Advanced Mathematical Programming

Leo Liberti[1]

[1] *CNRS LIX, École Polytechnique, F-91128 Palaiseau, France*
Email:liberti@lix.polytechnique.fr

February 17, 2022

Contents

Part I

Setting the scene

Chapter 1

Introduction

The Dean, in his office, is receiving a delegation from a prominent firm, with the aim of promoting a partnership between the two institutions. The goal is to pair a technical contact from the firm with a faculty member from the university. The hope is that the two will talk and find something interesting to do together. This might induce the firm to give some money to the university. It's a well-tested endeavour, which invariably ends up generating a "framework contract" (which mentions no money whatsoever), followed, sometimes, by more substantial contracts.

There's a physics professor who extols the virtues of a new revolutionary battery for storing energy. There's a mathematics professor who entered the now burgeoning field of "data science" and talks about how it took them two years to "clean" a certain vendor database, after which they could apparently forecast all future sales to a 1% precision. Another professor from the computer science department says "me too". When it's my turn, as usual, I mentally groan, and brace for impact.

— And now we come to Prof. Liberti, says the Dean with his usual sneer, for he's heard the communication failure of my trade too many times for him to be optimistic about it.

— We are listening, Prof. Liberti, says the CTO of the firm, tell us what you do. We're sure your research will be very useful to our firm.

— Well, ehm. OK. What I do is called "mathematical programming". It is a language for describing and then solving optimization problems.

— Interesting. And tell me, Prof. Liberti: what type of problems do you work on?

— All, really. Any optimization problem can be modelled as a mathematical program.

— Yes, yes of course. But could you tell us exactly *what* this mathematical thingy can be applied to?

— But this is exactly what I'm trying to say: it is a language, you can use it to describe *any* problem! So the application field really does not matter!

This can go on and on, with the industry leader who tries to extract an application field name out of me (optimal power flow in electrical grids, vehicle routing on road/rail networks, economic planning, shortest paths in urban settings, facility location in view of minimizing costs, crew rostering in the airline business, pricing of goods and services based on market demand, mixing and transforming materials, scheduling of tasks on processors, packing of crates/boxes in variously shaped containers, packing of said containers in a ship, docking the damn ships in the port, strategic decisions to address market segmentation, retrieving a clean signal from a noisy one, assigning personnel to different jobs, analysing data by clustering points, finding the shape of proteins and other molecules, aligning strands of RNA or DNA so that they match as closely as possible, and many, many more), while I attempt to wiggle out of having an application

label slapped on my work.

Once, when I was younger and more naive, I figured I might as well make the industrialist happy. During one of these meetings, after a couple of iterations of "what application field do you work on?" followed by "this is really the wrong question", I had decided to break the monotony and hopelessness of the exchange by stating I work on shortest paths in road networks.

— Ah, this is fantastic! came the reply, but unfortunately we don't do any of that.

This had two adverse effects: the first, and less serious, was that the industrialist did not wish to work with me. The second, much more damaging, was that I had inadvertently convinced the Dean that I worked on shortest paths. So he never invited me to any further industry meeting unless the words "shortest paths" appeared prominently in the firm's product descriptions.

On the other hand, the "stalling choice" I adopt today, while it keeps me invited to meetings with the Dean, is no less disastrous with industry leaders. At one point I thought to myself, I'll try and collaborate with firms which already employ an operations research team, who'll be obviously familiar with mathematical programming. So, additionally to the "Dean scenario", I sometimes also went to visit operations research teams in large firms directly. This approach elicited the polite answer "thank you, but we already know how to do that", followed by vague promises of future possible scientific collaborations, never actually pursued.

When the top boss of a large firm, which includes an operations research team internally, asks my Dean about mathematical programming or operations research, I immediately get called, and the meeting itself is usually successful. Unfortunately, however, the very need for hiring outside hands instead of relying on the internal team means that there is a disagreement between the latter and their top boss. I am henceforth directed to working with the internal team, which means that I rely on their availability for data, explanations, and so on; and then I am supposed to deliver what they could not. Quite aside from the difficulty of producing better results than a whole team of specialists, the real issue is that the internal team have all the motivations to sabotage the outsider's work, so it takes fairly advanced diplomatic skills to actually produce something.

To recap: mathematical programming is a general framework for describing and solving optimization problems. It is not only theoretical: quite on the contrary, it applies to practically everything. Its methods are used in engineering, chemistry, biology, mathematics, economics, psychology and even some social and human sciences. As far as communication is concerned, this polyvalence is its worst weakness. The generalization and efficiency of some mathematical programming algorithms, on the other hand, boggle the mind: it seems they can solve every optimization problem under the sun — which is exactly what they are meant to do.

1.1 Some easy examples

Look at the list of problems below. We start from a bank and its investment strategy. There follows a refinery wishing to make fuels from a blend of crudes. We then look at pairing jobs to machines in a production environment. We continue with a firm deciding some expenses to satisfy demands. Then there is a firm planning production and storage to match sales forecasts. Then a network operator deciding its data routing on two possible backbone links. Then a firm confronted with hiring and training decisions in view of demands. Then a hospital rostering nurses. The ninth problem requires decision on three issues: how to cover a set of customers with facilities, how to deal with a one-time cost, and how to transport the goods from the built facilities to the customers. Then there is a beer crate packing operation. Lastly, we look at a protein structure problem.

1. *Investments.* A bank needs to invest C gazillion dollars, and focuses on two types of investments: one, imaginatively called (a), guarantees a 15% return, while the other, riskier and called, surprise

surprise, (b), is set to a 25%. At least one fourth of the budget C must be invested in (a), and the quantity invested in (b) cannot be more than double the quantity invested in (a). How do we choose how much to invest in (a) and (b) so that revenue is maximized?

2. *Blending.* A refinery produces two types of fuel by blending three types of crude. The first type of fuel requires at most 30% of crude 1 and at least 40% of crude 2, and retails at 5.5EUR per unit[1] of volume. The second type requires at most 50% of crude 1 and at least 10% of crude 2, and retails at 4.5EUR. The availability of crude 1 is 3000 units, at a unit cost of 3EUR; for crude 2 we have 2000 units and a unit cost of 6EUR; for crude 3, 4000 and 4EUR. How do we choose the amounts of crude to blend in the two fuels so as to maximize net profit?

3. *Assignment.* There are n jobs to be dispatched to m identical machines. The j-th job takes time p_j to complete. Jobs cannot be interrupted and resumed. Each machine can only process one job at a time. Assign jobs to machines so the whole set of jobs is completed in the shortest possible time.

4. *Demands.* A small firm needs to obtain a certain number of computational servers on loan. Their needs change every month: 9 in January, 5 in February, 7 in March, 9 in April. The loan cost depends on the length: 200EUR for one month, 350 for two, and 450 for three. Plan the needed loans in the cheapest possible way.

5. *Multi-period production.* A manufacturing firm needs to plan its activities on a 3-month horizon. It can produce 110 units at a cost of 300$ each; moreover, if it produces at all in a given month, it must produce at least 15 units per month. It can also subcontract production of 60 supplementary units at a cost of 330$ each. Storage costs amount to 10$ per unit per month. Sales forecasts for the next three months are 100, 130, and 150 units. Satisfy the demand at minimum cost.

6. *Capacities.* A total of n data flows must be routed on one of two possible links between a source and a target node. The j-th data flow requires c_j Mbps to be routed. The capacity of the first link is 1Mbps; the capacity of the second is 2Mbps. Routing through the second link, however, is 30% more expensive than routing through the first. Minimize the routing cost while respecting link capacities.

7. *Demands, again.* A computer service firm estimates the need for service hours over the next five months as follows: 6000, 7000, 8000, 9500, 11000. Currently, the firm employs 50 consultants: each works at most 160 hours/month, and is paid 2000EUR/month. To satisfy demand peaks, the firm must recruit and train new consultants: training takes one month, and 50 hours of supervision work of an existing consultant. Trainees are paid 1000EUR/month. It was observed that 5% of the trainees leave the firm for the competition at the end of training. Plan the activites at minimum cost.

8. *Rostering.* A hospital needs to roster nurses: each can work 5 consecutive days followed by two days of rest. The demand for each day of the week (mon-sun) are: 11, 9, 7, 12, 13, 8, 5. Plan the roster in order to minimize the number of nurses.

9. *Covering, set-up costs and transportation.* A distribution firm has identified n candidate sites to build depots. The i-th candidate depot, having given capacity b_i, costs f_i to build (for $i \leq n$). There are m stores to be supplied, each having a minimum demand d_j (for $j \leq n$). The cost of transporting one unit of goods between depot i and store j is c_{ij}. Plan openings and transportation so as to minimize costs.

10. *Circle packing.* Maximize the number of cylindrical crates of beer (each having 20cm radius) which can be packed in the carrying area (6m long and 2.5m wide) of a pick-up truck.

11. *Molecular distance geometry.* A protein with n atoms is examined with a nuclear magnetic resonance experiment, which determines all and only the distances d_{ij} between the pairs of atoms closer than 5Å. Decide the atomic positions that best satisfy these distance data.

[1] Speaking of "units" in this context is not an implicit reference to an integrality constraints: one might well want to acquire a fractional number of units.

1.2 Real vs. didactical problems

What is the the most prominent common feature of the problems in Sect. 1.1? To a student, they possibly contribute to raise a state of mind between boredom and despair. To a professor, it is the fact that they can all be formulated by means of mathematical programming. To a practitioner, the fact that they are all invented for teaching purposes. No real world problem ever presents itself as clearly as the problems above.

Should any reader of this text ever become a mathematical programming consultant, let her or him be aware of this fact: on being assigned a task by your client, you shall be given almost no explanation and some incomprehensible data. The sketch below is sadly much closer to the truth than one might imagine.

— Dr. Liberti, thank you for visiting the wonderful, magnificent firm I preside. I'll come straight to the point: we have a problem we want you to solve.

— Why, certainly sir; what is the problem?

— Exactly this thing we want you to solve.

— I understand you want me to solve the problem; but what *is* it?

— My dear fellow, if we knew what the problem was, we very possibly wouldn't need your help! Now off you go and do your mathemathingy trick or whatever it is that you do, and make us rich! Oh and by the way we'll try and withhold any payment for your services with any excuse we deem fit to use. And if forced by law to actually pay you (God forbid!), we'll at least delay the payment indefinitely. You won't mind, will you?

Although it is usually not quite as bad, the "definition" of a problem, in an industrialist's view, is very often a partial set of database tables, with many essential columns missing for nondisclosure (or forgetfulness, or loss) purposes, accompanied by an email reading more or less as follows. *These 143 tables are all I found to get you going. I don't have any further time for you over the next three months. I don't know what the tables contain, nor what most of the column labels mean, but I'm almost sure that the column labeled* `t5y/3R` *denotes a complicated function – I forget which one – of the estimated contribution of the product indexed by the row to our fiscal imposition in 2005, scaled by a 2.2% inflation, well more or less. Oh, and we couldn't give you the unit costs or profits, since of course they are confidential. Your task is to optimize our revenues. You have three days. Best of luck.*

Coming back to the common features of the list of problems in Sect. 1.1, the majority concern industry. This is not by chance: most of the *direct* practical value provided by mathematical programming is to technological processes. In the last twenty years, however, the state-of-the-art has advanced enough so that mathematical programming methods now feature prominently as substeps of complex algorithmic frameworks designed to address systemic issues. This provides an indirect, though no less important, value to the real world.

1.3 Solutions of the easy problems

We now provide solutions for all of the problems listed in Sect. 1.1. Some of these solutions are heavily commented: I am hoping to address many of the issues raised by students when they learn how to model by mathematical programming.

1.3.1 Investments

We start with a simplistic bank wishing to invest an unknown, but given, budget C in two types of investments, (a) and (b). Let x_a denote the part of the budget C invested in (a), and x_b the same for (b): their relation is $x_a + x_b = C$. The revenue is $1.15x_a + 1.25x_b$. The "technical constraints" require that $x_a \geq \frac{1}{4}C$ and $x_b \leq 2x_a$. This might appear to be all: written properly, it looks like the following formulation.

$$\left.\begin{array}{rrcl} \max\limits_{x_a,x_b} & 1.15x_a + 1.25x_b & & \\ & x_a + x_b & = & C \\ & x_a & \geq & \frac{1}{4}C \\ & 2x_a - x_b & \geq & 0. \end{array}\right\}$$

1.3.1.1 Missing trivial constraints

Now, it is easy to see that $x_a = C + 1$ and $x_b = -1$ satisfies all the constraints, but what is the meaning of a negative part of a budget? This should make readers realize we had left out an apparently trivial, but crucial piece of information: $x_a \geq 0$ and $x_b \geq 0$.

Like all programming, mathematical programming is also subject to bugs: in our brains, the "part of a budget" is obviously nonnegative, but this is a meaning corresponding only to the natural language description of the problem. When we switch to formal language, anything not explicitly stated is absent: everything must be laid out explicitly. So the correct formulation is

$$\left.\begin{array}{rrcl} \max\limits_{x_a,x_b} & 1.15x_a + 1.25x_b & & \\ & x_a + x_b & = & C \\ & x_a & \geq & \frac{1}{4}C \\ & 2x_a - x_b & \geq & 0 \\ & x_a, x_b & \geq & 0. \end{array}\right\}$$

We remark that writing $x_a, x_b \geq 0$ (which is formally wrong) is just a short-hand for its correct counterpart $x_a \geq 0 \wedge x_b \geq 0$. Variable restrictions can also be stated under the minimum operator:

$$\max_{x_a,x_b \geq 0} 1.15x_a + 1.25x_b.$$

1.3.1.2 No numbers in formulations

Something else we draw from computer programming is that good coding practices forbid the use of numerical constants within the code itself: they should instead appear at the beginning of the coding section where they are used. Correspondingly, we let $c_a = 1.15$, $c_b = 1.25$, $p = \frac{1}{4}$ and $d = 2$, and rewrite the formulation as follows:

$$\left.\begin{array}{rrcl} \max\limits_{x_a,x_b \geq 0} & c_a x_a + c_b x_b & & \\ & x_a + x_b & = & C \\ & x_a & \geq & pC \\ & dx_a - x_b & \geq & 0. \end{array}\right\} \tag{1.1}$$

1.3.1.3 Formulation generality

Most mathematical writing attempts to set the object of interest in the most general setting. What if we had n possible investments rather than only two? We should rename our variables $x_1, \ldots, x_n$, and our

returns $c_1, \ldots, c_n$, yielding:

$$\left.\begin{array}{rrcl} \max\limits_{x \geq 0} & \sum\limits_{j \leq n} c_j x_j & & \\ & \sum\limits_{j \leq n} x_j & = & C \\ & x_1 & \geq & pC \\ & dx_1 - x_2 & \geq & 0, \end{array}\right\} \tag{1.2}$$

a formulation which generalizes Eq. (1.1) since the latter is an instance of Eq. (1.2) where $n = 2$. In particular, Eq. (1.1) is the type of formulation that can be an input to a mathematical programming solver (once C is fixed to some value), since it involves a list of (scalar) variables, and a list of (single-row) constraints. Such formulations are called *flat*. On the other hand, Eq. (1.2) involves a variable vector x (of unspecified length n) with some components x_j: it needs to be *flattened* before it can be passed to a solver. This usually involves fixing a parameter (in this case n) to a given value (in this case, 2), with all the consequences this entails on the formulation (see Sect. 2.2.6). Formulations involving parameter symbols and quantifiers are called *structured*.

1.3.1.4 Technical constraints

Lastly, there are two reasons why I called the last two constraints "technical". The first has to do with the application field: they provide a way to limit the risk of pledging too much money to the second investment. They would not be readily explained in a different field (whereas the objective maximizes revenue, something which is common to practically all business-related decisions). The second is that they cannot be generalized with the introduction of the parameter n: they still only concern the first two investments. Of course, had the problem been described as "the quantity invested in anything different from (a) cannot be more than double the quantity invested in (a)", then the corresponding formulation would have been:

$$\left.\begin{array}{rrcl} \max\limits_{x \geq 0} & \sum\limits_{j \leq n} c_j x_j & & \\ & \sum\limits_{j \leq n} x_j & = & C \\ & x_1 & \geq & pC \\ \forall j \in \{2, \ldots, n\} & dx_1 - x_j & \geq & 0, \end{array}\right\}$$

an even "more structured" formulation than Eq. (1.2) since it also involves a constraint vector (the last line in the formulation above) including $n - 1$ single-row constraints.

How far is it reasonable to generalize formulations? It depends on the final goals: in the real world of production, one is usually confronted with the problem of improving an existing system, so some of the sizes (e.g. the number of machines) might be fixed, while others (e.g. the demands, which might vary by day or month) are not.

1.3.2 Blending

We come to the world of oil production: when we refuel our vehicles, we are actually buying a blend of different crudes with different qualities. In Sect. 2.2.7.7 we shall look at blending operations with a decision on based on the qualities. Here we look at a simple setting where we have prescriptions on the fraction of each crude to put in the blend for a particular fuel.

1.3.2.1 Decision variables

Although I did not stress this fact in Sect. 1.3.1, the first (and most critical) decision to make when modelling a problem is to define the *decision variables*. In Sect. 1.3.1 it was quite obvious we needed

to decide the parts of budget x_a, x_b, so there was no need to reflect on the concept. Here, however, the natural language description of this problem is sufficiently fuzzy for the issue to deserve some remarks.

- The process of turning a problem description from natural language to formal language is called *modelling*. It involves human intelligence. So far, no-one has ever produced a computer program that is able to automate this task. The difficulty of modelling is that natural language is ambiguous. The natural language description is usually interpreted in each stakeholder's cultural context. When working with a client in the real world, the modelling part might well take the largest part of the meeting time.

- For most problems arising in a didactical setting, my advice is: start modelling by deciding a semi-formal meaning for the decision variables; then try and express the objective function and constraints in terms of these variables. If you cannot, or you obtain impossibly nonlinear expressions, try adding new variables. If you still fail, go back and choose a different set of decision variables. One is generally better off modelling structured formulations rather than flat formulations: so, while you decide the variables, you also have to decide how they are indexed, in what sets these indices range, and what parameters appear in the formulation.

- For real world problems, where information is painstakingly collected during many meetings with the client, the first task is to understand what the client has in mind. Usually, the client tries to describe the whole system in which she works. Naturally, she will give more information about the parts of the system which, to her mind, are most problematic. On the other hand, it might well be that the problematic parts are only symptoms of an issue originating because poor decisions are being taken elsewhere in the system. When you think you have a sufficiently clear picture of the whole system (meaning the interactions between all the parts, and every entity on which decisions can be taken), then you can start modelling as sketched above (think of decision variables first).

In the present case, the hint about "what decision variables to consider" is given in the final question "how do we choose the amounts of crude to blend in the two fuels?" So each choice must refer to crudes and fuels: as such, we need variables indexed over crudes and over fuels. Let us define the set of C of crudes, and the set F of fuels, and then decision variables x_{ij} indicating the fraction of crude $i \in C$ in fuel $j \in F$. What else does the text of the problem tell us? We know that each fuel $j \in F$ has a retail price, which we shall call r_j, and that each crude $i \in C$ has an availability in terms of units, which we shall call a_i, and a unit cost, called c_i.

1.3.2.2 Parameters

The other numeric information given in the text concerns upper or lower bounds to the amount of crude in each fuel. Since we are employing decision variables x_{ij} to denote this amount, we can generalize these lower and upper bounds by a set of intervals $[x^L_{ij}, x^U_{ij}]$, which define the ranges $x^L_{ij} \leq x_{ij} \leq x^U_{ij}$ of each decision variable. Specifically, we have $x^U_{11} = 0.3$, $x^L_{21} = 0.4$, $x^U_{12} = 0.5$ and $x^L_{22} = 0.1$. Unspecified lower bounds must be set to 0 and unspecified upper bounds to 1, since x_{ij} denotes a fraction. We remark that x^L, x^U, r, a are vectors of parameters, which, together with the index sets C, F, will allow us to write a structured formulation.

1.3.2.3 Objective function

Let us see whether this choice of decision variables lets us easily express the objective function: the text says "maximize the net profit". A net profit is the difference between revenue and cost. We have:

$$\begin{aligned} \text{revenue} &= \sum_{j \in F} r_j \, \text{fuel}_j \\ \text{cost} &= \sum_{i \in C} c_i \, \text{crude}_i. \end{aligned}$$

We have not defined decision variables denoting the amounts of produced fuels and crudes used for production. But these can be written in terms of the decision variables x_{ij} as follows:

$$\begin{aligned} \forall j \in F \quad \text{fuel}_j &= \sum_{i \in C} a_i x_{ij} \\ \forall i \in C \quad \text{crude}_i &= a_i \sum_{j \in F} x_{ij}. \end{aligned}$$

So, now, the objective function is:

$$\max \left(\sum_{j \in F} r_j \sum_{i \in C} a_i x_{ij} - \sum_{i \in C} c_i a_i \sum_{j \in F} x_{ij} \right).$$

We can rewrite the objective more compactly as follows:

$$\begin{aligned} & \max \left(\sum_{j \in F} r_j \sum_{i \in C} a_i x_{ij} - \sum_{i \in C} c_i a_i \sum_{j \in F} x_{ij} \right) = \\ = \quad & \max \left(\sum_{\substack{i \in C \\ j \in F}} a_i r_j x_{ij} - \sum_{\substack{i \in C \\ j \in F}} a_i c_i x_{ij} \right) = \\ = \quad & \max \sum_{\substack{i \in C \\ j \in F}} a_i (r_j - c_i) x_{ij}. \end{aligned}$$

1.3.2.4 Constraints

What about the constraints? As mentioned already, we have range constraints, which we express in tensor form:

$$x \in [x^L, x^U].$$

(The scalar form would need a quantification: $\forall i \in C, j \in F \; x^L_{ij} \leq x_{ij} \leq x^U_{ij}$).

Any other constraint? As in Problem 1 of Sect. 1.1, there is a risk of a bug due to a forgotten trivial constraint: we know that x_{ij} are supposed to indicate a fraction of crude $i \in C$ over all $j \in F$. So, for all $j \in F$, the sum of the fractions cannot exceed 1:

$$\forall j \in F \quad \sum_{i \in C} x_{ij} \leq 1.$$

It would not be wrong to opt for an equality constraint instead of an inequality. From the mathematical point of view, both options give rise to the same optimal objective function value, since maximization tends to decrease the difference RHS − LHS to zero by increasing the x_{ij} as much as possible. From a modelling point of view, we are told that x_{ij} are fractions, but we are never told that the fuels might not

contain other components than the crudes (e.g. stabilizing or anti-freezing agents), so modelling with $\leq$ is a safer options in case this formulation is a part of a larger, more complicated formulation.

Hence, the complete formulation is as follows:

$$\left.\begin{array}{lll} \max\limits_{0\leq x\leq 1} & \sum\limits_{\substack{i\in C\\ j\in F}} a_i(r_j-c_i)x_{ij} & \\ \forall j\in F & \sum\limits_{i\in C} x_{ij} \leq 1 & \\ & x \in [x^L, x^U]. & \end{array}\right\}$$

1.3.1 Exercise

Propose a formulation of this problem based on decision variables y_{ij} denoting the total amount (rather than the fraction) of crude i in fuel j.

Answer. Here it is:

$$\left.\begin{array}{lll} \max\limits_{0\leq y\leq a} & \sum\limits_{\substack{i\in C\\ j\in F}} (r_j-c_i)y_{ij} & \\ \forall i\in C & \sum\limits_{j\in F} y_{ij} \leq a_i & \\ \forall i\in C, j\in F & y_{ij} \in [a_i x^L_{ij}, a_i x^U_{ij}]. & \end{array}\right\}$$

1.3.2.5 Ambiguities in the text description

Unfortunately, whether the problem is didactical or from the real world, human communication is given in natural language, which is prone to ambiguities. Sometimes "almost the same text" yields important, or even dramatic differences in the formulation, as Exercise 1.3.2 shows. If you are working with a client, if in doubt ask — do not be afraid of being considered an idiot for asking multiple times: you will certainly be considered an idiot if you produce the wrong formulation. If the setting is didactical, perhaps the ambiguity is desired, and you have to deal with it as well as you can (you might be judged exactly for the way you dealt with a textual ambiguity).

1.3.2 Exercise

How would the formulation change if, instead of saying "fuel i requires at most/least a given fraction of crude i", the problem said "the amount of crude i must be at most/least a given fraction of fuel j"?

Answer. Denoting the given fractions with x^L, x^U as above, the range constraints $x^L \leq x \leq x^U$ would need to be changed into:

$$\begin{array}{lll} \forall i\in C, j\in F & x_{ij} \leq & x^U_{ij}\,\mathrm{fuel}_j \\ \forall i\in C, j\in F & x_{ij} \geq & x^L_{ij}\,\mathrm{fuel}_j, \end{array}$$

where fuel_j can be expressed in terms of the decision variables x_{ij} as above.

Wrong: use
fractions, a
of quantitie

1.3.2.6 Unused textual elements

Recall that the text mentioned three crudes. Although our structured formulations are invariant to the actual number of crudes, the data are such that the third crude is completely irrelevant to the problem. This is a typical feature of modelling real world problems: some (much?) of the information given to the modeller turns out to be irrelevant. The issue is that one may only recognize irrelevance *a posteriori*. During the modelling process, irrelevant information gives a nagging sensation of failure, which a good modeller must learn to recognize and ignore.

In fact, recognizing data irrelevance often provides a valuable feedback to the client: they may use this information in order to simplify and rationalize the dynamics of their system processes. Mostly, the reason why some data are irrelevant, and yet given to the modeller, is that, historically, those data were once relevant. In the present case, perhaps the firm once used all three crude types to produce other types of fuels. Some fuel types may have been discontinued to various reasons, but the database system remained unchanged.

1.3.3 Assignment

When hearing talk of "jobs" and "machines" (or "tasks" and "processors"), a mathematical programmer's brain pathways immediately synthesize the concept of "scheduling". Scheduling problems consist of an assignment (of jobs to machines or tasks to processors) and of a linear order (on jobs on a given machine). But sometimes text descriptions can be misleading, and the problem be simpler than it might appear.

In this case we have identical machines, jobs cannot be interrupted/resumed, each machine can process at most one job at a time, and the requirement is to assign jobs to machines so that the set of jobs is completed "in the shortest possible time". The appearance of the word "shortest time" seems to require an order of the jobs on each machine: after all completing a set of jobs requires the maximum completion time of the *last job* over all machines.

But now suppose someone already gave you the optimal assignment of jobs to machines: then the working time of each machine would simply be the sum of the completion times of all the jobs assigned to the machine. And the completion times for all jobs would involve minimizing the working time of the slowest machine. This does not involve an order of the jobs assigned to each machine. The problem can therefore be more simply formulated by means of an assignment.

1.3.3.1 Decision variables

Assignment problems are among the fundamental problems which we can solve efficiently. You should therefore learn to recognize and formulate them without even thinking about it.

The beginner's mistake is to define two sets of binary variables x_i (relating to job i) and y_j (relating to machine j) and stating that $x_i y_j$ denotes the assignment of job i to machine j, because "$x_i y_j = 1$ if and only if both are 1". This is a mistake for at least two reasons:

- the definition of a binary decision variable should reflect a boolean condition, whereas here the condition (the assignment of i to j) is related to the product of two variables;
- introducing a product of variables in the formulation introduces a nonlinearity, which yields more difficult formulations to solve. While sometimes nonlinearity is necessary, one should think twice (or more) before introducing it in a formulation.

Nonetheless, the first reason points us towards the correct formulation. Since the boolean condition is "whether job i is assigned to machine j", we need binary decision variables indexed on the pair i, j. So we introduce the index sets J of jobs and the set M of machines, and decision variables $x_{ji} \in \{0, 1\}$ for each $j \in J$ and $i \in M$. The only other formulation parameters are the completion times p_j for each job $j \in J$.

1.3.3.2 Objective function

We can write the working time μ_i of machine $i \in M$ as the sum of the completion times of the jobs assigned to it:

$$\forall i \in M \quad \mu_i = \sum_{j \in J} p_j x_{ji}. \tag{1.3}$$

Now the objective function minimizes the maximum μ_i:

$$\min_{\mu, x} \max_{i \in M} \mu_i.$$

Several remarks are in order.

- New decision variables μ_i (for $i \in M$) were "surreptitiously" introduced. This is a helpful trick in order to help us cope with increasing formulation complexity while modelling. It is clear that μ_i can be eliminated by the formulation by replacing it with the right-hand side (RHS) of Eq. (1.3), so they are inessential. They simply serve the purpose of writing the formulation more clearly.
- The function in Eq. (1.3.3.2) involves both a minimization and a maximization; at first sight, it might remind the reader of a a saddle point search. On closer inspection, the minimization operator occurs over decision variables, but the second only occurs over the indices of a given set: so the maximization does not involve an optimization procedure, but simply the choice of maximum value among $|M|$. It simply means that the function being optimized is $\max_{i \in M} \mu_i$. Such a function is piecewise linear and concave.
- Eq. (1.3.3.2) can be reformulated to a purely linear form by means of an additional decision variable t, as follows:

$$\min_{\mu,x,t} \quad t \tag{1.4}$$

$$\forall i \in M \quad t \quad \geq \mu_i. \tag{1.5}$$

It is clear that t will be at least as large as the maximum μ_i, and by minimizing it we shall select the "min max μ".

1.3.3.3 Constraints

The unmentioned constraint implicit in the term "assignment" is that each job j is assigned to exactly one machine i. This is written formally as follows:

$$\forall j \in J \quad \sum_{i \in M} x_{ji} = 1. \tag{1.6}$$

Eq. (1.6) is known as an *assignment constraint*. It does its work by stating that exactly one variable in the set $\{x_{ji} \mid i \in M\}$ will take value 1, while the rest will take value 0. All that is left to do is to state that the variables are binary:

$$\forall j \in J, i \in M \quad x_{ji} \in \{0,1\}. \tag{1.7}$$

Eq. (1.7), known as *boolean constraint*, are part of the wider class of *integrality constraints*.

The whole formulation can now be written as follows.

$$\left.\begin{array}{rrcl} \min\limits_{x,t} & t & & \\ \forall i \in M & \sum\limits_{j \in J} p_j x_{ji} & \leq & t \\ \forall j \in J & \sum\limits_{i \in M} x_{ji} & = & 1 \\ & x & \in & \{0,1\}^{|J||M|}. \end{array}\right\}$$

1.3.4 Demands

There are no new difficulties with this problem. The only thing worth pointing out is that we have $L \subset T$, and therefore we can perform arithmetic operations on indices, as both indices in L and T point to months.

1. **Index sets**
   (a) T: set of month indices ($\{1,2,3,4\}$)

   (b) L: set of loan indices ($\{1, 2, 3\}$)

2. **Parameters**

   (a) $\forall t \in T$ let d_t = demand in month t

   (b) $\forall \ell \in L$ let c_ℓ = cost of loan length ℓ

3. **Decision variables**
   $\forall t \in T, \ell \in L$ let $x_{t\ell}$ = number of servers loaned for ℓ months in month t

4. **Objective function**: $\min \sum_{\ell \in L} c_\ell \sum_{t \in T} x_{t\ell}$

5. **Constraints**
   demand satisfaction: $\forall t \in T \sum_{\substack{\ell \in L \\ \ell \leq t}} x_{t-\ell+1,\ell} \geq d_t$.

Again we emphasize that variables indexed by time involve some index arithmetic (see $x_{t-\ell+1,\ell}$ in the demand satisfaction constraint).

1.3.5 Multi-period production

1. **Index sets**

   (a) T: set of month indices ($\{1, 2, 3\}$)

   (b) $T' = T \cup \{0\}$

2. **Parameters**

   (a) $\forall t \in T$ let f_t = sales forecasts in month t

   (b) let p = max normal monthly production

   (c) let r = max subcontracted monthly production

   (d) let d = min monthly production level in normal production

   (e) let c^{normal} = unit cost in normal production

   (f) let c^{sub} = unit cost in subcontracted production

   (g) let c^{store} = unit storage cost

3. **Decision variables**

   (a) $\forall t \in T$ let x_t = units produced in month t

   (b) $\forall t \in T$ let $w_t = 1$ iff normal production active in month t, 0 otherwise

   (c) $\forall t \in T$ let y_t = units subcontracted in month t

   (d) $\forall t \in T'$ let z_t = units stored in month t

4. **Objective function**: $\min c^{\text{normal}} \sum_{t \in T} x_t + c^{\text{sub}} \sum_{t \in T} y_t + c^{\text{store}} \sum_{t \in T} z_t$

5. **Constraints**

   (a) demand satisfaction: $\forall t \in T \; x_t + y_t + z_{t-1} \geq f_t$

   (b) storage balance: $\forall t \in T \; x_t + y_t + z_{t-1} = z_t + f_t$

   (c) max production capacity: $\forall t \in T \; x_t \leq p$

   (d) max subcontracted capacity: $\forall t \in T \; x_t \leq r$

   (e) storage is empty when operation starts: $z_0 = 0$

(f) minimum normal production level:

$$\forall t \in T \; x_t \geq dw_t \tag{1.8}$$
$$\forall t \in T \; x_t \leq pw_t. \tag{1.9}$$

(g) integrality constraints: $\forall t \in T \; w_t \in \{0, 1\}$

(h) non-negativity constraints: $\forall t \in T \; x, y, z \geq 0$.

We remark that Eq. (1.8)-(1.9) ensure that $x_t \geq 0$ iff $w_t = 1$, and, conversely, $x_t = 0$ iff $w_t = 0$ (also see Sect. 2.2.7.5.1).

1.3.6 Capacities

Note that the text hints to an assignment of data flows to links, as well as to capacity constraints.

1. **Index sets**
   (a) F: set of data flow indices
   (b) L: set of links joining source an target
2. **Parameters**
   (a) $\forall j \in F$ let c_j = required capacity for flow j
   (b) $\forall i \in L$ let k_i = capacity of link i
   (c) $\forall i \in L$ let p_i = cost of routing 1Mbps through link i
3. **Decision variables**
   (a) $\forall i \in L, j \in F$ let $x_{ij} = 1$ iff flow j assigned to link i, 0 otherwise
4. **Objective function**: $\min \sum_{i \in L} p_i \sum_{j \in F} c_j x_{ij}$
5. **Constraints**
   (a) assignment: $\forall j \in F \; \sum_{i \in L} x_{ij} = 1$
   (b) link capacity: $\forall i \in L \; \sum_{j \in F} c_j x_{ij} \leq k_i$

1.3.7 Demands, again

1. **Index sets**
   (a) T: set of month indices
   (b) $T' = T \cup \{0\}$
2. **Parameters**
   (a) $\forall t \in T$ let d_t = service hour demands for month t
   (b) σ = starting number of consultants
   (c) ω = hours/month worked by each consultant
   (d) γ = monthly salary for each consultant
   (e) τ = number of hours needed for training a new consultant

(f) δ = monthly salary for trainee

(g) p = percentage of trainees who leave the firm after training

3. **Decision variables**

(a) $\forall t \in T$ let x_t = number of trainees hired at month t

(b) $\forall t \in T$ let y_t = number of consultants at month t

4. **Objective function**: $\min \gamma \sum_{t \in T} y_t + \tau \sum_{t \in T} x_t$

5. **Constraints**

(a) demand satisfaction: $\forall t \in T \ \omega y_t - \tau x_t \geq d_t$

(b) active consultants: $\forall t \in T \ y_{t-1} + (1-p)x_{t-1} = y_t$
note that $(1-p)x_{t-1}$ may not be integer even if x_{t-1} is, which means that in order to satisfy this equation the solver may need to increase x, y, exceeding the demands; the equation can always be satisfied as long as $p \in \mathbb{Q}$, of course (why?), but it may fail to be satisfied otherwise — think about a possible way out of this issue

(c) boundary conditions: $y_0 = \sigma$, $x_0 = 0$

(d) integrality and nonnegativity: $x, y \in \mathbb{Z}_+$.

We remark that nowhere does the text state that $x_0 = 0$. But this is a natural condition in view of the fact that hiring does not occur before the need arises, i.e. before the start of the planning horizon T.

1.3.8 Rostering

The text of this problem says nothing about the total number of nurses. In real life, the output of a rostering problem should be a timetable stating that, e.g., nurse John works from Monday to Friday, nurse Mary from Tuesday to Saturday, and so on. In this sense, the timetable assigns nurses to days in a similar way as scheduling assigns jobs to machines. But in this didactical setting the modeller is asked to work without the set of nurses. The trick is to define variables for the number of nurses starting on a certain day. This provides a neat way for decomposing the assignment problem from the resource allocation.

1. **Index sets**

(a) $T = \{0, \dots, |T| - 1\}$: set of days indices (periodic set)

2. **Parameters**

(a) $\forall t \in T$ let d_t = demand for day t

(b) let α = number of consecutive working days

3. **Decision variables**

(a) $\forall t \in T$ let x_t = number of nurses starting on day t

4. **Objective function**: $\min \sum_{t \in T} x_t$

5. **Constraints**

(a) resource allocation: $\forall t \in T \ \sum_{i=0}^{\alpha-1} x_{t+i} \geq d_t$

- Does it make sense to generalize the "days of the week" to a set T? After all, there will never be weeks composed of anything other than seven days. However, some day an administrator might plan the rostering over two weeks, or any other number of days. Moreover, this allows writing indexed variables (such as x_t for $t \in T$) rather than "flat" variables (e.g. a for Monday, b for Tuesday, and so on).
- The set T is interpreted as being a periodic set: it indexes $\{\text{Monday}, \text{Tuesday}, \ldots, \text{Sunday}\}$, where "Monday" obviously follows "Sunday": this is seen in the resource allocation constraints, where $t+i$ must be interpreted as modular arithmetic modulo $|T|$.

1.3.9 Covering, set-up costs and transportation

This problem embodies three important features seen in many real-world supply chain problems: covering a set of stores with facilities, set-up costs (already seen in Sect. 1.3.5, and modelled with the binary variables w), and transportation. While we shall look at covering in Sect. 2.2.7.4 and transportation in Sect. 2.2.7.2, this problem is typical in that it mixes these aspects together. Most real-world problems I came across end up being modelled by combining variables and constraints from classic examples (covering, packing, network flow, transportation, assignment, ordering and so on) in a smart way.

1. **Index sets**
   (a) $N = \{1, \ldots, n\}$: index set for candidate sites
   (b) $M = \{1, \ldots, m\}$: index set for stores
2. **Parameters**
   (a) $\forall i \in N$ let b_i = capacity of candidate depot i
   (b) $\forall i \in N$ let f_i = cost of building depot i
   (c) $\forall j \in M$ let d_j = minimum demand for store j
   (d) $\forall i \in N, j \in M$ let c_{ij} = cost of transporting one unit from i to j
3. **Decision variables**
   (a) $\forall i \in N, j \in M$ let x_{ij} = units transported from i to j
   (b) $\forall i \in N$ let $y_i = 1$ iff depot i is opened, 0 otherwise
4. **Objective function**: $\min \sum_{i \in N, j \in M} c_{ij} x_{ij} + \sum_{i \in N} f_i y_i$
5. **Constraints**
   (a) facility capacity: $\forall i \in N \sum_{j \in M} x_{ij} \leq b_i$
   (b) facility choice: $\forall i \in N, j \in M \; x_{ij} \leq b_i y_i$ (also see Sect. 2.2.7.5.1)
   (c) demand satisfaction: $\forall j \in M \sum_{i \in N} x_{ij} \geq d_j$
   (d) integrality: $\forall i \in N \; y_i \in \{0, 1\}$
   (e) nonnegativity: $\forall i \in N, j \in M \; x_{ij} \geq 0$

A few remarks are in order.

- The facility choice constraint forces x_{ij} to be zero (for all $j \in M$) if facility i is not built (i.e. $y_i = 0$); this constraint is inactive if $y_i = 1$, since it reduces to $x_{ij} \leq b_i$, which is the obvious production limit for candidate facility i (for $i \in N$).

- Facility capacity and demand satisfaction constraints encode the transportation problem within the formulation. Set-up costs are modelled using the binary variables y_i. Note that we do not need to ensure an "only if" direction on the relationship between x and y, i.e. if $y_i = 1$ we still allow $x_{ij} = 0$ (in other words, if a facility is built, it might not produce anything), since it is enforced by the objective function direction (a built facility which does not produce anything contradicts minimality of the objective, since otherwise we could set the corresponding y_i to zero and obtain a lower value). The covering aspect of the problem (i.e. selecting a minimum cardinality subset of facilities that cover the stores) is somewhat hidden: it is partly encoded in the term $\sum_i y_i$ of the objective (minimality of the covering subset), by the facility choice constraints (which let the y variables control whether the x variables are zero), and by the demand satisfaction constraints, ensuring that ever store is covered.

- An alternative formulation, with fewer constraints, can be obtained by combining facility capacity and facility choice as follows:
$$\forall i \in N \quad \sum_{j \in M} x_{ij} \leq b_i y_i, \tag{1.10}$$
and then removing the facility choice constraints entirely. The formulation was presented with explicit facility choice constraints because, in general, they are "solver-frendlier" (i.e. solvers work better with them). In this specific case, this may or may not be the case, depending on the instance. But suppose you knew the transportation capacity limits q_{ij} on each pair $i \in N, j \in M$. Then you could rewrite the facility choice constraints as:
$$\forall i \in N, j \in M \quad x_{ij} \leq q_{ij} y_i. \tag{1.11}$$
Obviously, the transportation capacities from each facility cannot exceed the production capacity, so $\sum_j q_{ij} \leq b_i$. This implies that by summing Eq. (1.11) over $j \in M$, we obtain Eq. (1.10). By contrast, we cannot "disaggregate" Eq. (1.10) to retrieve Eq. (1.11), which shows that the formulation using Eq. (1.11) is somehow tighter (more precisely, it yields a better continuous relaxation — we shall see this in Part III). Again, while this may not be the case in the formulation above (since we do not have the q_{ij} parameters as given), using facility choice constraints is nonetheless a good coding habit.

1.3.10 Circle packing

In this problem we have to make an assumption: i.e. that we know at least an upper bound ν to the maximum number of crates of beer. We can compute it by e.g. dividing the carrying area of the pick-up truck by the area of a circular base of the cylinder representing a beer crate.

1. **Index sets**
   (a) $N = \{1, \ldots, \nu\}$
2. **Parameters**
   (a) let r = radius of the circular base of the crates
   (b) let a = length of the carrying area of the pick-up truck
   (c) let b = width of the carrying area of the pick-up truck
3. **Decision variables**
   (a) $\forall i \in N$ let x_i = abscissa of the center of circular base of the i-th crate
   (b) $\forall i \in N$ let y_i = coordinate of the center of circular base of the i-th crate
   (c) $\forall i \in N$ let $z_i = 1$ iff the i-th crate can be packed, and 0 otherwise

4. **Objective function**: $\max \sum_{i \in N} z_i$

5. **Constraints**

   (a) packing (abscissae): $\forall i \in N \; rz_i \leq x_i \leq (a-r)z_i$

   (b) packing (coordinates): $\forall i \in N \; rz_i \leq y_i \leq (b-r)z_i$

   (c) non-overlapping: $\forall i < j \in N \; (x_i - x_j)^2 + (y_i - y_j)^2 \geq (2r)^2 z_i z_j$

   (d) integrality: $\forall i \in N \; z_i \in \{0, 1\}$

This problem is also known in combinatorial geometry as packing of equal circles in a rectangle. Note that any packing is invariant by translations, rotations and reflections. We therefore arbitrarily choose the origin as the lower left corner of the rectangle, and align its sides with the Euclidean axes. Note that the packing constraints ensure that either a circle is active, in which case it stays within the boundaries of the rectangle, or it is inactive (and its center is set to the origin) — also see Sect. 2.2.7.5.1. The non-overlapping constraints state that the centers of two active circles must be at least $2r$ distance units apart.

1.3.11 The distance geometry problem

See Ch. 10 for more information about this problem and its modelling by MP.

1. **Index sets**

   (a) V: set of atoms

   (b) E: set of unordered pairs of atoms at distance ≤ 5Å

2. **Parameters**

   (a) $\forall \{i, j\} \in E$ let d_{ij} = Euclidean distance between atoms i, j

   (b) let $K = 3$ be the number of dimensions of the Euclidean space we consider

3. **Decision variables**

   (a) $\forall i \in V, k \leq K$ let x_{ik} = value of the k-th component of the i-th atom position vector

4. **Objective function**: $\min \sum_{\{i,j\} \in E} (\|x_i - x_j\|_2^2 - d_{ij}^2)^2$

A few remarks follow.

- Notationwise, we write x_i to mean the vector $(x_{i1}, \ldots, x_{iK})$. This allows us to write the Euclidean distance between two vectors more compactly.
- Although the distance measure 5Å was given in the text of the problem, it is not a formulation parameter. Rather, it simply states that E might not contain all the possible unordered pairs (it usually doesn't).
- The objective function appears to be a sum of squares, but this is deceiving: if you write it in function of each component x_{ik} you soon discover that it is a quartic multivariate polynomial of the x variables.

- The formulation is unconstrained. Like packings, a set of vectors identified by pairwise distances is invariant to congruences. One way to make it invariant to rotations is to fix the barycenter of the vector set to the origin, which corresponds to the constraint:

$$\sum_{i \in N} x_i = 0.$$

- Most solvers are happier if you provide bounds to the variables (there are some exceptions to this rule). The worst that can happen is that all vectors are on a segment as long as $D = \sum_{\{i,j\} \in E} d_{ij}$. Since the barycenter is at the origin, you can impose the following variable bounds:

$$\forall i \in N, k \leq K \quad -\frac{D}{2} \leq x_{ik} \leq \frac{D}{2}.$$

Chapter 2

The language of optimization

Mathematical Programming (MP) is a formal language for describing and solving optimization problems.

In general, languages can be natural or formal. Natural languages are those spoken by people to communicate. Almost every sentence we utter is ambiguous, as it can be interpreted by different people in different ways. By contrast, its expressivity is incredibly rich, as it can be used to describe reality. Formal languages can be defined formally through recursive rules for interpreting their sentences within the confines of an abstract model of a tiny part of reality. If ambiguity occurs in formal languages, it is limited in scope, announced and controlled.

Optimization refers to the improvement of a process which an agent can modify through his/her/its decisions. For example, the airplane departure sequence is decided by controllers who usually try and reduce the average (or maximum) delay.

We use the word "problem" here in the semi-formal sense of computer science: a problem is a formal question relating some given input to a corresponding output: both input and output are encoded through a sequence of bits respecting some given formats and rules. For example, a decision problem might ask whether a given integer is even or odd: the input is an integer, and the output is a bit encoded by the mapping $1 \leftrightarrow$ YES, $0 \leftrightarrow$ NO. An optimization problem might ask the shortest sequence of consecutive vertices linking two given vertices in a graph (provided it exists).

2.1 MP as a language

In computer science, a (basic) *language* $\mathscr{L}$ is a collection of *strings*, each of which is a sequence of *characters* from a given *alphabet* $\mathscr{A}$. Composite languages can be formed as Cartesian products of basic languages. The fundamental problem, given a language and a string, is to determine whether or not the string belongs to the language. This is called *recognition problem*.

2.1.1 Exercise
Implement algorithms to recognize: (a) strings made entirely of digits; (b) strings consisting of a finite sequence of floating point numbers (in exponential notation) separated by any amount of spaces, tabs and commas; (c) strings consisting of multi-indexed symbols (use square brackets for indices, as in, e.g. $x[2,4]$*); (d) strings consisting of English words from a dictionary of your choice.*

2.1.2 Exercise
Implement an algorithm that recognizes a language consisting of strings in the union of (b) and (d) in Exercise 2.1.1 above.

2.1.1 The arithmetic expression language

The most basic language we use in this book includes all functions that can be expressed in "closed form" using a set of primitive operators, say $+, -, \times, \div, (\cdot)^{(\cdot)}, \log, \exp, \sin, \cos, \tan$. Some of these operators are binary, some are unary, and others, like $-$, can be both, depending on context: the number of arguments of an operator is its *arity*.

Operators recursively act on their arguments, which are either other operators, or symbols representing numerical quantities (which can also be considered as operators of zero arity). Ambiguous cases, such as $x_1 + x_2x_3$, are resolved by assigning to each operator a precedence: $\times$ trumps $+$, so the order of the operations is x_2x_3 first, and then $x_1+$the result. In cases of tie, the left-to-right scanning order is used.

Given a string such as $x_1 + \frac{x_1x_2}{\log x_2}$, the recognition problem can be solved by the following recognition algorithm, given here in pseudocode:

1. find the operator of leftmost lowest precedence in the string
2. identify its arguments and tag them by their arity, marking the characters of the string that compose them
3. recurse over all operators of positive arity

If all characters in the string have been marked at the end of this algorithm, the string is a valid sentence of the language. By representing each recursive call by a node, and the relation "parent-child" by an arrow, we obtain a directed tree (called *parsing tree*) associated with each execution of the recognition algorithm. If, moreover, we contract leaf nodes with equal labels, we obtain a *directed acyclic graph* (DAG) representation of the arithmetical expression (see Example 2.1.3). This representation is called *expression DAG* or *expression tree*.

2.1.3 Example
Consider the expression $x_1 + \frac{x_1x_2}{\log x_2}$. *Precedences may be re-written using brackets and implicit operators are written explicitly: this yields* $x_1 + (x_1 \times x_2)/\log(x_2)$. *The corresponding parsing tree and DAG are shown in Fig. 2.1.*

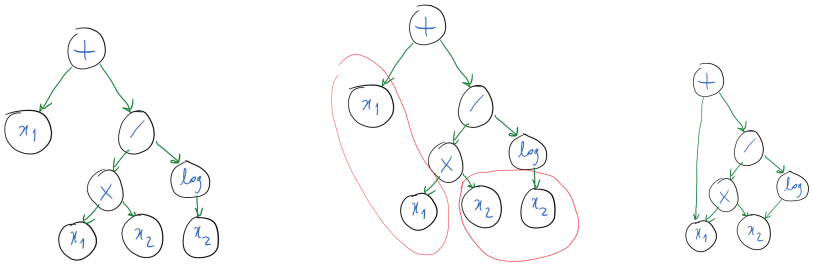


Figure 2.1: A parsing tree (left), the nodes being contracted (center), and the corresponding DAG (right).

Note that implementing this algorithm in code requires a nontrivial amount work, mostly necessary to accommodate operator precedence (using brackets) and scope within the string. On the other hand, easy modifications to the recognition algorithm sketched above yield more interesting behaviour. For example, a minor change allows the recursive recognition algorithm to compute the result of an arithmetical expression where all symbols representing numerical values are actually replaced by those values. This algorithm performs what is called *evaluation* of an arithmetic expression.

2.1.4 Exercise
Implement recognition and evaluation algorithms for an arithmetic expression language.

2.1.1.1 Semantics

By *semantics* of a sentence we mean a set of its interpretations. A possible semantics of an arithmetic expression language is the value of the function represented by the expression at a given point x. The algorithm for this semantics can be readily obtained by the evalution algorithm described in Sect. 2.1.1 above as follows:

- replace each symbol (operator of zero arity) with the value of the corresponding component of the vector x;
- evaluate the arithmetic expression.

Other semantics are possible, of course.

2.1.5 Exercise
Define a semantics for SAT *sentences, namely conjunctions of disjunctions of literals such as* $(x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$*, where* $\bar{x}_i$ *is equivalent to* $\neg x_i$*, i.e. the negation of the literal* x_i*.*

> **Answer**. For all i, assign to x_i a value in $\{0, 1\}$, then use the compositional syntax to evaluate the truth/falsity of the given SAT sentence. The fundamental decision problem SAT asks whether there exists an assignment of binary values to the literals such as to make the sentence true, or not. Such an assignment, if it exists, is called a *certificate*.

2.1.6 Exercise
How would you define a semantics for the English language?

> **Answer**. Unlike formal languages, for which ambiguities are absent or controlled, English is a *natural language*. Whether formal semantics of natural languages exist is a question that belongs to the realm of philosophy [228, 229]; since no attempt was ever successful aside from very limited subset of English, we might as well assume formal semantics cannot generally be defined on natural languages. The definition of practically useful semantics for natural languages falls within the realm of computational linguistics [207].

2.1.2 MP entities

The sentences of the MP formal language consist of symbols belonging to five categories: parameters, decision variables, objective functions, functional constraints and implicit constraints. The parameters encode the problem input, the decision variables encode the problem output (the solution), an objective function defines a search direction of optimality, the functional constraints are expressed as equations ($=$) or (non-strict) inequalities ($\leq$, $\geq$) involving functions, and the implicit constraints express range constraints, integrality, membership in a cone, and so on.

The distinction between functional and implicit constraints need not be strict. For example, one can write $x \in \{0, 1\}$ (an implicit integrality constraint) as $x^2 = x$ (a functional constraint), and vice versa. In such cases, the choice often depends on the solution algorithm. For example, an algorithm that can accept nonlinear functions as part of its input, but is limited to continuous variables, will accept $x^2 = x$. Conversely, an algorithm that only accepts linear functions but can deal with integrality constraints directly will accept $x \in \{0, 1\}$.

2.1.2.1 Parameters

The input of the optimization problem that the MP describes is contained in various data structures, which can be vectors, matrices, arrays and so on. Each component of these data structures appears as a symbol in the MP formulation. These symbols are known as *parameters*. The parameter language must simply recognize an appropriately structured set of parameter symbols (which are initially declared as such) or numerical values.

2.1.2.2 Decision variables

The decisions that can be taken by the agent interested in solving the optimization problems are also contained in various data structures, each component of which appears as a symbol in the MP formulation. These symbols are known as *decision variables*. The decision variable language must simply recognize an appropriately structured set of decision variable symbols (which are initially declared as such). After the problem is solved, these symbols are assigned the values of the optimal solutions found.

2.1.2.3 Objective functions

An objective function string is an arithmetical expression having decision variables and (optionally) parameter symbols as operators of zero arity, together with an *optimization direction*, denoted by min or max. Depending on whether the optimization problem is feasibility-only, single-objective or multi-objective, there may be zero, one or a finite number of objective functions in the problem.

The single objective function language is a trivial extension of the arithmetical expression language: it must also recognize an optimization direction. The objective function language must be able to recognize a sequence of single objective function language sentences.

2.1.7 Exercise
Consider an optimization problem that aims at minimizing the objective function $f(x) = \sum_{j \leq n} c_j x_j$. *Typically, people use the 'x' symbols to denote decision variables: thus each component of the vector* $x = (x_1, \ldots, x_n)$ *is a decision variable. Since we wrote* f *as a function of* x*, we implicitly mean that the symbol* $c = (c_1, \ldots, c_n)$ *is a parameter vector.*

2.1.2.4 Functional constraints

A constraint string is an arithmetical expression having decision variables and (optionally) parameter symbols as operators of zero arity, together with a relational sign, denoted by $=, \leq, \geq$ and a numerical value. Depending on whether the problem is constrained or unconstrained, there may be multiple constraint strings in the MP formulation or none.

The single constraint function language is a trivial extension of the arithmetical expression language: it must also recognize the relational operator and the numerical value. The functional constraint function langauge must be able to recognize a sequence of single constraint function language sentences.

2.1.2.5 Implicit constraints

Constraints are *implicit* when their description does not include an arithmetical expression string. We shall consider two types:

- integrality constraints, which require that a certain subset of decision variables must attain integer values to be feasible;
- semidefinite programming (SDP) constraints, which require that a certain square symmetric matrix of decision variables must attain a positive semidefinite (PSD) matrix of values to be feasible.

An integrality constraint string simply consists of a tuple of indices denoting the decision variable symbols required to be integer. Equivalently, an SDP constraint indicates that the decision variable matrix is required to attain a PSD matrix value.

2.1.3 The MP formulation language

A MP *formulation* is a description of a MP problem. It consists of a tuple (P, V, O, F, I) where P is a parameter string, V a decision variable string, O an objective function string, F a functional constraint string, and I an implicit constraint string. An MP formulation indicates a optimization problem, if at least one parameter is a symbol, or an *instance* of this problem if all parameters have numerical values.

A problem is nontrivial if the parameter symbols can range over infinite sets of values. Each assignment of values to the parameter symbols yields an instance, so that problems can also be considered as sets of instances.

2.1.3.1 Solvers as interpreters

The semantics of an MP formulation are given by the assignment of optimal solution values to the decision variable symbols. The algorithms for computing optima of MP formulations are called *solvers*. There exist different solvers for MP, each targeting a particular MP restriction.

For example, if the objective and constraint functions are linear, several off-the-shelf solvers exist, most of which are commercial, e.g. CPLEX [145], XPress-MP [128]. Many of these solvers have lately endowed their products with algorithms for dealing also with nonlinear terms involving specific operators. There are also some free and open-source solvers, though: look at the COIN-OR project [199] page `www.coin-or.org`, as well as GLPK [206].

A solver establishes the semantics of an MP language sentence P. In this sense, it can be seen as a natural *interpreter* for the MP language. Its input is the set of parameter values, i.e. the instance, which suffices to reconstruct the formulation P. Its output (if the solver is successful) is usually the pair (f^*, x^*) where f^* is the optimal objective function value and x^* an optimum with value f^*. We denote this by $[\![P]\!] = (f^*, x^*)$.

Most theories of programming languages manage to control the relationship between syntax and semantics, since the formal grammar parsers they employ to define recognition algorithms can easily be modified to construct the sentence semantics (as underlined in the discussion on recognition and evaluation of arithmetic expression sentences in Sect. 2.1.1). In other words, most programming languages are defined in such a way that recursion in syntax parallels recursion in semantics (this is particularly true of imperative languages, see Sect. 2.1.3.2).

On the other hand, the algorithms implemented by solvers have generally nothing to do with the recursive syntactical structure of the formal grammar used to recognize the sentences. Hence, unfortunately, the relationship between syntax and semantics is much harder to establish in a theoretical sense. There is a general consensus to the effect that efficient (e.g. polynomial time) algorithms provide a "desirable semantics".

Not only can different solvers employ very different algorithms in order to compute $[\![P]\!]$; they may also have different *formats* for their semantics, which could be, e.g.: a single optimum, or many, or even all if there are finitely many optima; the optima might be defined only with respect to (w.r.t.) a neighbourhood, or they may be global; the results might be guaranteed or not (see Defn. 6.0.1 for a formal definition of optima).

One of the major efforts in the theory of MP is that of deriving new implied constraints (generally called *valid cuts*) to be added to existing hard MP formulations so that: (a) at least one (and hopefully all) optimal solutions remains feasible; (b) they help make other problematic constraints (such as integrality constraints) redundant; (c) they themselves are not problematic. This allows the use of an efficient solver to solve the instance. Valid cuts can be seen as a way to keep $[\![P]\!]$ invariant while making the solution searching process more efficient.

2.1.3.2 Imperative and declarative languages

Computer programming languages such as C/C++, Java, Pascal, Basic, Fortran, Python, Matlab (and more) are known as *imperative* languages: a program in any of these languages will be a set of statements in any of four broad categories: (a) storage, or assignments of values to program variables; (b) tests, or verification of given conditions on program variables; (c) loops, or repeated execution of some parts of the code; (d) input/output (I/O), i.e. interaction with the hardware layer or external world. Most languages can also encapsulate some parts of code into *callable functions*: this also allows the implementation of loops by means of *recursion*.

By contrast, *declarative*[1] languages only make logical statements about variables, and then let a general algorithm search for the variable values that satisfy those statements. It should be clear that MP is a declarative language. Programming in declarative languages is both simpler and harder than programming in an imperative language. It is simpler because one need not conceive nor implement the algorithm — it is by definition the language interpreter. It is harder because humans are better suited to understand the very simple blocks *assignments, tests, loops* than to design multiple logical statements which, conjunctively, are supposed to describe the output of a given process (without actually detailing the process itself). Programming in a declarative language is also somtimes called *modelling*, although this term also has a different meaning (see Sect. 1.3.2.1). The MP language shifts the focus of optimization from algorithmics to modelling.

Suppose someone asks you to find the largest stable set in a given graph $G = (V, E)$, i.e. the largest subset $S \subseteq V$ such that no unordered pair $\{s, t\}$ of vertices in S is an edge in E. In an imperative language, you would need to decide how to encode a graph and a subgraph, how to enumerate subgraphs in the most efficient possible way, and how to verify whether a subgraph is a stable set (i.e. it has induces no edges in E) or not. In MP, you would simply write:

$$\max \left\{ \sum_{v \in V} x_v \;\middle|\; \forall \{u, v\} \in E \; (x_u + x_v \leq 1) \wedge \forall v \in V \; x_v \in \{0, 1\} \right\}, \tag{2.1}$$

and then deploy an appropriate solver on the formulation in Eq. (2.1).

Note that most imperative and declarative languages are Turing-complete, (see Sect. 4.1.1). This means one can program or design a universal Turing machine (UTM) with them (again, see Sect. 4.1.1). Hence their expressive power is equivalent.

2.1.8 Exercise

Consider the imperative pseudocode

$i = 1$
$A = \varnothing$
while $i \leq 50$ ***do***
  $i \leftarrow i + 1$
  $A \leftarrow A \cup \{2x\}$
end while
return A

Write a declarative language equivalent of this code.

> **Answer**. The pseudocode returns the set of even numbers > 1 and ≤ 100, therefore a declarative formulation is given by $\{x \in \mathbb{Z} \mid \exists y \in \{1, \ldots, 50\} \; (x = 2y)\}$.

[1] Some computer scientists might call such languages "descriptive", and reserve "declarative" for a different type of programming languages.

2.2 Definition of MP and basic notions

The formal definition of an MP formulation on n decision variables and m constraints is as follows:

$$\left.\begin{array}{lrcl} \min\limits_{x\in\mathbb{R}^n} & f(p,x) & & \\ \forall i \le m & g_i(p,x) & \le & 0 \\ \forall j \in Z & x_j & \in & \mathbb{Z} \\ & x & \in & X, \end{array}\right\} \quad [P] \tag{2.2}$$

where $Z \subseteq \{1,\ldots,n\}$, $X \subseteq \mathbb{R}^n$ is the set of implicit constraints (see Sect. 2.1.2.5) x is a vector of n decision variables taking values in $\mathbb{R}$, p is a vector of parameters which, once fixed to values, make the functions f and g_i (for $i \le m$) functions $\mathbb{R}^n \to \mathbb{R}$. The symbol $P = P(n,m,p,Z,X)$ denotes the MP formulation, i.e. the MP instance for given n,m,p,Z,X.

For a MP formulation P, we let $\mathsf{feas}(P)$ be the feasible set of P. We let $\mathsf{val}(P)$ be the globally optimal value of P if $\mathsf{feas}(P) \neq \varnothing$ and if a global optimum of P exists (i.e. if $\mathsf{feas}(P)$ is bounded in the optimization direction). By convention, if P is a minimization problem, we let $\mathsf{val}(P) = \infty$ if $\mathsf{feas}(P) = \varnothing$, and $\mathsf{val}(P) = -\infty$ if $\mathsf{feas}(P)$ is unbounded in the optimization direction (if P is a maximization problem, we multiply $\mathsf{val}(P)$ by -1).

We let $\overline{\mathbb{MP}}$ be the class of all instances of Eq. (2.2) as n,m range over all sizes, Z over all subsets of $\{1,\ldots,n\}$ and p over all possible sizes and values, and f,g over all functions $\mathbb{R}^n \to \mathbb{R}$.

The set of vectors in $\mathbb{R}^n$ that satisfy all constraints of the instance P is called the *feasible set* of P and denoted by $\mathcal{F}(P)$ or simply $\mathcal{F}$. The set of feasible vectors that are also optima (i.e. achieve the minimum objective function value) in P is called the *optimal set* of P and denoted by $\mathcal{G}(P)$ or simply $\mathcal{G}$.

There are three possibilities for a given a MP instance P, which we assume without loss of generality (wlog) in minimization form:

1. there exists at least an optimum x^* of P with optimal objective function value f^*;
2. P is *unbounded*, i.e. for any feasible solution x' with objective function value f', there always exist another feasible solution x'' with objective function value f'' such that $f'' < f'$;
3. P is *infeasible*, i.e. $\mathcal{F}(P) = \varnothing$.

This ternary classification corresponds to two hierarchically dependent binary classifications: whether P is feasible or not, and, if P is feasible, whether it has an optimum or not.

2.2.1 Certifying feasibility and boundedness

In general, it is possible to certify feasibility in practice: given a solution x', it suffices to verify whether x' satisfies all the constraints. Certifying optimality is more difficult in general: whether it can be done or not depends on many factors, the most important of which the behaviour of the objective over the feasible region, the presence of linear/nonlinear terms in the description of the objective function, whether there are finitely or infinitely many feasible solutions.

Infeasibility is also hard to certify: in general, given a MP formulation as in Eq. (2.2), assuming we have an algorithm for optimizing over the implicit constraints $x \in X$, we can add non-negative *slack variables* $s_1,\ldots,s_m$ to the rhs of the inequality constraints and minimize their sum:

$$\left.\begin{array}{lrcl} \min\limits_{x\in\mathbb{R}^n, s\ge 0} & \sum\limits_{i\le m} s_i & & \\ \forall i \le m & g_i(p,x) & \le & s_i \\ \forall j \in Z & x_j & \in & \mathbb{Z} \\ & x & \in & X. \end{array}\right\} \quad [F_P] \tag{2.3}$$

The original instance P is infeasible if and only if (iff) the globally optimal objective function value of Eq. (2.3) is strictly greater than zero. This poses two issues: (a) globally optimizing Eq. (2.3) is generally just as hard as optimizing Eq. (2.2); and (b) in practice, using floating point computations, we might well obtain optimal objective function values in the order $O(10^{-5})$ where the parameters are in the range $O(1)$-$O(100)$: how do we decide whether the problem is infeasible or it is feasible up to some floating point error?

2.2.2 Cardinality of the $\overline{\mathbb{MP}}$ class

Since p might encode continuously varying parameters, and f, g range over all possible real functions of n arguments, the class $\overline{\mathbb{MP}}$ must contain uncountably many instances.

In practice, however, any continuous parameter in p must in fact range over the rationals, of which there are countably many (in fact continuous parameters usually range over the floating point numbers, of which there are finitely many). As for f, g, they must be recognized by the expression languages described in Sect. 2.1 above. Since there are countably many arithmetical expressions, and only finitely many of a given size, the subclass of instances in $\overline{\mathbb{MP}}$ which can be explicitly described (a necessary condition in order for an instance to be solved — if nothing else because the instance description is the input to the solver) is countably infinite in theory. Given our language $\mathscr{L}$, we call $\mathbb{MP}_{\mathscr{L}}$ the subclass of instances in $\overline{\mathbb{MP}}$ having a finite description in $\mathscr{L}$. If $\mathscr{L}$ is fixed *a priori* (which it usually is), we drop the subscript and simply write $\mathbb{MP}$.

This means we cannot express (and hence solve) uncountably many instances. But the good news is that, given any instance (and so any functions f, g described in some language $\mathscr{L}'$), we can extend our language $\mathscr{L}$ to integrate $\mathcal{L}'$ so that f, g can also be described in $\mathcal{L}$.

2.2.3 Reformulations

Several features of Eq. (2.2) might appear arbitrary to the untrained eye. We will show in this section that every assumption was made wlog.

2.2.3.1 Minimization and maximization

In Sect. 2.1.2.3 we stated that objective functions have a direction, either min or max. But in Eq. (2.2) we only wrote min. However, given any MP formulation

$$P \equiv \min\{f(x) \mid x \in \mathcal{F}\} \tag{2.4}$$

can be written as

$$Q \equiv \max\{-f(x) \mid x \in \mathcal{F}\} \tag{2.5}$$

while keeping the set $\mathcal{G}$ of optima invariant. The only thing that changes is the value of the optimal objective function: it is f^* in P iff it is $-f^*$ in Q.

Some solvers only accept one of the two optimization directions (typically the min direction is more common). So there is sometimes a practical need for carrying out this transformation.

2.2.3.2 Equation and inequality constraints

Eq. (2.2) only lists m inequality constraints $g_i(p, x) \leq 0$ (for $i \leq m$). However, we can express an equation constraint $h(p, x) = 0$ by requiring that there exist two indices $i, j \leq 0$ such that $g_i(p, x) \equiv h(p, x)$ and

$g_j(p,x) \equiv h(p,x)$. This yields

$$\begin{aligned} h(p,x) &\leq 0 \\ -h(p,x) &\leq 0, \end{aligned}$$

which hold for fixed p iff $h(p,x) = 0$. Note that for each equation constraint we need two inequality constraints, so the size m of the constraint list changes.

In practice, the large majority of solvers accept both inequality and equation constraints explicitly.

2.2.3.3 Right-hand side constants

Although all RHSs in Eq. (2.2) have been set to zero, any other constant can simply be brought over (with opposite sign) to the left-hand side (LHS) and considered part of the function $g_i(p,x)$ (for $i \leq m$).

2.2.3.4 Symbolic transformations

All the simple transformations given in Sect. 2.2.3.1-2.2.3.3 are part of a much larger family of symbolic and numerical transformations on MP formulations collectively known as *reformulations* [171, 306], some of which will be discussed in more depth below.

2.2.3.5 Linearization

We note two easy, but very important types of reformulations.

- The *linearization* consists in identifying a nonlinear term $t(x)$ appearing in f or g_i, replacing it with an added variable y_t, and then adjoining the *defining constraint* $y_t = t(x)$ to the formulation.
- The *constraint relaxation* consists in removing a constraint: since this means that the feasible region becomes larger, the optima can only improve with respect to those of the original problem. Thus, relaxing constraints yields a relaxation of the problem.

These two reformulation techniques are often used in sequence: one identifies problematic nonlinear terms, linearizes them, and then relaxes the defining constraints. Carrying this out recursively for every term in an NLP [215], and only relaxing the nonlinear defining constraints yields an LP relaxation of an NLP [277, 285, 38].

2.2.4 Coarse systematics

As we shall see below, MP formulations are classified in many different ways [178, 306]. For example,

- according to the properties of their descriptions: e.g. an MP where all the functions are linear and $Z = \varnothing$ is called a Linear Program (LP), whereas if $Z \neq \varnothing$ it is called a Mixed-Integer Linear Program (MILP);
- according to their mathematical properties: e.g. if f is nonlinear but convex in x and $\mathcal{F}$ is a convex set, P is a convex Nonlinear Program (convex NLP or cNLP);
- according to whether a certain class of solvers can solve them: this classification is obviously *a posteriori* w.r.t. the solution process, and often changes in time, since solvers, operating systems and hardware all evolve.

Unions of properties are also possible: for example an MP involving both integer variables and nonlinear functions is called a Mixed-Integer Nonlinear Program (MINLP). If, when relaxing the integer variables to take continuous values, the resulting problem is a cNLP, then the MINLP is referred to as a convex MINLP (cMINLP). We remark that a cMINLP has a nonconvex feasible region (by virtue of the integrality constraints).

2.2.5 Solvers state-of-the-art

In this section we examine the state of the art of current MP solvers according to our systematics (Sect. 2.2.4) as regards robustness and solution size.

- Currently, LP solvers are the most advanced. Most of them are very robust. For sparse data, current LP solvers might well solve instances with $O(10^6)$ variables and constraints. For dense data, unfortunately, the situation varies wildly depending on the data themselves: but most input data in MP is sparse, and dense MPs mostly occur in specific fields, such as quantile regression or signal processing.
- The most advanced MILP solvers are based on the Branch-and-Bound (BB) algorithm, which essentially solves a sequence of LPs (this makes MILP solvers quite robust). However, this sequence is exponentially long in the worst case. It is very hard to make predictions on the maximal size of MILPs that current BB technology is able to solve in "acceptable times", as there are minuscule examples known to force the worst-case, as well as practical cases of MILPs with tens of thousands of variables and/or constraints being solved very fast.
- SDP solvers (and conic solvers in general) are quite robust, but compared to LP solvers their "size bottleneck" is much more restricted. You can hope to solve SDPs with $O(10^3)$ variables and constraints, and possibly even $O(10^4)$ (but remember that SDPs involves matrices of decision variables, so the number of decision variables is usually quite large).
- Local NLP solvers guarantee global optimality on the vast majority of practical cNLP instances; and most of them will also be rather efficient at finding such optima. One might hope to solve instances with tens of thousands of variables and/or constraints as long as the "linear part" of the problem is large, sparse, and the solver is able to exploit it. Today, however, many cNLP applications are from Machine Learning (ML), and their sizes are so large that special purpose solvers have to be designed — and a guarantee of optimality often forsaken. While a local NLP solver deployed on a cNLP is reasonably robust, the same solver deployed on a nonconvex NLP typically has many more chances of failures, specially if an infeasible starting point was provided (which is often the case — finding a feasible solution in a nonconvex NLP is as hard as finding an optimal one, in terms of computational complexity in the worst case).
- The state of the art of global NLP solvers for nonconvex NLPs is less advanced than then the previous ones. Currently, all of the solvers that are able to certify global optimality (to within a given $\varepsilon > 0$ tolerance on the optimal objective function value) are based on a BB variant called "spatial BB" (or sBB). The sBB algorithm solves a sequence of LP or cNLP relaxations of the given (nonconvex) NLP, as well as locally solving the given NLP every so often. This procedure is as fragile as its local solvers warrant — and since local NLP solvers are not so robust, sBB is no different. Current sBBs perform reasonably well on problems with a quadratic structure, as those are the best studied. sBB implementations can often exhibit their exponential worst-case behaviour even on tiny instances, unfortunately. Expect some results in the size range $O(10^2)$-$O(10^3)$ variables/constraints.
- cMINLP solvers are somewhat more robust than nonconvex NLP solvers, but much less advanced than MILP solvers; they can probably be deployed on instance sizes of around $O(10^3)$.
- General MINLP solvers implement sBB algorithms that can also branch on integer variables. There is currently no sBB solver that can only deal with nonconvex NLPs but not with MINLP. Since

MILP solution technology is reasonably advanced, endowing an sBB implementation with mixed-integer capabilities does not worsen its robustness or the limits of its maximal sizes.

You should take all of the information given above with a pinch of salt. It is the author's own opinion, formed after close to twenty years' experience in the field. It is not a scientific statement, and has been left suitably vague on purpose. Algorithmic research in MP is intense, and major improvements, even on much smaller subclasses than our systematics above lists, are often picked up by the major commercial codes rather rapidly.

If you have a large instance for which your solver takes too long you should: (b) ask yourself whether there exists a simple and efficient algorithm that can solve your problem independently of its MP formulation; (b) attempt to reformulate the problem in order to be able to solve it with a more robust/efficient solver; (c) give your solver a time limit, forsake its guarantees, and hope it finds a good solution within the allotted time. While advice (c) sounds close to "give up", remember that (i) in the large majority of practical cases, having any solution is better than having none at all; (ii) for many classes of problems, local optima tend to cluster together, and local optima close to the global optimum might be quite close to it.[2]

2.2.6 Flat versus structured formulations

A formulation is *flat* if it has no quantifier over variable indices, and *structured* otherwise.

We have seen two examples of MP formulations so far: Eq. (2.1) and Eq. (2.2). Eq. (2.1) involves the maximization of the sum of all the decision variables (one per vertex in the graph), subject to the condition that, for each edge in the graph, the decision variables corresponding to the adjacent vertices cannot both be set to one. Eq. (2.2) is completely general: it minimizes an objective subject to a list of functional constraints introduced by a universal ("for all") quantifier. In both formulations there appear symbols quantified by indices (u, v in Eq. (2.1), i in Eq. (2.2)), and quantifiers ($\sum, \forall$ in Eq. (2.1), $\forall$ in Eq. (2.2)). Therefore, both are structured.

The following is a simple example of a flat formulation:

$$\left.\begin{array}{rrcl} \min & x_1 + x_2 + x_3 + x_4 & & \\ & x_1 + x_2 & \leq & 1 \\ & x_1 + x_3 & \leq & 1 \\ & x_2 + x_4 & \leq & 1 \\ & x_3 + x_4 & \leq & p \\ & x_1, x_2, x_3, x_4 & \in & \{0, 1\}, \end{array}\right\} \tag{2.6}$$

where p is a (scalar) parameter. If we assign a fix value to p, e.g. $p = 1$, we obtain a flat formulation representing an instance of the MP formulation in Eq. (2.1).

2.2.1 Exercise

Since Eq. (2.6) is an instance of Eq. (2.1) when $p = 1$, its optima are stable sets in a graph: recover the graph from the formulation.

Answer. The graph is $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\}$.

2.2.2 Exercise

Is Eq. (2.6) an instance of Eq. (2.1) whenever $p \neq 1$? What about $p = 0$ or $p = 2$? Does it make sense to consider other values of p?

Answer. (a) no, since Eq. (2.1) specifically lists a rhs with value 1. (b) If $p = 0$ the formulation finds a stable set not involving vertices 3 and 4. (c) If $p = 2$ the formulation allows both 3 and 4 in the solution, but selects a stable

[2] This is mostly an empirical observation [57]. Intuitive explanations have sometimes been attempted, based on the distribution of optima in all points satisfying first-order conditions [40].

set in the remaining subgraph (the edge $\{1,2\}$). (c) No: since x_3, x_4 are binary, the values of the lhs can only be in $\{0,1,2\}$.

2.2.6.1 Modelling languages

The distinction between flat and structured formulations is that solvers only accept flat (instance) formulations, whereas humans naturally model optimization problems using structured formulations. This calls for two MP languages: one that understand parameters and variable symbols with indices, as well as quantifiers over those indices, and one that does not. It also calls for a translator between the two languages.

In MP, translators from structured to flat formulations are called *modelling languages*. In fact, they do more than just translate. Since each solver accepts its input in a flat formulation language with its own distinct syntax, most modelling languages act as interfaces to a set of solvers, each of which requires a specific translation module.

2.2.3 Remark (Quantifiers over indices)
Quantifiers are only allowed to range over indices belonging to a set given as part of the parameters (the input). In no case should a quantifier ever involve a decision variable symbol. The issue arises because translating from a structured formulation to a flat one would be impossible if a quantifier involved a decision variable (the value of which is not known before the solution process occurs). Though this remark might appear innocuous at this time, wait until you need a "conditional constraint" (see the discussion at the end of Sect. 2.2.7.5).

The best known purely MP-related translators are AMPL [115] and GAMS [58]. AMPL structured formulation language syntax is very similar to the mathematical form of MP languages, and hence very close to human-readable syntax, and can interface with many solvers. Its weak point is that it has very a limited imperative sub-language, which makes it hard to reformulate and post-process solutions. GAMS has a better imperative sub-language, but its structure formulation language syntax has a steep learning curve (at best).

On the other hand, with the increasing need for complicated algorithms involving MP as elementary steps, many imperative programming languages have been endowed with capabilities for modelling and solving MP formulations. Python has a growing number of translators (e.g. PyOMO [132] and PICOS [258]), and Matlab [208] has many extremely good translators, some commercial and some not. The most complete Matlab translator appears to be TOMLAB [142], which interfaces with a large number of solvers. A free equivalent to TOMLAB is the OPTI toolbox [83], which, however, unfortunately only works on the Windows version of Matlab. Another excellent free translator for Matlab is YALMIP [198], which focuses mostly (but not only) on conic optimization.

2.2.7 Some examples

2.2.7.1 Diet problem

A diet involving m nutrients (vitamins, proteins, fats, fibers, iron, etc.) is healthy if it has at least quantities $b = (b_1, \ldots, b_m)$ of the nutrients. In order to compose such a diet, we need to buy quantities $x = (x_1, \ldots, x_n)$ of n types of food having unit costs $c = (c_1, \ldots, c_n)$. Food $j \leq n$ contains a_{ij} units of nutrient $i \leq m$. The solution of the following LP problem yields an x that satisfies the requirements of a

healthy diet at minimum cost [88].

$$\left.\begin{array}{rrcl} \min_x & \sum_{i=1}^{n} c^\top x & & \\ & Ax & \geq & b \\ & x & \geq & 0. \end{array}\right\} \tag{2.7}$$

The parameters are A, b, c. The decision variables are the components of the vector x. The functional constraints are the rows of the linear inequality system $Ax \leq b$. The *non-negativity* constraints $x \geq 0$ can be interpreted both as functional constraints (since $x = f(x)$ where f is the identity function) and as implicit constraints: in the latter case, the sentence $x \geq 0$ encodes membership in the non-negative orthant.

2.2.7.2 Transportation problem

Consider a transportation network modelled by a weighted bipartite directed graph $B = (U, V, A, d)$ with a set of departure vertices U, a set of destinations V, a set of arcs $A = \{(u, v) \mid u \in U, v \in V\}$ weighted by a nonnegative distance function $d : A \to \mathbb{R}_+$. A certain amount of material a_u is stored at each departure vertex $u \in U$. We associate to each destination $v \in V$ a given demand b_v of the same material. The cost of routing a unit of material from $u \in U$ to $v \in V$ is directly proportional to the distance d_{uv}. We have to determine the transportation plan of least cost satisfying the demands at the destinations. The variables x_{uv} in the LP formulation below, associated to each arc $(u, v) \in A$, denote the amount of material routed on the arc.

$$\left.\begin{array}{rrcl} \min_x & \sum_{(u,v)\in A} d_{uv} x_{uv} & & \\ \forall u \in U & \sum_{v\in V} x_{uv} & \leq & a_u \\ \forall v \in V & \sum_{u\in U} x_{uv} & \geq & b_v \\ \forall (u,v) \in A & x_{uv} & \geq & 0. \end{array}\right\} \tag{2.8}$$

The parameters are all encoded in the weighted bipartite directed graph (digraph) B. The decision variables represented by the double-indexed symbols x_{uv} (for $(u, v) \in A$). This structure can also be seen as a partial[3] $|U| \times |V|$ matrix X with component (u, v) being present iff $(u, v) \in A$. There are two sets of functional constraints and a non-negativity constraint.

2.2.7.3 Network flow

Given a network on a directed graph $G = (V, A)$ with a source node s, a destination node t, and capacities u_{ij} on each arc (i, j). We have to determine the maximum amount of material flow that can circulate on the network from s to t. The variables x_{ij} in the LP formulation below, defined for each arc (i, j) in the graph, denote the quantity of flow units.

$$\left.\begin{array}{rrcl} \max_x & \sum_{i\in\delta^+(s)} x_{si} & & \\ \forall\, i \leq V, i \neq s, i \neq t & \sum_{j\in N^+(i)} x_{ij} & = & \sum_{j\in N^-(i)} x_{ji} \\ \forall (i,j) \in A & 0 \leq x_{ij} \leq u_{ij}. & & \end{array}\right\} \tag{2.9}$$

The parameters include the array $\{u_{ij} \mid (i, j) \in A$ of upper bounds and the digraph G, represented in Eq. (2.9) in the node adjacencies $N^+(i)$ (set of *outgoing* nodes j such that $(i, j) \in A$) and $N^-(i)$ (set

[3] A *partial matrix* is a matrix with some components replaced by a placeholder indicating their absence.

of *incoming* nodes j such that $(j,i) \in A$). The decision variables are x_{ij} for $(i,j) \in A$. The functional constraints are in the form of equations (we remark that the form of the equation is not $f(x) = 0$ but $f^1(x) = f^2(x)$, trivially reducible to the former standard form). The *range* constraints $0 \leq x_{ij} \leq u_{ij}$ can be seen as functional constraints. This is obtained by re-writing them as $x_{ij} - u_{ij} \leq 0$ and $x_{ij} \geq 0$ or as an implicit constraint $X \in [0, U]$ where X is the partial $|V| \times |V|$ matrix of decision variables, and U is the partial $|V| \times |V|$ matrix of upper bounds.

2.2.7.4 Set covering problem

A certain territory containing n cities is partitioned in m regions. We must decide whether to build a facility on region $i \leq m$ or not. For each $i \leq m$ and $j \leq n$, the parameter a_{ij} is equal to 1 iff a facility on region i can serve city j (otherwise $a_{ij} = 0$). The cost of building a facility on region i is c_i. We require that each city is served by at least one facility, and we want to find the construction plan of minimum cost. We model the problem as the following MILP:

$$\left.\begin{array}{rrcl} \min_x & \sum_{i=1}^{m} c_i x_i & & \\ \forall\, j \leq n & \sum_{i=1}^{m} a_{ij} x_i & \geq & 1 \\ & x & \in & \{0,1\}^n. \end{array}\right\} \tag{2.10}$$

The parameters are (c, A) where A is the $m \times n$ matrix having a_{ij} as its (i,j)-th component (A is an adjacency matrix). There is a decision variable vector x, a set of linear functional constraints and an implicit integrality constraints of binary type. Note that, although Eq. (2.10) is a MILP, it only has binary variables and no continuous ones. We usually call such problems Binary Linear Programs (BLP).

2.2.7.5 Multiprocessor scheduling with communication delays

A *scheduling problem* generally has two main types of decision variables: assignment (of tasks to processors) and sequencing (order of tasks for each processor). The *makespan* is the time taken to finish processing the last task: an optimal schedule usually minimizes the makespan. The Multiprocessor Scheduling Problem with Communication Delays (MSPCD) consists in finding a minimum makespan schedule in order to run a set V of tasks with given duration L_i (for $i \in V$) on an arbitrary network P of homogeneous processors. Tasks must be executed according to a certain partial precedence order encoded as a digraph $G = (V, A)$, where $(i,j) \in A$ if task i must be completed before task j can start. If $i, j \in V$ are assigned to different processors $h, k \in P$, they incur a communication cost penalty γ_{ij}^{hk}, which depends on the (given) distance d_{hk}, which is in fact the length of a shortest path from h to k in P, as well as on the (given) amount of exchanged data c_{ij} that needs to be passed from task i to task j. In summary, $\gamma_{ij}^{hk} = \Gamma c_{ij} d_{hk}$, where Γ is a given constant.

2.2.4 Remark (Meanings of V)

We assume that V is actually a set of integers $\{1, \ldots, |V|\}$ indexing the tasks, so that we can talk about task i (for $i \in V$) but also the s-th task on some processor (for $s \in V$). While syntactically this notation is valid, it might engender some confusion to do with the fact that task i might not be the i-th task on the processor it is assigned to. You should consider that V stands in fact for two different collections: one of the task indices, and the other of the order indices. □

We define two sets of decision variables as follows:

- binary variables y denoting assignment and sequencing:

$$\forall i, s \in V, k \in P \; y_{ik}^s = \begin{cases} 1 & \text{task } i \text{ is the } s\text{-th task on processor } k \\ 0 & \text{otherwise,} \end{cases}$$

- continuous variables $t_i \geq 0$ determining the starting time for task i (for $i \in V$).

The MSPCD can now be formulated as follows:

$$\left.\begin{array}{rrcll} \min\limits_{y,t} & \max\limits_{i \in V}(t_i + L_i) & & & (1) \\ \forall i \in V & \sum\limits_{k \in P} \sum\limits_{s \in V} y_{ik}^s & = & 1 & (2) \\ \forall k \in P & \sum\limits_{i \in V} y_{ik}^1 & \leq & 1 & (3) \\ \forall k \in P, s \in V \setminus \{1\} & \sum\limits_{i \in V} y_{ik}^s & \leq & \sum\limits_{i \in V} y_{ik}^{s-1} & (4) \\ \forall j \in V, i \in N^-(j) & t_i + L_i + \sum\limits_{h \in P} \sum\limits_{s \in V} \sum\limits_{k \in P} \sum\limits_{r \in V} \gamma_{ij}^{hk} y_{ih}^s y_{jk}^r & \leq & t_j & (5) \\ \forall i, j \in V, k \in P, s \in V \setminus \{n\} & t_i + L_i - M\left(2 - \left(y_{ik}^s + \sum_{r=s+1}^n y_{jk}^r\right)\right) & \leq & t_j & (6) \\ \forall i, s \in V, k \in P & y_{ik}^s & \in & \{0,1\} & (7) \\ \forall i \in V & t_i & \geq & 0, & (8) \end{array}\right\} \quad (2.11)$$

where $M \gg 0$ is a "sufficiently large" (given) penalty coefficient, and $N^-(j)$ is standard graph theoretical notation to mean the *incoming neighbourhood* of node $j \in V$, i.e. the set of nodes $i \in V$ such that $(i, j) \in A$.

Equation (2) ensures that each task is assigned to exactly one processor. Inequalities (3)-(4) state that each processor can not be simultaneously used by more than one task: (3) means that at most one task will be the first one at k, while (4) ensures that if some task is the s^{th} one (for $s \geq 2$) scheduled to processor $k \in P$ then there must be another task assigned as $(s-1)$-st to the same processor. Inequality (5) expresses the precedence constraints together with communication time required for tasks assigned to different processors. Inequality (6) defines the sequence of the starting times for the set of tasks assigned to the same processor: it expresses the fact that task j must start at least L_i time units after the beginning of task i whenever j is executed after i on the same processor k; the M parameter must be large enough so that constraint (6) is active only if i and j are executed on the same processor k and $r > s$ (for $i, j, r, s \in V$, $k \in P$).

2.2.5 Remark (Presenting a MP formulation)
Each row of a MP formulation is split in four columns.

- *In the first column, quantifiers:* min *and* max *for objective functions,* $\forall$ *or none for constraints;*
- *In the second column, functional forms (objective and constraints);*
- *In the third column, equality sign or* $\leq, \geq$ *signs for functional constraints,* $\in$ *for implicit constraints (nothing for objectives);*
- *In the fourth column, constants or functional forms for functional constraints, set names or labels for implicit constraints (nothing for objectives).*

2.2.6 Remark (Objective function)
The objective function has a min max *form, and is therefore not linear. Formulations exhibiting a* min max *optimization direction are known as* saddle problems. *Though this might deceptively look like one, it is not: in saddle problems you minimize over a subset of variables and maximize over the rest. In Eq. (2.11) the inner maximization occurs w.r.t. an index ranging over a set given as an input parameter. In other words, the objective function expression* $f(p, x)$ *given in Eq. (2.2) is* $\max\limits_{i \in V}(t_i + L_i)$*: it corresponds to the ending time of the last task, i.e. the makespan.*

2.2.7 Exercise
Reformulate the objective function (1) of Eq. (2.11) exactly so that it becomes linear in the decision variables.

Answer. It suffices replace the objective expression with a new decision variable μ (to denote the makespan) and adjoin some new constraints:

$$\forall i \in V \quad \mu \geq t_i + L_i, \qquad (1a)$$

so the whole Eq. (2.11) becomes $\min\{\mu \mid (1a),(2)\text{-}(8)\}$.

2.2.8 Remark (Products of binary variables)
Note that Constraint (5) in Eq. (2.11) involves a sum of products of two binary variables. Without any further analysis on the problem structure, we are forced to conclude that this formulation belongs to the (nonconvex) MINLP class. This is bad news according to Sect. 2.2.5. Fortunately, however, roducts of binary variables can be reformulated exactly to a linear form. Whenever any MP involves a product $x_i x_j$ where $x_i, x_j \in \{0,1\}$, proceed as follows: replace the product with an additional variable $X_{ij} \in [0,1]$ (this is the linearization process described in Sect. 2.2.3.5), and adjoin the following constraints:

$$X_{ij} \leq x_i \qquad (2.12)$$
$$X_{ij} \leq x_j \qquad (2.13)$$
$$X_{ij} \geq x_i + x_j - 1. \qquad (2.14)$$

This reformulation was first proposed in [113]. It can be generalized to products where only one of the variables is binary, as long as the other is bounded above and below [178]. If both variables are continuous and bounded, Eq. (2.12)-(2.14) can be generalized to obtain a convex envelope of the set $\mathcal{X}^{ij} = \{(X_{ij}, x_i, x_j) \in B^{ij} \mid X_{ij} = x_i x_j\}$ (where B^{ij} is a box defined by lower and upper bounds on x_i, x_j and the corresponding bounds on X_{ij} obtained by interval analysis) [215].

2.2.9 Exercise
With reference to Rem. 2.2.8, prove that Eq. (2.12)-(2.14) provide an exact reformulation of the set $\mathcal{X}^{ij} \cap [0,1] \times \{0,1\}^2$.

Answer. We proceed by cases: if at least one of x_i, x_j has value 0, their product is 0 and $X_{ij} = 0$ by either Eq. (2.12) or Eq. (2.13). If $x_i = x_j = 1$ then the product has value 1 and $X_{ij} = 1$ by Eq. (2.14).

The formulation in Eq. (2.11) uses a rather standard set of variables for most scheduling problems. It is not easy to solve, however, even for medium-sized instances, since it has $O(|V|^3)$ variables and $O(|V|^3|P|)$ constraints. See [93] for a more compact formulation.

2.2.7.5.1 The infamous "big M" Anyone who is familiar with the integer programming literature knows about the "big M", its widespread use, and its unanimous condemnation. The need for a "large" constant M arises when a constraint needs to be activated or deactivated according to the value of a given binary decision variable (such constrained are known as conditional constraints). Suppose we have a constraint $g(x) \leq g_0$ which should only be active when the variable $y \in \{0,1\}$ has value 1.

A maximally inexperienced reader would probably come up with the expression "$\forall y \in \{1\}\ g(x) \leq g_0$", which unfortunately involves the decision variable y appearing in a constraint quantifier, against Rem. 2.2.3. A naive reader might propose multiplying both sides of the constraint by y, so that the constraint (now reformulated as $yg(x) \leq yg_0$) will be inactive when $y = 0$ and active when $y = 1$, as desired. The issue here is that this introduces unnecessary products in the formulation, which, according to our analysis of the state of the art in solvers (Sect. 2.2.5) should be best avoided.

The common practice is to assume that $g(\cdot)$ is bounded above over its domain, say $g(x) \leq g^U$ for all possible values of the decision variable vector x. One would then reformulate the constraint by choosing some parameter $M \geq g^U - g_0$ and writing:

$$g(x) \leq g_0 + M(1 - y), \qquad (2.15)$$

so that when $y = 1$ we retrieve $g(x) \leq g_0$, whereas with $y = 0$ we simply have $g(x) \leq g_0 + M$, which implies $g(x) \leq g^U$, which is always true (and therefore does not actively constrain the problem anymore).

MILP formulations involving "big M"s are often condemned because their continuous relaxations (obtained by relaxing the integrality constraints) yield a bound that is known to often have a large gap with the optimal objective function value of the original problem. MILP solvers therefore take much longer to achieve termination. It is also known that this adverse impact is lessened when M is as small as possible. So when people write "big M" what they really mean is "big enough": make an effort to correctly estimate M as an upper bound to the (possibly unknown) g^U.

2.2.7.6 Graph partitioning

The Graph Partitioning Problem (GPP), also known as min-k-cut problem is part of a larger family of clustering problems, which form the methodological basis of unsupervised ML. Given an undirected graph $G = (V, E)$ and two integers $L, k \geq 2$, it asks for a partition of V into k (disjoint) subsets, called *clusters*, such that each cluster has cardinality bounded above by L, and such that the number of edges $\{i, j\} \in E$ such that i, j belong to different clusters is minimized [116, ND14].

In general, the GPP is **NP**-hard for all $k \geq 3$. The particular case $k = 2$ is known as the min cut problem, and can be solved in polynomial time by computing the solution of its dual,[4] the max flow problem (see Sect. 2.2.7.3).

The parameters of the problem are given by an encoding of G and the two integers L, k. As decision variables we employ two-indexed binary variables in order to assign vertices to clusters:

$$\forall i \in V, h \leq k \quad x_{ih} = \begin{cases} 1 & \text{if vertex } i \text{ is in cluster } h \\ 0 & \text{otherwise.} \end{cases}$$

The formulation is as follows:

$$\left.\begin{array}{rrcll} \min\limits_{x} & \sum\limits_{\{i,j\}\in E} \sum\limits_{h\leq k} x_{ih}(1 - x_{ih}) & & & (1) \\ \forall h \leq k & \sum\limits_{i\in V} x_{ih} & \leq & L & (2) \\ \forall i \in V & \sum\limits_{h\leq k} x_{ih} & = & 1 & (3) \\ \forall i \in V, h \leq k & x_{ih} & \in & \{0, 1\}. & (4) \end{array}\right\} \tag{2.16}$$

We remark that, for each edge $\{i, j\} \in E$ and $h \leq k$, the product $x_{ih}(1 - x_{jh})$ has value 1 iff i, j are assigned to different clusters: thus the objective function (1).

2.2.10 Remark (Data structure representation for edges)
If you implement Eq. (2.16), be aware that if $\{i, j\}$ is represented as a couple (i, j) rather than an unordered pair, then you are likely to have to multiply the objective function by $\frac{1}{2}$, unless you remove the repeated couples by means of a condition $i < j$. In general, however, whether you are allowed to express $i < j$ also depends on the other features of the MP you are considering; sometimes the most expedited solution is to count edges twice and divide by two.

As for Constraint (2), it expresses the fact that each cluster must have cardinality bounded above by L. Constraint (3) states that each vertex must be assigned to exactly one cluster (i.e. the clustering is a partition).

The formulation in Eq. (2.16) belongs to the class of MINLP, since the objective function is quadratic, and possibly nonconvex. An exact reformulation to a MILP can be achieved by employing Fortet's reformulation technique (see Remark 2.2.8).

[4] See e.g. [240] for an introduction to LP duality, as well as the special form of "combinatorial" duality given by the pair max flow – min cut.

2.2.7.7 Haverly's Pooling Problem

Haverly's Pooling Problem (HPP), first introduced in [134], is described visually in Fig. 2.2. We have three input feeds with different levels of percentage of sulphur in the crude. Accordingly, the unit costs of the input feeds vary. Two of the feeds go into a mixing pool, which makes the level of sulphur percentage the same in the two streams coming out of the pool. The various streams are then blended to make two end products with different requirements on the sulphur percentage and different unit revenues. We have some market demands on the end products and we want to determine the quantities of input crude of each type to feed into the networks so that the operations costs are minimized. This problem can be

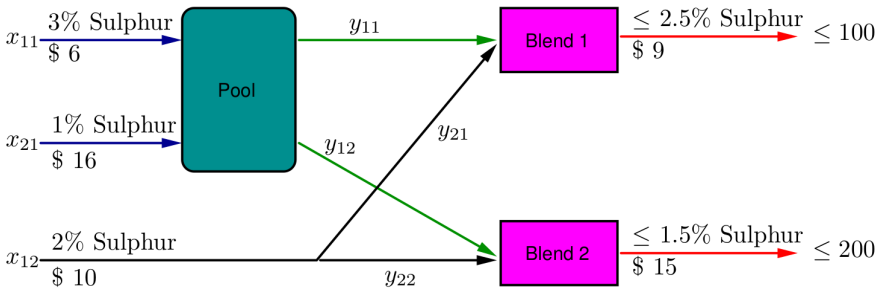


Figure 2.2: Haverly's Pooling Problem.

formulated in various ways. We present here what is known as the "p-formulation" of the HPP [25], with comments on the constraints explaining their meaning.

$$
\begin{array}{lll}
\min\limits_{x,y,p\geq 0} & 6x_{11} + 16x_{21} + 10x_{12} - 9(y_{11} + y_{21}) - 15(y_{12} + y_{22}) & \text{cost} \\
\text{s.t.} & x_{11} + x_{21} - y_{11} - y_{12} = 0 & \text{mass balance} \\
& x_{12} - y_{21} - y_{22} = 0 & \text{mass balance} \\
& y_{11} + y_{21} \leq 100 & \text{demands} \\
& y_{12} + y_{22} \leq 200 & \text{demands} \\
& 3x_{11} + x_{21} - p(y_{11} + y_{12}) = 0 & \text{sulphur balance} \\
& py_{11} + 2y_{21} \leq 2.5(y_{11} + y_{21}) & \text{quality req} \\
& py_{12} + 2y_{22} \leq 1.5(y_{12} + y_{22}) & \text{quality req,}
\end{array}
$$

where x, y are decision variable vectors representing input and intermediate stream quantities (as shown in Fig. 2.2) and p is the decision variable scalar representing the percentage of sulphur in the streams out of the pool. Notice that this NLP has three constraints involving bilinear terms, so Global Optimization (GO) techniques are mandatory for its solution. Since bilinear terms can (and usually do) yield nonconvex feasible regions when used in constraints, we can only say that this is a nonconvex NLP. Note that the implicit non-negativity constraints appear under the minimization direction operator min.

The formulation presented above is the flat formulation (it has no quantifiers ranging over indices) of a single instance (it has no parameters). This instance is part of the problem that can be obtained by replacing the numbers $6, 16, 10, 9, 15, 100, 200, 2.5, 1.5$ in the objective and constraints by parameter symbols ranging over non-negative real scalars. It is this problem that is actually called HPP. Many generalizations of the HPP have been studied in the literature, most notably with respect to the network topology, to the types of oil impurity (e.g. the sulphur) and to the activation/de-activation of conducts in the oil network [225], see Sect. 2.2.7.8 below.

2.2.7.8 Pooling and Blending Problems

The HPP (Sect. 2.2.7.7) is a subclass of more general problems termed Pooling/Blending problems. Various types of crude oils of different qualities, coming from different feeds, are mixed together in

various pools to produce several end-products subject to quality requirements and quantity demands; the objective is to minimize the net costs. These problems are usually modelled as continuous NLPs involving bilinear terms. The literature on Pooling/Blending problems is vast (see e.g. [3, 114, 300, 286, 25, 191]). We present the general blending problem formulation found in [3, p. 1958]. The formulation refers to a setting with q pools each with n_j input streams ($j \leq q$) and r end products. Each stream has l qualities.

$$\left.\begin{array}{rll} \min_{x,y,p\geq 0} \sum_{j=1}^{q}\sum_{i=1}^{n_j} c_{ij}x_{ij} - \sum_{k=1}^{r} d_k \sum_{j=1}^{q} y_{jk} & & \\ \sum_{i=1}^{n_j} x_{ij} = \sum_{k=1}^{r} y_{jk} & \forall j \leq q & \\ \sum_{j=1}^{q} y_{jk} \leq S_k & \forall k \leq r & \\ \sum_{i=1}^{n_j} \lambda_{ijw} x_{ij} = p_{jw} \sum_{k=1}^{r} x_{jk} & \forall j \leq q\ \forall w \leq l & \\ \sum_{j=1}^{q} p_{jw} x_{jk} \leq z_{kw} \sum_{j=1}^{q} y_{jk} & \forall k \leq r\ \forall w \leq l & \\ x^L \leq x \leq x^U, p^L \leq p \leq p^U, y^L \leq y \leq y^U, & & \end{array}\right\} \quad (2.17)$$

where x_{ij} is the flow of input stream i into pool j, y_{jk} is the total flow from pool j to product k and p_{jw} is the w-th quality of pool j; $c_{ij}, d_k, S_k, Z_{kw}, \lambda_{ijw}$ are, respectively: the unit cost of the i-th stream into pool j, the unit price of product k, the demand for product k, the w-th quality requirement for product k and the w-th quality specification of the i-th stream into pool j.

The variable symbols are x, y, p, as emphasized under the optimization direction symbol min. The other symbols indicate parameters.

2.2.7.9 Euclidean Location Problems

There are n plants with geographical positions expressed in Euclidean coordinates (a_i, b_i) $(1 \leq i \leq n)$ w.r.t. to an arbitrary point $(0, 0)$. Daily, the i-th plant needs r_i tons of raw material, which can be delivered from m storage points each with storage capacity c_j $(1 \leq j \leq m)$. For security reasons, each storage point must be located at least at $D = 1$ Km distance from the plant. The cost of raw material transportation between each storage point and each plant is directly proportional to their distance as well as the quantity of raw material. Find geographical positions where to place each storage point so that the transportation costs are minimized. The MP formulation is as follows:

$$\left.\begin{array}{lll} \min_{x,y,d,w} & \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}d_{ij} & \\ \text{s.t.} & \sum_{i=1}^{n} w_{ij} \leq c_j \quad \forall j \leq m & \text{storage capacity} \\ & \sum_{j=1}^{m} w_{ij} \geq r_i \quad \forall i \leq n & \text{plant requirement} \\ & d_{ij} = \sqrt{(x_j - a_i)^2 + (y_j - b_i)^2} \quad \forall i \leq m, j \leq n & \text{Euclidean distance} \\ & d_{ij} \geq D \quad \forall i \leq n, j \leq m & \text{minimum distance} \end{array}\right\}$$

where (x_j, y_j) is the geographical position of the j-th storage point, d_{ij} is the distance between the i-th plant and the j-th storage point, w_{ij} is the quantity of raw material transported from the j-th storage point to the i-th plant $(i \leq n, j \leq m)$. The decision variable symbols are x, y, d, w. The rest are parameter symbols.

2.2.7.10 Kissing Number Problem

When billiard balls touch each other they are said to "kiss". Accordingly, the *kissing number* in D dimensions is the number of D-dimensional spheres of radius R that can be arranged around a central

D-dimensional sphere of radius R so that each of the surrounding spheres touches the central one without their interiors overlapping. Determining the maximum kissing number in various dimensions has become a well-known problem in combinatorial geometry. Notationally, we indicate the Kissing Number Problem in D dimensions by KNP(D). In $\mathbb{R}^2$ the result is trivial: the maximum kissing number is 6.

2.2.11 Exercise

Prove that the kissing number in 2 *dimensions is* 6.

> **Answer**. Construct an arrangement of 6 unit circles around a central unit circle and prove it leaves no space for any other circle.

The situation is far from trivial in $\mathbb{R}^3$. The problem earned its fame because, according to Newton, the maximum kissing number in 3D is 12, whereas according to his contemporary fellow mathematician David Gregory, the maximum kissing number in 3D is 13 (this conjecture was stated without proof). This question was settled, at long last, more than 250 years after having been stated, when J. Leech finally proved that the solution in 3D is 12 [169]. As you can see in Fig. 2.3, there is some slack around the sphere, so it is not obvious that 13 non-overlapping spheres cannot fit. Given parameters D (number

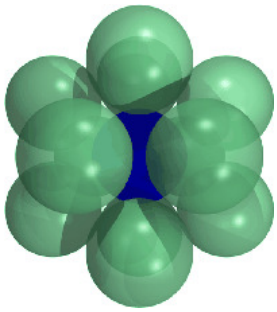


Figure 2.3: A solution of Kissing Number Problem in 3D with 12 spheres.

of dimensions) and N (number of spheres), the variables $x^i = (x^i_1, \ldots, x^i_D)$, $1 \leq i \leq N$ determine the position of the center of the i-th sphere around the central one. We maximize a decision variable $\alpha \geq 0$ which represents the minimum pairwise sphere separation distance in the N-sphere configuration being tested, subject to the necessary geometric constraints. Since the constraints are nonconvex, there may be multiple local minima. If the solution of the formulation determines that the global maximum is at $\alpha \geq 1$, then there is enough space for N spheres; if the globally optimal α is strictly less than 1, it means that the N configuration has overlapping spheres, hence the kissing number is $N - 1$. By solving this decision problem repeatedly for different values of N, we are able to quickly pinpoint the maximum N for which $\alpha > 1$. The following formulation correctly models the problem:

$$\begin{aligned}
\max \quad & \alpha && (2.18)\\
\forall i \leq N \quad & ||x^i||_2 = 2R && (2.19)\\
\forall i < j \leq N \quad & ||x^i - x^j||_2 \geq 2R\alpha && (2.20)\\
& \alpha \geq 0 && (2.21)\\
\forall i \leq N \quad & x^i \in \mathbb{R}^D && (2.22)
\end{aligned}$$

Constraints Eq. (2.19) ensure that the centers of the N spheres all have distance $2R$ from the center of the central sphere (i.e., the N spheres kiss the central sphere). Constraints Eq. 2.20 make the N spheres non-overlapping. This problem has been solved correctly up to $D = 4$ (and $N = 24$, which is KNP(4)) [187].

Chapter 3

The AMPL language

This chapter provides a brief tutorial to the AMPL language. A much better reference is the AMPL book [115], the chapters of which can be downloaded from `www.ampl.com` free of charge. AMPL stands for "A Mathematical Programming Language". It was initially conceived by Robert Fourer, David Gay and Brian Kernighan, although the current software and book are only authored by Fourer and Gay.

AMPL's strong point is that can interface to many different solvers, for LP, MILP, NLP and MINLP. Another endearing feature is that MP formulations written in AMPL are very close to the mathematical syntax (much closer to it than, say, what the GAMS [58] language achieves). Its weak point is that its algorithmic capabilities are quite limited.

AMPL is a commercial software: even with academic pricing, a yearly license runs into the hundreds of dollars (but GAMS costs much more, and MATLAB [209] more than either, specially if you order a decent set of add-ons). A subset of the AMPL language is implemented within the open-source GNU-licensed GLPK [206] solver; but GLPK only offers two solvers (a good LP solver, and a somewhat primitive MILP solver). Note that with both AMPL and GAMS the cost of the solvers is extra — and often you need to purchase solvers from other distributors. Luckily, AMPL makes it possible for teachers to obtain time-limited teaching licenses free of charge, which come with a lot of embedded solvers.

We will not cover installation here: we assume you obtained an AMPL distribution with a good set of solvers (particularly CPLEX, IPOPT, BonMin and Couenne), and that paths are correctly set, so that if you type `ampl` or `cplex` on the command line, you will invoke the correct executables.

3.1 The workflow

Although AMPL comes with a rather primitive GUI, we focus on a command-line driven workflow. You can see the `ampl` executable as an interpreter: a program which inputs a text file containing instructions and executes them one by one. Instruction files traditionally have the extension `.run`. So the basic calling sequence (from Linux, MacOSX or Windows) is:

```
ampl < file.run
```

If you are on a Unix system (e.g. Linux and MacOSX, but to some extent this might also work with Windows), you can exploit the operating system's "pipe"-style filters: you instruct AMPL to provide its output as formatted text, and pass the output to a second `command` which accepts that format:

```
ampl < file.run | command
```

In this setting, `command` might perform tasks such as writing a graphical file with a visualization of the output of your `file.run` program, or perform further algorithmics on the output of your program.

3.2 Input files

The simplest way to solve a MP formulation using AMPL involves three text files:

- a `.run` file, containing some imperative instructions (e.g. read/write data from/to the console or files, perform loops and/or tests);
- a `.mod` file, containing some declarative instructions (e.g. declare and define index sets, parameters, variables, objectives, constraints, formulations)
- a `.dat` file, containing the values to assign to the index sets and parameters.

Typically, all files share the same name (e.g. `mymodel.run`, `mymodel.mod`, `mymodel.dat`). The `.run` file usually instructs AMPL to read the `.mod` and the the `.dat` file.

Dividing a formulation into two files (`.mod` and `.dat`) is meant to help separating symbolic entities from numerical data. In a real world setting, one might need to solve large amounts of instances of the same problem: in that case one would only need to update the `.dat` file, never the `.mod` file.

On the other hand, this division in three text files serves the purpose of making the organization of information clearer to the user. AMPL does not care if you use one, two, three or more files (which might be the case if solving your problem involves more than one formulation).

3.3 Basic syntax

AMPL embeds three sublanguages: one is imperative (used mostly in `.run` files), one is declarative (used mostly in `.mod` files), and the third (the simplest) is only used within `.dat` files.

For all three sublanguages, comments are one line long and are introduced by the hash (`#`) character. Every instruction is completed using a semicolon (`;`), with one exception: within the AMPL imperative sublanguage, if the body of loops or tests is delimited by braces (`{`, `}`), there need not be a semicolon after the closing brace. E.g.,

```
# whole line comment
param i default 0; # rest of line is comment
let i := 4;
```

but

```
if (i == 4) then {
  let i := i + 1;
} else {
  let i := i - 1;
}
```

and

```
for {i in 1..5} {
  let i := i + 1;
}
```

Note that imperative program variables are declared to be formulation parameters (keyword `param`). Assignment of values to program variables requires the keyword `let` and the assignment operator `:=`, while the equality test uses `==` (the operator `=` is reserved for use in the definition of equality constraints). The notation `1..5` denotes the set $\{1, 2, 3, 4, 5\}$. For more about test/loop syntax, see the AMPL book [115].

3.4 LP example

Let us model the following LP using AMPL:

$$\min\{c^\top x \mid Ax \geq b \wedge x \geq 0\},$$

where c, A, b are parameters and x decision variables. AMPL will not accept matrix notation, so we really have to write the formulation as follows:

$$\left.\begin{array}{rll} \min & \sum\limits_{j \in N} c_j x_j & \\ \forall i \in M & \sum\limits_{j \in N} A_{ij} x_j & \geq \quad b_i, \end{array}\right\}$$

which involves two index sets M (for the constraints) and N (for the variables), a cost vector $c \in \mathbb{R}^n$, a RHS vector $b \in \mathbb{R}^m$ and an $m \times n$ matrix A.

3.4.1 The `.mod` file

The syntax of a `.mod` file is declarative. It segments a MP formulation into five different entities: sets (`set`), parameters (`param`), variables (`var`), objective (introduced by either `minimize` or `maximize`) and constraints (introduced by `subject to`). Every entity is named. The name may be quantified over an index set. For example if `N` is an index set, you can declare variables x_i for $i \in$ `N` as follows:

```
var x{i in N};
```

We write the `.mod` file as follows.

```
## .mod file for an LP in form min cx : Ax >= b
# parameters and sets
param m integer, >0, default 50; # number of constraints
param n integer, >0, default 10; # number of variables
set M := 1..m; # index set for constraints
set N := 1..n; # index set for variables
param c{N} default Uniform01();  # objfun coeff vector
param A{M,N} default Uniform(-1,1); # constr matrix
param b{M} default Uniform(1,2); # RHS vector
# decision variables
var x{N} >= 0;
# objective function
minimize myObj: sum{j in N} c[j]*x[j];
# constraints
```

```
subject to myConstr{i in M}: sum{j in N} A[i,j]*x[j] >= b[i];
# solve the problem
option solver cplex;  # choose the solver
solve;  # solve it
# display the output
display x, objective, solve_result;
```

Some remarks follow.

- Every entity declaration supports a list of modifiers separated by commas: the parameter symbol `m` is required to be integer, positive, and, if uninitialized elsewhere, be assigned the value 50. The decision variable symbol `x`, quantified over `N`, declares a variable vector each component of which is constrained to be nonnegative.
- Modifiers such as `integer`, `>0` and `>=0` are applied to both parameters and variables: their meaning in these different context is correspondingly different. When a parameter is constrained, it will force AMPL to abort its execution whenever it is assigned (within the `.run` or `.dat` file) a value which does not satisfy the constraints. When a variable is constrained, the constraint will be passed to the solver. Constraining parameters is therefore a form of *input data validation*, whereas constraining variables is an essential part of an MP formulation.
- The parameters `c`, `A`, `b` are quantified over index sets, and describe vectors and matrices. Notwithstanding, their assigned default value is the output of the functions `Uniform01` and `Uniform`, which is a scalar. This means that every component of the vectors and matrix will be assigned the scalar returned by the function, which is called on every component. In other words, `c`, `A`, `b` are random vectors and a random matrix. Suppose we defined a parameter `C` quantified over `N`, to which we would like to assign default values taken from `c`. In this case, the declaration should be written as follows:

  ```
  param C{j in N} default c[j];
  ```

  Note that we explicitly mention the index `j` ranging over `N` in order to use it to index `c` (which is also quantified over the same set `N`).
- In order to initialize the index sets `M` and `N`, we use two scalar integer parameters `m` and `n`, which allow us to define `M` and `N` as sequence of consecutive integers from 1 to `m` (respectively, `n`). This is often done when defining consecutive integer ranges. You can also initialize a set within a `.mod` file by using normal set notation:

  ```
  set M := {1,3,6};
  ```

  or range notation:

  ```
  set M := 1..m;
  ```

  We remark that initialization of sets in `.dat` files follows a different syntax (see Sect. 3.4.2).
- Initializing a parameter or set with a value within a declaration instruction in a `.mod` can be done with two constructs: the assignment operator `:=` and the `default` keyword, e.g.:

  ```
  param m default 50;
  param n := 10;
  set M := 1..50;
  set N default {1,2,3,4};
  ```

Those sets/parameters initialized with an assignment operator `:=` are initialized once and for all during the AMPL execution. The initialization value assigned with the `default` keyword can be changed (e.g. using `let`) during the AMPL execution.

- The label `myObj` is the name we chose for the objective function. The label`myConstr` is the name we chose for the constraints. Since LPs only have one objective, we do not need to quantify `myObj` over an index set. On the other hand, we need to quantify `myConstr` over the constraint index set `M`.
- The choice of solver is made by specifying the `option solver`. The name which follows (in this case, `cplex`) must correspond to a solver executable file by the same name available to AMPL for calling (which requires adding the solver executable directory to the default path). The instruction `solve` will flatten the formulation, pass it to the solver, retrieve the (flat) solution from the solver, and re-organize the solution components in structured format.
- The `display` command is the simplest of AMPL's output commands. See the AMPL book [115] for more information.
- The variable `solve_result` is a system variable (i.e. it exists in AMPL's namespace by default — no need to declare it), in which solvers may store a string description of their return status.

3.4.2 The `.dat` file

An AMPL `.dat` file simply stores the input data in an AMPL-specific format. Users can define parameter values and set contents in a `.dat` file.

Sets are initialized in `.dat` files by providing a list of elements separated by spaces, without brace enclosure:

```
set N := 1 2 3;
set M := 1 2;
```

Parameters are initialized in various ways (see the AMPL book for more details). We shall only cover the basics here.

- Initializing a vector is done as follows.

```
param c :=
  1  0.4
  2  0.1
  3  1.23 ;
```

- Supposing you did not initialize the index set `N` in the `.mod` file, and you cannot be bother to list its contents explicitly using the constructs above, you can do it implicitly in the definition of the first parameter indexed on `N`.

```
param : N : c :=
  1  0.4
  2  0.1
  3  1.23 ;
```

- If two (or more) vectors (such as `c`, `C`) are defined over the same index set `N`, you can list the index set once only.

```
param : c C :=
  1  0.4   0.9
  2  0.1   0.8
  3  1.23 -0.7 ;
```

- You can also initialize `N` while defining both `c` and `C`.

```
param : N : c C :=
  1  0.4   0.9
  2  0.1   0.8
  3  1.23 -0.7 ;
```

- Initializing a matrix (or a tensor) simply requires more index columns.

```
param A :=
  1 1   1.1
  1 2   0.0
  1 3   0.3
  2 1   0.0
  2 2  -0.31
  2 3   0.0;
```

- If your matrix is sparse, make sure you initialize it to zero in the `.mod` file (using the `default 0` modifier in the declaration), and then simply initialize the nonzeros in the `.dat` file (otherwise your `.dat` files will be enormous).

```
param A :=
  1 1   1.1
  1 3   0.3
  2 2  -0.31;
```

- Note that the format (columns, line breaks) is arbitrary. By leveraging its knowledge of the index sets of the parameters, necessarily declared in the `.mod` file, AMPL will not mind whether you write the previous matrix definition as follows.

```
param A := 1 1 1.1 1 3 0.3 2 2 -0.31;
```

  Since the previous syntax is clearer (to a human), it is preferable.

3.4.3 The `.run` file

A minimal `.run` file, sufficient for reading the model, the data, selecting the solver and solving the instance is as follows.

```
## .run file
# read the model file
model file.mod;
# read the data file
model file.dat;
# choose the solver
option solver cplex;
# solve the problem
solve;
# display the solver status
display solve_result;
```

3.5 The imperative sublanguage

Although some heuristics can easily be coded in AMPL, more complicated algorithms might be unwieldy, as AMPL has (among other limitations) no construct for function calls. Mostly, the imperative sublanguage is very useful for formatting the output. You might for example instruct AMPL to write the solution of a circle packing problem (Sect. 1.3.10) in Python, and then use the latter to display the solution graphically, as in the following AMPL code snippet:

```
print "import matplotlib.pyplot as plt" > circlepacking_out.py;
for {i in N} {
  printf "circle%d = plt.Circle((%g,%g),%g)\n", i, x[i], y[i], r >> circlepacking_out.py;
}
print "fig = plt.gcf()" >> circlepacking_out.py;
for {i in N} {
  printf "fig.gca().add_artist(circle%d)\n", i >> circlepacking_out.py;
}
print "fig.savefig('circlepacking_out.png')" >> circlepacking_out.py;
```

Part II

Computability and complexity

Chapter 4

Computability

In this chapter we treat the very basic question "can we even solve an optimization problem using a computer?" In general, this question can be asked of any type of broad problem category, be it decision, search or optimization. The field of logic that studies this question is called *computability*. The questions are asked in a given *computing model*, which is usually the Turing machine (TM) model of computation (though it need not be). We want the answers to be valid for every problem in the class of interest. For example, every (finite) LP with rational input data can be solved using a TM that implements the simplex method. Conversely, there can be no TM that is able to solve every MINLP.

4.1 A short summary

We first give a very short summary of concepts in computability theory.

4.1.1 Models of computation

The computer was first conceived by Alan Turing in 1936 [288]. Turing's mathematical model of the computer is called *Turing machine*. A TM consists of an infinite tape, divided into a countably infinite number of cells, with a device (called *head*) that can read or write symbols out of a given alphabet A on each cell of the tape. According to the state $s \in S$ the TM is in, the head either reads, or writes, or moves its position along the tape. The description of any TM includes a set of instructions which tell it how to change its state. Turing showed that there exist TMs which can simulate the behaviour of any other TM: such TMs are called *Universal TMs*

Turing's work spawned further research, from the 1950s onwards, aimed at simplifying the description of UTMs, involving scientists of the caliber of Shannon [271] and Minsky [224]. More recently, Rogozhin [254] described UTMs with low values of $(|S|, |A|)$, e.g. $(24, 2)$, $(10, 3)$, $(7, 4)$, $(5, 5)$, $(4, 6)$, $(3, 10)$, $(2, 18)$. It appears clear that there is a trade-off between number of states and number of symbols in the alphabet.

UTMs are not the only existing models of computation — several others exist, sometimes very different from each other (see e.g. the *Game of Life*, a model for bacteria diffusion [42]). Such models are said to be *Turing-complete* if they can simulate a UTM. "Simulating", in this context, means using a different model C of computation in order to mimick the behaviour of a UTM. To prove this, it suffices to show that every instruction, state and symbol of the UTM can be simulated by C. If the UTM can also simulate C, then C is said to be *Turing-equivalent*.

Church's Thesis is the statement that every Turing-complete model of computation is also Turing-

equivalent. So far, no Turing-complete model of computation was found to be more "powerful" than the UTM.

4.1.2 Decidability

Functions $\mathbb{N} \to \mathbb{N}$ described by a TM are called *computable*. The term *decidable* applies to relations instead of functions: a k-ary relation R on D^k is decidable if, given $d_1, \ldots, d_k \in D$, there exists a TM that decides whether the k-tuple $d = (d_1, \ldots, d_k)$ is in R. Since "being in R" can be encoded as a YES/NO type of question, decidability applies to decision problems.

Sets $V \subseteq \mathbb{N}$ that are domains (or co-domains) of computable functions are called *recursively enumerable*. If V and $\mathbb{N} \setminus V$ are both recursively enumerable, then they are both also called *recursive* or *decidable*.

Given a set $V \subseteq \mathbb{N}$ and an integer $v \in \mathbb{N}$, one may ask whether $v \in V$ or not. This is a fundamental decision problem (in number theory — but a computer represents every data structure with numbers in base 2, so this limitation is only formal). It turns out that V is recursively enumerable if and only if there is a program that answers YES when $v \in V$, but may answer wrongly or even fail to terminate when $v \notin V$; moreover, V is decidable if and only if there is a program that answers YES when $v \in V$ and NO when $v \notin V$.

It should appear clear that recursively enumerable sets are of limited value as far as proving that an answer to a problem is correct: if the algorithm answers YES, it may be a true positive or a false negative, and there is no way to tell. Moreover, how is a user supposed to know whether a program fails to terminate or is simply taking a long time? Accordingly, we hope to consider problems so that the solution set is decidable. On the other hand, many interesting sets arising in optimization problems are only recursively enumerable (or worse).

4.2 Solution representability

The solutions of LPs and MILP have rational components as long as the input data is rational. For MINLP, the situation is not so clear: solution components might involve irrational numbers (both algebraic and transcendental, depending on the involved functions). We list four approaches that attempt to deal with the issue of representing such solutions within a finite amount of storage.

4.2.1 The real RAM model

The computation model of Blum, Shub and Smale [47] (often referred to as *real RAM* model) essentially eschews the problem of representation, as it defines the real equivalent of computational complexity classes such as **P**, **NP** and their relationships. In the real RAM model, storing, reading or performing arithmetic on real numbers has unit cost [47, p. 24]. This model of computation is mostly used for theoretical results concerning algorithms in numerical analysis and scientific computing.

4.2.2 Approximation of the optimal objective function value

The approach taken by the GO community, specially those who develop sBB algorithms, consists in finding a (rational, or rather floating point encoded) solution x^* yielding an objective function value $f^* = f(x^*)$ that is within a given $\varepsilon > 0$ of the true globally optimal function value $\tilde{f}$ at a true global

optimum $\tilde{x}$:

$$|f^* - \tilde{f}| \leq \varepsilon. \tag{4.1}$$

Since the true optimal value $\tilde{f}$ is not known, in practice Eq. (4.1) is enforced by requiring that $|f^* - \bar{f}| \leq \varepsilon$, where $\bar{f}$ is a guaranteed lower bound to the optimal objective function value, computed e.g. by solving an LP relaxation of Eq. (2.2). One of the issues with this approach is that, depending on the instance being solved, Eq. (4.1) might be satisfied even if $\|x^* - \tilde{x}\|$ is arbitrarily large.

4.2.3 Approximation of the optimal solution

The approach taken by Dorit Hochbaum [141] is more accurate, as it insists that the solution x^* is within a given $\varepsilon > 0$ tolerance of the true optimum $\tilde{x}$:

$$\|x^* - \tilde{x}\|_\infty \leq \varepsilon. \tag{4.2}$$

This means that x^* is the same as $\tilde{x}$ in $O(\log \frac{1}{\varepsilon})$ decimal digits. Other than that, all arithmetical operations on reals take $O(1)$ time. Since one only needs to consider $O(\log \frac{1}{\varepsilon})$ decimal digits, this assumption places this representation approach within the TM model of computation.

4.2.4 Representability of algebraic numbers

A more credible attempt at representing an algebraic number α has been made using of minimal polynomials. The most common is the *Thom encoding*, a pair (p_α, σ), where $p_\alpha(x)$ is the minimal polynomial of α (of degree d, say) and $\sigma : \{0, \ldots, d\} \to \{0, -1, 1\}$ encodes the sign of the k-th derivative of p_α at α [33, Prop. 2.28]. Simpler representations are possible for specific tasks, e.g. σ might be replaced by a rational number a which is closer to α than to any other real root of p_α [35].

4.2.4.1 Solving polynomial systems of equations

Solving polynomial systems of equations exactly is sometimes possible by "diagonalizing" them using *Gröbner bases* and then perform back-substitution, similar to Gaussian elimination.

Gröbner bases can be found by Buchberger's algorithm [59]. It takes as input a rational multivariate polynomial equation system:

$$\forall i \leq m \quad p_i(x) = 0. \tag{4.3}$$

It then proceeds by diagonalizing Eq. (4.3) to a new polynomial system:

$$\forall i \leq m' \quad q_i(x) = 0, \tag{4.4}$$

such that the leading terms of p_i's and q_i's, w.r.t. some given monomial order, generate the same ideal. Let $F = \{p_1, \ldots, p_m\}$. Buchberger's algorithm proceeds as follows:

Fix an ordering $<_F$ on the monomials of F
Let $G \leftarrow F$
repeat
  Let $H \leftarrow G$
  for $p, q \in G$ s.t. $p \neq q$ **do**
    Let $\hat{p}, \hat{q}$ be the leading terms of p, q w.r.t. $<_F$
    Let a be the least common multiple (lcm) of $\hat{p}, \hat{q}$
    Let $s \leftarrow \frac{a}{\hat{p}} p - \frac{a}{\hat{q}} q$ *# This cancels the leading terms of p, q in s*
    Reduce s w.r.t. G using multivariate polynomial division
    if $s \not\equiv 0$ **then**
      Update $G \leftarrow G \cup \{s\}$

12: **end if**
13: **end for**
14: **until** $G = H$

Buchberger's algorithm monotonically increases the size of the ideal generated by the leading terms of G. The algorithm terminates because, by Hilbert's basis theorem, ascending chains of ideals must eventually become stationary. The termination condition is verified even if the last considered polynomial is not univariate.

Like Gaussian elimination, this diagonalization can be used for performing back-substitution as long as the last considered equation $q(x)$ only depends on a single variable. Unlike Gaussian elimination, however, m' generally exceeds m, and it does not yield a guarantee that q_i will contain strictly fewer variables than q_{i+1}: the diagonal structure might contain blocks of polynomials depending on the same set of variables. Even if the back-substitution procedure fails to provide a solution, the diagonal form will still hold after a failure occurs, a condition which might be described as "back-substitute as far as possible".

Unfortunately, Buchberger's algorithm has doubly exponential worst-case complexity in general (see Ch. 5), though it behaves singly exponentially in some cases [28].

An interesting development of Gröbner's bases is given by *chordal networks* [68]. They supply polynomial systems that are cheaper to construct, provide the same back-substitution functionality for finding a solution of a polynomial system of equations. Unfortunately, their size is larger than for Gröbner bases.

4.2.4.2 Optimization using Gröbner bases

There are at least two articles [129, 66] where Gröbner bases are used to diagonalize the first-order optimality conditions, also known as Karush-Kuhn Tucker (KKT) system (see Thm. 6.2.7), of a Polynomial Programming (PP) problem. In both articles, the last phase of the procedure performs back-substitution on the diagonalized polynomial system using floating point numbers, thereby forsaking exactness; but those floating point numbers could in principle be used as "closest rationals" in the representation mentioned above.

4.3 Computability in MP

As mentioned above, we can solve LPs (by means of the simplex method, see Sect. 7.1). We can solve MILPs (by means of the BB algorithm). We cannot solve all cNLPs (and hence even the including classes NLPs, cMINLPs and MINLPs) at least because we have issues with representing the solution exactly, as detailed in Sect. 4.2.

We focus on MINLP. While we cannot solve them in full generality (see Sect. 4.3.2 below), the full answer is rather more interesting: MINLP is a class of problems containing two bordering theories[1], one of which is decidable, while the other is not.

4.3.1 Polynomial feasibility in continuous variables

MINLP contains (as a strict subset) all Polynomial Feasibility Problems (PFP) with variables ranging over the reals:

$$\forall i \leq m \qquad p_i(x) \quad \mathscr{R} \quad 0, \tag{4.5}$$

[1] By *theory* we mean a class of formal sentences provable from a given set of axioms; also see Footnote 3 on p. 70.

where the p_i's are rational polynomials and the relation symbol $\mathscr{R}$ is one of the relations in the set $\{\geq, =\}$. We question the existence of a general method for deciding whether there exists a solution to Eq. (4.5).

4.3.1 Remark (Polynomial equations and inequalities)
Note that Eq. (4.5) can in fact be expressed with a single relation $\mathscr{R}$, either $\geq$ or $=$, since every equation can be expressed by a pair of inequalities with opposite sign, and every inequality $p_i(x) \geq 0$ can be rewritten as $p_i(x) + s_i^2 = 0$, where s_i is an additional continuous variable.

4.3.1.1 Quantifier elimination

Tarski proved in [284] that systems like Eq. (4.5) are decidable. Most algorithms for deciding Eq. (4.5) are based on a technique known as *quantifier elimination*. This is an algorithm which turns a quantified logical sentence with variable terms into one without quantifiers or free variables. Forthwith, deciding truth or falsity can be achieved by elementary manipulations leading to either a tautology $1 = 1$ or a contradiction $0 = 1$. Note that Tarski's algorithm can be applied to a larger class of polynomial systems than Eq. (4.5): the relation symbol $\mathscr{R}$ can actually range over $\{\geq, >, =, \neq\}$.

Tarski's proof extends for 57 pages of a RAND Corporation technical report [284]. To showcase an easier quantifier elimination example, Lyndon discusses dense linear orders in his book [202, p. 38]. This theory includes logical formulæ involving $\forall, \exists, \neg, \wedge, \vee$, the variable symbols $x_1, x_2, \ldots$, the terms True, False, the real numbers as constants, and the relation symbols $<, =$. Sentences of this theory can be inductively reduced to a disjunctive normal form (DNF) such that each clause is $\exists x_i$ s.t. $q_i(x)$, where q_i is an unquantified conjunction of sentences $\bigwedge q_{ij}$, where, in turn, each q_{ij} has one of the forms $s = t$ or $s < t$ with s, t are either variables or real constants. Each q_{ij} is then reduced to either True or False: if s, t are both constants, the conditions $s < t$ or $s = t$ are immediately verified and can be replaced with True or False. If both s and t are x_i, then $s = t \Rightarrow x_i = x_i$ is True, and $s < t \Rightarrow x_i < x_i$ is False: so s, t can be assumed to be different. If s is the variable x_i, a formula $x_i = t$ (independently of whether t is a constant or another variable) can be interpreted to mean "replace x_i with t", so the variable x_i can be eliminated. The remaining case is that q_i is a conjunction of formulae of the form $s < x_i$ and $x_i < t$ (with s, t either other variables or constants), which amounts to writing $s < t$ — again, this eliminates x_i. Since every occurrence of x_i can be eliminated from q_i, $\exists x_i q_i$ can be more simply re-written as q_i: hence the name *quantifier elimination.*.

4.3.1.2 Cylindrical decomposition

Well after Tarski's 1948 quantifier elimination algorithm, Collins discovered [71] that the decision procedure of systems such as Eq. (4.5) is in some sense a nontrivial extension of the dense linear order case. The solution set of Eq. (4.5) consists of finitely many connected components, which can be recursively built out of cylinders with bases shaped as points or intervals the extremities of which are either points, or $\pm\infty$, or algebraic curves depending on variables involved in previous recursion levels. Although Collins' algorithm is doubly exponential in the number of variables, a singly exponential algorithm was described in [33].

The cylindrical decomposition result of [71] consists of a topological and a geometric part. The topological part states that the feasible regions of Eq. (4.5) (where $\mathscr{R}$ ranges in $\{>, \geq, =, \neq\}$), also called *semi-algebraic sets*, consist of a finite number of connected components. The geometrical part gives the recursive description mentioned above in terms of cylinders. The topological part was known previous to Collins' contribution, see e.g. [222].

4.3.2 Polynomial feasibility in integer variables

In this section, we shall consider Eq. (4.5) subject to integrality constraints on the vector x of decision variables. In fact, it suffices to consider equations only (see Rem. 4.3.1):

$$\left.\begin{array}{rrcl} \forall i \leq m & p_i(x) & = & 0 \\ \forall j \leq n & x_j & \in & \mathbb{Z}. \end{array}\right\} \tag{4.6}$$

Given a system Eq. (4.6) of polynomial equations in integers, can we decide whether it has a solution?

4.3.2.1 Undecidability versus incompleteness

The question of deciding whether a polynomial system has integer solutions is related but different from completeness[2]. A consistent formal system[3] S is *complete* when, for any sentence p of the system, either p is provable within S, or $\neg p$ is. A system S is incomplete when there are sentences p such that neither p nor $\neg p$ are provable within S. Gödel's (first) incompleteness theorem states that any S capable of representing $\mathbb{N}$ and its arithmetic is either inconsistent or incomplete. This leaves open the question of decidability, i.e. of whether there exists an algorithm that, given any p in S, decides whether p is provable within S or not.

4.3.2 Remark (Decidability does not imply completeness)
There is a common misconception that decidability should imply completeness. The (faulty) argument goes as follows. If a formal system S is decidable, then for any given p the system S can be used to decide whether p is provable in S (e.g. by exhibiting a proof of p within S). So, in case p is provable, then $\neg p$ cannot be, and vice versa. Hence, either p or $\neg p$ must be provable within S, which implies that S is complete.

The error resides in thinking that the "vice versa" above exhausts all possibilities. For two sentences p and $\neg p$ in S, there are four assignments of "provability within S" to the two sentences: (i) both are provable, (ii) the first is provable and the second is not, (iii) the first is not and the second is, and (iv) neither is provable. The first assignment leads to the inconsistency of S: it should be discarded since we assumed that S is consistent. The second and third assignments are part of the above (faulty) argument. The error stems from forgetting the fourth assignment: i.e., that both p and $\neg p$ may fail to be provable within S. This condition is described by saying that p is independent *of S. A decision algorithm for provability in S might answer NO when given either p or $\neg p$ as input. So a formal system can be decidable but incomplete (e.g., the Wikipedia page https://en.wikipedia.org/wiki/Decidability_(logic)#Relationship_with_completeness states that the theory of algebraically closed fields bears this property).*

As it turns out, if S encodes arithmetic in $\mathbb{N}$ and is consistent, it is not only incomplete (by Gödel's first incompleteness theorem) but also undecidable. This was settled by Turing [288] using a diagonalization argument involving another undecidable decision problem called the *halting problem* — is there an algorithm for deciding whether a given TM terminates or not?

4.3.2.2 Hilbert's 10^{th} problem

We now come back to our MINLP feasibility problem in Eq. (4.6). Each equation in the system is known as a *Diophantine equation* (DE). Let S be a formal system encoding arithmetic in $\mathbb{N}$. Systems of DEs are

[2] We mean "completeness" as in Gödel's *incompleteness* theorem, rather than Gödel's *completeness* theorem — these two notions, though bearing related names, are distinct.

[3] A formal system S is a finite language L, together with a grammar that defines a set of well-formed formulæ (called *sentences*) over L, a chosen set of sentences called *axioms*, and a set of *inference rules* (e.g. *modus ponens*) which define relations over sentences. Given a sentence p, a *proof* for p in S is a sequence of iterative applications of inference rules to sentences that are either axioms or for which a proof was already provided.

certainly well-formed formulæ within S. Although the set of *all* well-formed formulæ in S is undecidable, we still do not know whether the limited subset described by the form of Eq. (4.6) is "general enough" to be undecidable. The common property of all feasibility systems is that the integer variables $x_1, \dots, x_n$ are implicitly quantified by *existential* quantifiers "$\exists$", but no *universal* quantifier "$\forall$". Moreover, Eq. (4.6) only consists of sentences involving polynomials, whereas exponentiation is also part of arithmetic in integers. We can now frame the question as follows: is the set of all existentially quantified polynomial sentences of S decidable or not? This can be seen as a modern restatement of Hilbert's 10^{th} problem.

From the proof of Gödel's first incompleteness theorem [122], it is clear that Gödel only used a finite number of *bounded* universal quantifiers. Davis, in [94], shows that a single bounded universal quantifier is sufficient to encode undecidable problems:

$$\exists y \forall z \leq y \exists x_1, \dots, x_n \quad p(y, z, x) = 0, \tag{4.7}$$

where p is an integral polynomial, and $x_1, \dots, x_n, y, z$ are all in $\mathbb{N}$. In [95], it is shown that there exist undecidable problems that are almost in the desired form:

$$\exists x_1, \dots, x_n \in \mathbb{N} \quad \eta(x) = 0, \tag{4.8}$$

but where η is an *exponential* DE (EDE), i.e. a function that can be written using arithmetical operations and exponentiation. In a landmark result, Matiyasevich proved in 1970 [210] that the exponential relationship $a = b^c$ can be expressed by means of a DE (i.e. without exponentiation), thereby settling Hilbert's 10^{th} problem in the negative. This result is now known as the Matiyasevich-Davis-Putnam-Robinson (MDPR) theorem. In the setting of this paper, the MDPR theorem means that

$$\exists x_1, \dots, x_n \in \mathbb{N} \quad p(x) = 0, \tag{4.9}$$

where p is a polynomial, is generally undecidable. This makes Eq. (4.6) undecidable, which, by inclusion, makes MINLP undecidable, too.

4.3.3 Remark (DE systems)
A system of DEs such as Eq. (4.6) is equivalent to a single DE such as Eq. (4.9): it suffices to write

$$p(x) = \sum_{i \leq m} (p_i(x))^2.$$

4.3.4 Remark (Undecidability in $\mathbb{Z}$)
The undecidability results relate to $\mathbb{N}$ rather than $\mathbb{Z}$, but this is wlog: we can transform any DE with solutions in $\mathbb{Z}$ to one with solutions in $\mathbb{N}$ by writing $x_i = y_i^+ - y_i^-$ (where $y_i^+, y_i^- \in \mathbb{N}$), and, conversely, since every non-negative integer is the sum of four squares, the opposite transformation as $x_i = a_i^2 + b_i^2 + c_i^2 + d_i^2$ for the i-th component x_i occurring in the arguments of $p(x)$.

4.3.5 Remark (Polynomials over $\mathbb{Q}$)
The result also holds for polynomials over $\mathbb{Q}$: it suffices to write rational coefficients as fractions, find the lcm L of the denominators, and consider $Lp(x)$ instead of $p(x)$.

It might seem strange to stare at a single polynomial equation such as Eq. (4.9), and claim that it is undecidable: it sounds like saying that a single instance of a problem is undecidable. In fact, the variable vector of $p(x)$ is partitioned in two subsequences $\alpha = (\alpha_1, \dots, \alpha_\ell)$ and $y = (y_1, \dots, y_N)$, so that α encodes the parameter of an "instance", and y the solution. Since:

(a) every recursively enumerable set W can be encoded by a single polynomial equation $p^W(\alpha, y) = 0$ such that any string w is encoded in α, and $w \in W$ iff $p(\alpha, y) = 0$ has a solution in y;

(b) there exist recursively enumerable sets that are undecidable,

the MDRP result follows.

4.3.3 Universality

As remarked in [154], recursively enumerable sets can be enumerated in some order $W_1, W_2, \ldots$ such that the relation $w \in W_v$ for some v is also recursively enumerable. This implies the existence of a DE

$$U(w, v, y) = 0 \tag{4.10}$$

for some parameters w and v such that $U(\alpha, v, y) = 0$ has an integer solution y iff $w \in W_v$. Such polynomials U are known as *universal Diophantine equations* (UDE). This is equivalent to a UTM, which takes as input an encoding of any TM T_v and any instance w, and simulates $T_v(w)$. In this setting, T_v is a TM that is supposed to determine whether $w \in W_v$.

Finding a UDE presents potential applications, as it would allow the encoding of any computer program into a single polynomial. Studying its properties might help shed light on the program itself. Jones [154] conducted a thorough study of two complexity measures for UDEs: their degrees δ and the number ν of variables occuring in them.

The minimum known value for δ is 4. It suffices to take *any* UDE, and repeatedly replace any degree 2 term with a new variable until we obtain a system of DEs $\forall i \leq m\ (t_i(x) = 0)$ where each $t_i(x)$ consists of monomials of degrees 1 and 2. The equation

$$\sum_{i \leq m} (t_i(x))^2 = 0 \tag{4.11}$$

(see Rem. (4.3.3)) is therefore a UDE with degree $\delta = 4$. In passing, we also conclude that, in general, systems of (many) quadratic DEs (QDE) are also undecidable. Jones reports the existence of a UDE with $(\delta = 4, \nu = 58)$. Decreasing ν unfortunately yields large increases on δ: Jones [154] reports a new (δ, ν) pair due to Matiyasevich valued at $(1.638 \times 10^{45}, 9)$.

4.3.6 Remark (Proofs in $\mathbb{N}$ involve at most 100 operations)
By way of an application, Jones exploits the $(4, 58)$ *UDE, suitably modified to minimize the number* B *basic operations (additions and multiplications) required to evaluate the polynomial, and obtains* $B = 100$. *This implies that for any statement* p *that can be proved within the formal system* S *of arithmetic in* $\mathbb{N}$, p *has at least a proof that only involves 100 integer additions and multiplications.*

Coming back to MINLP, since UDEs are subsets of MINLP, MINLP inherits their properties, which means that there exist universal MINLP formulations, though they may be hard to exploit in practice. The dynamics of a universal register machine (another computation model which is more similar to modern computers than TMs are) have been modelled in [188] using an infinite MINLP. This MINLP becomes finite, and hence practically solvable, on *bounded* executions, as boundedness ensures termination of the underlying TM.

4.3.4 What is the cause of MINLP undecidability?

Considering Sect. 4.3.1 and Sect. 4.3.2, we can ascribe the undecidability of MINLP to the presence of integer variables. This is apparently in contrast with the fact that any integer can be expressed within a theory encoding polynomials and continuous variables: given any $a \in \mathbb{Z}$, the equation

$$x - a = 0$$

is polynomial and encodes the fact that the continuous variable x should take the integer value a. Taking this further, we can write $x \in \{a_1, \ldots, a_k\}$ for any integer values $a_1, \ldots, a_k$ using the polynomial equation

$$(x - a_1) \cdots (x - a_k) = 0.$$

A different approach uses the length ℓ of the finite sum $y + y + \cdots + y$ (say it has ℓ occurrences of y) to express the integer ℓ using the polyomial equation $xy = \sum_\ell y$. Aside from the zero solution, any other solution with $y > 0$ yields $x = \ell$.

However, every finite set F is decidable, at least if its elements are all given explicitly: any total order on F provides an enumeration algorithm. Obviously, the complement $\bar{F}$ of F w.r.t. $\mathbb{N}$ is recursively enumerable: just list $\mathbb{N}$ and verify if the current element is in F or not.

To achieve undecidability, we have to look at infinite subsets of $\mathbb{N}$. Is it possible to encode any such set as solutions over $\mathbb{R}$ of some (multivariate) polynomial equation? The answer is no: supposing there exists a real polynomial system with a countably infinite set of (integer) solutions would yield an infinite number of connected components in the corresponding variety, contrary to Milnor's topological result [222]. Another way of proving this is that, if it were possible to encode $\mathbb{N}$ as solutions of a real polynomial system in any (even nonstandard) way, then the theory of polynomials over $\mathbb{R}$ would be undecidable, contrary to Tarski's decision algorithm. It is common knowledge that, in order to encode all integers in a system of nonlinear equations with variables ranging over $\mathbb{R}^n$, one needs at least one periodic function, e.g. the set of solutions of $\sin(\pi x) = 0$ is $\mathbb{Z}$. If the polynomials range over $\mathbb{C}^n$, the exponential function (which is periodic over the complex numbers) is also a candidate.

These arguments suggest that a cause of undecidability is the issue of boundedness versus unboundedness of the decision variables. This is well known in polynomial optimization theory. With unbounded variables, even the continuous relaxation of a MILP may need to be reformulated using nonlinear functions for practical usefulness [138].

4.3.5 Undecidability in MP

Although the theories of polynomial equations over the reals and the natural numbers belongs *de re* to MINLP, since PFP is clearly a subclass of MINLP, the results above are traditionally a part of logic and axiomatic set theory. Here follow two results from the MP community.

The earliest paper investigating MP and (un)decidability is possibly Jeroslow's 1971 paper [151] about the undecidability of Integer Quadratic Programming. Consider the formulation

$$\left.\begin{array}{rrcl} \min & c^\top x & & \\ \forall i \leq m & x^\top Q^i x + a_i^\top x & \leq & b_i \\ \forall j \leq n & x_j & \in & \mathbb{Z}, \end{array}\right\} \tag{4.12}$$

where $c \in \mathbb{Q}^n$, Q^i are rational $n \times n$ matrices for each $i \leq m$, $A = (a_i \mid i \leq m)$ is a rational $m \times n$ matrix, and $b \in \mathbb{Q}^m$. Witzgall's algorithm [307] solves Eq. (4.12) when each quadratic form $x^\top Q^i x + a_i^\top x$ is *parabolic*, i.e. each Q^i is diagonal with non-negative diagonal entries (for $i \leq m$).

In contrast, Jeroslow observed that this is a limiting case, since if the quadratic forms are diagonal but are allowed to have some negative entries, then they can encode an undecidable set, via the undecidability theory of DEs. The proof is as follows: given an undecidable DE $p(x) = 0$, we consider the following subclass of Eq. (4.12):

$$\left.\begin{array}{rrcl} \min & x_{n+1} & & \\ & (1 - x_{n+1})p(x) & = & 0 \\ & x_{n+1} & \geq & 0 \\ \forall j \leq n & x_j & \in & \mathbb{Z}. \end{array}\right\} \tag{4.13}$$

Obviously, the minimum of Eq. (4.13) is 0 iff $p(x) = 0$ has a solution, and 1 otherwise. Now we follow the argument given above Eq. (4.11), and iteratively linearize products occurring in the k monomials of $p(x)$ until p can be reformulated to a linear form $y_1 + \cdots + y_k$, subject to a set of *defining constraints* in the form $y_h = \tau_h(x, y)$, where each τ_h only involves linear occurrences of variables, or their squares, or bilinear products of two variables. This is already undecidable, since finding the optimum of Eq. (4.13) would solve the undecidable DE $p(x) = 0$, but Jeroslow notes that one can write each bilinear product

of decision variables uv (where u, v are decision variable symbols occurring in either x or y), by means of an additional variable w, as $w = u + v$ while replacing uv by $\frac{1}{2}(w^2 - u^2 - v^2)$. This yields the required form.

Jeroslow also notes that whenever the integer variables are bounded, the problem becomes decidable (since there is only a finite number of possible solutions of $p(x) = 0$).

Another paper about optimization and undecidability is [315], which focuses on undecidable problems in MINLP and GO as detailed here below.

1. **PP in integers is undecidable**. Consider:

$$\left.\begin{array}{lll} \min & q(x) & \\ \forall j \leq n & x_j & \in \mathbb{Z}, \end{array}\right\} \tag{4.14}$$

where q is a polynomial. We define $q(x) = (p(x))^2$ so that $\min_x q(x) = 0$ iff $p(x) = 0$, since q is a square and its minimum value must be ≥ 0. If we could compute the minimum of $q(x)$, we could decide on the solvability of $p(x) = 0$ according to whether $\min_x q(x) = 0$ or > 0. But this is impossible since $p(x) = 0$ is undecidable.

2. **Solving system of nonlinear equations is undecidable**. Consider:

$$\forall i \leq m \quad g_i(x) = 0, \tag{4.15}$$

where $g_i : \mathbb{R}^n \to \mathbb{R}$ are nonlinear functions: is this problem decidable? The paper [315] remarks that if $p(x) = 0$ is an undecidable DE, then

$$(p(x))^2 + \sum_{j \leq n} (\sin(\pi x_j))^2 = 0, \tag{4.16}$$

where x ranges in $\mathbb{R}^n$, is precisely a restatement of the DE, and so it is undecidable. Note that Eq. (4.16) is obviously a subclass of Eq. (4.15).

3. **Unconstrained NLP is undecidable**. Consider:

$$\min_x q(x), \tag{4.17}$$

for some nonlinear function $q(x)$. Let $p(x) = 0$ be an undecidable DE. Let $q(x) = (p(x))^2 + \sum_{j \leq n} (\sin(\pi x_j))^2$, and suppose $q^* = \min_x q(x)$ is computable. Then if $q^* = 0$ we have $p(x) = 0$ and $x_j \in \mathbb{Z}$ for all $j \leq n$, otherwise we do not, which is equivalent to the decidability of $p(x) = 0$, against assumption.

4. **Box-constrained NLP is undecidable**. Consider

$$\min_{x^L \leq x \leq x^U} q(x), \tag{4.18}$$

for some nonlinear function $q(x)$ and some variable bound vectors x^L, x^U. Let $p(y) = 0$ be an undecidable DE. Let

$$q(x) = (p(\tan x_1, \ldots, \tan x_n))^2 + \sum_{j \leq n} (\sin(\pi \tan x_j))^2,$$

then we can enforce $-\frac{\pi}{2} \leq x_j \leq \frac{\pi}{2}$ for all $j \leq n$, i.e. $x^L = -\frac{\pi}{2}$ and $x^U = \frac{\pi}{2}$ without changing the values of $q(x)$. The argument runs as above: if we could compute the minimum q^* of $q(x)$ over $x^L \leq x \leq x^U$, we could establish whether $p(y) = 0$ has a solution or not, which is impossible.

Chapter 5

Complexity

In this chapter, we look at the computational complexity of some MINLP problems. We shall mostly focus on nonconvex NLPs, since the presence of integer variables, even bounded to $\{0, 1\}$, makes it very easy to prove MINLP hardness, as shown in Sect. 5.2.

5.1 Some introductory remarks

Many variants of TMs have been proposed in the first years of research on theoretical computer science: larger alphabets, half-infinite or multiple tapes to name a few. For most of these variants, simulation proofs were quickly devised to show that none of them was *essentially* more powerful than the "basic" TM. Driven by the need for faster computation, the issue of computational complexity became more pressing: given an algorithm and some input data, how many steps[1] would a TM need to perform before termination? Given that the same algorithm is routinely run on different inputs, the paradigm of *worst case complexity* was brought to attention. Infinite sets of input data of the same type and increasing size, together with a specification of the goal the algorithm is supposed to achieve, were grouped in classes named *problems*; conversely, each member of a given problem is an *instance* (also see Sect. 2.1.3). We recall that a decision problem consists in answering a YES/NO question.

Given any problem P, if there exists an algorithm taking a number of steps bounded by a polynomial function of the instance size n in the worst case, we denote its complexity by $O(n^d)$, where d is the degree of the polynomial, and call the algorithm *polynomial-time* (or simply *polytime*). If we are only able to establish an exponential bound, we denote it by $O(2^n)$, and call the algorithm *exponential-time*.

5.1.1 Problem classes

5.1.1.1 The class P

The focus of computational complexity rapidly shifted from algorithms to problems. Given a decision problem P, the main issue of computational complexity is: what is the *worst-case* complexity of the *best* algorithm for solving P? Edmonds [104] and Cobham [69] remarked around 1965 that problems having a polytime complexity are invariant w.r.t. changes in the TM definitions concerning alphabet size (as long as it contains at least 3 characters), number of tapes, and more. This is the reason why today we partition decision problems into two broad categories called "tractable" and "intractable". The tractable problems are those having polytime complexity. The class of such problems is called **P**.

[1] The issue of whether the TM would actually stop or not was discussed in Ch. 4.

5.1.1.2 The class NP

For those decision problems for which no-one was able to find a polytime algorithm, it was remarked that a somewhat "magic" TM, one that is able to pursue every test branch concurrently (this is called *nondeterministic* TM), gives another useful partition on the class of all decision problems. Those for which there exists a nondeterministic TM that runs in polytime are all grouped in a class called **NP**.

5.1.1 Remark (NP and certificates)
An equivalent definition of **NP**, *which is often used, is the following:* **NP** *is the class of all problems for which YES instances come with a polytime verifiable* certificate, *i.e. a proof that the instance is indeed YES, which can be checked in polytime.*

To see this, we first define **NP** *as the class of all problems which can be solved by a nondeterministic TM in polytime. We run the solution algorithm nondeterministically (i.e. by following all test branches concurrently) on a given instance. At termination we look at the* trace *of the algorithm, i.e. the sequence of instructions followed by the algorithm: we "unfold" loops, and list every occurrence of repeated steps. This will yield a set of step sequences each pair of which shares an initial subsequence, i.e. a tree where branches correspond to tests. Each tree leaf identifies the end of the sequence it corresponds to. If the algorithm terminates with a YES, it is because one of the sequences ends with a solution of the instance. We identify this sequence with the certificate: it has polynomial length since the nondeterministic algorithm runs in polytime; and by following its steps we can convince ourselves that the answer is indeed YES, as we verify each computation along the terminating YES sequence. We note that NO answers generally occur because every branch was explored, and not necessarily because a particular sequence has some specific property (but see Ex. 5.1.2), which makes this argument non-symmetric w.r.t. YES and NO.*

Conversely, we now define **NP** *as the class of problems for which YES instances have a polytime verifiable certificate. We assume that the given certificate is a binary string with length bounded by a (given) polynomial in function of the instance size. We devise a brute-force algorithm which explores the set of all such strings by branching (on zeros and ones) at each component. Since the TM is nondeterministic, all branches will be followed concurrently, and since we only need to explore polynomially bounded strings, it will terminate in polytime. The certificate will be found if it exists (i.e. if the instance is YES).*

Consider the CLIQUE problem: given a graph G and an integer k, does G possess a clique of size at least k? If the instance is YES, then it has a clique of size at least k, and this clique can be supplied as certificate to a verification algorithm which is able to establish in polytime whether the certificate is valid. Therefore CLIQUE is in **NP**.

While it is easy to show that $\mathbf{P} \subseteq \mathbf{NP}$, no-one was ever able to establish whether **P** is different or equal to **NP**. This is the famous **P** vs. **NP** question.

5.1.2 Exercise
Prove that $\mathbf{P} \subseteq \mathbf{NP}$.

> **Answer**. We consider a problem P in **P**. By definition there is an algorithm that decides whether an instance of **P** is YES or NO in polytime. Its trace, in particular, provides a polytime verifiable certificate for YES instances. By Rem. 5.1.1, P is in **NP**.

5.1.2 Reductions

The class **NP** is very useful because, in absence of a complement of **P** allowing us to tell problems apart into "easy" and "hard", it provides a good proxy.

Given two problems P and Q in **NP**, suppose we know a polytime algorithm A to solve P but we know of no such algorithm for Q. If we can identify a polytime algorithm R that transforms a YES instance of Q into a YES instance of P and, conversely, a NO instance of Q into a NO instance of P, then we

can compose R and A to obtain a polytime algorithm for Q. An algorithm R mapping instances $Q \to P$ while keeping their membership in YES and NO classes invariant is called a *polynomial reduction*[2] from Q to P.

5.1.2.1 The hardest problem in the class

This gives an interesting way to define the "hardest problem in **NP**". Let P be a problem in **NP** such that any problem in **NP** polynomially reduces to P. Pick any $Q \in$ **NP**: can Q be harder to solve than P? Since there is a polynomial reduction $R : Q \to P$, if we know how to solve P then we know how to solve Q, so there is no real additional difficulty in solving Q other than knowing how to solve P. In this sense, P is hardest in the class **NP**. The set of such problems are said to be **NP**-*complete*. Since this notion of hardness only depends on polynomial reductions rather than membership in **NP**, we also define as **NP**-*hard* those problems which are as hard as any problem in **NP** without necessarily being members of **NP**. Thus, **NP**-complete problems are the subset of **NP**-hard problems which also belong to **NP**.

In a landmark result, S. Cook proved in [74] that the class of **NP**-complete problems is non-empty, since it contains the satisfiability problem (SAT). The SAT problem involves a decision about the truth or falsity of the sentences defined over variable symbols, the $\neg$ unary operator, the $\wedge$ and $\vee$ binary operators, and the constants True and False (represented by 1 and 0). Traditionally, in SAT notation, one writes $\bar{x}$ instead of $\neg x$. An instance (i.e. a sentence) is YES if there is an assignment of 1 and 0 to the variables for which the sentence is True, and NO otherwise. The evaluation uses the common meaning of the boolean operators $\neg, \wedge, \vee$. Cook's result is based on the idea that SAT can be used as a declarative language to model the dynamics of *any* polytime TM. The modelling exploits the fact that TMs (tapes, instructions, states) can be described by 0-1 vectors each component of which is modelled as a SAT variable. Since the definition of **NP** involves a polytime-checkable certificate, SAT can be used to model the polytime TM used to verify the certificate. The SAT instance has finite length because we know that the number of steps of the TM is bounded by a polynomial in the input size.

5.1.3 Remark (Cook's Theorem)
In order to prove that SAT *is* **NP**-*complete, we must prove that it is in* ***NP*** *and that is* **NP**-*hard.*

The hard part is the reduction of any problem in ***NP*** *to* SAT. *We use the definition of* ***NP****: given a problem in this class, there exists a polytime nondeterministic TM which decides each instance of the problem. Cook showed that the dynamics of a nondeterministic polytime TM can be modelled by a polynomially sized* SAT *instance. Since this is a book about MP, we reduce to a MILP instead: so, instead of proving that* SAT *is* ***NP****-complete, we shall prove that MILP is* ***NP****-hard.*

A nondeterministic TM M is described by a tuple $(Q, \Sigma, s, F, \delta)$ where Q is the set of states of the machine, Σ is the alphabet (characters to be written on the tape), $s \in Q$ is the initial state, $F \subseteq Q$ is the set of termination states, and δ is the transition relation, a subset of $(Q \setminus F \times \Sigma) \times (Q \times \Sigma \times \{-1, 1\})$ (the component $\{-1, 1\}$ signals the left or right movement of the head on the tape). Let n be the size of the input. We know that M terminates in polytime $O(p(n))$, where p is a polynomial. So we only need tape cells indexed by i where $-p(n) \leq i \leq p(n)$, and steps indexed by k where $0 \leq k \leq p(n)$. We index alphabet characters by $j \in \Sigma$. The dynamics of the TM can be described by the following statements [116].

1. *Initialization:*
   (a) *the tape has an initial string at step $k = 0$;*
   (b) *M has an initial state s at step $k = 0$;*

[2] Technically, this is known as a *Karp reduction*. A *Cook reduction* occurs from Q to P if Q can be solved by a polynomial number of standard operations and calls to an oracle that solves P. A Karp reduction is like a Cook reduction allowing for a single oracle call. Karp reductions allow for a distinction between hard problems where polynomial certificates are available for YES instances and for NO ones — in other words, it allows for the definition of the class co-**NP**.

(c) *the initial position of the read/write head at $k = 0$ is on cell $i = 0$;*

2. *Execution:*

   (a) *at each step k each cell i contains exactly one symbol j;*

   (b) *if cell i changes symbol between step k and $k+1$, the head must have been on cell i at step k;*

   (c) *at each step k, M is in exactly one state;*

   (d) *each step k, cell i and symbol σ lead to possible head positions, machine states and symbols as prescribed by the transition relation δ;*

3. *Termination:*

   (a) *M must reach a termination state in F at some step $k \leq p(n)$.*

Next, we translate the above description into a set of MILP constraints. As parameters we have the initial tape string τ (a vector indexed by $-p(n) \leq i \leq p(n)$). We introduce the following binary (polynomially many) decision variables:

- *$\forall$ cell i, symbol j, step k $t_{ijk} = 1$ iff tape cell i contains symbol j at step k;*
- *$\forall$ cell i, step k $h_{ik} = 1$ iff head is at tape cell i at step k;*
- *$\forall$ state ℓ, step k $q_{\ell k} = 1$ iff M is in state ℓ at step k.*

Finally, we express the above statements by means of constraints in function of the variables t, h, q:

1. *Initialization:*

   (a) $\forall i \ (t_{i,\tau_i,0} = 1)$;

   (b) $q_{s,0} = 1$;

   (c) $h_{0,0} = 1$;

2. *Execution:*

   (a) $\forall i, k \ (\sum_j t_{ijk} = 1)$;

   (b) $\forall i, j \neq j', k < p(n) \ (t_{ijk}\, t_{i,j',k+1} = h_{ik})$;

   (c) $\forall k \ \sum_i h_{ik} = 1$;

   (d) $\forall i, \ell, j, k \ (|\delta(\ell, j)|\, h_{ik}\, q_{\ell k}\, t_{ijk} = \sum_{((\ell,j),(\ell',j',d))\in\delta} h_{i+d,k+1}\, q_{\ell',k+1}\, t_{i,j',k+1})$;

3. *Termination:*

   (a) $\sum_{k, f\in F} q_{fk} = 1$.

Again, the number and size of these constraints are bounded by polynomials in the input size. We observe that some constraints involve products of binary variables, but these can all be reformulated exactly to linear, by means of a polynomial quantity of additional variables and constraints (see Rem. 2.2.8).

5.1.4 Exercise

Prove that feasibility-only MILPs are in **NP**.

> **Answer**. First, remark that a feasibility-only MILP is a decision (rather than an optimization) problem. Next, we take a feasible solution as a YES certificate. We can verify it is a correct solution by replacing decision variables with corresponding values (polytime in the instance size since the instance size is a polynomial in the number of variables), and by evaluating whether the constraints are satisfied (polytime in the instance size since evaluating linear constraints takes time proportional to the number of the constraints multiplied by the number of variables in the worst case).

5.1.2.2 The reduction digraph

After Cook's result, SAT became known as the "primitive problem" in the theory of **NP**-hardness. While the *first* proof of **NP**-hardness was difficult to devise (since it necessarily has to encode *every* TM, following the definition), subsequent proofs are of an altogether different nature, as they can rely on the mechanism of reduction.

Let P be an **NP**-hard problem. We want to establish the **NP**-hardness of Q. Can Q be any easier than P? Suppose we find a polynomial reduction $R : P \to Q$: if Q could be solved by an algorithm A that is asymptotically faster than that of P, then we could compose R with A to derive a better algorithm for P, which is impossible since P was defined to be hardest for **NP**. In [157], R. Karp used this idea to prove that many common problems are **NP**-complete. Since then, when discussing a new problem, an effort is made to establish either membership in **P** or **NP**-hardness.

Note that polytime reductions define an asymmetric relation on the class **NP**, i.e. two problems P, Q in **NP** are related if there is a reduction $R : P \to Q$. This turns **NP** in an (infinite) digraph, called the *reduction digraph*, where the problems are represented by nodes, and reductions by arcs. This digraph is strongly connected by Cook's result, which proves exactly that there is a reduction from any problem in **NP** to SAT. This makes the reduction digraph strongly connected.

5.1.2.3 Decision vs. optimization

As mentioned above, a problem P is **NP**-complete if it is **NP**-hard and belongs to **NP**. Many theoretical results in computational complexity concern decision problems, but in practice we also need to solve function evaluation and optimization problems. In the Wikipedia page about "Decision problem" (en.wikipedia.org/wiki/Decision_problem), we read:

> There are standard techniques for transforming function and optimization problems into decision problems, and vice versa, that do not significantly change the computational difficulty of these problems.

The "decision version" of the MINLP in Eq. (2.2) adds a *threshold value* ϕ to the input and asks whether the system

$$\left.\begin{array}{rcl} f(x) & \leq & \phi \\ g(x) & \in & [g^L, g^U] \\ x & \in & [x^L, x^U] \\ \forall j \in Z \quad x_j & \in & \mathbb{Z} \end{array}\right\} \tag{5.1}$$

is feasible. The class of optimization problems that can be reducible to their decision version Eq. (5.1) is called **NPO** (the "O" stands for "optimization"). It has the following properties: (i) the feasibility of any solution can be established in polytime; (ii) every feasible solution has polynomially bounded size; (iii) the objective function and constraints can be evaluated in polytime at any fasible solution.

5.1.2.4 When the input is numeric

For problems involving arrays of rational numbers in the input data, such as vectors or matrices, more notions are used to distinguish complexity in function of the numerical value of the input versus the number of bits of memory required to store it.

An algorithm is *pseudopolynomial* if it is polynomial in the value of the numbers in the input, but not in their storage size. E.g., testing whether a number n is prime by trying to divide it by $2, \ldots, \lfloor\sqrt{n}\rfloor$ takes $\sqrt{n}$ divisions. In the classic TM computational model, the number of divisions is polynomial in the value of the input n, but exponential in the size of the input $\lceil\log_2(n)\rceil$, since $\sqrt{n} = 2^{\frac{1}{2}\log_2(n)}$.

5.1.5 Remark (Pseudopolynomial and unary encoding)
An equivalent definition of a pseudopolynomial algorithm is that it is polytime in the unary encoding of the data (but not in the binary encoding). The equivalence of these two definitions is readily seen because an integer in encoded in base 1 needs as many bits of storage as its value, whereas in base 2 it can be encoded in $\lceil \log_2(n) \rceil$ *bits.*

In the arithmetic computational model, where elementary operations on numbers take unit time, an algorithm is *strongly polytime* if its worst case running time is bounded above by a polynomial in function of the *length* of the input arrays, and the worst-case amount of memory required to run it (a.k.a. the *worst-case space*) is bounded by a polynomial in the number of bits required to store their values, a.k.a. the size of the input (e.g. Dijkstra's shortest path algorithm on weighted graphs is strongly polytme). By contrast, if the running time also depends nontrivially (i.e. aside from read/write and elementary arithmetic operations) on the actual input storage size, including the bits required to represent the numbers in the arrays, the algorithm is *weakly polytime* (e.g. the ellipsoid method for LP is weakly polytime: whether there exists a strongly polytime algorithm for LP is a major open question).

A problem P is *strongly* **NP**-hard if there is a strongly polytime reduction algorithm R from every problem in **NP** to P, or, equivalently, from a single **NP**-hard problem to P. P is *strongly* **NP**-complete if it is strongly **NP**-hard and also belongs to the class P. An equivalent definition of strong **NP**-hardness is the class of those **NP**-hard problems that remain **NP**-hard even if the input is encoded in unary (see Rem. 5.1.5), i.e. when the value of all of their numerical data is bounded by a polynomial in the input size. A problem P is *weakly* **NP**-hard when it is **NP**-hard but there exists a pseudopolynomial algorithm that solves it. E.g. often dynamic programming based solutions search the set of values that a given variable can take, so that the number of iterations might remain polynomial in the value rather than the number of bits required to store it. When proving **NP**-hardness for a new problem P, using reductions from a strongly **NP**-hard problem gives more flexibility, as one can use a pseudopolynomial reduction and still claim **NP**-hardness of P. This technique was used in [56].

5.1.6 Remark (Strong and weak)
Algorithms may be strongly or weakly polytime, and problems may be strongly or weakly **NP***-hard. But the opposition of the terms "strong" and "weak" are expressed in different terms for algorithms and problems. I.e. the weak notion of problem* **NP***-hardness rests on pseudopolynomial (rather than "weak") reductions.*

5.2 Complexity of solving general MINLP

MINLP is not in **NP** because it is not a decision problem. It is **NP**-hard because it includes MILP as a subclass. MILP, in turn, is **NP**-hard by means of a trivial reduction from SAT. Transform the SAT instance to conjunctive normal form[3] (CNF), write $\bar{x}$ as $1 - x$, $\vee$ as $+$ and let $\wedge$ mark the separation between constraints. This yields a set of MILP constraints that are feasible if and only if the SAT instance is YES.

This argument, however, only brushes the surface of the issue, as it is essentially an argument about MILP. We know by Sect. 4 that nonlinearity makes unbounded MINLPs undecidable, whereas finite bounds make it decidable. But what about the continuous variables? We know that PP without integer variables is decidable, but is it **NP**-hard? Again, the answer is yes, and it follows from the fact that polynomials can express a bounded number of integer variables (see the beginning of Sect. 4.3.4). All that the SAT $\rightarrow$ MILP reduction above needs is a sufficient number of binary (boolean) variables, which can be defined by requiring a continuous variable x to satisfy the polynomial $x(1 - x) = 0$.

In the rest of this section, we shall survey the field of computational complexity of several different MINLP problems, with a particular attention to NLPs (which only involve continuous variables).

[3] I.e. a sequence of clauses involving exclusively $\vee$, each separated from the next by $\wedge$.

5.3 Quadratic programming

The general Quadratic Programming (QP) problem is as follows:

$$\left.\begin{array}{lrcl} \min & \frac{1}{2}x^\top Qx & + & c^\top x \\ & Ax & \geq & b. \end{array}\right\} \tag{5.2}$$

5.3.1 NP-hardness

QP was shown in [259] to contain an **NP**-hard subclass of instances (and hence QP itself is **NP**-hard by inclusion). The reduction is from an **NP**-complete problem called SUBSET-SUM: given a list s of non-negative integers $s_1, \ldots, s_n$ and an integer σ, is there an index set $J \subseteq \{1, \ldots, n\}$ such that $\sum_{j \in J} s_j = \sigma$? Consider the QP formulation:

$$\left.\begin{array}{lrcl} \max & f(x) = \sum\limits_{j \leq n} x_j(x_j - 1) & + & \sum\limits_{j \leq n} s_j x_j \\ & \sum\limits_{j \leq n} s_j x_j & \leq & \sigma \\ \forall j \leq n & 0 \leq x_j & \leq & 1. \end{array}\right\} \tag{5.3}$$

This defines a transformation from an instance of SUBSET-SUM to one of QP. Obviously, it can be carried out in polytime. We now prove that it maps YES instances of SUBSET-SUM to QP instances where $f(x) < \sigma$ and NO instances to instances of QP where $f(x) = \sigma$.

Assume (s, σ) is a YES instance of SUBSET-SUM with solution J. Then setting $x_j = 1$ for all $j \in J$ satisfies all constraints of Eq. (5.3): in particular, the constraint is satisfied at equality. Solution integrality mkes the first sum in the objective function yield value zero, which yields $f(x) = \sigma$. Conversely, assume (s, σ) is a NO instance, and suppose first that Eq. (5.3) has an integer solution: then the constraint must yield $\sum_j s_j x_j < \sigma$; solution integrality again zeroes the first sum in the objective function, so $f(x) < \sigma$. Next, if Eq. (5.3) has no integer solution, there is at least one $j \leq n$ such that $0 < x_j < 1$, which makes the objective function term $x_j(x_j - 1)$ negative, yielding again $f(x) < \sigma$, as claimed.

The paper [259] also contains another proof that shows that a Nonlinear Programming (NLP) problem with just one nonlinear constraint of the form $\sum_j x_j(x_j - 1) \geq 0$ is also **NP**-hard.

5.3.1.1 Strong NP-hardness

A different proof, reducing from SAT, was given in [295, Thm. 2.5]. It shows that the following QP subclass is **NP**-hard:

$$\left.\begin{array}{lrl} \min & f(x) = \sum\limits_{j \leq n} x_j(1 - x_j) \\ & \text{SAT} \rightarrow \text{MILP} \\ \forall j \leq n & 0 \leq x_j \leq 1, \end{array}\right\} \tag{5.4}$$

where SAT $\rightarrow$ MILP indicates the MILP constraints encoding a SAT instances which were discussed in the first paragraph of this section. The proof shows that the given SAT instance is YES iff $f(x) = 0$. Assume first that the SAT instance is YES: then there is an assignment of boolean values to the SAT variables that satisfies the SAT $\rightarrow$ MILP constraints. Moreover, since the solution is integral, we get $f(x) = 0$. Conversely, assume the SAT instance is NO, and suppose, to aim at a contradiction, that a solution x^* to Eq. (5.4) exists, and is such that $f(x^*) = 0$. Since the quadratic form $\sum_j x_j(1 - x_j)$ is zero iff each $x_j \in \{0, 1\}$, we must conclude that x^* is integral. Since the SAT $\rightarrow$ MILP constraints are feasible iff the SAT instance is YES, and x^* satisfies those constraints, it follows that the SAT instance is YES, against assumption. Hence, if the SAT instance is NO, either Eq. (5.4) is infeasible or $f(x^*) > 0$.

These two arguments prove the same fact, i.e. that QP is **NP**-hard. However, the first reduction is from SUBSET-SUM, while the second is from SAT. This has some consequences, since while SAT is

strongly **NP**-hard, SUBSET-SUM is not: namely, the first reduction only proves the weak **NP**-hardness of QP, leaving the possibility of a pseudopolynomial algorithm open. The second reduction rules out this possibility.

5.3.2 NP-completeness

While QP cannot be in **NP** since it is not a decision problem, the decision problem corresponding to QP is a candidate. The question is whether the following decision problem is in **NP**:

$$\left.\begin{array}{rcl} \frac{1}{2}x^\top Qx + c^\top x & \leq & \phi \\ Ax & \geq & b, \end{array}\right\} \tag{5.5}$$

where ϕ is a threshold value that is part of the input.

As shown in [295], the proof is long and technical, and consists of many intermediate results that are very important by themselves.

1. If the QP of Eq. (5.2) is convex, i.e. Q is positive semidefinite (PSD), and has a global optimum, then it has a globally optimal solution vector where all the components are rational numbers — this was shown in [253]. The main idea of the proof is as follows: only consider the active constraints in $Ax \geq b$, then consider the KKT conditions, which consist of linear equations in x, and derive a global optimum of the convex QP as a solution of a set of linear equations.

2. Again by [253], there is a polytime algorithm for solving convex QPs (cQP): in other words, cQP is in **P**. This is a landmark result by itself — we note that the algorithm is an interior point method (IPM) and that it is weakly polytime. Specifically, though, this result proves that the size of the rational solution is bounded by a polynomial in the input size. By [295, p. 77], this is enough to prove the existence of a polynomially sized certificate in case Eq. (5.5) is bounded.

3. The unbounded case is settled in [292].

This allows us to conclude that the decision version of QP is **NP**-complete.

5.3.3 Box constraints

A QP is *box-constrained* if the constraints $Ax \geq b$ in Eq. (5.2) consist of variable bounds $x^L \leq x \leq x^U$. The hyper-rectangle defined by $[x^L, x^U]$ is also known as a "box". As for the **NP**-hardness proof of QP, there are two reductions from **NP**-complete problems to box-constrained QP: one from SUBSET-SUM [235, Eq. (3)] and one from SAT [295, §4.2]. Since reductions from SAT imply strong **NP**-hardness, we only focus on the latter.

We actually reduce from 3SAT in CNF, i.e. instances consisting of a conjunction of m clauses each of which is a disjunction of exactly 3 literals. As before, we write the i-th literal as x_i or $1 - x_i$ if negated. A disjunction $x_i \vee \bar{x}_j \vee x_k$, for example, is written as a linear constraint $x_h + (1 - x_j) + x_k \geq 1$, yielding

$$\forall i \leq m \quad a_i^\top x \geq b_i \tag{5.6}$$

for appropriate vectors $a_i \in \mathbb{Z}^m$ and $b \in \mathbb{Z}$. By adding a slack variable $v_\ell \in [0, 2]$ to each of the m clauses, Eq. (5.6) becomes $\forall i \leq m\ (a_i^\top x = b_i + v_i)$. Next, we formulate the following box-constrained QP:

$$\left.\begin{array}{ll} \min & f(x, v) = \sum\limits_{j \leq n} x_j(1 - x_j) + \sum\limits_{i \leq m} (a_i^\top x - b_i - v_i)^2 \\ & x \in [0, 1]^n, \quad v \in [0, 2]^m. \end{array}\right\} \tag{5.7}$$

We are going to show that the 3SAT instance is YES iff the globally optimal objective function value of Eq. (5.7) is zero.

Obviously, if the 3SAT formula is satisfiable, there exists a feasible solution (x^*, v^*) where $x^* \in \{0,1\}^n$ and $v^* \in \{0,1,2\}^m$ that yields a zero objective function value. Conversely, suppose there exists (x^*, v^*) such that $f(x^*, v^*) = 0$, which yields

$$a_i^\top x^* = b_i + v_i^* \tag{5.8}$$

for all $i \leq m$. Supposing some of the x variables take a fractional value, the corresponding term $x_j^*(1-x_j^*)$ would be nonzero, against the assumption $f(x^*, v^*) = 0$. From integrality of the x variables, Eq. (5.8) ensures that v_i^* is integer, for all $i \leq m$. Since integrality and feasibility w.r.t. Eq. (5.8) are equivalent to Eq. (5.6) and therefore encode the clauses of the 3SAT instance, x^* is a YES certificate for the 3SAT instance. Thus, finding the global optimum of the box-constrained QP in Eq. (5.7) is **NP**-hard.

5.3.4 Trust region subproblems

Trust Region Subproblems (TRS) take their name from the well-known trust region method for black-box optimization. TRSs are essentially instances of QP modified by adding a single (convex) norm constraint $\|x\| \leq r$, where r (which is part of the input) is the *radius* of the trust region. If the norm is ℓ_2 and $Ax \geq b$ has a special structure, then the problem can be solved in polytime [44]. In this case the solution is generally irrational (due to the nonlinear norm constraint — note that because of that constraint, the TRS is not formally a subclass of QP in general), though for the simple case of $\|x\|_2 \leq 1$ and no linear constraints, the technical report [294] states that the decision problem is actually in **P**.

Most TRSs arising in practical black-box optimization problems, however, are formulated with an ℓ_∞ norm constraint. This makes the resulting QP easier to solve numerically, to a given $\varepsilon > 0$ approximation tolerance, using local NLP solvers. This is because the ℓ_∞ norm yields simple box constraints on the decision variables, and therefore the ℓ_∞ TRS *does* belong to the class QP, specifically it is a box-constrained QP. From a worst-case computational complexity point of view, however, we recall that box-constrained QPs form an **NP**-hard subclass of QP, as mentioned above.

5.3.5 Continuous Quadratic Knapsack

The reduction is from SUBSET-SUM. Consider the following formulation, called *continuous Quadratick Knapsack Problem* (cQKP):

$$\left.\begin{array}{lrcl} \min & x^\top Q x & + & c^\top x \\ & \sum\limits_{j \leq n} a_j x_j & = & \gamma \\ & x & \in & [0,1]^n, \end{array}\right\} \tag{5.9}$$

where Q is a square diagonal matrix, $a, c \in \mathbb{Q}^n$, and $\gamma \in \mathbb{Q}$.

We encode a SUBSET-SUM instance $(\{s_1, \ldots, s_n\}), \sigma)$ in Eq. (5.9) as follows [295, §4.2]:

$$\left.\begin{array}{ll} \min & f(x) = \sum\limits_{j \leq n} x_j(1-x_j) \\ & \sum\limits_{j \leq n} s_j x_j = \sigma \\ & x \in [0,1]^n, \end{array}\right\} \tag{5.10}$$

We are going to show that the instance is YES iff the global optimum of Eq. (5.10) is zero. If the instance is YES, then there is a subset $J \subset \{1, \ldots, n\}$ such that $\sum\limits_{j \in J} s_j = \sigma$. Let $x_j^* = 1$ for all $j \in J$ and $x_j^* = 0$ for all $j \notin J$: then x^* is feasible in the constraints of Eq. (5.9), and yields $f(x^*) = 0$. Conversely, if x^* is a feasible solution of Eq. (5.9) yielding $f(x^*) = 0$, then each term $x_j^*(1-x_j^*)$ in $f(x)$ has value zero, which implies $x^* \in \{0,1\}^n$. Now let $J = \{j \leq n \mid x_j^* = 1\}$ be the support of x^*. By construction, J is a YES certificate for the SUBSET-SUM instance. Thus finding a global optimum of a cQKP is **NP**-hard.

5.3.5.1 Convex QKP

Since a convex QKP is a subclass of convex QP, it can be solved in polytime — but no strongly polytime algorithm is known. On the other hand, if Q is a diagonal matrix, then the objective function of Eq. (5.9) is separable.[4] As remarked in [295, §3.1] there is an $O(n \log n)$ algorithm for solving this specific variant of convex QKP [135].

5.3.6 The Motzkin-Straus formulation

In 1965, Motzkin and Straus established an interesting relationship between the maximum clique C in a graph $G = (V, E)$ and the following QP [232]:

$$\left.\begin{array}{rrcl} \max & f(x) & = & \sum\limits_{\{i,j\}\in E} x_i x_j \\ & \sum\limits_{j\in V} x_j & = & 1 \\ \forall j \in V & x_j & \geq & 0\ ; \end{array}\right\} \tag{5.11}$$

namely, that there exists a global optimum x^* of Eq. (5.11) such that

$$f^* = f(x^*) = \frac{1}{2}\left(1 - \frac{1}{\omega(G)}\right),$$

where $\omega(G)$ is the size of a maximum cardinality clique in G. In other words, this gives the following formula for computing the *clique number* of a graph:

$$\omega(G) = \frac{1}{1 - 2f^*}.$$

Moreover, a maximum clique is encoded in a global optimum x^* of Eq. (5.11), which has the form

$$\forall j \in V \quad x_j^* = \begin{cases} \frac{1}{\omega(G)} & \text{if } j \in C \\ 0 & \text{otherwise.} \end{cases}$$

Eq. (5.11) is called the *Motzkin-Straus formulation* for the MAX CLIQUE optimization problem. Its decision version CLIQUE is well known to be **NP**-complete, which makes the Motzkin-Straus formulation an appropriate candidate for reductions from CLIQUE to various problems related to QP and NLP.

We follow the proof of Motzkin and Straus as related in [10]. Let x^* be a global optimum of Eq. (5.11) with as many zero components as possible. Consider $C = \{j \in V \mid x_j^* > 0\}$. We claim that C is a maximum clique in G.

- First, we claim C is a clique. Suppose this is false to aim at a contradiction, and assume wlog that $1, 2 \in C$ and $\{1, 2\} \notin E$. For some ϵ in the interval $[-x_1^*, x_2^*]$, let $x^\epsilon = (x_1^* + \epsilon, x_2^* - \epsilon, x_3^*, \ldots, x_n^*)$. Note that x^ϵ satisfies the simplex constraint $\sum_j x_j = 1$, as well as the non-negativity constraints. Since the edge $\{1, 2\}$ is not in E, the product $x_1 x_2$ does not appear in the objective function $f(x)$. This means that ϵ^2 never occurs in $f(x^\epsilon)$. In particular, $f(x)$ is linear in ϵ. We temporarily look at f as a function f_ϵ of ϵ, parametrized by x^*. By the choice of x^* (a global optimum), f_ϵ achieves its maximum at $\epsilon = 0$. Since $\epsilon = 0$ is in the interior of its range $[-x_1^*, x_2^*]$ and f_ϵ is linear in ϵ, f_ϵ must necessarily be constant over this range. Hence setting $\epsilon = -x_1^*$ and $\epsilon = x_2^*$ yields global optima with a smaller number of nonzero components x^*, which is a contradiction as x^* was chosen with the maximum possible number of zero components. Thus C is a clique.

[4] A multivariate function is separable if it can be written as a sum of univariate functions.

- Now, we claim C has maximum cardinality $|C| = \omega(G)$. Consider the simplex constraint $\sum_j x_j = 1$ and square both sides:

$$1 = \left(\sum_{j \in V} x_j \right)^2 = 2 \sum_{i<j \in V} x_i x_j + \sum_{j \in V} x_j^2. \tag{5.12}$$

Since by construction of C we have $x_j^* = 0$ iff $j \notin C$, the above reduces to

$$\psi(x^*) = 2 \sum_{i<j \in C} x_i^* x_j^* + \sum_{j \in C} (x_j^*)^2.$$

Moreover, $\psi(x) = 1$ for all feasible x by Eq. (5.12). So the objective function $f(x) = \sum_{i,j} x_i x_j$ achieves its is maximum when the second term of $\psi(x)$ is minimum (given that the sum of the two terms is constant by Eq. (5.12)), i.e. when $\sum_{j \in C}(x_j^*)$ attains its minimum, which occurs at $x_j^* = \frac{1}{|C|}$. Again by the simplex constraint, we have

$$f(x^*) = 1 - \sum_{j \in C} (x_j^*)^2 = 1 - |C| \frac{1}{|C|^2} = 1 - \frac{1}{|C|} \leq 1 - \frac{1}{\omega(G)},$$

with equality when $|C| = \omega(G)$, as claimed.

By reduction from CLIQUE, it follows therefore that Eq. (5.11) describes an **NP**-hard subclass of QP. Using the same basic ideas, Vavasis gives a proof by induction on $|V|$ in [295, Lemma 4.1]. A bijection between the local optima of Eq. (5.11) and the maximal cliques of G is discussed in [49].

5.3.6.1 QP on a simplex

Consider the following formulation:

$$\left.\begin{array}{rlcl} \min & x^\top Q x & + & c^\top x \\ & \sum\limits_{j \leq n} x_j & = & 1 \\ \forall j \leq n & x_j & \geq & 0, \end{array}\right\} \tag{5.13}$$

where Q is a square $n \times n$ rational matrix and $c \in \mathbb{Q}^n$. Since Eq. (5.11) describes a subclass of Eq. (5.13), the latter is **NP**-hard by inclusion.

5.3.7 QP with one negative eigenvalue

So far, we established that cQPs are in **P**, and that a variety of nonconvex QPs are **NP**-hard (with their decision version being **NP**-complete). Where does efficiency end and hardness begin? In an attempt to provide an answer to this question, Pardalos and Vavasis proved in [241] that an objective function $\min x^\top Q x$ where Q has rank one and has a single negative eigenvalue suffices to make the problem **NP**-hard. The problem of solving a QP with one negative eigenvalue is denoted by QP1NE.

The construction is very ingenious. It reduces CLIQUE to QP1NE in two stages. First, it provides a QP formulation attaining globally optimal objective function value zero iff the main decision variable vector x takes optimal values in $\{0, 1\}$. Second, it encodes a clique of given cardinality k in a given graph $G = (V, E)$ by means of the following constraints:

$$\forall \{i, j\} \notin E \quad x_i + x_j \leq 1 \tag{5.14}$$

$$\sum_{j \in V} x_j = k \tag{5.15}$$

$$0 \leq x \leq 1. \tag{5.16}$$

The rest of the formulation, which essentially ensures integrality of the decision variables, is as follows:

$$\begin{aligned} \min \quad z &= w^2 && (5.17)\\ w &= \sum_{j\in V} 4^j x_j && (5.18)\\ z &= \sum_{j\in V} 4^{2j} x_j + 2\sum_{i<j} 4^{i+j} y_{ij} && (5.19)\\ \forall i < j \in V \quad y_{ij} &\geq \max(0, x_i + x_j - 1). && (5.20) \end{aligned}$$

Eq. (5.17) clearly is of rank one and has a single nonzero eigenvalue which is negative. Note that the definitions of w^2 (by means of Eq. (5.18)) and z in Eq. (5.19) almost match: we should have $y_{ij} = x_i x_j$ for them to be equal. Eqs. (5.16), (5.20) and (5.19) ensure that z cannot be negative, which also holds for w^2 since it is a square. A couple of technical lemmata ensure that the QP in Eqs. (5.17)-(5.20) has optimal value zero iff the optimal solution is binary. Integrating the constraints Eqs. (5.14)-(5.16) encodes CLIQUE in this QP so that G has a clique of cardinality k iff the optimal objective function value is zero, as claimed.

5.3.8 Bilinear programming

The BILINEAR PROGRAMMING PROBLEM (BPP) reads as follows:

$$\left.\begin{array}{ll} \min & \sum_{j\leq n} x_j y_j \\ & Ax + By \geq b. \end{array}\right\} \tag{5.21}$$

The **NP**-hardness of BPP has been established in [41], by a reduction from a problem in computational geometry called 2-LINEAR SEPARABILITY (2LS), consisting in determining whether two sets of points in a Euclidean space can be separated by a piecewise linear curve consisting of two pieces. In turn, 2LS was proven **NP**-complete in [216].

Bennett and Mangasarian show in [41] that the 2LS reduces to one of three possible BPP formulations, all of which with general inequality constraints and non-negativity constraints on the decision variables.

5.3.8.1 Products of two linear forms

In [293, p. 37], Vavasis proposed as an open question whether the following QP:

$$\left.\begin{array}{ll} \min & f(x) = (c^\top x + \gamma)(d^\top x + \delta) \\ & Ax \geq b \end{array}\right\} \tag{5.22}$$

is **NP**-hard. Note that that although Eq. (5.22) is a subclass of QP, which is itself an **NP**-hard problem, there is the possibility that this might be a tractable case. This question was eventually settled in the negative in [213], which gives an **NP**-hardness proof for Eq. (5.22).

The proof is constructed very ingeniously by borrowing the functional form $\min x_1 - x_2^2$ from QP1NE. First, it presents a reduction of the (**NP**-complete) SET PARTITION problem to the formulation

$$\left.\begin{array}{rl} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} p^{i+j} y_{ij} - \left(\sum_{i=1}^{n} p^i x_i\right)^2 \\ & Sx = 1 \\ \forall i \neq j \leq n & y_{ij} \leq x_i \\ \forall i \neq j \leq n & y_{ij} \leq x_j \\ \forall i \neq j \leq n & y_{ij} \geq x_i + x_j - 1 \\ \forall i \leq n & y_{ii} = x_i \\ & x \in [0,1]^n \\ & y \in [0,1]^{n^2}, \end{array}\right\} \tag{5.23}$$

where p is any positive integer and $Sx = 1$ is a set of constraints modelling SET PARTITION. This is achieved by showing that the objective function is positive if and only if there are fractional feasible solutions x' with $0 < x'_i < 1$ for some $i \leq n$. The crucial point of the proof is that the reformulation constraints $x_i + x_j + 1 \leq y_{ij} \leq \min(x_i, x_j)$ and $y_{ii} = x_i$ are exact only for $x \in \{0, 1\}$. A technical (but easy to follow) argument shows that the objective function is positive at the optimum if and only if some fractional solution exists.

This proof is then extended to the generalization of Eq. (5.23) where the objective function is replaced by a general function $g(x_0, y_0)$ where $x_0 = \sum_i p^i x_i$, $y_0 = \sum_{i,j} p^{i+j} y_{ij}$ and two more conditions on x_0, y_0 designed to endow g with positivity/negativity properties similar to that of the objective function of Eq. (5.23). This new problem also has non-positive objective function value at the optimum if and only if the optimum is binary. This provides the reduction mechanism from SET PARTITION (encoded in the constraints $Sx = 1$).

Finally, the function $g(x_0, y_0) = (y_0 - p + 2p^{4n})^2 - 4p^{4n}x_0^2 - 4p^{8n}$ is shown to satisfy the requirements on g and also to factor as a product of two linear forms plus a constant, which provides the necessary form shown in Eq. (5.22).

The same proof mechanism can also prove **NP**-hardness of the related problems

$$\min \quad \{x_1 x_2 \mid Ax \geq b\} \tag{5.24}$$

$$\min \quad \left\{x_1 - \frac{1}{x_2} \mid Ax \geq b\right\} \tag{5.25}$$

$$\min \quad \left\{\frac{1}{x_1} - \frac{1}{x_2} \mid Ax \geq b\right\}. \tag{5.26}$$

5.3.9 Establishing local minimality

The problem of deciding whether a given x is a local minimum of a QP or not has attracted quite a lot of attention in the late 1980s and early 1990s. There appear to be three distinct proofs of this fact: in [235, Problem 1], in [244, §3], and in [295, §5.1]. The first [235] reduces SUBSET-SUM to the problem of deciding whether a quadratic form over a translated simplex is negative, shown to be equivalent to unboundedness of a constrained quadratic form, and to the problem of deciding whether a given point is not a local optimum of a given QP. Moreover, [235] also shows that deciding copositivity of a given matrix is co-**NP**-hard; it further proves co-**NP**-hardness of verifying that a point is *not* a local optimum in unconstrained minimization (of a polynomial of degree 4). The second [244] presents a QP where a certain point is not a local minimum iff a given 3SAT instance is YES. The third [295] is based on an unconstrained variant of the Motzkin-Straus formulation, where the zero vector is a local minimum iff a given CLIQUE instance is NO.

These proofs all show that it is as hard to prove that a given point is *not* a local minimum of a QP as to show that a given **NP**-hard problem instance is YES. Implicitly, using the reduction transformation, this provides certificates for the NO instances of the QP local minimality problem (QPLOC) rather than for the YES instances. In computational complexity, this shows that QPLOC is co-**NP**-hard[5] rather than **NP**-hard.

Currently, no-one knows whether **NP** = co-**NP**. Assume we could prove that **NP** $\neq$ co-**NP** (†); we know that **P** = co-**P** (the whole polynomially long trace of the algorithm provides a polynomially long certificate of both YES and NO instances). Suppose now **P** = **NP**: then it would follow from **P** = co-**P** that **NP** = co-**NP**, a contradiction with (†), which would prove **P** $\neq$ **NP**, the most famous open question in computer science and one of the most famous in mathematics. This tells us that proving **NP** $\neq$ co-**NP** looks exceedingly difficult.

[5] See Footnote 2.

It is remarkable that many **NP**-hard problems have QP formulations (proving that solving QPs is **NP**-hard), but verifying whether a given point is a local optimum is actually co-**NP**-hard. Looking in more depth, most of the **NP**-hardness reductions about QP actually reduce to deciding whether a given QP has zero optimal objective function value or not. On the other hand, the co-**NP**-hardness reductions in [235, 244, 295] all establish *local optimality of a given solution vector*. The former setting sometimes allows for clear dichotomies: for example, for the box-constrained QP in [235, Eq. (8)], it is shown that there is a finite gap between the optimal value being zero or less than a finite negative value (this implies that the corresponding decision problem is actually in **NP**). In practice, a sufficiently close approximation to zero can be rounded to zero exactly, which means that a certificate for "equal to zero" can be exhibited. Moreover, a YES instance of a problem might have multiple certificates, each of which is mapped (through the reduction) to a different solution of the corresponding QP having the same objective function value. The latter setting (about local optimality), on the other hand, is about a specific point. Since QP involves continuous functions of continuously varying variables, it seems hard to find cases where gap separations are possible. Furthermore, encoding a YES instance, possibly with multiple certificates, in the verification of optimality of a single point appears to be a hard task. These two considerations might be seen as an intuitive explanation about why QPLOC is co-**NP**-hard.

A completely different (again, intuitive) argument could be brought against the fact that QPLOC should be a hard problem at all. Given a candidate local minimum $\bar{x}$ in a QP, why not simply verify first and second order conditions? First-order conditions are equivalent to KKT conditions, and second-order conditions are well-known to involve the Hessian, i.e. the square symmetric matrix having second derivatives w.r.t. two decision variables as components. The main issue with this approach is that all QPs that have been proved hard in this section are constrained, so the Hessian alone is not sufficient: indeed, local solution algorithms for constrained QPs all look at the Hessian of the Lagrangian function, which involves the constraints; moreover, the Hessian of a QP objective is constant, so it cannot depend on $\bar{x}$. The second issue is that, in general, local and global optima of QPs may involve algebraic numbers, for which we do not know a compact representation in terms of the length of the input (see Sect. 4.2).

5.4 General Nonlinear Programming

The general NLP formulation is like Eq. (2.2) where $Z = \varnothing$. It obviously contains QP, so NLP is **NP**-hard in general. Murty and Kabadi's proof of co-**NP**-hardness of verifying whether a given point is a local minimum (see previous section) uses the following formulation

$$\min_u (u^2)^\top Q u^2,$$

where u^2 denotes the vector $(u_1^2, \ldots, u_n^2)$. This is a minimization of a quartic polynomial, and so it is not, strictly speaking, quadratic (although it is readily seen to be equivalent to the quadratic problem $\min_{x \geq 0} x^\top Q x$).

In Sect. 4.3.5 we discussed *undecidable* NLP problems, so the hardness issue is somewhat moot. Two hardness results in NLP are nonetheless worth mentioning: deciding whether a given function is convex, and optimization over the copositive cone. These two problems are related by the notion of convexity — which, according to Rockafellar, is the real barrier between easy and difficult optimization problems. Given any function, can we efficiently decide whether it is convex? If so, then we may hope to optimize it efficiently using a special-purpose algorithm for convex functions. This paradigm was doubly shattered by relatively recent results. On the one hand, Ahmadi et al. proved **NP**-hardness of deciding convexity of polynomials of 4th degree. On the other, the realization that many **NP**-hard problems have natural copositive formulations [103] made Rockafellar's remark obsolete.

5.4.1 Verifying convexity

In [242], Pardalos and Vavasis presented a list of open questions in the field of computational complexity in numerical optimization problems. Problem 6, due to Shor, reads as follows.

> Given a degree-4 polynomial of n variables, what is the complexity of determining whether this polynomial describes a convex function [over the whole variable range]?

Polynomials of degree smaller than 4 are easy to settle: degree-1 are linear and hence convex, degree-2 have constant Hessian which can be computed in polytime in order to decide their convexity, and odd-degree polynomials are never convex over their whole domain (although they may be pseudo- or quasi-convex, see [163, Thm. 13.11]). Degree-4, the smallest open question concerning the efficient decidability of polynomial convexity, was settled in [9] in the negative.

The proof provided in [9] reduces from the problem of determining whether a given *biquadratic form* $\sum_{\substack{i\leq j\\ k\leq \ell}} \alpha_{i,j,l\ell} x_i x_j y_k y_\ell$ is non-negative over all its range is strongly **NP**-hard by reduction from CLIQUE [194] by way of the Motzkin-Straus formulation Eq. (5.11). It shows how to construct a (variable) Hessian form from any biquadratic form such that the former is non-negative iff the latter is. This Hessian form allows the construction of a quartic polynomial that is convex over its range iff the original biquadratic form is non-negative.

This result is extended to deciding other forms of convexity as well: strict and strong convexity, as well as pseudo- and quasi-convexity.

5.4.1.1 The copositive cone

First, a square symmetric matrix A is *copositive* if $x^\top Ax \geq 0$ for all $x \geq 0$. If we removed the non-negativity condition $x \geq 0$, we would retrieve a definition of A being PSD: while every PSD matrix is copositive, the converse is not true. Establishing copositivity of non-PSD matrices is **NP**-hard, as it involves verifying whether the optimal objective function value of the QP $P \equiv \min\{x^\top Ax \mid x \geq 0\}$ is ≥ 0 or either < 0 or unbounded [235].

We consider a variant of Eq. (5.13) where c is the zero vector (so the objective function is purely quadratic). This variant is called STANDARD QUADRATIC PROGRAMMING (StQP). Surprisingly, the (**NP**-hard) StQP can be exactly reformulated to a *convex* continuous cNLP, as reported here below.

(a) We linearize each product $x_i x_j$ occurring in the quadratic form $x^\top Qx$ to a new variable X_{ij}. This yields:

$$\left.\begin{array}{rrcl} \min & Q \bullet X & & \\ & \sum\limits_{j\leq n} x_j & = & 1 \\ \forall j \leq n & x_j & \geq & 0 \\ & X & = & xx^\top, \end{array}\right\}$$

where $\bullet$ denotes $\mathsf{trace}(Q^\top X)$. Note that $X = xx^\top$ encodes the whole constraint set

$$\forall i < j \leq n \quad X_{ij} = x_i x_j.$$

(b) We square both sides of the simplex constraint $\sum_j x_j = 1$ to obtain $\sum_{i,j} x_i x_j = 1$, which can be written as $\mathbf{1} \bullet X = 1$, where $\mathbf{1}$ is the all-one $n \times n$ matrix.

(c) We replace the (nonconvex) set $C = \{X \mid X = xx^\top \wedge x \geq 0\}$ by its convex hull $\mathcal{C} = \mathsf{conv}(C)$. This

yields:

$$\left.\begin{array}{lrcl} \min & Q \bullet X & & \\ & \mathbf{1} \bullet X & = & 1 \\ & X & \in & \mathcal{C}. \end{array}\right\} \tag{5.27}$$

The set $\mathcal{C}$ is a convex combination of rank-one matrices, and as such forms a convex cone.

(d) We write the dual of Eq. (5.27):

$$\left.\begin{array}{lrcl} \max & y & & \\ & Q - y\mathbf{1} & \in & \mathcal{C}^*, \end{array}\right\} \tag{5.28}$$

where $\mathcal{C}^*$ is the dual cone of $\mathcal{C}$. Unlike the PSD cone, which is self-dual, $\mathcal{C}^* \neq \mathcal{C}$. In fact, $\mathcal{C}^*$ turns out to be the cone of all copositive matrices:

$$\mathcal{C}^* = \{A \mid \forall x \geq 0 \; (x^\top A x \geq 0)\},$$

see [130, Thm. 16.2.1] for a detailed proof. Both cones are convex and achieve strong duality. Hence Eq. (5.28) is a cNLP, as claimed.

This is extremely surprising, as cNLPs are known to be "easy to solve", since every local optimum is also global: thus a local solution algorithm should immediately find the global optimum (this proabably motivated Rockafellar to put the barrier of complexity in MP at the frontier of convexity and nonconvexity, see Sect. 5.4).

The issue is that no-one knows of any efficient algorithm for solving the cNLP in Eq. (5.28), and this is exactly because of the copositive cone $\mathcal{C}^*$ [50, 103]. Those cNLPs which we know how to solve efficiently are usually defined over the non-negative orthant or the PSD cone. We can obviously test whether a vector is in the non-negative orthant in linear time, and we can also test whether a given matrix is PSD in polytime [269, p. 1152]. The polynomially long traces of the checking algorithms provide compact certificates for both YES and NO instances. On the other hand, since testing copositivity is **NP**-hard, there is no hope of achieving such compact descriptions for the copositive cone $\mathcal{C}^*$.

A different way to see this issue is to consider that cNLPs over matrix cones are routinely solved by IPMs, the polytime analysis of which rest on the existence of some functions called "self-concordant barriers", that deviate the search path for the optimum away from the border of the cone. Barrier functions are used as penalty functions to be added to the objective function, which is made to increase towards the boundary of the cone. Self-concordant barriers can be optimized efficiently using Newton's method. Self-concordant barriers are known for many convex cones (orthants, PSD and more), but not for $\mathcal{C}^*$. In [51], an IPM-based heuristic is proposed to solve copositive programs such as [51].

Another interpretation of the complexity inherent in the copositive cone is the classification of extremal matrices of copositive cones of small dimension, pursued by Roland Hildebrand [140]. He emphasizes that the description complexity of these extreme rays increases dramatically as the dimension increases.

Part III

Mathematical Programming

Chapter 6

Convex analysis

This chapter contains a condensed treatment of the basics of convex analysis and duality.

6.0.1 Definition
Given a set $X \subseteq \mathbb{R}^n$, a function $f : X \to \mathbb{R}$ and a point $x' \in X$:

- *we say that x' is a local minimum of f if there is a ball $S(x', \epsilon)$ (for some $\epsilon > 0$) such that $\forall x \in S(x', \epsilon) \cap X$ we have $f(x') \leq f(x)$ — the local minimum is strict if $f(x') < f(x)$;*
- *we say that x' is a global minimum of f if $\forall x \in X$ we have $f(x') \leq f(x)$ — again, the global minimum is strict if $f(x') < f(x)$.*

Similar definitions hold for (strict) local/global maxima. If the objective function direction is not specified, the corresponding points are called *optima*.

6.1 Convex analysis

6.1.1 Definition
A set $S \subseteq \mathbb{R}^n$ is convex if for any two points $x, y \in S$ the segment between them is wholly contained in S, that is, $\forall \lambda \in [0, 1]$ $(\lambda x + (1 - \lambda)y \in S)$.

A linear equation $a^\top x = b$ where $a \in \mathbb{R}^n$ defines a hyperplane in $\mathbb{R}^n$. The corresponding linear inequality $a^\top x \leq b_0$ defines a closed half-space. Both hyperplanes and closed half-spaces are convex sets. Since any intersection of convex sets is a convex set, the subset of $\mathbb{R}^n$ defined by the system of closed half-spaces $Ax \geq b, x \geq 0$ is convex (where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$).

6.1.2 Definition
A subset $S \subseteq \mathbb{R}^n$ defined by a finite number of closed half-spaces is called a polyhedron. A bounded, non-empty polyhedron is a polytope.

Polyhedra and polytopes are very important in optimization as they are the geometrical representation of a feasible region expressed by linear constraints.

6.1.3 Definition
Let $S = \{x_i \mid i \leq n\}$ be a finite set of points in $\mathbb{R}^n$.

1. *The set span(S) of all linear combinations $\sum_{i=1}^{n} \lambda_i x_i$ of vectors in S is called the linear hull (or linear closure, or span) of S.*

2. *The set aff(S) of all affine combinations $\sum_{i=1}^n \lambda_i x_i$ of vectors in S such that $\sum_{i=1}^n \lambda_i = 1$ is called the affine hull (or affine closure) of S.*

3. *The set cone(S) of all conic combinations $\sum_{i=1}^n \lambda_i x_i$ of vectors in S such that $\forall i \leq n$ $(\lambda_i \geq 0)$ is called the conic hull (or conic closure, or cone) of S. A conic combination is strict if for all $i \leq n$ we have $\lambda_i > 0$.*

4. *The set conv(S) of all convex combinations $\sum_{i=1}^n \lambda_i x_i$ of vectors in S such that $\sum_{i=1}^n \lambda_i = 1$ and $\forall i \leq n$ $(\lambda_i \geq 0)$ is called the convex hull (or convex closure) of S. A convex combination is strict if for all $i \leq n$ we have $\lambda_i > 0$.*

6.1.4 Definition
Consider a polyhedron P and a closed half-space H in $\mathbb{R}^n$. Let $\overline{H \cap P}$ be the affine closure of $H \cap P$ and $d = \dim(\overline{H \cap P})$. If $d < n$ then $H \cap P$ is a face of P. If $d = 0$, $H \cap P$ is called a vertex of P, and if $d = 1$, it is called an edge. If $d = n - 1$ then $H \cap P$ is a facet of P.

The following technical result is often invoked in the proofs of LP theory.

6.1.5 Lemma
Let $x^* \in P = \{x \geq 0 \mid Ax \geq b\}$. Then x^* is a vertex of P if and only if for any pair of distinct points $x', x'' \in P$, x^* cannot be a strict convex combination of x', x''.

Proof. ($\Rightarrow$) Suppose x^* is a vertex of P and there are distinct points $x', x'' \in P$ such that there is a $\lambda \in (0,1)$ with $x^* = \lambda x' + (1-\lambda)x''$. Since x^* is a vertex, there is a half-space $H = \{x \in \mathbb{R}^n \mid h^\top x \leq d\}$ such that $H \cap P = \{x^*\}$. Thus $x', x'' \notin H$ and $h^\top x' > d$, $h^\top x'' > d$. Hence $h^\top x^* = h^\top(\lambda x' + (1-\lambda)x'') > d$, whence $x^* \notin H$, a contradiction. ($\Leftarrow$) Assume that x^* cannot be a strict convex combination of any pair of distinct points x', x'' of P, and suppose that x^* is not a vertex of P. Since x^* is not a vertex, it belongs to a face $H \cap P$ of P (with H a closed half-space) with $d = \dim \text{aff}(H \cap P) > 0$. Then there must be a ball S (of dimension d) within $H \cap P$, having radius $\varepsilon > 0$ and center x^*. Let $x' \in S$ such that $||x' - x^*|| = \frac{\varepsilon}{2}$. Let $x'' = (x^* - x') + x^*$. By construction, x'' is the point symmetric to x' on the line through x' and x^*. Thus, $||x'' - x^*|| = \frac{\varepsilon}{2}$, $x'' \neq x'$, $x'' \in P$, and $x^* = \frac{1}{2}x' + \frac{1}{2}x''$, which is a strict convex combination of x', x'', a contradiction. □

Having defined convex sets, we now turn our attention to convex functions.

6.1.6 Definition
A function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ is convex if for all $x, y \in X$ and for all $\lambda \in [0,1]$ we have:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

Note that for this definition to be consistent, $\lambda x + (1-\lambda)y$ must be in the domain of f, i.e. X must be convex. This requirement can be formally relaxed if we extend f to be defined outside X.

The main theorem in convex analysis, and the fundamental reason why convex functions are useful in optimization, is that a local optimum of a convex function over a convex set is also a global optimum, which we prove in Thm. 6.1.7. The proof is described graphically in Fig. 6.1.

6.1.7 Theorem
Let $X \subseteq \mathbb{R}^n$ be a convex set and $f : X \to \mathbb{R}$ be a convex function. Given a point $x^* \in X$, suppose that there is a ball $S(x^*, \varepsilon) \subset X$ such that for all $x \in S(x^*, \varepsilon)$ we have $f(x^*) \leq f(x)$. Then $f(x^*) \leq f(x)$ for all $x \in X$.

Proof. Let $x \in X$. Since f is convex over X, for all $\lambda \in [0,1]$ we have $f(\lambda x^* + (1-\lambda)x) \leq \lambda f(x^*) + (1-\lambda) f(x)$. Notice that there exists $\bar{\lambda} \in (0,1)$ such that $\bar{\lambda} x^* + (1-\bar{\lambda})x = \bar{x} \in S(x^*, \varepsilon)$. Consider $\bar{x}$: by

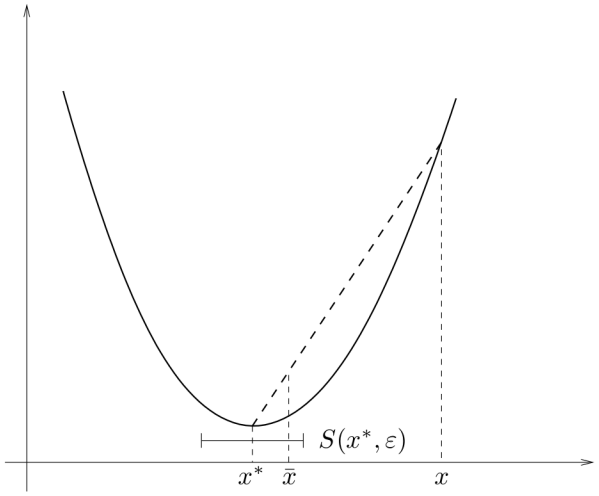


Figure 6.1: Graphical sketch of the proof of Thm. 6.1.7. The fact that $f(x^*)$ is the minimum in $S(x^*, \varepsilon)$ is enough to show that for any point $x \in X$, $f(x^*) \leq f(x)$.

convexity of f we have $f(\bar{x}) \leq \bar{\lambda} f(x^*) + (1 - \bar{\lambda}) f(x)$. After rearrangement, we have

$$f(x) \geq \frac{f(\bar{x}) - \bar{\lambda} f(x^*)}{1 - \bar{\lambda}}.$$

Since $\bar{x} \in S(x^*, \varepsilon)$, we have $f(\bar{x}) \geq f(x^*)$, thus

$$f(x) \geq \frac{f(x^*) - \bar{\lambda} f(x^*)}{1 - \bar{\lambda}} = f(x^*),$$

as required. □

6.2 Conditions for local optimality

In this section we shall give necessary and sufficient conditions for a feasible point x^* to be locally optimal. Most of the work deals with deriving necessary conditions. The sufficient conditions are in fact very close to requiring convexity of the objective function and feasible region, as we shall see later.

For a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ we define the function vector ∇f as $(\nabla f)(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)^\top$, where $x = (x_1, \ldots, x_n)^\top$. We denote by $\nabla f(x')$ the evaluation of ∇f at x'. If g is a vector-valued function $g(x) = (g_1(x), \ldots, g_m(x))$, then ∇g is the set of function vectors $\{\nabla g_1, \ldots, \nabla g_m\}$, and $\nabla g(x')$ is the evaluation of ∇g at x'.

6.2.1 Equality constraints

We first deal with the case of an optimization problem as in Eq. (2.2) with all equality constraints, unbounded variables and $Z = \varnothing$. Consider the following NLP:

$$\left.\begin{array}{lll} \min\limits_{x \in \mathbb{R}^n} & f(x) & \\ \text{s.t.} & g(x) & = \; 0, \end{array}\right\} \tag{6.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are C^1 (i.e., continuously differentiable) functions.

A *constrained critical point* of Eq. (6.1) is a point $x^* \in \mathbb{R}^n$ such that $g(x^*) = 0$ and the directional derivative[1] of f along g is 0 at x^*. It is easy to show that such an x^* is a (local) maximum, or a minimum, or a saddle point of $f(x)$ subject to $g(x) = 0$.

6.2.1 Theorem (Lagrange Multiplier Method)
If x^* is a constrained critical point of Eq. (6.1), $m \leq n$, and $\nabla g(x^*)$ is a linearly independent set of vectors, then $\nabla f(x^*)$ is a linear combination of the set of vectors $\nabla g(x^*)$.

Proof. Suppose, to get a contradiction, that $\nabla g(x^*)$ and $\nabla f(x^*)$ are linearly independent. In the case $\nabla f(x^*) = 0$, the theorem is trivially true, so assume $\nabla f(x^*) \neq 0$. Choose vectors $w_{m+2}, \ldots, w_n$ such that the set $J = \{\nabla g_1(x^*), \ldots, \nabla g_m(x^*), \nabla f(x^*), w_{m+2}, \ldots, w_n\}$ is a basis of $\mathbb{R}^n$. For $m + 2 \leq i \leq n$ define $h_i(x) = \langle w_i, x \rangle$. Consider the map

$$F(x) = (F_1(x), \ldots, F_n(x)) = (g_1(x), \ldots, g_m(x), f(x), h_{m+2}(x), \ldots, h_n(x)).$$

Since the Jacobian (i.e., the matrix of the first partial derivatives of each constraint function) of F evaluated at x^* is J, and J is nonsingular, by the inverse function theorem F is a local diffeomorphism of $\mathbb{R}^n$ to itself. Thus, the equations $y_i = F_i(x)$ $(i \leq n)$ are a local change of coordinates in a neighbourhood of x^*. With respect to coordinates y_i, the surface S defined by $g(x) = 0$ is the coordinate hyperplane $0 \times \mathbb{R}^{n-m}$. Notice that the $(k + 1)$-st coordinate, $y_{k+1} = f(x)$, cannot have a critical point on the coordinate plane $0 \times \mathbb{R}^{n-m}$, so x^* is not a constrained critical point, which is a contradiction. □

In the proof of the above theorem, we have implicitly used the fact that the criticality of points is invariant with respect to diffeomorphism (this can be easily established by showing that the derivatives of the transformed functions are zero at the point expressed in the new coordinates). The classical proof of the Lagrange Multiplier Theorem 6.2.1 is much longer but does not make explicit use of differential topology concepts (see [22], p. 153). The proof given above was taken almost verbatim from [249].

By Thm. 6.2.1 there exist scalars $\lambda_1, \ldots, \lambda_m$, such that

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) = 0.$$

The above condition is equivalent to saying that if x^* is a constrained critical point of f s.t. $g(x^*) = 0$, then x^* is a critical point of the following (unconstrained) function:

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x). \tag{6.2}$$

Eq. (6.2) is called the *Lagrangian* of f w.r.t. g, and $\lambda_1, \ldots, \lambda_m$ are called the *Lagrange multipliers*. Intuitively, when we are optimizing over a subspace defined by $Ax = b$, the optimization direction must be a linear combination of the vectors which are normal to the hyperplane system $Ax = b$. The situation is shown in Fig. 6.2.

Hence, in order to find the solution of Eq. (6.1), by Thm. 6.2.1, one should find all critical points of $L(x, \lambda)$ and then select the one with minimum objective function value. This approach is of limited use on all but the simplest formulations. It does, however, provide at least necessary conditions for the characterization of a constrained minimum.

[1] A formal definition is given in [223], p. 7-8.

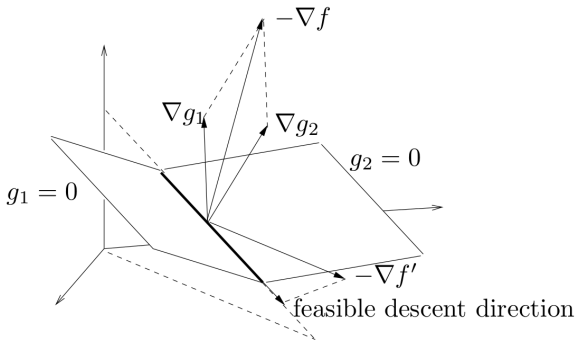


Figure 6.2: If ∇f is linearly dependent on the constraint gradients, there is no feasible descent direction. $\nabla f'$, which does not have this property, identifies a feasible descent direction when projected on the feasible space (the thick line).

6.2.2 Inequality constraints

Notice, however, that Eq. (6.1) only deals with equality constraints; moreover, the proof of Thm. 6.2.1 is heavily based on the assumption that the only constraints of the problem are equality constraints. In order to take into account inequality constraints, we introduce the following simple example.

6.2.2 Example
Consider the problem $\min\{-x_1 - x_2 \mid x_1 - 1 \leq 0, x_2 - 1 \leq 0\}$. *The minimum is obviously at* $x^* = (1,1)$ *as shown in Fig. 6.3 (the figure only shows the non-negative quadrant; the feasible region actually extends to the other quadrants). By inspection, it is easy to see that at the point* $x^* = (1,1)$ *any further movement*

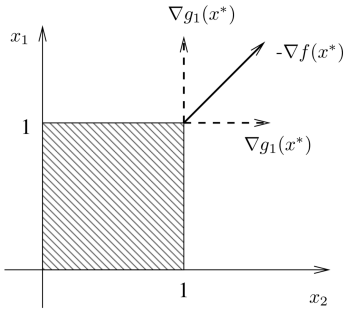


Figure 6.3: The problem of Example 6.2.2.

towards the direction of objective function decrease would take x^* *outside the feasible region, i.e. no feasible direction decreases the objective function value. Notice that the descent direction at* x^* *is in the cone defined by the normal vectors to the constraints in* x^*. *In this particular case, the descent direction of the objective function is the vector* $-\nabla f(x^*) = (1,1)$. *The normal vector to* $g_1(x) = x_1 - 1$ *at* x^* *is* $\nabla g_1(x^*) = (1,0)$ *and the normal vector to* $g_2(x) = x_2 - 1$ *at* x^* *is* $\nabla g_2(x^*) = (0,1)$. *Notice that we can express* $-\nabla f(x^*) = (1,1)$ *as* $\lambda_1(1,0) + \lambda_2(0,1)$ *with* $\lambda_1 = 1 > 0$ *and* $\lambda_2 = 1 > 0$. *In other words,* $(1,1)$ *is a conic combination of* $(1,0)$ *and* $(0,1)$.

If we now consider the problem of minimizing $\bar{f}(x) = x_1 + x_2$ *subject to the same constraints as above, it appears clear that* $x^* = (1,1)$ *cannot be a local minimum, as there are many feasible descent directions. Take for example the direction vector* $y = (-1,-1)$. *This direction is feasible w.r.t. the*

constraints: $\nabla g_1(x^*)y = (1,0)(-1,-1) = (-1,0) \leq 0$ *and* $\nabla g_2(x^*)y = (0,1)(-1,-1) = (0,-1) \leq 0$. *Moreover, it is a nonzero descent direction:* $-\nabla f(x^*)y = -(1,1)(-1,-1) = (1,1) > 0$.

In summary, either the descent direction at x^* is a conic combination of the gradients of the constraints (and in this case x^* may be a local minimum), or there is a nonzero feasible descent direction (and in this case x^* cannot be a local minimum).

The example above is an application of a well known theorem of alternatives called Farkas' Lemma. The following three results are necessary to introduce Farkas' Lemma, which will then be used in the proof of Thm. 6.2.7.

6.2.3 Theorem (Weierstraß)
Let $S \subseteq \mathbb{R}^n$ be a non-empty, closed and bounded set and let $f : S \to \mathbb{R}$ be continuous on S. Then there is a point $x^* \in S$ such that f attains its minimum value at x^*.

Proof. Since f is continuous on S and S is closed and bounded, then f is bounded below on S. Since S is non-empty, there exists a greatest lower bound α for the values of f over S. Let ε be such that $0 < \varepsilon < 1$ and consider the sequence of sets $S_k = \{x \in S \mid \alpha \leq f(x) \leq \alpha + \varepsilon^k\}$ for $k \in \mathbb{N}$. By the definition of infimum, S_k is non-empty for each k, so we can select a point $x^{(k)} \in S_k$ for each k. Since S is bounded, there exists a convergent subsequence of $x^{(k)}$ which converges to a point x^*. Since S is closed, $x^* \in S$. By continuity of f, and because $\alpha \leq f(x^{(k)}) \leq \alpha + \varepsilon^k$, we have that the values $f(x^{(k)})$ (taken on the convergent subsequence) converge to α. Thus x^* is a point where f attains its minimum value $f(x^*) = \alpha$ (see Fig. 6.4). □

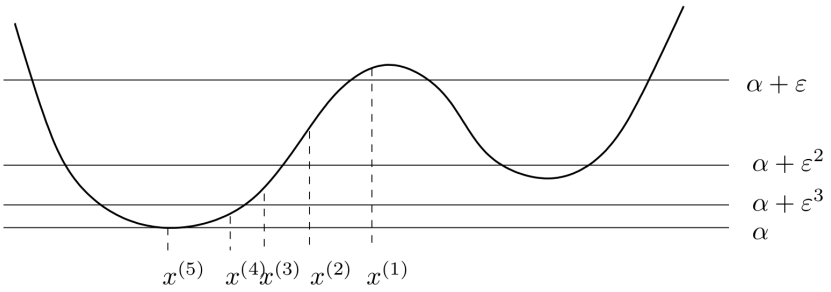


Figure 6.4: Weierstraß' Theorem 6.2.3.

6.2.4 Proposition
Given a non-empty, closed convex set $S \subseteq \mathbb{R}^n$ and a point $x^* \notin S$, there exists a unique point $x' \in S$ with minimum distance from x^*. Furthermore, x' is the minimizing point if and only if for all $x \in S$ we have $(x^* - x')^\top (x - x') \leq 0$.

Proof. Let $x' \in S$ be the point minimizing $f(x) = ||x^* - x||$ subject to $x \in S$ (which exists by Weierstraß Theorem 6.2.3). To show that it is unique, notice first that f is convex: for $\lambda \in [0,1]$ and $x_1, x_2 \in S$ we have $f(\lambda x_1 + (1-\lambda)x_2) \leq f(\lambda x_1) + f((1-\lambda)x_2) = \lambda f(x_1) + (1-\lambda) f(x_2)$ by triangular inequality and homogeneity of the norm. Suppose now $y \in \mathbb{R}^n$ such that $f(y) = f(x')$. Since $x', y \in S$ and S is convex, the point $z = \frac{x'+y}{2}$ is in S. We have $f(z) \leq \frac{f(x')}{2} + \frac{f(y)}{2} = f(x')$. Since $f(x')$ is minimal, $f(z) = f(x')$. Furthermore, $f(z) = ||x^* - z|| = ||x^* - \frac{x'+y}{2}|| = ||\frac{x^*-x'}{2} + \frac{x^*-y}{2}||$. By the triangle inequality, $f(z) \leq \frac{1}{2}||x^* - x'|| + \frac{1}{2}||x^* - y||$. Since equality must hold as $f(z) = f(x') = f(y)$, vectors $x^* - x'$ and $x^* - y$ must be collinear, i.e. there is $\theta \in \mathbb{R}$ such that $x^* - x' = \theta(x^* - y)$. Since $f(x') = f(y)$, $|\theta| = 1$. Supposing $\theta = -1$ we would have $x^* = \frac{x'+y}{2}$, which by convexity of S would imply $x^* \in S$, contradicting the hypothesis. Hence $\theta = 1$ and $x' = y$ as claimed. For the second part of the proposition, assume x'

is the minimizing point in S and suppose there is $x \in S$ such that $(x^* - x')^\top (x - x') > 0$. Take a point $y \neq x'$ on the segment $\overline{x, x'}$. Since S is convex, $y \in S$. Furthermore, since $y - x'$ is collinear to $x - x'$, $(x^* - x')^\top (y - x') > 0$. Thus, the angle between $y - x'$ and $x^* - x'$ is acute. Choose y close enough to x' so that the largest angle of the triangle T having x^*, x', y as vertices is the angle in y (such a choice is always possible) as in Fig. 6.5. By elementary geometry, the longest side of such a triangle is the segment $\overline{x^*, x'}$ opposite to the angle in y. This implies $||x^* - y|| < ||x^* - x'||$, which is a contradiction, as x' was chosen to minimize the distance from x^*. Conversely, suppose for all $x \in S$ we have $(x^* - x')^\top (x - x') \leq 0$. This means that the angle in x' is obtuse (and hence it is the largest angle of T), and consequently the opposite side $\overline{x, x^*}$ is longer than the side $\overline{x', x^*}$. □

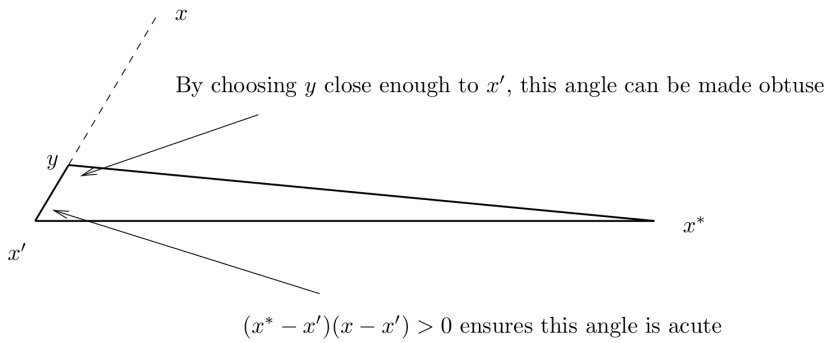


Figure 6.5: Second part of the proof of Prop. 6.2.4: the point y can always be chosen close enough to x' so that the angle in the vertex y is obtuse. The figure also shows that it is impossible for x' to be the minimum distance point from x^* if the angle in x' is acute and the set S containing points x', x, y is convex.

6.2.5 Proposition (Separating hyperplane)
Given a non-empty, closed convex set $S \subseteq \mathbb{R}^n$ and a point $x^* \notin S$, there exists a separating hyperplane $h^\top x = d$ (with $h \geq 0$) such that $h^\top x \leq d$ for all $x \in S$ and $h^\top x^* > d$.

Proof. Let x' be the point of S having minimum (strictly positive, since $x^* \notin S$) distance from x^*. Let $y = \frac{x' + x^*}{2}$ be the midpoint between x', x^*. The plane normal to the vector $x^* - y$ and passing through y separates x^* and S (see Fig. 6.6). □

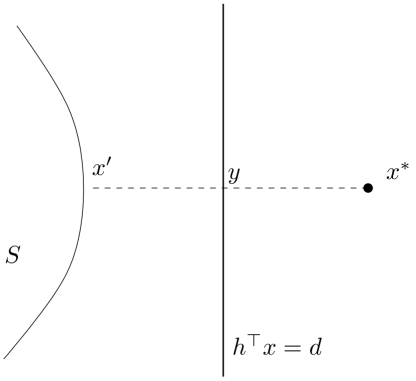


Figure 6.6: Prop. 6.2.5: separating hyperplane between a point x^* and a convex set S.

6.2.6 Theorem (Farkas' Lemma)
Let A be an $m \times n$ matrix and c be a vector in $\mathbb{R}^n$. Then exactly one of the following systems has a solution: (a) $Ax \leq 0$ and $c^\top x > 0$ for some $x \in \mathbb{R}^n$; (b) $\mu^\top A = c^\top$ and $\mu \geq 0$ for some $\mu \in \mathbb{R}^m$.

Proof. Suppose system (b) has a solution. Then $\mu^\top Ax = c^\top x$; supposing $Ax \leq 0$, since $\mu \geq 0$, we have $c^\top x \leq 0$. Conversely, suppose system (b) has no solution. Let $\mathsf{lm}_+(A) = \{z \in \mathbb{R}^n \mid z^\top = \mu^\top A, \mu \geq 0\}$; $\mathsf{lm}_+(A)$ is convex, and $c \notin \mathsf{lm}_+(A)$. By Prop. 6.2.5, there is a separating hyperplane $h^\top x = d$ such that $h^\top z \leq d$ for all $z \in \mathsf{lm}_+(A)$ and $h^\top c > d$. Since $0 \in \mathsf{lm}_+(A)$, $d \geq 0$, hence $h^\top c > 0$. Furthermore, $d \geq z^\top h = \mu^\top Ah$ for all $\mu \geq 0$. Since μ can be arbitrarily large, $\mu^\top Ah \leq d$ implies $Ah \leq 0$. Take $x = h$; then x solves system (a). □

We can finally consider the necessary conditions for local minimality subject to inequality constraints: Consider the following NLP:

$$\left.\begin{array}{lll} \min\limits_{x \in \mathbb{R}^n} & f(x) & \\ \text{s.t.} & g(x) \leq & 0, \end{array}\right\} \tag{6.3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are C^1 functions. A *constrained optimum* of Eq. (6.3) is an optimum x^* of $f(x)$ such that $g(x^*) = 0$ — a condition described as the constraint g being *active* at x^*. It can be shown that if x^* is a constrained minimum then there is no nonzero feasible descent direction at x^* (see [34], p. 141). We shall define a *feasible direction at* x^* as a direction vector y such that $(\nabla g(x^*))^\top y \leq 0$, and a *nonzero descent direction at* x^* as a direction vector y such that $(-\nabla f(x^*))^\top y > 0$.

6.2.7 Theorem (KKT)
If x^* is a constrained minimum of Eq. (6.3), I is the maximal subset of $\{1, \ldots, m\}$ such that $g_i(x^*) = 0$ for all $i \in I$, and $\nabla \bar{g}$ is a linearly independent set of vectors (where $\bar{g} = \{g_i(x^*) \mid i \in I\}$), then $-\nabla f(x^*)$ is a conic combination of the vectors in $\nabla \bar{g}$, i.e. there exist scalars λ_i for all $i \in I$ such that the following conditions hold:

$$\nabla f(x^*) + \sum_{i \in I} \lambda_i \nabla g_i(x^*) = 0 \tag{6.4}$$

$$\forall i \in I \quad (\lambda_i \geq 0). \tag{6.5}$$

Proof. Since x^* is a constrained minimum and $\nabla \bar{g}$ is linearly independent, there is no nonzero feasible descent direction at x^* such that $(\nabla \bar{g}(x^*))^\top y \leq 0$ and $-\nabla f(x^*) y > 0$. By a direct application of Farkas' Lemma 6.2.6, there is a vector $\lambda \in \mathbb{R}^{|I|}$ such that $\nabla(\bar{g}(x^*))\lambda = -\nabla f(x^*)$ and $\lambda \geq 0$. □

The KKT necessary conditions (6.4)-(6.5) can be reformulated to the following:

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) = 0 \tag{6.6}$$

$$\forall i \leq m \quad (\lambda_i g_i(x^*) = 0) \tag{6.7}$$

$$\forall i \leq m \quad (\lambda_i \geq 0). \tag{6.8}$$

This is easily verified by defining $\lambda_i = 0$ for all $i \notin I$. Conditions (6.7) are called *complementary slackness conditions*, and they express the fact that if a constraint is not active at x^* then its corresponding Lagrange multiplier is 0. A point x^* satisfying the KKT conditions is called a *KKT point*.

6.2.3 General NLPs

Consider now a general NLP with both inequality and equality constraints:

$$\left.\begin{array}{lrcl} \min\limits_{x\in\mathbb{R}^n} & f(x) & & \\ \text{s.t.} & g(x) & \leq & 0 \\ & h(x) & = & 0, \end{array}\right\} \tag{6.9}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^p$ are C^1 functions.

By applying theorems 6.2.1 and 6.2.7, we can define the Lagrangian of Eq. (6.9) by the following:

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{i=1}^{p} \mu_i h_i(x), \tag{6.10}$$

and the corresponding KKT conditions as:

$$\left.\begin{array}{rcll} \nabla f(x^*) + \sum\limits_{i=1}^{m} \lambda_i \nabla g_i(x^*) + \sum\limits_{i=1}^{p} \mu_i \nabla h_i(x^*) & = & 0 & \\ \lambda_i g_i(x^*) & = & 0 & \forall i \leq m \\ \lambda_i & \geq & 0 & \forall i \leq m. \end{array}\right\} \tag{6.11}$$

In order to derive the general KKT conditions (6.11), it is sufficient to sum Eq. (6.4) and (6.6) and then divide by 2.

This completes the discussion as regards the necessary conditions for local optimality. If a point x^* is a KKT point, the objective function is convex in a neighbourhood of x^*, and the objective direction is minimization, then x^* is a local minimum These are the sufficient conditions which are used in practice in most cases. It turns out, however, that they are not the most stringent conditions possible: many variants of convexity have been described in order to capture wider classes of functions for which all local optima are also global.

6.2.8 Definition
Consider a function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}^n$.

1. *f is quasiconvex if for all $x, x' \in X$ and for all $\lambda \in [0, 1]$ we have:*

$$f(\lambda x + (1 - \lambda)x') \leq \max\{f(x), f(x')\}.$$

2. *f is pseudoconvex if for all $x, x' \in X$ such that $f(x') < f(x)$ we have:*

$$(\nabla f(x))^\top (x' - x) < 0.$$

We state the sufficiency conditions in the following theorem; the proof can be found in [34, p. 155].

6.2.9 Theorem
Let x^* be a KKT point of Eq. (6.3), I be as in Thm. 6.2.7, and X be the feasible region of a relaxation of Eq. (6.9) where all constraints g_i such that $i \notin I$ have been discarded from the formulation. If there is a ball $S(x^*, \varepsilon)$ with $\varepsilon > 0$ such that f is pseudoconvex over $S(x^*, \varepsilon) \cap X$ and g_i are differentiable at x^* and quasiconvex over $S(x^*, \varepsilon) \cap X$ for all $i \in I$, then x^* is a local minimum of Eq. (6.9).

6.3 Duality

Given an MP formulation P, the term "dual" often refers to an auxiliary formulation Q such that the decision variables of P are used to index the constraints of Q, and the constraints of P are used to index

the variables of Q. For minimization problems, this is shown by setting up a special saddle problem can sometimes be re-cast in terms of an ordinary MP where variables and constraint indices are swapped (this is not the only relationship between primal and dual formulations, of course). Duality is important because of weak and strong duality theorems. The former guarantees that dual formulations provide lower bounds to primal minimization formulations. The latter guarantees that the "gap" between this lower bound and the optimal objective function value of the original primal formulation is zero.

In the following, we shall employ the terms *primal variables*, *dual variables*, *primal constraints*, *dual constraints*, *primal objective*, *dual objective* to denote variables, constraints and objectives in the primal and dual problems in primal/dual problem pairs.

6.3.1 The Lagrangian function

The primordial concept in duality theory is the Lagrangian function (see Eq. (6.2)) of a given formulation, which, for a problem

$$\min\{f(x) \mid g(x) \leq 0\} \tag{6.12}$$

is given by $L(x, y) = f(x)+yg(x)$. The row vector $y \in \mathbb{R}^m$ is called the vector of the Lagrange multipliers.

The Lagrangian aggregates the constraints on the objective, penalizing them by means of the multipliers. The solution $x^*(y)$ (itself a function of the multiplier vector y) to $\min_x L(x, y)$ must surely have $g(x^*(y)) \leq 0$ as long as $y \geq 0$, since we are minimizing w.r.t. x (the term $yg(x)$ must become negative for L to attain its minimum). In other words, the solution $x^*(y)$ is feasible in the original problem Eq. (6.12). We write $L(y) = \min_x L(x, y) = \min_x(f(x) + yg(x))$.

It is easy to show that for all $y \geq 0$, $L(y)$ is a lower bound to the optimal objective function value f^* of the original formulation. Thus, it makes sense to find the *best* (i.e. maximum) achievable lower bound. This leads us to maximize $L(y)$ with respect to y and subject to $y \geq 0$. The Lagrangian dual problem of Eq. (6.12) is a saddle problem defined as:

$$p^* = \max_{y \geq 0} \min_x L(x, y). \tag{6.13}$$

The considerations above show that

$$p^* \leq f^* \tag{6.14}$$

which is known as the weak duality theorem.

6.3.2 The dual of an LP

Consider the LP in *standard form*

$$\min\{c^\top x \mid Ax = b \wedge x \geq 0\}. \tag{6.15}$$

Its Lagrangian function is $L(x, y) = c^\top x + y(b - Ax)$. We have

$$\begin{aligned} & \max_{y \geq 0} \min_x L(x, y) = \\ = \; & \max_{y \geq 0} \min_x (c^\top x + y(b - Ax)) = \\ = \; & \max_{y \geq 0} \min_x (c^\top x + yb - yAx) = \\ = \; & \max_{y \geq 0} (yb + \min_x (c^\top - yA)x). \end{aligned}$$

In general, if y increases, the expression above may become unbounded. Assume therefore that the equation $yA = c^\top$ holds; this implies $\min_x (c^\top - yA)x = \min_x 0 = 0$, and so the above reduces to finding

$$\max_{y \geq 0, yA = c^\top} yb.$$

We define the above maximization problem to be the dual of the LP Eq. (6.15), which is usually written as follows:

$$\left.\begin{array}{lrcl} \max_y & yb & & \\ & yA & = & c^\top \\ & y & \geq & 0. \end{array}\right\} \tag{6.16}$$

6.3.2.1 Alternative derivation of LP duality

Eq. (6.16) can also be achieved as follows: we seek the best possible lower bound for Eq. (6.15) by considering a weighted sum of the constraints $Ax \geq b$, using weights $y_1, \ldots, y_m$. We obtain $yAx \geq yb$, which are only valid if $y \geq 0$. To get a lower bound, we need $yb \leq c^\top x$: since $yAx \geq yb$, we must require $yA = c^\top$. To make the lower bound tightest, we maximize yb subject to $y \geq 0$ and $yA = c^\top$, obtaining Eq. (6.16) again.

6.3.2.2 Economic interpretation of LP duality

Consider the diet problem of Sect. 2.2.7.1. Its dual, $\max\{yb \mid yA \leq c^\top \wedge y \geq 0\}$, can be interpreted as follows. A megalomaniac pharmaceutical firm wishes to replace human food with nutrient pills: it wishes to set the prices of the pills as high as possible, whilst being competitive with the cost of the foods. In this setting, y are the prices of the m nutrient pills, b is the quantity of nutrients required, and c are the costs of the food.

6.3.3 Strong duality

In this section we shall prove the strong duality theorem for cNLP: namely that for any cNLP Eq. (6.12) having feasible region with a non-empty interior, strong duality holds:

$$p^* = f^* \tag{6.17}$$

holds. The condition on the non-emptiness of the interior of the feasible region is called *Slater's constraint qualification*, and it asserts that:

$$\exists x' \ (g(x') < 0). \tag{6.18}$$

6.3.1 Theorem (Strong Duality)
Consider a cNLP Eq. (6.12) s.t. Eq. (6.18) holds. Then we have $p^* = f^*$.

Proof. Consider the sets $\mathcal{A} = \{(\lambda, t) \mid \exists x \ (\lambda \geq g(x) \wedge t \geq f(x))\}$ and $\mathcal{B} = \{(0, t) \mid t < f^*\}$. It is easy to show that $\mathcal{A} \cap \mathcal{B} = \emptyset$, for otherwise f^* would not be optimal. Furthermore, both $\mathcal{A}$ and $\mathcal{B}$ are convex sets. Thus, there must be a separating hyperplane defined by $(u, \mu) \neq 0$ and $\alpha \in \mathbb{R}$ such that

$$\forall (\lambda, t) \in \mathcal{A} \ (u\lambda + \mu t \geq \alpha) \tag{6.19}$$
$$\forall (\lambda, t) \in \mathcal{B} \ (u\lambda + \mu t \leq \alpha). \tag{6.20}$$

Since both λ and t can increase indefinitely, in order for the expression $u\lambda + \mu t$ to be bounded below in Eq. (6.19), we must have

$$u \geq 0, \mu \geq 0. \tag{6.21}$$

Condition (6.20) is equivalent to $\mu t \leq \alpha$ for all $t < f^*$, that is $\mu f^* \leq \alpha$, since $\lambda = 0$ in $\mathcal{B}$. Combining the latter with (6.19) we conclude that for all x (in particular, for all feasible x),

$$ug(x) + \mu f(x) \geq \mu f^*. \tag{6.22}$$

Suppose now that $\mu = 0$: this implies, by Eq. (6.22), that $ug(x) \geq 0$ for all feasible x. In particular, by Eq. (6.18) there exists x' feasible such that $g(x') < 0$, which implies $u \leq 0$, and by Eq. (6.21), this means $u = 0$, yielding $(u, \mu) = 0$ which contradicts the separating hyperplane theorem (Prop. 6.2.5). Thus $\mu > 0$ and we can set $y = \frac{1}{\mu}u$ in Eq. (6.22):

$$f(x) + yg(x) \geq f^*. \tag{6.23}$$

This implies that for all feasible x we have $L(x, y) \geq f^*$. The result follows from the weak duality theorem Eq. (6.14). □

The above proof applies to LPs Eq. (6.15) as a special case.

Chapter 7

Linear Programming

This chapter is devoted to LP. We summarize the main LP solution methods: simplex, ellipsoid, and interior point.

7.1 The Simplex method

The simplex method[1] is the fundamental algorithm used in LP. We remark straight away that it is one of the practically most efficient algorithms for LP, although all known implementations have exponential worst-case behaviour.

The simplex algorithms rests on four main observations.

- The LP in *canonical form*
$$\min\{c^\top x \mid Ax \leq b\} \tag{7.1}$$
is equivalent to minimizing a linear form over a polyhedron.
- The minimum of a linear form over a polyhedron is attained at a vertex of the polyhedron (cf. Thm. 7.1.9): since there are finitely many vertices in a polyhedron in $\mathbb{R}^n$, there exists a finite search procedure to solve the (continuous) LP.
- Verifying whether a vertex of a polyhedron is a local minimum w.r.t. a linear form is easy: it suffices to check that all adjacent vertices have higher associated objective function value.
- Polyhedra are convex sets and linear forms are convex functions, so any local minimum is also a global minimum.

Obviously, corresponding statements can be made for maximization.

The simplex algorithm starts from a feasible polyhedron vertex and moves to an adjacent vertex with lower associated objective function value. When no such vertex exists, the vertex is the minimum and the algorithm terminates. The simplex algorithm can also be used to detect unboundedness and infeasibility.

[1] The simplex method was invented by G. Dantzig in the late forties [89]; rumour has it that the young Dantzig approached J. Von Neumann to present his results, and met with the response that his algorithm was correct but not very innovative. Dantzig himself, for the very same reason, chose not to immediately publish his algorithm in a scientific journal paper, although, to date, the simplex algorithm is the most famous algorithm in Operations Research and one of the most famous in the whole field of applied mathematics.

7.1.1 Geometry of Linear Programming

The material in this section is taken from [240, 111]. Consider the LP (in standard form)

$$\min_{x \in P} c^\top x \tag{7.2}$$

where P is the polyhedron $\{x \in \mathbb{R}^n_+ \mid Ax = b\}$ and $c \in \mathbb{R}^n$.

7.1.1 Definition
A set $\{A_i \mid i \in \beta\}$ of m linearly independent columns of A is a basis of A. The variables $\{x_i \mid i \in \beta\}$ corresponding to the indices β of the basis are called basic variables. All other variables are called nonbasic variables.

Defn. 7.1.1 suggests that we can partition the columns of A in $(B|N)$ where B is the nonsingular, square matrix of the basic columns and N are the nonbasic columns. Correspondingly, we partition the variables x into (x_B, x_N).

7.1.2 Definition
Given a polyhedron $P = \{x \in \mathbb{R}^n_+ \mid Ax = b\}$, the feasible vectors x having $x_B = B^{-1}b \geq 0$ and $x_N = 0$ are called basic feasible solutions (bfs) of P.

7.1.3 Lemma
Given a polyhedron $P = \{x \in \mathbb{R}^n_+ \mid Ax = b\}$ and a bfs x^* for P, there exists a cost vector $c \in \mathbb{R}^n$ such that x^* is the unique optimal solution of the problem $\min\{c^\top x \mid x \in P\}$.

Proof. Let $c_j = 0$ for all j such that x_j is a basic variable, and $c_j = 1$ otherwise. □

The most important result in this section states that bfs's correspond to vertices of P.

7.1.4 Theorem
Given a polyhedron $P = \{x \in \mathbb{R}^n_+ \mid Ax = b\}$, any bfs for P is a vertex of P, and vice versa.

Proof. ($\Rightarrow$) Let $x^* = (x^*_B, 0)$, with $x^*_B \geq 0$, be a bfs for P. By Lemma 7.1.3 there is a cost vector c such that for all $x \in P$, x^* is the unique vector such that $c^\top x^* \leq c^\top x$ for all $x \in P$. Thus, the hyperplane $c^\top(x - x^*) = 0$ intersects P in exactly one point, namely x^*. Hence x^* is a vertex of P. ($\Leftarrow$) Assume now that x^* is a vertex of P and suppose, to get a contradiction, that it is not a bfs. Consider the columns A_j of A such that $j \in \beta = \{j \leq n \mid x^*_j > 0\}$. If A_j are linearly independent, we have immediately that x^* a bfs for P, which contradicts the hypothesis. Thus, suppose A_j are linearly dependent. This means that there are scalars d_j, not all zero, such that $\sum_{j \in \beta} d_j A_j = 0$. On the other hand, since x^* satisfies $Ax = b$, we have $\sum_{j \in \beta} x_j A_j = b$. Thus, for all $\varepsilon > 0$, we obtain $\sum_{j \in \beta}(x_j - \varepsilon d_j)A_j = b$ and $\sum_{j \in \beta}(x_j + \varepsilon d_j)A_j = b$. Let x' have components $x_j - \varepsilon d_j$ for all $j \in \beta$ and 0 otherwise, and x'' have components $x_j + \varepsilon d_j$ for all $j \in \beta$ and 0 otherwise. By choosing a small enough ε we can ensure that $x', x'' \geq 0$. Since $Ax' = Ax'' = b$ by construction, both x' and x'' are in P. Thus, $x^* = \frac{1}{2}x' + \frac{1}{2}x''$ is a strict convex combination of two points of P, hence by Lemma 6.1.5 it cannot be a vertex of P, contradicting the hypothesis. □

We remark that Thm. 7.1.4 does *not* imply that there is a bijection between vertices and bfs. In fact, multiple bfs may correspond to the same vertex, as Example 7.1.5 shows.

7.1.5 Example
Consider the trivial LP $\min\{x_1 \mid x_1 + x_2 \leq 0 \wedge x_1, x_2 \geq 0\}$. The feasible polyhedron $P \equiv \{(x_1, x_2) \geq 0 \mid x_1 + x_2 \leq 0\} = \{0\}$ consists of a single vertex at the origin. On the other hand, we can partition the constraint matrix $(1\ 1)$ so that the basic column is indexed by x_1 (and the nonbasic by x_2) or, conversely,

so that the basic column is indexed by x_2 (and the nonbasic by x_1): both are bfs of the problem, both correspond to a feasible vertex, but since there is only one vertex, obviously both must correspond to the origin.

7.1.6 Remark
Geometrically, multiple bfs corresponding to a single vertex is a phenomenon known as degeneracy, which corresponds to more than n constraint hyperplanes passing through a single point (where n is the number of variables). Degenerate vertices correspond to bfs having strictly less than m nonzero components (where m is the number of constraints).

In the case of our example, $n = 2$ but there are three lines passing through the (only) feasible vertex $(0, 0)$: $x_2 = 0$, $x_1 = 0$ and $x_1 + x_2 = 0$.

The event of more than n hyperplanes intersecting in a single point has probability zero if their coefficients are sampled from a uniform distribution: this might lead the reader to believe that degeneracy is a rare occurrence. This, however, is not the case: many formulations put together by humans are degenerate. This is sometimes explained by noticing that LPs are used to model technological processes conceived by humans, which — according to a human predisposition to simplicity — involve a fair amount of symmetry, which, in turn, is likely to generate degeneracy.

7.1.7 Exercise
"A bfs is degenerate iff it has fewer than m nonzero components": is this statement true or false?

> **Answer**. The statement is false: the polyhedron $P \equiv \{(x_1, x_2) \geq 0 \mid x_1 + x_2 \leq 1\}$ has three vertices: $(0, 0), (0, 1), (1, 0)$, each of which corresponds to a distinct bfs (intersection of $x_1 = 0$ and $x_2 = 0$, intersection of $x_2 = 0$ and $x_1 + x_2 = 1$, and intersection of $x_1 = 0$ and $x_1 + x_2 = 1$ respectively). So none of the bfs is degenerate, and yet $(0, 0)$ has no nonzero components (whereas $m = 1$).

7.1.8 Exercise
Prove the statement "degenerate vertices correspond to bfs having strictly less than m nonzero components" in Rem. 7.1.6.

> **Answer**. Suppose vertex v is degenerate: then by definition it must correspond to at least two distinct bfs, say x, y. Suppose both have exactly m nonzero components. Since both bfs correspond to the same vertex, we must have $x = y$. However, since the bfs are distinct, their basic and nonbasic column partitions must be different. Let i be the smallest index of a column that is basic in x and nonbasic in y. Since y_i is nonbasic, by definition we have $y_i = 0$; however, because $x = y$, we must also have $x_i = 0$. Since only basic columns of bfs may have nonzero values, and there are at most m of them, we found a basic column of x which has zero value, which implies that at most $m - 1$ may be nonzero. The claim follows.

By the following theorem, in order to solve an LP all we have to do is compute the objective function at each vertex of the feasible polyhedron.

7.1.9 Theorem
Consider Eq. (7.2). If P is non-empty, closed and bounded, there is at least one bfs which solves the problem. Furthermore, if x', x'' are two distinct solutions, any convex combination of x', x'' is also a solution.

Proof. Since the polyhedron P is closed and bounded, the function $f(x) = c^\top x$ attains a minimum on P, say at x' (and by convexity of P, x' is the global minimum). Since $x' \in P$, x' is a convex combination of the vertices $v_1, \ldots, v_p$ of P, say $x' = \sum_{i=1}^{p} \lambda_i v_i$ with $\lambda_i \geq 0$ for all $i \leq p$ and $\sum_{i=1}^{p} \lambda_i = 1$. Thus,

$$c^\top x' = \sum_{i=1}^{p} \lambda_i c^\top v_i.$$

Let $j \leq p$ be such that $c^\top v_j \leq c^\top v_i$ for all $i \leq p$. Then

$$\sum_{i=1}^{p} \lambda_i c^\top v_i \geq c^\top v_j \sum_{i=1}^{p} \lambda_i = c^\top v_j,$$

whence $c^\top x' \geq c^\top v_j$. But since $c^\top x'$ is minimal, we have $c^\top x' = c^\top v_j$, which implies that there exists a vertex of P, which by Thm. 7.1.4 corresponds to a bfs, having mimum objective function value; in other words, there is a bfs which solves the problem. For the second part of the theorem, consider a convex combination $x = \lambda x' + (1-\lambda)x''$ with $\lambda \geq 0$. We have $c^\top x = \lambda c^\top x' + (1-\lambda)c^\top x''$. Since x', x'' are solutions, we have $c^\top x' = c^\top x''$, and hence

$$c^\top x = c^\top x'(\lambda + (1-\lambda)) = c^\top x',$$

which shows that x is also a solution. □

Thm. 7.1.9 states that P should be closed and bounded, so it requires that P should in fact be a polytope (polyhedra may be unbounded). In fact, this theorem can be modified [43, Prop. 2.4.2] to apply to unbounded polyhedra by keeping track of the unboundedness directions (also called extreme rays).

7.1.2 Moving from vertex to vertex

Since vertices of a polyhedron correspond to bfs by Thm. 7.1.4, and a bfs has at most m nonzero components out of n, there are at worst (nm) vertices in a given polyhedron. Thus, unfortunately, polyhedra may possess a number of vertices which is exponential in the size of the instance, so the above approach is not practical.

However, it is possible to look for the optimal vertex by moving from vertex to vertex along the edges of the polyhedron, following the direction of decreasing cost, and checking at each vertex if an optimality condition is satisfied: this is a summary description of the *simplex method*. In order to fully describe it, we need an efficient way of moving along a path of edges and vertices.

Consider a bfs x^* for $P = \{x \geq 0 \mid Ax = b\}$ and let β be the set of indices of the basic variables. Let A_i be the i-th column of A. We have:

$$\sum_{i \in \beta} x_i^* A_i = b. \tag{7.3}$$

Now, fix a $j \notin \beta$; A_j is a linear combination of the A_i in the basis. Thus, there exist multipliers x_{ij} such that for all $j \notin \beta$,

$$\sum_{i \in \beta} x_{ij} A_i = A_j. \tag{7.4}$$

Multiply Eq. (7.4) by a scalar θ and subtract it from Eq. 7.3 to get:

$$\sum_{i \in \beta} (x_i^* - \theta x_{ij}) A_i + \theta A_j = b. \tag{7.5}$$

Now suppose we want to move (by increasing θ from its initial value 0) from the current bfs to another point inside the feasible region. In order to move to a feasible point, we need $x_i^* - \theta x_{ij} \geq 0$ for all $i \in \beta$. If $x_{ij} \leq 0$ for all i, then θ can grow indefinitely (this means the polyhedron P is unbounded). Assuming a bounded polyhedron, we have a bounded $\theta > 0$. This means

$$\theta = \min_{\substack{i \in \beta \\ x_{ij} > 0}} \frac{x_i^*}{x_{ij}}. \tag{7.6}$$

If $\theta = 0$ then there is $i \in \beta$ such that $x_i^* = 0$. This means that the bfs x^* is degenerate (see Example 7.1.5 and Remark 7.1.6). We assume a nondegenerate x^* in this summary treatment. Let $k \in \beta$ be the index

minimizing θ in the expression above. The coefficient of A_k in Eq. 7.5 becomes 0, whereas the coefficient of A_j is nonzero.

7.1.10 Proposition
Let x^* be a bfs, $j \notin \beta$, x_{ij} be as in Eq. (7.5) for all $i \in \beta$ and θ be as in Eq. (7.6). The point $x' = (x'_1, \ldots, x'_n)$ defined by

$$x'_i = \begin{cases} x^*_i - \theta x_{ij} & \forall i \in \beta \setminus \{k\} \\ \theta & i = k \\ 0 & \textit{otherwise} \end{cases}$$

is a bfs.

Proof. First notice that x' is a feasible solution by construction. Secondly, for all $i \notin \beta, i \neq$ we have $x'_i = x^*_i = 0$ and for all $i \in \beta, i \neq k$ we have $x'_i \geq 0$. By the definition of x', we have $x'_j \geq 0$ and $x'_k = 0$. Thus, if we define $\beta' = \beta \backslash \{k\} \cup \{j\}$, it only remains to be shown that $\{x_i \mid i \in \beta'\}$ is a set of basic variables for A. In other words, if we partition the columns of A according to the index set β', obtaining $A = (B'|N')$, we have to show that B' is nonsingular. Notice $(B|N) = A = (B'|N')$ implies $B^{-1}A = (I|B^{-1}N) = (B^{-1}B'|B^{-1}N')$ (here the equality sign is considered "modulo the column order": i.e. the two matrices are equal when considered as sets of columns). Eq. (7.4) can be stated in matrix form as $BX^\top = N$ where X is the $(n-m) \times m$ matrix whose (p,q)-th entry is x_{pq}, thus $B^{-1}N = X^\top$. By construction of B' we have $B' = B \backslash \{A_k\} \cup \{A_j\}$. Thus, $B^{-1}B' = (e_1, \ldots, e_{k-1}, B^{-1}A_j, e_{k+1}, \ldots, e_n)$, where e_i is the vector with i-th component set to 1 and the rest set to zero, and $B^{-1}A_j$ is the j-th column of X, i.e. $(x_{1j}, \ldots, x_{nj})^\top$. Hence, $|\det(B^{-1}B')| = |x_{kj}| > 0$, since $\theta > 0$ implies $x_{kj} \neq 0$. Thus, $\det B' \neq 0$ and B' is nonsingular. □

Intuitively, Thm. 7.1.10 says that any column A_j outside the basis can replace a column A_k in the basis if we choose k as the index that minimizes θ in Eq. 7.6. Notice that this implies that the column A_k exiting the basis is always among those columns i in Eq. 7.4 which have multiplier $x_{ij} > 0$. In other words, a column A_j can replace column A_k in the basis only if the linear dependence of A_j on A_k is nonzero (A_k is a "nonzero component" of A_j). Informally, we say that x_j enters the basis and x_k leaves the basis.

In order to formalize this process algorithmically, we need to find the value of the multipliers x_{ij}. By the proof of Prop. 7.1.10, x_{ij} is the (i,j)-th component of a matrix X satisfying $X^\top = B^{-1}N$. Knowing x_{ij} makes it possible to calculate β' and the corresponding partition B', N' of the columns of A. The new bfs x' can be obtained as $(B')^{-1}b$, since $Ax' = b$ implies $(B'|N')x' = b$ and hence $Ix' = (B')^{-1}b$ (recall $N'x' = 0$ since x' is a bfs).

7.1.3 Decrease direction

All we need now is a way to identify "convenient" variables to enter the basis. In other words, from a starting bfs we want to move to other bfs (with the method described above) so that the objective function value decreases. To this end, we iteratively select the variable x_j having the most negative *reduced cost* to enter the basis (the reduced costs are the coefficients of the objective function expressed in terms of the current nonbasic variables). Writing c as (c_B, c_N) according to the current basic/nonbasic partition, the reduced costs $\bar{c}^\top$ are obtained as $c^\top - c_B B^{-1}A$. The algorithm terminates when there is no negative reduced cost (i.e. no vertex adjacent to the current solution has a smaller objective function value). This criterion defines a local optimum. Since LP problems are convex, any local minimum is also a global one by Thm. 6.1.7.

7.1.4 Bland's rule

When a vertex is degenerate, an application of the simplex algorithm as stated above may cause the algorithm to cycle. This happens because a reduced cost in the objective function identifies a variable x_j to enter the basis, replacing x_k, with (degenerate) value $x'_j = \theta = 0$. This results in a new basis yielding exactly the same objective function value as before; at the next iteration, it may happen that the selected variable to enter the basis is again x_k. Thus, the current basis alternatively includes x_k and x_j without any change to the objective function value. It can be shown that the following simple rule entirely avoids these situations: whenever possible, always choose the variable having the lowest index for entering/leaving the basis [67, Thm. 3.3].

7.1.5 Simplex method in matrix form

Here we give a summary of the algebraic operations involved in a simplex algorithm step applied to a LP in standard form

$$\min\{c^\top x \mid Ax = b \wedge x \geq 0\}. \tag{7.7}$$

Suppose we are given a current bfs x ordered so that variables $x_B = (x_1, \ldots, x_m, 0, \ldots, 0)$ are basic and $x_N = (0, \ldots, 0, x_{m+1}, \ldots, x_n)$ are nonbasic. We write $A = B + N$ where B consists of the basic columns $1, \ldots m$ of A and 0 in the columns $m+1, \ldots n$, and N consists of 0 in the first m columns and then the nonbasic columns $m+1, \ldots n$ of A.

1. *Express the basic variables in terms of the nonbasic variables.* From $Ax = b$ we get $Bx_B + Nx_N = b$. Let B^{-1} be the inverse of the square submatrix of B consisting of the first m columns (i.e. the basic columns). We pre-multiply by B^{-1} to get:

$$x_B = B^{-1}b - B^{-1}Nx_N. \tag{7.8}$$

2. *Select an improving direction.* Express the objective function in terms of the nonbasic variables: $c^\top x = c_B^\top x_B + c_N^\top x_N = c_B^\top(B^{-1}b - B^{-1}Nx_N) + c_N^\top x_N$, whence:

$$c^\top x = c_B^\top B^{-1}b + \bar{c}_N^\top x_N, \tag{7.9}$$

   where $\bar{c}_N^\top = c_N^\top - c_B^\top B^{-1}N$ are the reduced costs. If all the reduced costs are nonnegative there is no nonbasic variable which yields an objective function decrease if inserted in the basis, since the variables must also be nonegative. Geometrically speaking it means that there is no adjacent vertex with a lower objective function value, which in turn, by Thm. 6.1.7, means that we are at an optimum, and the algorithm terminates. Otherwise, select an index $h \in \{m+1, \ldots n\}$ of a nonbasic variable x_h with negative reduced cost (or, as in Section 7.1.4, select the least such h). We now wish to insert x_h in the basis by increasing its value from its current value 0. Geometrically, this corresponds to moving along an edge towards an adjacent vertex.

3. *Determine the steplength.* Inserting x_h in the basis implies that we should determine an index $l \leq m$ of a basic variable which exits the basis (thereby taking value 0). Let $\bar{b} = B^{-1}b$, and let $\bar{a}_{ij}$ be the (i,j)-th component of $B^{-1}N$ (notice $\bar{a}_{ij} = 0$ for $j \leq m$). By Eq. (7.8) we can write $x_i = \bar{b}_i - \sum_{j=m+1}^{n} \bar{a}_{ij}x_j$ for all $i \leq m$. Since we only wish to increase the value of x_h and all the other nonbasic variables will keep value 0, we can write $x_i = \bar{b}_i - \bar{a}_{ih}x_h$. At this point, increasing the value of x_h may impact on the feasibility of x_i only if it becomes negative, which can only happen if $\bar{a}_{ih}$ is positive. Thus, we get $x_h \leq \frac{\bar{b}_i}{\bar{a}_{ih}}$ for each $i \leq m$ and $\bar{a}_{ih} > 0$, and hence:

$$l = \operatorname{argmin}\{\frac{\bar{b}_i}{\bar{a}_{ih}} \mid i \leq m \wedge \bar{a}_{ih} > 0\} \tag{7.10}$$

$$x_h = \frac{\bar{b}_l}{\bar{a}_{lh}}. \tag{7.11}$$

The above procedure fails if $\bar{a}_{ih} \leq 0$ for all $i \leq m$. Geometrically, it means that x_h can be increased in value without limit: this implies that the value of the objective function becomes unbounded. In other words, the problem is unbounded.

7.1.6 Sensitivity analysis

Material from this section has been taken from [111]. We write the optimality conditions as follows:

$$\bar{b} = B^{-1}b \geq 0 \tag{7.12}$$
$$\bar{c}^\top = c^\top - c_B^\top B^{-1}A \geq 0. \tag{7.13}$$

Eq. (7.12) expresses primal feasibility and Eq. (7.13) expresses dual feasibility.

Suppose now we have a variation in b, say $b \to b + \Delta b$: Eq. (7.12) becomes $B^{-1}b \geq -B^{-1}\Delta b$. This system defines a polyhedron in Δb where the optimal basis does not change. The variable values and objective function obviously change. The variation of the objective function is $(c_B^\top B^{-1})\Delta b = y^*\Delta b$, where y^* are the optimal dual variable values: these, therefore, can be seen to measure the sensitivity of the objective function value to a small change in the constraint coefficients.

7.1.7 Simplex variants

We briefly describe some of the most popular variants of the simplex algorithm. Material in this section has been taken from [240, 111].

7.1.7.1 Revised Simplex method

The revised simplex mehod is basically a smart storage and update scheme for the data of the current iteration of the simplex algorithm. In practice, we only need to store B^{-1}, as the rest of the data can be obtained via premultiplication by B^{-1}. The disadvantages of this method reside in the high numerical instability of updating B^{-1} directly. This issue is usually addressed by storing B^{-1} in various factorized forms.

7.1.7.2 Two-phase Simplex method

If no starting bfs is available for Eq. (7.7), we can artificially look for one by solving the auxiliary LP:

$$\left.\begin{array}{ll} \min\limits_{x,y} & \mathbf{1}y \\ \text{s.t.} & Ax + yI = b \\ & x \in \mathbb{R}^n_+ \\ & y \in \mathbb{R}^m_+ \end{array}\right\}$$

where $\mathbf{1}$ is the row vector consisting of all 1's. Here, the bfs where $B = I$ is immediately evident (all x are nonbasic, all y are basic), and if Eq. (7.7) is feasible, the optimal objective function value of the auxiliary LP will be 0 with $y = 0$, yielding a bfs in the x variables only. This solution can then be used as a starting bfs for the original LP.

7.1.7.3 Dual Simplex method

Another way to deal with the absence of a starting bfs is to apply the dual simplex method. What happens in the (primal) simplex algorithm is that we maintain primal feasibility by moving from vertex

to vertex, while trying to decrease the objective function until this is no longer possible. In the dual simplex algorithm, we maintain the optimality of the objective function (in the sense that the current dual simplex objective function value is always a lower bound with respect to the primal minimum value) whilst trying to achieve feasibility. We use the same notation as in Section 7.1.5.

We start with a (possibly infeasible) basic primal solution where all the reduced costs are nonnegative. If $\bar{b} = B^{-1}b = x_B \geq 0$ the algorithm terminates: if x_B is primal feasible, then we have an optimal basis. Otherwise, the primal problem is infeasible. If $b \ngeq 0$, we select $l \leq m$ such that $\bar{b}_l < 0$. We then find $h \in \{m+1, \ldots, n\}$ such that $\bar{a}_{lh} < 0$ and a_{lh} is minimum among $\{\frac{\bar{c}_j}{|\bar{a}_{lj}|} \mid j \leq n \wedge \bar{a}_{lj} < 0\}$ (this ensures that the reduced costs will stay nonnegative), and we swap x_l with x_h in the current basis. If $\bar{a}_{lj} \geq 0$ for all $j \leq n$, then the primal problem is infeasible.

The advantage of the dual simplex method is that we can add cutting planes (valid inequalities) to the main data structure of the simplex algorithm (called *simplex tableau*) during the execution of the algorithm. A valid cut should make the current optimal solution infeasible, but since dual simplex bases are not primal feasible, this is not an issue.

7.1.8 Column generation

It may sometimes happen that the number of variables in a LP is much larger than the number of constraints. Consequently, while the basis has a manageable cardinality, the computational costs of running the simplex algorithm on the LP are huge. If there exists an efficient procedure for finding the reduced cost of minimum value, we can insert columns in the simplex tableau as the need arises, thus eliminating the need of dealing with all variables at once. The problem of determining the reduced cost of minimum value (of a variable that is not yet in the simplex tableau) is called *pricing problem*. Column generation techniques are only useful if the pricing problem can be solved efficiently. The procedure stops when the pricing problem determines a minimum reduced cost of non-negative value.

One example of application of column generation is the multicommodity network flow problem formulated so that each variable corresponds to a path on the network. There are exponentially many paths, but the pricing problem is a shortest path problem, which can be solved very efficiently.

7.2 Polytime algorithms for LP

Material for this section (except for Sect. 7.3) has been taken from [34, 112, 283, 150, 55].

L. Khachiyan showed in 1979 that finding an optimal solution to a LP problem has polynomial worst-case complexity. Even though Kachiyan's *ellipsoid algorithm* has polynomial worst-case complexity, it does not work well in practice. In 1984, Karmarkar presented another polynomial algorithm for solving LP which was claimed to have useful practical applicability. Gill et al. showed in 1985 that Karmarkar's algorithm was in fact an IPM with a log-barrier function applied to an LP. IPMs had been applied to nonlinear problems with considerable success in the late 60's [110]. This spawned considerable interest in IPMs applied to LP; so-called *barrier solvers* for LP were incorporated in most commercial software codes [145]. Barrier solvers compete well with simplex-based implementations especially on large-scale LPs. We look at these methods below.

7.3 The ellipsoid algorithm

Material for this section has been taken from [237, 160, 240]. The ellipsoid algorithm actually solves a different problem, which is called

LINEAR STRICT INEQUALITIES (LSI). Given $b \in \mathbb{Q}^m$ and a rational $m \times n$ matrix A, decide whether there is a rational vector $x \in \mathbb{Q}^n$ such that $Ax < b$.

First, we show a computational complexity equivalence between LP and LSI, by reducing the former to the latter in polytime.

7.3.1 Equivalence of LP and LSI

By the term LP we mean both a MP formulation and a problem. Formally, but rather unusually, LP "as a problem" is stated as a conditional sequence of (formal) subproblems:

LINEAR OPTIMIZATION PROBLEM (LOP). Given a vector $c \in \mathbb{Q}^n$, an $m \times n$ rational matrix A, and a vector $b \in \mathbb{Q}^m$, consider the LP formulation in Eq. (7.7), and:

1. decide whether the feasible set is empty;
2. if not, decide whether Eq. (7.7) is unbounded in the objective direction
3. if not, find the optimum and the optimal objective function value.

Namely, we check feasibility, unboundedness (two decision problems) and then optimality (an optimization problems). We must show that all of these tasks can be carried out in a polynomial number of calls to the "oracle" algorithm $\mathscr{A}$.

7.3.1.1 Reducing LOP to LI

Instead of reducing LOP to LSI directly, we show a reduction to

LINEAR INEQUALITIES (LI). Given $b \in \mathbb{Q}^m$ and a rational $m \times n$ matrix A, decide whether there is a rational vector $x \in \mathbb{Q}^n$ such that $Ax \leq b$.

7.3.1.1.1 Addressing feasibility First, we note that if $\mathscr{A}$ is an algorithm for LI, $\mathscr{A}$ also trivially solves (with just one call) the feasibility subproblem 1 of LOP: it suffices to reformulate $\{x \geq 0 \mid Ax = b\}$ to $\{x \geq 0 \mid Ax \leq b \wedge Ax \geq b\}$.

7.3.1.1.2 Instance size For boundedness and optimality, we need to introduce some notions about the maximum possible size of LOP solutions (if they exist) in function of the input data A, b, c. We define the size $\mathscr{L} = mn + \lceil \log_2 P \rceil$ of the LOP instance (A, b, c), where P is the product of all numerators and denominators occurring in the components of A, b, c ($\mathscr{L}$ is an upper bound to the storage required to write A, b, c in binary representation).

7.3.1.1.3 Bounds on bfs components By [240, Lemma 8.5], bfs consist of rational components such that the absolute values of numerators and denominators are bounded above by 2^L (while this is not the strictest possible bound, it is a valid bound). This bound arises a consequence of three facts: (i) bfs can be computed as $x_B = B^{-1}b$ (see Eq. (7.8) and set nonbasic variables x_N to zero), where B is a basis of A; (ii) components of B^{-1} can be expressed in terms of the determinants of the adjoints of B; (iii) the absolute value of the determinant of B is bounded above by the product of the absolute values of the numerators of the components of B. Thus, we know that

$$x \text{ is a bfs of Eq. (7.7)} \iff \forall j \leq n \ (0 \leq x_j \leq 2^L), \tag{7.14}$$

which implies in particular that if the instance is feasible and bounded and x^* is a bfs which solves Eq. (7.7), then $x^* \in [0, 2^L]$.

7.3.1.1.4 Addressing unboundedness This observation allows us to bound the optimal objective function $\min c^\top x$ below: since $c_j \geq -2^L$ for each $j \leq n$, $c^\top x^* \geq -n2^{2L}$. On the other hand, if the LOP instance is unbounded, there are feasible points yielding any value of the objective function, namely strictly smaller than $n2^{2L}$. This allows us to use LSI in order to decide the boundedness subproblem 2 of LOP: we apply $\mathscr{A}$ (the LI oracle) to the system $Ax \leq b \wedge Ax \geq b \wedge x \geq 0 \wedge c^\top x \leq -n2^{2L} - 1$. If the system is feasible, then the LOP is unbounded.

7.3.1.1.5 Approximating the optimal bfs If the LOP instance was not found infeasible or unbounded, we know it has an optimal bfs x^*. We first find an approximation $\hat{x}$ of x^* using *bisection search*, one of the most efficient generic algorithms in combinatorial optimization More precisely, we determine $\hat{x}$ and an integer $K \in [-2^{4L}, 2^{4L}]$ such that $K2^{2L} < c^\top \hat{x} \leq (K+1)2^{2L}$ as follows:

1. Let $a^L = -2^{2L}$, $a^U = 2^{2L}$;
2. Let $a = \frac{a^L + a^U}{2}$;
3. Solve the LI instance:
$$Ax \leq b \ \wedge \ Ax \geq b \ \wedge \ x \geq 0 \ \wedge \ c^\top x \leq a\,; \tag{7.15}$$
using the oracle $\mathscr{A}$;
4. If Eq. (7.15) is infeasible:
   - update $a^L \leftarrow a$ and repeat from Step 2;
5. If Eq. (7.15) is feasible:
   - let $\hat{x}$ be the solution of Eq. (7.15)
   - if $a^U - a^L < 2^{-2L}$ return $\hat{x}$ and stop, else $a^U \leftarrow a$ and repeat from Step 2.

We note this bisection search makes $O(\log_2(2^{2L+1}) = 2L + 1$ calls to the oracle $\mathscr{A}$.

7.3.1.1.6 Approximation precision Terminating the bisection search when the objective function approximation of $\hat{x}$ is within an $\epsilon = 2^{-2L}$ has the following significance: by [240, Lemma 8.6], if there are two bfs x, y of Eq. (7.7) and $K \in \mathbb{Z}$ such that
$$K2^{-2L} < c^\top x \leq (K+1)2^{-2L} \quad \wedge \quad K2^{-2L} < c^\top y \leq (K+1)2^{-2L}, \tag{7.16}$$
then $c^\top x = c^\top y$. Note that Eq. (7.16) states that the two rational numbers $q_1 = c^\top x$ and $q_2 = c^\top y$ are as close to each other as 2^{-2L}. We suppose $q_1 \neq q_2$ to aim at a contradiction: since these are different rational numbers with denominators bounded above by 2^L [240, Lemma 8.5], we must have $|c^\top x - c^\top y| \geq 2^{-2L}$, which contradicts Eq. (7.16).

7.3.1.1.7 Approximation rounding Since we found a point $\hat{x}$ with $a^L < c^\top \hat{x} \leq a^U$, and we know that requiring the objective function value to be equal to a^L yields an infeasible LI, by the existence of optimal bfs, there must be an optimal bfs x^* such that $a^L < c^\top x^* \leq a^U$. We shall now give an algorithm for finding x^* using $\mathscr{A}$, which works as long as there is no degeneracy (see Rem. 7.1.6).

- Let $S = \varnothing$.
- For $j \in \{1, \ldots, n\}$ repeat:
  1. Append j to S;

2. Solve the LI instance:

$$Ax \leq b \ \wedge \ Ax \geq b \ \wedge \ x \geq 0 \ \wedge \ c^\top x \leq a^U \ \wedge \ c^\top x \geq a^L \ \wedge \ \forall j \in S \, (x_j \leq 0) \, ; \tag{7.17}$$

using the oracle $\mathscr{A}$;

3. If Eq. (7.17) is infeasible then remove j from S.

- Let B be square $m \times m$ submatrix of A indexed by any subset of m columns indexed by $\bar{S} = \{1, \ldots, n\} \setminus S$
- Let $x^* = B^{-1}b$.

This algorithm is correct since it chooses basic columns from A indexed by variables which could not be set to zero in Eq. (7.17) (and hence they could not be nonbasic).

This completes the polynomial reduction from LOP to LI.

7.3.1.2 Reducing LI to LSI

By [240, Lemma 8.7], a LI instance $Ax \leq b$ is feasible if and only if the LSI instance $Ax < b + \epsilon$, where $\epsilon = 2^{-2L}$, is feasible. Let x' satisfy $Ax \leq b$; then by definition it satisfies $Ax \leq (b - \epsilon) + \epsilon$, i.e. $Ax < b + \epsilon$. Now let x^* satisfy $Ax < b + \epsilon$. If x^* also satisfies $Ax < b$ then it satisfies $Ax \leq b$ by definition, so let I be the largest set I of constraint indices of $Ax < b + \epsilon$ such that $A_I = \{a_i \mid i \in I\}$ is linearly independent. A technical argument (see [240, p. 174]) shows that the solution $\hat{x}$ to $A_I x = b_I$ also satisfies $Ax \leq b$.

The ellipsoid algorithm presentation in [268] works with closed polyhedra and hence does not need this last reduction from LI to LSI (although there is a simpler equivalence between $Ax \leq b$ and $Ax \leq b + \epsilon$ based on Farkas' lemma [268, p. 169]).

7.3.2 Solving LSIs in polytime

Let P_ε be the (open) polyhedron corresponding to the feasible set of the given system of strict linear inequalities, and let E_0 be an ellipsoid such that $P_\varepsilon \subseteq E_0$. The ellipsoid algorithm either finds a feasible solution $x^* \in P_\varepsilon$ or determines that P_ε is empty. We assume that the volume of P_ε is strictly positive if P_ε is nonempty, i.e. $P_\varepsilon \neq \emptyset \rightarrow \mathrm{Vol}(P_\varepsilon) > 0$. Let v be a number such that $\mathrm{Vol}(P_\varepsilon) > v$ if $P_\varepsilon \neq \emptyset$ and $V = \mathrm{Vol}(E_0)$, and let

$$K = \lceil 2(n+1)(\ln V - \ln v) \rceil .$$

Initially, $k = 0$.

1. Let $x^{(k)}$ be the centre of the ellipsoid E_k. If $x^{(k)} \in P_\varepsilon$, the algorithm terminates with solution $x^{(k)}$. Otherwise, there exists a violated strict constraint $A_i x < b_i$ such that $A_i x^{(k)} \geq b_i$. The hyperplane $A_i x = b_i$ slices E_k through the centre; the part which does not contain P_ε can be discarded.

2. Find the minimum volume ellipsoid E_{k+1} containing the of part of E_k containing P_ε, and let $x^{(k+1)}$ be the centre of E_{k+1}.

3. If $k \geq K$, stop: $P_\varepsilon = \varnothing$.

4. Repeat from 1.

An ellipsoid E_k with centre $x^{(k)}$ is described by $\{x \in \mathbb{R}^n \mid (x - x^{(k)})^\top D_k^{-1}(x - x^{(k)}) \leq 1\}$ where D_k is an $n \times n$ PSD matrix. We compute $x^{(k+1)}$ and D_{k+1} as follows:

$$\begin{aligned} \Gamma_k &= \frac{D_k A_i}{\sqrt{{A_i}^\top D_k A_i}} \\ x^{(k+1)} &= x^{(k)} + \frac{\Gamma_k}{n+1} \\ D_{k+1} &= \frac{n^2}{n^2-1}\left(D_k - \frac{2\Gamma_k {\Gamma_k}^\top}{n+1}\right). \end{aligned}$$

It turns out that E_{k+1} defined by the matrix D_{k+1} contains the part of E_k containing P_ε, and furthermore

$$\mathrm{Vol}(E_{k+1}) \leq e^{-\frac{1}{2(n+1)}} \mathrm{Vol}(E_k). \tag{7.18}$$

The volume of E_K is necessarily smaller than v: by Eq. (7.18) we have

$$\mathrm{Vol}(E_K) \leq e^{-\frac{K}{2(n+1)}} \mathrm{Vol}(E_0) \leq V e^{-\frac{K}{2(n+1)}} \leq V e^{-\ln(V/v)} = v,$$

hence by the assumption on v we must have $P_\varepsilon = \emptyset$ if $k \geq K$. Since it can be shown that K is polynomial in the size of the instance (which depends on the number of variables, the number of constraints as well as the bits needed to store A and b), the ellipsoid algorithm is a polytime algorithm which can be used to solve LPs.

One last point to be raised is that in Eq. (7.18) we rely on an irrational computation, which is never precise on a computer. It can be shown that computations carried out to a certain amount of accuracy can still decide whether P_ε is empty or not (see [240, §8.7.4]).

7.4 Karmarkar's algorithm

Karmarkar's algorithm addresses all LPs with constraints $Ax = 0, x \geq 0$ and $\mathbf{1}^\top x = 1$, and known *a priori* to have optimal objective function value zero. The constraints evidently imply that $\bar{x} = \mathbf{1}/n$ is an initial feasible solution (where $\mathbf{1} = (1, \ldots, 1)^\top$.

It can be shown that any LP can be reduced to this special form.

- If no constraint $\sum_j x_j = 1$ is present in the given LP instance, one can be added as follows:
  - adjoin a constraint $\sum_j x_j \leq M = O(2^L)$ (where L is the size of the instance) to the formulation: this will not change optimality properties, though it might turn unboundedness into infeasibility;
  - add a new (slack) variable x_{n+1} and rewrite the new constraint as $\sum_{j \leq n+1} x_j = M$;
  - scale all variables so that $x \leftarrow \frac{1}{M}x$, and accordingly scale the constraint RHSs $b \leftarrow \frac{1}{M}b$, yielding $\sum_j x_j = 1$.
- The system $Ax = b$ appearing in Eq. (7.7) can be written as

  $$\forall i \leq m \quad \sum_{j \leq n} a_{ij} x_j - b \sum_{j \leq n} x_j = 0$$

  since we have the constraint $\mathbf{1}^\top x = 1$; this yields

  $$\forall i \leq m \quad \sum_{j \leq n} (a_{ij} - b_i) x_j = 0,$$

  which is homogeneous in x, as desired.

- If the optimal objective function value is known to be $\bar{c} \neq 0$, it can be "homogenized" in the same way as $Ax = b$, namely optimize the objective $c^\top x - \bar{c}$ (adding constants does not change the optima) and rewrite it as $c^\top x - \bar{c}\sum_j x_j$ based on the constraint $\mathbf{1}^\top x = 1$.
- If the optimal objective function value is unknown, proceed with a bisection search (see Sect. 7.3.1).

Let $X = \text{diag}(\bar{x})$ and $B = \begin{pmatrix} AX \\ \mathbf{1} \end{pmatrix}$. The algorithm is as follows:

1. Project Xc into the null space of B: $c^* = (I - B^\top(BB^\top)^{-1}B)Xc$.
2. Normalize the descent direction: $d = \gamma \frac{c^*}{||c^*||}$. If $c^* = 0$ terminate with optimal solution x^*.
3. Move in projected space: $y = e/n - sd$ where s is a fixed step size.
4. Project back into x-space: $\bar{x} \leftarrow \frac{Xy}{e^\top Xy}$.
5. Repeat from 1.

Taking $\gamma = \frac{1}{\sqrt{n(n-1)}}$ and $s = \frac{1}{4}$ guarantees that the algorithm has polynomial complexity.

Let us look at the first iteration of Karmarkar's algorithm in more detail. The initial point $\bar{x}$ is taken to be the feasible solution $\bar{x} = \mathbf{1}/n$; notice $\mathbf{1}/n$ is also the centre of the simplex $\mathbf{1}^\top x = 1 \wedge x \geq 0$. Let $S_r = \{x \in \mathbb{R}^n \mid ||x - x^*|| \leq r \wedge \mathbf{1}^\top x = 1 \wedge x \geq 0\}$ be the largest sphere that can be inscribed in the simplex, and let S_R be the smallest concentric sphere that circumscribes the simplex. It can be shown that $R/r = n - 1$. Let x_r be the minimum of $c^\top x$ on $F_r = S_r \cap \{x \mid Ax = 0\}$ and x_R the minimum on $F_R = S_R \cap \{x \mid Ax = 0\}$; let $f_r = c^\top x_r$, $f_R = c^\top x_R$ and $\bar{f} = c^\top \bar{x}$. By the linearity of the objective function, we have $\frac{\bar{f} - f_R}{\bar{f} - f_r} = n - 1$. Since F_R contains the feasible region of the problem, $f_R \leq f^* = 0$, and so $(n-1)\bar{f} - (n-1)f_r = \bar{f} - f_R \geq \bar{f} - f^* = \bar{f}$, whence

$$f_r \leq \frac{n-2}{n-1}\bar{f} < e^{-\frac{1}{n-1}}\bar{f}. \tag{7.19}$$

Finally, we update $\bar{x}$ with x_r and repeat the process. If this reduction in objective function value could be attained at each iteration, $O(nL)$ iterations would be required to get within 2^{-L} tolerance from $f^* = 0$, and the algorithm would be polynomial. The only issue lies in the fact that after the first iteration the updated current point x_r is not the centre of the simplex anymore. Thus, we use a projective transformation (step 4 of the algorithm) to "re-shape" the problem so that the updated current point is again at the centre of the simplex. The linearity of the objective function is not preserved by the transformation, so a linear approximation is used (step 1 of the algorithm). Karmarkar showed that Eq. (7.19) holds for the modified objective function too, thus proving correctness and polytime complexity.

7.5 Interior point methods

Karmarkar's algorithm turns out to be equivalent to an IPM applied to an LP in standard form with orthant constraints $x \geq 0$ reformulated by means of log-barrier penalties on the objective function [118]. We consider the LP in Eq. (7.7), and reformulate it as follows:

$$\left.\begin{array}{ll} \min_x & c^\top x - \beta \sum_{j=1}^{n} \ln x_j \\ \text{s.t.} & Ax = b, \end{array}\right\} \tag{7.20}$$

where $\beta > 0$ is a parameter. Notice that since $-\ln x_j$ tends to ∞ as $x_j \to 0^+$, minimizing the objective function of Eq. (7.20) automatically implies that $x > 0$. Furthermore, as β decreases towards 0, Eq. (7.20)

describes a continuous family of NLP formulations converging to Eq. (7.7). This suggests a solution method based on solving a discrete sequence of convex problems

$$\left.\begin{array}{ll} \min_{x(\beta)} & c^\top x(\beta) - \beta \sum_{j=1}^{n} \ln x_j(\beta) \\ \text{s.t.} & Ax(\beta) = b \end{array}\right\} \tag{7.21}$$

for $\beta = \beta_1, \ldots, \beta_k, \ldots$. We expect the solutions $x^*(\beta_k)$ of Eq. (7.21) to tend to x^* as $k \to \infty$, where x^* is the solution of Eq. (7.7). The set $\{x^*(\beta) \mid \beta > 0\}$ is a path in $\mathbb{R}^n$ called the *central path*.

7.5.1 Primal-Dual feasible points

We show that each point $x(\beta)$ on the central path yields a dual feasible point. The Lagrangian $L_1(x, \lambda, \nu)$ for Eq. (7.7) (where λ are the dual variables for the constraints $-x \leq 0$ and ν for $Ax = b$) is:

$$L_1(x, \lambda, \nu) = c^\top x - \sum_{j=1}^{n} \lambda_j x_j + \nu(Ax - b), \tag{7.22}$$

while the Lagrangian $L_2(x, \nu)$ for Problem (7.21) is:

$$L_2(x, \nu) = c^\top x(\beta) - \beta \sum_{j=1}^{n} \ln x_j(\beta) + \nu(Ax - b).$$

Deriving the KKT condition (6.6) from L_1 we get:

$$\forall j \leq n \ (c_j - \lambda_j + \nu A^j = 0),$$

where A^j is the j-th column of A. From L_2 we get:

$$\forall j \leq n \ (c_j - \frac{\beta}{x_j} + \nu A^j = 0).$$

Therefore, by letting

$$\lambda_j = \frac{\beta}{x_j} \tag{7.23}$$

for all $j \leq n$, we show that each point $x(\beta)$ on the central path gives rise to a dual feasible point (λ, ν) for the dual of Eq. (7.7):

$$\left.\begin{array}{lrcl} \max\limits_{\nu} & b^\top \nu & & \\ \text{s.t.} & A^\top \nu + \lambda & = & c \\ & \lambda & \geq & 0. \end{array}\right\} \tag{7.24}$$

Notice that the dual Problem (7.24) has been derived from Eq. (6.16) by setting $\nu = -\mu$ and using λ as slack variables for the inequalities in Eq. (6.16). Since λ, ν also depend on β, we indicate them by $\lambda(\beta), \nu(\beta)$. That $(\lambda(\beta), \nu(\beta))$ is dual feasible in Eq. (7.24) follows because $(x(\beta), \lambda(\beta), \nu(\beta))$ satisfies Eq. (6.6) as shown above. Now, notice that by definition $\lambda_j(\beta) = \frac{\beta}{x_j(\beta)}$ implies

$$\forall j \leq n \ (\lambda_j(\beta) x_j(\beta) = \beta) \tag{7.25}$$

which means that, as β converges to 0, the conditions (7.25) above imply the KKT complementarity conditions (6.8). Since $\lambda(\beta) \geq 0$ by definition Eq. (7.23), we have that $(x^*, \lambda^*, \nu^*) = (x(0), \lambda(0), \mu(0))$ is an optimal primal-dual pair solving Eq. (7.7) and Eq. (7.24).

7.5.2 Optimal partitions

LPs may have more than one optimal solution. In such cases, all optimal solutions are on a single (non full-dimensional) face of the polyhedron. The simplex method would then find more than one optmial bfs. The analysis of the IPM sketched above provides a unique characterization of the optimal solutions even when there is no unique optimal solution. We show that the central path converges to a strictly complementary optimal solution, i.e. an optimal solution for which

$$\begin{aligned} (x^*)^\top \lambda^* &= 0 \\ x^* + \lambda^* &> 0. \end{aligned}$$

These solutions are used to construct the *optimal partition*,, i.e. a partition of the n solution components in two index sets B, N such that:

$$\begin{aligned} B &= \{j \leq n \mid x_i^* > 0\} \\ N &= \{j \leq n \mid \lambda_i^* > 0\}. \end{aligned}$$

The partition obtained in this way is unique, and does not depend on the strictly complementary optimal solution used to define it. Optimal partitions provide a unique and well-defined characterization of optimal faces.

In the rest of this section we write $x = x(\beta)$ and $\lambda = \lambda(\beta)$ to simplify notation. The proof of the following theorem was taken from [150].

7.5.1 Theorem (Goldman-Tucker, 1956)
With the same notation of this section, the limit point (x^*, λ^*) of the primal-dual central path $(x(\beta), \lambda(\beta))$ is a strictly complementary primal-dual optimal solution for Eq. (7.7).

Proof. By Eq. (7.25) we have that $x^\top \lambda = n\beta$, thus $(x^*)^\top \lambda^*$ converges to zero as $\beta \to 0$. Now, both x^* and x are primal feasible, so $Ax^* = Ax = b$, hence $A(x^* - x) = 0$. Furthermore, since both λ^* and λ are dual feasible, we have $\nu^* A + \lambda^* = \nu A + \lambda = c$, i.e. $(\nu - \nu^*)A = \lambda^* - \lambda$. In other words, $x^* - x$ is in the null space of A and $\lambda^* - \lambda$ is in the range of $A^\top$: thus, the two vectors are orthogonal. Hence we have:

$$0 = (x^* - x)^\top (\lambda^* - \lambda) = (x^*)^\top \lambda^* + x^\top \lambda - (x^*)^\top \lambda - x^\top \lambda^*,$$

and thus

$$(x^*)^\top \lambda + x^\top \lambda^* = n\beta.$$

Dividing throughout by $\beta = x_j \lambda_j$ we obtain:

$$\sum_{j=1}^{n} \left(\frac{x_j^*}{x_j} + \frac{\lambda_j^*}{\lambda_j} \right) = n.$$

Since

$$\lim_{\beta \to 0} \frac{x_j^*}{x_j} = \begin{cases} 1 & \text{if } x_i^* > 0 \\ 0 & \text{otherwise} \end{cases}$$

and similarly for the λ component, for each $j \leq n$ exactly one of the two components of the pair (x_j^*, λ_j^*) is zero and the other is positive. □

7.5.3 A simple IPM for LP

The prototypical IPM for LP is as follows:

1. Consider an initial point $x(\beta_0)$ feasible in Eq. 7.21, a parameter $\alpha < 1$ and a tolerance $\varepsilon > 0$. Let $k = 0$.
2. Solve Eq. (7.21) with initial point $x(\beta_k)$, to get a solution x^*.
3. If $n\beta_k < \varepsilon$, stop with solution x^*.
4. Update $\beta_{k+1} = \alpha\beta_k$, $x(\beta_{k+1}) = x^*$ and $k \leftarrow k + 1$.
5. Go to Step 2.

By Eq. (7.22), $L_1(x, \lambda, \nu) = c^\top x - n\beta_k$, which means that the duality gap is $n\beta_k$. This implies that $x(\beta_k)$ is never more than $n\beta_k$-suboptimal. This is the basis of the termination condition in Step 3. Each subproblem in Step 2 can be solved by using Newton's method.

7.5.4 The Newton step

In general, the Newton descent direction d for an unconstrained problem $\min f(x)$ at a point $\bar{x}$ is given by:

$$d = -(\nabla^2 f(\bar{x}))^{-1} \nabla f(\bar{x}). \tag{7.26}$$

If $\nabla^2 f(\bar{x})$ is PSD, we obtain

$$(\nabla f(\bar{x}))^\top d = -(\nabla f(\bar{x}))^\top (\nabla^2 f(\bar{x}))^{-1} \nabla f(\bar{x}) < 0,$$

so d is a descent direction. In this case, we need to find a feasible descent direction such that $Ad = 0$. Thus, we need to solve the system

$$\begin{pmatrix} \nabla^2 f(\bar{x}) & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} d \\ \nu^\top \end{pmatrix} = \begin{pmatrix} -\nabla f(\bar{x}) \\ 0 \end{pmatrix},$$

for (d, ν), where ν are the dual variables associated with the equality constraints $Ax = b$. Step 4 in the IPM of Section 7.5.3 becomes $x(\beta_{k+1}) = x(\beta_k) + \gamma d$, where γ is the result of a line search (see Rem. 9.1.1), for example

$$\gamma = \text{argmin}_{s \geq 0} f(\bar{x} + sd). \tag{7.27}$$

Notice that feasibility with respect to $Ax = b$ is automatically enforced because $\bar{x}$ is feasible and d is a feasible direction.

Chapter 8

Mixed-Integer Linear Programming

This chapter is devoted to some basic notions arising in MILP. We discuss total unimodularity, valid cuts, the Branch-and-Bound (BB) algorithm, and Lagrangean relaxation.

8.1 Total unimodularity

The continuous relaxation of a MILP is a MILP formulation where the integrality constraints have been dropped: in other words, an LP. In Sect. 7.1.1 and 7.1.5 we characterized LP solutions as bfs, which encode the combinatorics and geometry of an intersection of at least m hyperplanes in $\mathbb{R}^m$.

The question we try and answer in this section is: when does a solution of the continuous relaxation of a MILP automatically satisfy the integrality constraints too? In other words, when can we solve a MILP by simply solving an LP? The question is important because, as we noted above (see Sect. 5.2) MILP is **NP**-hard, whereas we can solve LPs in polytime (see Sect. 7.2): and this is a worst-case analysis which applies to practical computation (see Sect. 2.2.5).

As mentioned in Sect. 7.1.5, if we consider the continuous relaxation of a MILP as an LP in standard form (7.7), a bfs x^* corresponds to a basis B of the constraint matrix A appearing in the linear system $Ax = b$. The values of the basic variables in x^* are computed as $x^*_B = B^{-1}b$, whereas the nonbasic variables have value zero. So the question becomes: what are the conditions on a square nonsingular matrix B and on a vector b such that $B^{-1}b$ is an integer vector?

We first note that any MILP with rational input (A, b, c) can be reformulated to a MILP with integer input by simply rescaling by the minimum common multiple (mcm) of all the denominators (a better rescaling can be carried out by considering each constraint in turn). So we shall assume in the rest of this chapter that A, b, c all have integer components.

8.1.1 Definition
An $m \times n$ integral matrix A such that each $m \times m$ invertible square submatrix has determinant ± 1 is called unimodular.

8.1.2 Proposition
If A is unimodular, the vertices of the polyhedron $P_s = \{x \geq 0 \mid Ax = b\}$ all have integer components.

8.1.3 Exercise
Prove Prop. 8.1.2.

Answer. Each vertex of P_s corresponds to one or more bfs, so it suffices to prove the result for bfs. Since each bfs x^*_B corresponds to a square invertible submatrix B of A, and is obtained as $x^*_B = B^{-1}b$, since $|B^{-1}| = \frac{1}{|B|} = \pm 1$,

and b is integer, it follows that x^*_B is also integer.

Proving unimodularity is hard (computationally) for any instance, and even harder (theoretically) for problems. We now consider a stronger condition.

8.1.4 Definition
An $m \times n$ integral matrix A where all square submatrices of any size have determinant in $\{0, 1, -1\}$ is called totally unimodular (TUM).

8.1.5 Example
The identity matrix is TUM: square submatrices on the diagonal have determinant 1, while off-diagonal square submatrices have determinant zero.

The following propositions give some characterizations of TUM matrices which can be used to prove the TUM property for some interesting problems.

8.1.6 Proposition
A matrix A is TUM iff $A^\top$ is TUM.

8.1.7 Exercise
Prove Prop. 8.1.6.

Answer. Since for any square matrix B we have $|B| = |B^\top|$, the determinants of all the square invertible submatrices of A remain invariant under transposition of A.

8.1.8 Proposition
If a matrix is TUM, then its entries can only be $\{0, 1, -1\}$.

8.1.9 Exercise
Prove Prop. 8.1.8.

Answer. Since components of any matrix are 1×1 square submatrices, their determinants must be $\{0, 1, -1\}$ because of the TUM property.

8.1.10 Proposition
The $m \times n$ matrix A is TUM iff the matrix (A, I), obtained by appending an $m \times m$ identity to A, is TUM.

Proof. Let B be a square submatrix of (A, I): if it is wholly contained in A or in I, then its determinant is $\{0, 1, -1\}$ because A is TUM. Otherwise, by permuting its rows, we can write B as follows:

$$B = \left(\begin{array}{c|c} A' & 0 \\ \hline A'' & I_\ell \end{array}\right),$$

where A' is a $k \times k$ submatrix of A and A'' is an $\ell \times k$ submatrix of A. By elementary linear algebra, $|B| = |A'| \in \{0, 1, -1\}$ since A' is a square submatrix of a TUM matrix. □

We now give a sufficient condition for checking the TUM property.

8.1.11 Proposition
An $m \times n$ matrix A is TUM if: (a) for all $i \le m$, $j \le n$ we have $a_{ij} \in \{0, 1, -1\}$; (b) each column of A contains at most two nonzero coefficients; (c) there is a partition R_1, R_2 of the set of rows such that for all columns j having exactly two nonzero coefficients, we have $\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} = 0$.

Proof. Suppose that conditions (a), (b), (c) hold but A is not TUM. Let B be the smallest square submatrix of A such that $\det(B) \notin \{0, 1, -1\}$. B cannot contain a column with just one nonzero entry, otherwise B would not be minimal (just eliminate the row and column of B containing that single nonzero entry). Hence B must contain two nonzero entries in every column. By condition (c), adding the rows in R_1 to those in R_2 gives the zero vector, and hence $\det(B) = 0$, contradicting the hypothesis. □

8.1.12 Theorem
If A is TUM, the vertices of the polyhedron $P_c = \{x \geq 0 \mid Ax \geq b\}$ all have integer components.

Proof. Let x^* be a vertex of P_c, and $s^* = Ax^* - b$. Then (x^*, s^*) is a vertex of the standardized polyhedron $\bar{P}_c = \{(x, s) \geq 0 \mid Ax - s = b\}$: suppose it were not, then by Lemma 6.1.5 there would be two distinct points (x_1, s_1), (x_2, s_2) in $\bar{P}_c$ such that (x^*, s^*) is a strict convex combination of (x_1, s_1), (x_2, s_2), i.e. there would be a $\lambda \in (0, 1)$ such that $(x^*, s^*) = \lambda(x_1, s_1) + (1 - \lambda)(x_2, s_2)$. Since $s_1 = Ax_1 - b \geq 0$ and $s_2 = Ax_2 - b \geq 0$, both x_1 and x_2 are in P_c, and thus $x^* = \lambda x_1 + (1 - \lambda)x_2$ is not a vertex of P_c since $0 < \lambda < 1$, which is a contradiction. Now, since A is TUM, $(A, -I)$ is unimodular by Prop. 8.1.10, and (x^*, s^*) is an integer vector. □

8.1.13 Example
The transportation formulation Eq. (2.8) and network flow formulation Eq. (2.9) both have TUM constraint matrices. Solving them using the simplex method yields an integer solution vector.

8.1.14 Exercise
Prove that network flow constraints (see Eq. (2.9)) yield a TUM constraint matrix.

Answer. *Hint:* apply Prop. 8.1.11 with $R_2 = \varnothing$.

8.1.15 Exercise
Prove that transportation constraints (see Eq. (2.8)) yield a TUM constraint matrix.

8.2 Cutting planes

The convex hull of the feasible region of any MILP is a polyhedral set. Moreover, optimizing over the convex hull yields the same optima as optimizing over the mixed-integer feasible region of the MILP. Therefore, potentially, any MILP could be reformulated to an LP. Let me refrain the reader from jumping to the conclusion that the complexity of MILP (**NP**-hard) is the same as that of LP (**P**): there are obstacles of description and size. First, in order to even cast this equivalent LP formulation, we need to explicitly know all integer vectors in the feasible region of the MILP, including the optimal one: so already writing this LP formulation is as hard as solving the MILP. Moreover, there might be exponentially (or even infinitely) many feasible vectors to a given MILP instance, yielding an exponentially large description of the convex hull. On the other hand, finding some inequalities belonging (or just close) to the convex hull improves the performance of MILP solution methods, which is the subject of this section.

Consider a MILP $\min\{c^\top x \mid x \in \mathbb{Z}^n \cap P\}$ where $P = \{x \geq 0 \mid Ax \geq b\}$. Let $\bar{P}$ be the convex hull of its integer feasible solutions (thus $\min\{c^\top x \mid x \in \bar{P}\}$ has the same solution as the original MILP). We are interested in finding linear constraints defining the facets of $\bar{P}$. The polyhedral analysis approach is usually expressed as follows: given an integral vector set $X \subseteq \mathbb{Z}^n$ and a valid inequality $h^\top x \leq d$ for X, show that the inequality is a facet of $\text{conv}(X)$. We present two approaches below.

1. Find n points $x_1, \ldots, x_n \in X$ such that $h^\top x_i = d \forall i \leq n$ and show that these points are affinely independent (i.e. that the $n - 1$ directions $x_2 - x_1, \ldots, x_n - x_1$ are linearly independent).

2. Select $t \geq n$ points $x_1, \ldots, x_t$ satisfying the inequality. Suppose that all these points are on a generic hyperplane $\mu^\top x = \mu_0$. Solve the equation system

$$\forall k \leq t \sum_{j=1}^{n} x_{kj}\mu_j = \mu_0$$

in the $t+1$ unknowns (μ_j, μ_0). If the solution is $(\lambda h_j, \lambda d)$ with $\lambda \neq 0$ then the inequality $h^\top x \leq d$ is facet defining.

One of the limits of polyhedral analysis is that the theoretical devices used to find facets for a given MILP are often unique to the problem it models.

8.2.1 Separation Theory

Finding all of the facets of $\bar{P}$ is overkill, however. Since the optimization direction points us towards a specific region of the feasible polyhedron, what we really need is an oracle that, given a point $x' \in P$, tells us that $x' \in \bar{P}$ or else produces a separating hyperplane $h^\top x = d$ such that for all $h^\top \bar{x} \leq d$, $\bar{x} \in \bar{P}$ and $h^\top x' > d$, adjoining the constraint $h^\top x \geq d$ to the MILP formulation tightens the feasible region P of the continuous relaxation. The problem of finding a valid separating hyperplane is the *separation problem*. It is desirable that the separation problem for a given **NP**-hard problem should have polynomial complexity.

8.2.1 Example (Valid cuts for the TSP)
A classic example of a polynomial separation problem for a **NP***-hard problem is the* Travelling Salesman Problem *(TSP): given a nonnegatively edge-weighted clique K_n, with weight $c_{ij} \geq 0$ on each edge $\{i,j\} \in E(K_n)$, find the shortest Hamiltonian cycle in K_n.*

Now consider the following MILP:

$$\left.\begin{array}{lrcll} \min & \sum\limits_{i \neq j \in V} c_{ij}x_{ij} & & & \\ s.t. & \sum\limits_{i \neq j \in V} x_{ij} & = & 2 & \forall\, i \in V \\ & x_{ij} \in \{0,1\} & & \forall\, i \neq j \in V. & \end{array}\right\} \tag{8.1}$$

Some of its feasible solutions are given by disjoint cycles. However, we can exclude them from the feasible region by requesting that each cut should have cardinality at least 2, as follows:

$$\forall\, S \subsetneq V \,(\sum_{i \in S, j \notin S} x_{ij} \geq 2).$$

There is an exponential number of such constraints (one for each subset S of V), however we do not need them all: we can add them iteratively by identifying cuts $\delta(S)$ where $\sum_{\{i,j\} \in \delta(S)} x_{ij}$ is minimum. We can see this as the problem of finding a cut of minimum capacity in a flow network (so the current values x_{ij} are used as arc capacities — each edge is replaced by antiparallel arcs). The formulation we look for is the LP dual of Eq. (2.9); we can solve it in weakly polytime with the ellipsoid method or in strongly polytime with a combinatorial algorithm ($O(n^3)$ if we use the Goldberg-Tarjan push-relabel algorithm [123]). So if each nontrivial cut has capacity at least 2, the solution is an optimal Hamiltonian cycle.

Otherwise, supposing that $\delta(S)$ has capacity $K < 2$, the following is a valid cut:

$$\sum_{\{i,j\} \in \delta(S)} x_{ij} \geq 2,$$

which can be added to the formulation. The problem is then re-solved iteratively to get a new current solution until the optimality conditions are satisfied. This approach is also known as row generation.

In the following sections 8.2.2-8.2.6 we shall give examples of four different cut families which can be used to separate the incumbent (fractional) solution from the convex hull of the integer feasible set.

8.2.2 Chvátal Cut Hierarchy

A *cut hierarchy* is a finite sequence of cut families that generate tighter and tighter relaxed feasible sets $P = P_0 \supseteq P_1 \supseteq \ldots \supseteq P_k = \bar{P}$, where $\bar{P} = \text{conv}(X)$.

Given the relaxed feasible region $P = \{x \geq 0 \mid Ax = b\}$ in standard form, define the *Chvátal first-level closure* of P as $P_1 = \{x \geq 0 \mid Ax = b, \forall u \in \mathbb{R}^n_+ \lfloor uA \rfloor \leq \lfloor ub \rfloor\}$. Although formally the number of inequalities defining P_1 is infinite, it can be shown that these can be reduced to a finite number [308]. We can proceed in this fashion by transforming the Chvátal first-level inequalities to equations via the introduction of slack variables. Reiterating the process on P_1 to obtain the Chvátal second-level closure P_2 of P, and so on. It can be shown that any fractional vertex of P can be "cut" by a Chvátal inequality.

8.2.3 Gomory Cuts

Gomory cuts are a special kind of Chvátal cuts. Their fundamental property is that they can be inserted in the simplex tableau very easily. This makes them a favorite choice in cutting plane algorithms, which generate cuts iteratively in function of the current incumbent.

Suppose x^* is the optimal solution found by the simplex algorithm deployed on a continuous relaxation of a given MILP. Assume the component x^*_h is fractional. Since $x^*_h \neq 0$, column h is a basic column; thus there corresponds a row t in the simplex tableau:

$$x_h + \sum_{j \in \nu} \bar{a}_{tj} x_j = \bar{b}_t, \tag{8.2}$$

where ν are the nonbasic variable indices, $\bar{a}_{tj}$ is a component of $B^{-1}A$ (B is the nonsingular square matrix of the current basic columns of A) and $\bar{b}_t$ is a component of $B^{-1}b$. Since $\lfloor \bar{a}_{tj} \rfloor \leq \bar{a}_{tj}$ for each row index t and column index j,

$$x_h + \sum_{j \in \nu} \lfloor \bar{a}_{tj} \rfloor x_j \leq \bar{b}_t.$$

Furthermore, since the LHS must be integer, we can restrict the RHS to be integer too:

$$x_h + \sum_{j \in \nu} \lfloor \bar{a}_{tj} \rfloor x_j \leq \lfloor \bar{b}_t \rfloor. \tag{8.3}$$

We now subtract Eq. (8.3) from Eq. (8.2) to obtain the *Gomory cut*:

$$\sum_{j \in \nu} (\lfloor \bar{a}_{tj} \rfloor - \bar{a}_{tj}) x_j \leq (\lfloor \bar{b}_t \rfloor - b_t). \tag{8.4}$$

We can subsequently add a slack variable to the Gomory cut, transform it to an equation, and easily add it back to the current dual simplex tableau as the last row with the slack variable in the current basis.

8.2.3.1 Cutting plane algorithm

In this section we illustrate the application of Gomory cut in an iterative fashion in a cutting plane algorithm. This by solving a continuous relaxation at each step. If the continuous relaxation solution fails to be integral, a separating cutting plane (a valid Gomory cut) is generated and added to the formulation, and the process is repeated. The algorithm terminates when the continuous relaxation solution is integral.

Let us solve the following MILP in standard form:

$$\left.\begin{array}{lrcl} \min & x_1 - 2x_2 & & \\ t.c. & -4x_1 + 6x_2 + x_3 & = & 9 \\ & x_1 + x_2 + x_4 & = & 4 \\ & x \geq 0 & , & x_1, x_2 \in \mathbb{Z} \end{array}\right\}$$

In this example, x_3, x_4 can be seen as slack variables added to an original formulation in canonical form with inequality constraints expressed in x_1, x_2 only.

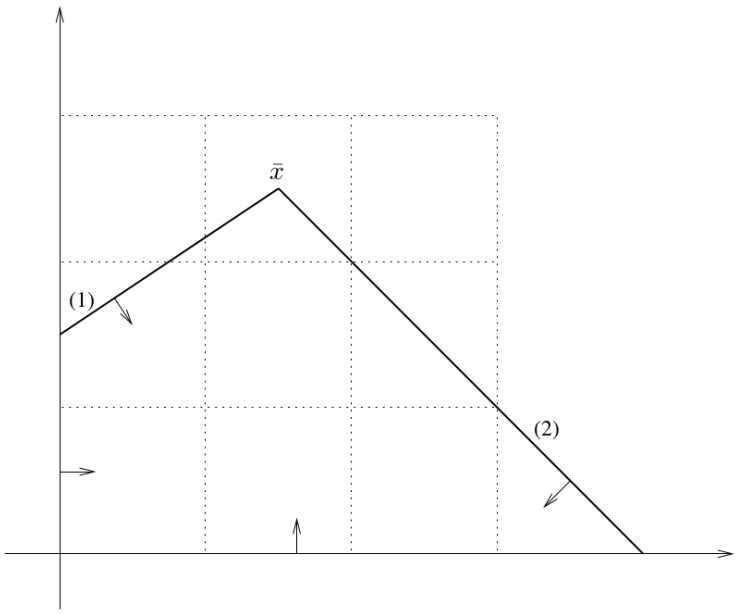
We identify $x_B = (x_3, x_4)$ as an initial feasible basis; we apply the simplex algorithm obtaining the following tableau sequence, where the pivot element is framed.

	x_1	x_2	x_3	x_4
0	1	-2	0	0
9	-4	6	1	0
4	1	1	0	1

	x_1	x_2	x_3	x_4
3	$-\frac{1}{3}$	0	$\frac{1}{3}$	0
$\frac{3}{2}$	$-\frac{2}{3}$	1	$\frac{1}{6}$	0
$\frac{5}{2}$	$\frac{5}{3}$	0	$-\frac{1}{6}$	1

	x_1	x_2	x_3	x_4
$\frac{7}{2}$	0	0	$\frac{3}{10}$	$\frac{1}{5}$
$\frac{5}{2}$	0	1	$\frac{1}{10}$	$\frac{2}{5}$
$\frac{3}{2}$	1	0	$-\frac{1}{10}$	$\frac{3}{5}$

The solution of the continuous relaxation is $\bar{x} = (\frac{3}{2}, \frac{5}{2})$, where $x_3 = x_4 = 0$.



We derive a Gomory cut from the first row of the optimal tableau: $x_2 + \frac{1}{10}x_3 + \frac{2}{5}x_4 = \frac{5}{2}$. The cut is formulated as follows:

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor, \tag{8.5}$$

where N is the set of nonbasic variable indices and i is the index of the chosen row. In this case we obtain the constraint $x_2 \leq 2$.

We introduce this Gomory cut in the current tableau. Note that if we insert a valid cut in a simplex tableau, the current basis becomes primal infeasible, thus a dual simplex iteration is needed. First of all, we express $x_2 \leq 2$ in terms of the current nonbasic variables x_3, x_4. We subtract the i-th optimal tableau

row from Eq. (8.5), obtaining:

$$\begin{aligned} x_i + \sum_{j \in N} \bar{a}_{ij} x_j &\leq \bar{b}_i \\ \Rightarrow \sum_{j \in N} (\lfloor \bar{a}_{ij} \rfloor - \bar{a}_{ij}) x_j &\leq (\lfloor \bar{b}_i \rfloor - \bar{b}_i) \\ \Rightarrow -\frac{1}{10} x_3 - \frac{2}{5} x_4 &\leq -\frac{1}{2}. \end{aligned}$$

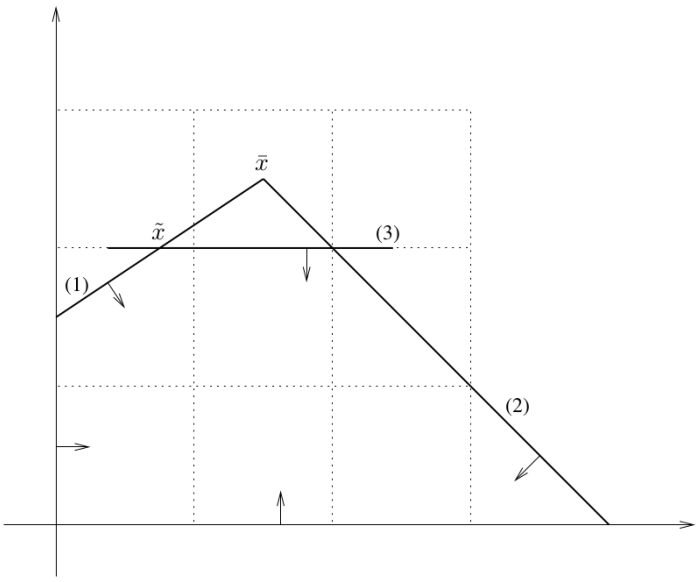
Recall that the simplex algorithm requires the constraints in equation (rather than inequality) form, so we add a slack variable $x_5 \geq 0$ to the formulation:

$$-\frac{1}{10} x_3 - \frac{2}{5} x_4 + x_5 = -\frac{1}{2}.$$

We ajoin this constraint as the bottom row of the optimal tableau. We now have a current tableau with an additional row and column, corresponding to the new cut and the new slack variable (which is in the basis):

	x_1	x_2	x_3	x_4	x_5
$\frac{7}{2}$	0	0	$\frac{3}{10}$	$\frac{1}{5}$	0
$\frac{5}{2}$	0	1	$\frac{1}{10}$	$\frac{2}{5}$	0
$\frac{3}{2}$	1	0	$-\frac{1}{10}$	$\frac{3}{5}$	0
$-\frac{1}{2}$	0	0	$-\frac{1}{10}$	$\boxed{-\frac{2}{5}}$	1

The new row corresponds to the Gomory cut $x_2 \leq 2$ (labelled "constraint (3)" in the figure below).



We carry out an iteration of the dual simplex algorithm using this modified tableau. The reduced costs are all non-negative, but $\bar{b}_3 = -\frac{1}{2} < 0$ implies that $x_5 = \bar{b}_3$ has negative value, so it is not primal feasible (as $x_5 \geq 0$ is now a valid constraint). We pick x_5 to exit the basis. The variable j entering the basis is given by:

$$j = \text{argmin}\{\frac{\bar{c}_j}{|\bar{a}_{ij}|} \mid j \leq n \wedge \bar{a}_{ij} < 0\}.$$

In this case, $j = \text{argmin}\{3, \frac{1}{2}\}$, corresponding to $j = 4$. Thus x_4 enters the basis replacing x_5 (the pivot element is indicated in the above tableau). The new tableau is:

	x_1	x_2	x_3	x_4	x_5
$\frac{13}{4}$	0	0	$\frac{1}{4}$	0	$\frac{1}{2}$
2	0	1	0	0	1
$\frac{3}{4}$	1	0	$-\frac{1}{4}$	0	$\frac{3}{2}$
$\frac{5}{4}$	0	0	$\frac{1}{4}$	1	$-\frac{5}{2}$

The optimal solution is $\tilde{x} = (\frac{3}{4}, 2)$. Since this solution is not integral, we continue. We pick the second tableau row:

$$x_1 - \frac{1}{4}x_3 + \frac{3}{2}x_5 = \frac{3}{4},$$

to generate a Gomory cut

$$x_1 - x_3 + x_5 \leq 0,$$

which, written in terms of the variables x_1, x_2 is

$$-3x_1 + 5x_2 \leq 7.$$

This cut can be written as:

$$-\frac{3}{4}x_3 - \frac{1}{2}x_5 \leq -\frac{3}{4}.$$

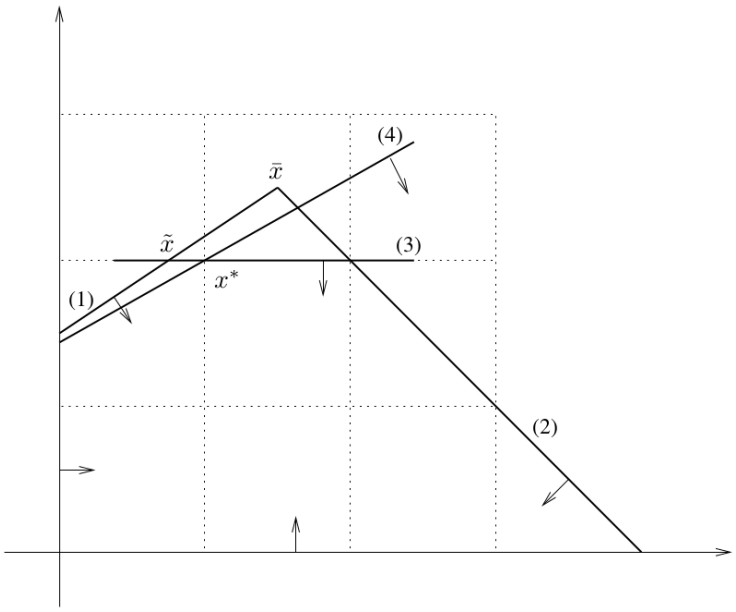
The new tableau is:

	x_1	x_2	x_3	x_4	x_5	x_6
$\frac{13}{4}$	0	0	$\frac{1}{4}$	0	$\frac{1}{2}$	0
2	0	1	0	0	1	0
$\frac{3}{4}$	1	0	$-\frac{1}{4}$	0	$\frac{3}{2}$	0
$\frac{5}{4}$	0	0	$\frac{1}{4}$	1	$-\frac{5}{2}$	0
$-\frac{3}{4}$	0	0	$\boxed{-\frac{3}{4}}$	0	$-\frac{1}{2}$	1

The pivot is framed; the exiting row 4 was chosen because $\bar{b}_4 < 0$, the entering column 5 because $\frac{\bar{c}_3}{|\bar{a}_{43}|} = \frac{1}{3} < 1 = \frac{\bar{c}_5}{|\bar{a}_{45}|}$). Pivoting, we obtain:

	x_1	x_2	x_3	x_4	x_5	x_6
3	0	0	0	0	$\frac{1}{3}$	$\frac{1}{3}$
2	0	1	0	0	1	0
1	1	0	0	0	$\frac{5}{3}$	$-\frac{1}{3}$
1	0	0	0	1	$-\frac{8}{3}$	$\frac{1}{3}$
1	0	0	1	0	$\frac{2}{3}$	$-\frac{4}{3}$

This tableau has optimal solution $x^* = (1, 2)$, which is integral (and hence optimal). The figure below shows the optimal solution and the last Gomory cut to be generated.



8.2.4 Disjunctive cuts

A condition such as "either... or..." can be modelled using disjunctive constraints (stating e.g. membership of a variable vector in a disjunction of two or more sets), wchich are themselves written using binary variables.

8.2.2 Example
Consider two polyhedra $P = \{x \in \mathbb{R}^m \mid Ax \leq b\}$ and $Q = \{x \in \mathbb{R}^n \mid Cx \leq d\}$. Then the constraint $x \in P \cup Q$ can be written as $(x \in P) \vee (x \in Q)$ by means of an additional binary variable $y \in \{0, 1\}$:

$$\begin{aligned} yAx &\leq yb \\ (1-y)Cx &\leq (1-y)d. \end{aligned}$$

We remark that the above constraints are bilinear, as they involve products of variables yx_j for all $j \leq n$. This can be dealt with using the techniques in Rem. 2.2.8. On the other hand, if P, Q are polytopes (i.e. they are bounded), it suffices to find a large constant M such that $Ax \leq b + M$ and $Cx \leq d + M$ hold for all $x \in P \cup Q$, and then adjoin the linear constraints:

$$\begin{aligned} Ax &\leq b + My \\ Cx &\leq d + M(1-y) \end{aligned}$$

We caution the reader against the indiscriminate use of "big M" constants (see Sect. 2.2.7.5.1). How would one find a "good" big M? First of all, if one really needed to find a single constant M valid for $x \in P \cup Q$, one possible way to go about it would be to find the smallest possible hyper-rectangle containnig P and Q, describe it by $x^L \leq x \leq x^U$, and then compute M as the maximum upper bound w.r.t. each row in $Ax - b$: this involves using interval arithmetic on $A[x^L, x^U] - b$ [230]. Secondly, we need not use the same constant M to bound both $Ax \leq b$ and $Cx \leq d$: we might be able to find smaller M_1, M_2 such that $Ax \leq b + M_1$ and $Cx \leq d + M_2$ are always valid for all $x \in P \cup Q$. This would allow us to write

$$\begin{aligned} Ax &\leq b + M_1 y \\ Cx &\leq d + M_2(1-y). \end{aligned}$$

8.2.3 Exercise
Propose an algorithm for finding the smallest hyper-rectangle $[x^L, x^U]$ *containing a given polytope* $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$.

Answer. Solve the sequence of $2n$ LPs:

$$\forall j \leq n \quad \left(\begin{array}{rcl} x_j^L = \min_x & x_j & \\ Ax & \leq & b \end{array} \right) \wedge \left(\begin{array}{rcl} x_j^U = \max_x & x_j & \\ Ax & \leq & b, \end{array} \right) \tag{8.6}$$

giving an interval vector $[x^L, x^U]$ with the given properties. This method is called *Optimization-Based Bounds Tightening* (OBBT), also see Sect. 9.4.1.1.

We now describe a method for computing a cut valid for the union of two polyhedra $P \cup Q$, where $P = \{x \geq 0 \mid Ax \leq b\}$ and $Q = \{x \geq 0 \mid ; Cx \leq d\}$ (note that we are explicitly requiring $x \geq 0$ for both P, Q). We construct our cut from row i in the description of P and row ℓ in the description of Q. For each $j \leq n$ choose some $h_j \leq \min(A_{ij}, C_{\ell j})$ and some $h_0 \geq \max(b_i, d_\ell)$. Then the inequality

$$h^\top x \leq h_0 \tag{8.7}$$

is valid for $P \cup Q$.

8.2.4 Exercise
Prove that Eq. (8.7) *is valid for* $P \cup Q$.

Answer. Suppose not. Then there must be $x' \in P \cup Q$ such that $h^\top x' > h_0$. Wlog we assume $x' \in P$: then by replacing each $h_j \neq A_{ij}$ with A_{ij} we simply increase the LHS, so the inequality still holds. Moreover, by replacing h_0 with b_i we decrease the RHS, so the inequality again holds: but by these replacements we have $A_i x' > b_i$, which means that $Ax \not\leq b$, implying $x' \notin P$, against the assumption.

8.2.5 Lifting

It is possible to derive valid inequalities for a given MILP

$$\min\{cx \mid Ax \leq b \wedge x \in \{0,1\}^n\} \tag{8.8}$$

by *lifting* lower-dimensional inequalities to a higher-dimensional space. If the inequality $\sum_{j=2}^n \pi_j x_j \leq \pi_0$ (with $\pi_j \geq 0$ for all $j \in \{2, \ldots, n\}$) is valid for the restricted

$$\min\{\sum_{j=2}^n c_j x_j \mid \sum_{j=2}^n a_{ij} x_j \leq b_i \forall i \leq m \wedge x_j \in \{0,1\} \; \forall \, j \in \{2, \ldots, n\}\},$$

then by letting

$$\left. \begin{array}{rcll} \pi_1 = \max & (\pi_0 - \sum_{j=2}^n \pi_j x_j) & & \\ & \sum_{j=2}^n a_{ij} x_j & \leq & b_i - a_{i1} \qquad \forall i \leq m \\ & x_j & \in & \{0,1\} \qquad \forall j \in \{2, \ldots, n\}. \end{array} \right\}$$

we can find the inequality $\sum_{j=1}^n \pi_j x_j \leq \pi_0$ which is valid for (8.8), as long as the above formulation is not infeasible; if it is, then $x_1 = 0$ is a valid inequality. Of course the lifting process can be carried out with respect to any variable index (not just 1).

8.2.6 RLT cuts

Reformulation-Linearization Techniques (RLT) cuts, described in [274], are a form of *nonlinear lifting*. From each constraint $a_i^\top x \geq b_i$ of a feasible set

$$F = \{x \in \{0,1\}^n \mid Ax \leq b\}$$

we derive the i-th *constraint factor*

$$\gamma(i) = a_i^\top x - b_i$$

(for $i \leq m$). We then form the *bound products*

$$p(I, J) = \prod_{j \in I} x_j \prod_{\ell \in J} (1 - x_\ell)$$

for all partitions I, J of the index set $\{1, \ldots, d\}$. The set of all of the cuts

$$p(I, J)\, \gamma(i) \geq 0 \tag{8.9}$$

obtained by multiplying each constraint factor $\gamma(i)$ (for $i \leq m$) by all the bound products $p(I, J)$ over all I, J s.t. $I \cup J = \{1, \ldots, d\}$, is called the *RLT d-th level closure*. We remark that d refers to the maximum degree of the bound products.

Notice that Eq. (8.9) are nonlinear inequalities. As long as they only involve binary variables, we can apply a linearization which generalizes the one in Rem. 2.2.8: after replacing each occurrence of x_j^2 by x_j (because $x_j \in \{0, 1\}$) for each j, we linearize the inequalities Eq. (8.9) by lifting them in a higher-dimensional space: replace all products $\prod_{j \in H} x_j$ by a new added variable $w_H = \prod_{j \in H} x_j$, to obtain linearized cuts of the form

$$\mathscr{L}^i_{IJ}(x, w) \geq 0, \tag{8.10}$$

where $\mathscr{L}^i_{IJ}$ is a linear function of x, w. We define the RLT d-th level closure as:

$$P_d = \{x \mid \forall i \leq m \text{ and } I, J \text{ partitioning } \{1, \ldots, d\},\ \mathscr{L}^i_{IJ}(x, w) \geq 0\}.$$

The *RLT cut hierarchy* is defined as the union of all the RLT d-th level closures P_d up to and including level n. It was shown in [273] that

$$\mathsf{conv}(F) = \mathsf{proj}_x(P_n) \subseteq \mathsf{proj}_x(P_{n-1}) \subseteq \ldots \subseteq \mathsf{proj}_x(P_0) = \mathsf{relax}(F), \tag{8.11}$$

where $\mathsf{conv}(X)$ denotes the convex hull of a set X, $\mathsf{proj}_x(X)$ denotes the projection on the x variables of a set description X involving the x variables and possibly other variables (e.g. the w linearization variables), and $\mathsf{relax}(X)$ denotes the continuous relaxation of a mixed-integer set X. In other words, the whole RLT hierarchy yields the convex hull of any linear set involving binary variables.

This cut hierarchy has two main shortcomings: (a) the high number of additional variables and (b) the huge number of generated constraints. It is sometimes used heuristically to generate tighter continuous relaxations various mixed-integer formulations.

8.3 Branch-and-Bound

BB is an iterative method that endows the search space with a tree structure. Each node of this tree defines a subproblem. BB processes each subproblem in turn. Lower and upper bounds to the objective function value are computed at the current subproblem, and globally valid upper and lower bounds (w.r.t. all remaining subproblems) are maintained during the search. Usually, and assuming a minimization direction, upper bounds are derived from feasible solutions, computed using various heuristics. Lower bounds are computed by solving a relaxation of the subproblem (unlike upper bounds, lower bounds do not always correspond to a feasible solution). If the lower bound at the current subproblem is larger than the globally valid upper bound, the current subproblem cannot yield a global optimum, so it is *pruned by bound* (i.e., discarded because its lower bound is worse than a global upper bound). If the lower and upper bounds are equal (or sufficiently close), then the current upper bound solution is assumed to be globally optimal for the subproblem — and if it improves the globally valid optimum (called *incumbent*), it is used to update it. Otherwise, a partition of the feasible region of the current subproblem is defined,

and the search is *branched*, yielding as many new subproblems as there are sets of the partition. The BB algorithm terminates when there are no more subproblems to process.

Most BB implementations define the search tree implicitly by storing subproblems into a priority queue, with highest priority given to subproblems with lowest lower bound. This is because, intuitively, relaxations are expected to yield lowest lower bounds in part of the feasible region containing global optima. The queue is initialized with the original formulation. In a typical iteration, a subproblem is extracted from the priority queue, processed as described above, and then discarded.

For MILP, lower bounds are usually computed by solving the continuous relaxations. If the lower bound does not prune the current subproblem by bound, the most common branching strategy in MILP selects an integer variable $x_j \in [x_j^L, x_j^U] \cap \mathbb{Z}$ where the lower bounding solution $\bar{x}$ obtained from the continuous relaxation is such that $\bar{x}_j \notin \mathbb{Z}$. The feasible set is partitioned in two: a part where $x_j \leq \lfloor \bar{x}_j \rfloor$, and the other where $x_j \geq \lceil \bar{x}_j \rceil$. This is equivalent to splitting the parent subproblem range $[x_j^L, x_j^U]$ in two ranges $[x_j^L, \lfloor \bar{x}_j \rfloor]$ and $[\lceil \bar{x}_j \rceil, x_j^U]$. If $\bar{x} \in \mathbb{Z}^n$, lower and upper bounds for the current subproblem clearly have the same value, which is used to update the incumbent if needed. Then the problem is discarded.

8.3.1 Example

Initially, we let $x^* = (0, 0)$ e $f^* = -\infty$. Let L be the list (priority queue) containing the unsolved subproblems, initialized to the original formulation N_1:

$$\left.\begin{array}{rrcl} \max & 2x_1 + 3x_2 & & \\ & x_1 + 2x_2 & \leq & 3 \\ & 6x_1 + 8x_2 & \leq & 15 \\ & x_1, x_2 & \in & \mathbb{Z}_+. \end{array}\right\} \tag{8.12}$$

We remark that in this situation we are maximizing rather than minimizing the objective, which implies that the roles of lower and upper bounds are exchanged.

The solution of the continuous relaxation is P (corresponding to the intersection of line (1) $x_1+2x_2 = 3$ with (2) $6x_1 + 8x_2 = 15$), as shown in the Fig. 8.1 We obtain an upper bound $f = \frac{21}{4}$ corresponding to a

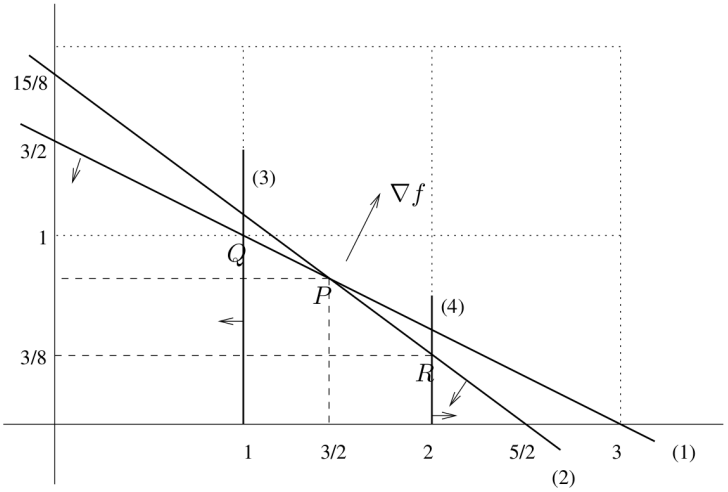


Figure 8.1: Graphical representation of the problem in Eq. (8.12).

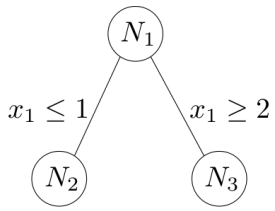
fractionary solution $x = P = (\frac{3}{2}, \frac{3}{4})$. We choose the fractionary component x_1 as the branching variable,

and form two subproblems N_2, N_3. We adjoin the constraint $x_1 \leq 1$ (labelled (3)) to N_2 and $x_1 \geq 2$ (labelled (4)) to N_3. Finally, we insert N_2 and N_3 in L with priority $f = \frac{21}{4}$.

Next, we choose N_2 and remove it from L. The solution of the continuous relaxation of N_2 is at the intersection Q of (1) and (3): we therefore obtain $x = Q = (1, 1)$ and $f = 5$. Since the solution is integral, we do not branch. Furthermore, since $f > f^*$ we update $x^* = x$ and $f^* = f$.

Finally, we choose N_3 and remove it from L. The solution of the continuous relaxation of N_3 is at the intersection R of (2) and (4): we obtain $x = R = (2, \frac{3}{8})$ and $f = \frac{41}{8}$. Since the upper bound is $41/8$, and $\lfloor \frac{41}{8} \rfloor = 5$, each integral solution found in the feasible region of N_3 will never attain a better objective function value than f^*; hence, again, we do not branch.

Since the L is now empty, the algorithm terminates with solution $x^* = (1, 1)$. The algorithmic process can be summarized by the following tree:



8.3.2 Branch-and-Cut

Branch-and-Cut (B&C) algorithms are a mixture of BB and cutting plane algorithms (Sect. 8.2.3.1). In practice, they follow a BB framework where valid cuts are generated at each subproblem [239].

8.3.3 Branch-and-Price

Branch-and-Price might be described as a mixture of BB and column generation algorithms (Sect. 7.1.8). Extremely large LP formulations are solved using column generation at each BB subproblem [290].

8.4 Lagrangean relaxation

Although the most common way to obtain a lower bound for a MILP is to solve its continuous relaxation, as mentioned in Sect. 8.3, there are other techniques. The technique we sketch in this section leverages duality theory (see Sect. 6.3) in order to exploit the formulation structure, which may be a great help for large-scale instances.

Consider the following MILP formulation:

$$\left.\begin{array}{rrcl} \min_{x,y} & c_1^\top x + c_2^\top y & & \\ & A_1 x & \geq & b_1 \\ & A_2 y & \geq & b_2 \\ & Bx + Dy & \geq & d \\ & x \in \mathbb{Z}_+^{n_1} & , & y \in \mathbb{Z}_+^{n_2} \end{array}\right\} \tag{8.13}$$

where x, y are two sets of integer variables, $c_1^\top, c_2^\top, b_1, b_2, d$ are vectors and A_1, A_2, B, D are matrices of appropriate sizes. Observe that, were it not for the "complicating constraints" $Bx + Dy \geq d$, we would be

able to decompose Eq. (8.13) into two smaller independet subproblems, and then solve each separately (usually a much easier task). Such formulations are sometimes referred to as "nearly decomposable".

A tight lower bound for nearly decomposable formulations can be obtained by solving a *Lagrangean relaxation*. The complicating constraints are penalized as part of the objective function, with each constraint weighted by a Lagrange multiplier:

$$\left.\begin{array}{rrcl} L(\lambda) = \min_{x,y} & c_1^\top x + c_2^\top y & + & \lambda(Bx + Dy - d) \\ & A_1 x & \geq & b_1 \\ & A_2 y & \geq & b_2 \\ & x \in \mathbb{Z}_+^{n_1} & , & y \in \mathbb{Z}_+^{n_2} \end{array}\right\} \tag{8.14}$$

For each vector $\lambda \geq 0$, $L(\lambda)$ is a lower bound to the solution of the original formulation. Since we want to find the tightest possible lower bound, we maximize $L(\lambda)$ w.r.t. λ:

$$\max_{\lambda \geq 0} L(\lambda). \tag{8.15}$$

The lower bound obtained by solving (8.15) is guaranteed to be at least as tight as that obtained by solving the continuous relaxation of the original problem: let us see why. Consider the following formulation:

$$\left.\begin{array}{rrcl} \min_x & c^\top x & & \\ & Ax & \leq & b \\ & x & \in & X, \end{array}\right\} \tag{8.16}$$

where X is a discrete but finite set $X = \{x^1, \ldots, x^\ell\}$, and its Lagrangean relaxation obtained as if each constraint were complicating:

$$\max_{\lambda \geq 0} \min_{x \in X} c^\top x + \lambda(Ax - b). \tag{8.17}$$

Let p^* be the solution of Eq. (8.17). By definition of X, Eq. (8.17) can be written as follows:

$$\left.\begin{array}{rrcll} \max_{\lambda,u} & u & & & \\ & u & \leq & c^\top x^t + \lambda(Ax^t - b) & \forall t \leq \ell \\ & \lambda & \geq & 0. & \end{array}\right\}$$

The above formulation is linear. Hence, its dual:

$$\left.\begin{array}{rrcll} \min_v & \sum_{t=1}^\ell v_t (c^\top x^t) & & & \\ & \sum_{t=1}^\ell v_t (A_i x^t - b_i) & \leq & 0 & \forall i \leq m \\ & \sum_{t=1}^\ell v_t & = & 1 & \\ & v_t & \geq & 0 & \forall t \leq \ell \end{array}\right\}$$

(where A_i is the i-the row of A), has the same optimal objective function value p^*. Notice now that $\sum_{t=1}^\ell v_t x^t$ subject to $\sum_{t=1}^\ell v_t = 1$ and $v_t \geq 0$ for all $t \leq \ell$ is the convex hull of $\{x^t \mid t \leq \ell\}$, and so the above dual formulation can be written as follows:

$$\left.\begin{array}{rrcl} \min_x & c^\top x & & \\ & Ax & \leq & b \\ & x & \in & \mathsf{conv}(X). \end{array}\right\}$$

Thus, the bound given by the Lagrangean relaxation is at least as strong as that given by the continuous relaxation restricted to the convex hull of the original discrete domain. Specifically, for problems with the integrality property, the Lagrangean relaxation bound is no tighter than the continuous relaxation bound. On the other hand, the above result leaves the possibility open that Lagrangian relaxations may be tighter.

Solving (8.15) is not straightforward, as $L(\lambda)$ is a piecewise linear (but non-differentiable) function. The most popular method for solving such problems is the *subgradient method*. Its description is very simple, although the implementation is not.

1. Start with an initial multiplier vector λ^*.
2. If a pre-set termination condition is verified, exit.
3. Evaluate $L(\lambda^*)$ by solving Eq. (8.14) with fixed λ, yielding a solution x^*, y^*.
4. Choose a vector d in the subgradient of L w.r.t. λ and a step length t, and set $\lambda' = \max\{0, \lambda^* + td\}$. **define sub**
5. Update $\lambda^* = \lambda'$ and go back to step 2.

Choosing appropriate subgradient vectors and step lengths at each iteration to accelerate convergence is, unfortunately, a non-trivial task that takes a lot of testing.

Chapter 9

Nonlinear Programming

This chapter is devoted to NLP. As we have seen in Chapter 6, much of the existing theory in NLP concerns local optimality. We start our treatment by summarizing a well-known local optimization algorithm called *sequential quadratic programming* (SQP), which can be seen as a solution method for cNLP as well as a heuristic algorithm for nonconvex NLP. We then overview the field of GO, focusing on general NLP.

9.1 Sequential quadratic programming

There are many different algorithmic approaches to local optimization of NLPs in the form (6.9), most of which are based on the theoretical results of Section 6.2. The vast majority of local optimization methods will take in input a NLP formulation as well as an "starting point" assumed to be feasible: as an output, it will return a local optimum close to the starting point. The assumption that a feasible starting point is known *a priori* is often false in practice: which implies that some of the theoretical guarantees might not hold. On the other hand, many local NLP algorithms deliver good computational performance and good quality solutions even from infeasible starting points.

One of the best known local optimization algorithms for NLP is SQP: it aims at solving the KKT conditions (6.6)-(6.8) iteratively by solving a sequence of quadratic approximations of the objective function subject to linearized constraints.

We start with a given starting point $x \in X$ and generate a sequence $x^{(k)}$ of points using the update relation:

$$x^{(k+1)} = x^{(k)} + \alpha d,$$

where α is the step length and d the search direction vector. In other words, at each iteration we move from $x^{(k)}$ in the direction of d by an amount αd. Both α and d are updated at each iteration. The step length $\alpha \in (0, 1]$ is updated at each iteration using a line search optimization method.

9.1.1 Remark (Line search)
A line search is a lifting of an optimization problem in a single scalar variable into a higher-dimensional space. Given a multivariate function $f(x)$ where $t \in [x^L, x^U] \subseteq \mathbb{R}^n$ is a decision variable vector, it sometimes happens that one would wish to optimize $f(x)$ for x belonging to a line, half-line or segment. Thus, if $x = x^0 + \alpha d$, where $x^0, d \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}_+$, x belongs to a half-line, and $f(x) = f(x^0 + \alpha d) = u_f(\alpha)$ becomes the problem of optimizing the univariate function u_f in the single variable α.

If f is monotonic, line searches can be carried out to a given $\epsilon > 0$ precision using a bisection search (see Sect. 7.3.1.1.5). Otherwise, bisection searches might help finding local optima. If f is a polynomial,

global optima can be found by listing all the roots $A = \{\alpha_1, \ldots, \alpha_\ell\}$ *of the polynomial equation* $\frac{df}{d\alpha} = 0$ *and choosing the* α_i *such that* $u_f(\alpha_i)$ *is minimum in* $U = \{u_f(\alpha_j) \mid j \leq \ell\}$.

The search direction vector d is obtained at each step by solving the following QP, where m, p, f, g and h are as in Eq. (6.9):

$$\left.\begin{array}{lrcll} \min\limits_{d} & (\nabla f(x^{(k)}))^\top d + \frac{1}{2} d^\top H(x^{(k)}) d & & & \\ \text{s.t.} & (\nabla g_i(x^{(k)}))^\top d & \leq & 0 & \forall i \leq m \\ & (\nabla h_i(x^{(k)}))^\top d & = & 0 & \forall i \leq p. \end{array}\right\} \tag{9.1}$$

In Eq. (9.1), $H(x^{(k)})$ is a PSD approximation to the Hessian of the objective function evaluated at $x^{(k)}$. As remarked in Sect. 5.4.1, a quadratic function can be shown to be convex by verifying that its Hessian is PSD, so Eq. (9.1) is the best convex quadratic approximation of Eq. (6.9). Thus, a KKT point for (9.1) is a globally optimal solution of (9.1) by Thm. 6.2.9.

The KKT conditions for each subproblem (9.1) are solved directly (e.g. by Newton's method) to obtain the solution of (9.1). If $\|d\|$ is smaller than a pre-defined $\varepsilon > 0$ tolerance, the current point $x^{(k)}$ solves the KKT conditions for (6.9) and the process terminates. Notice that solving the KKT conditions for Eq. (9.1) also provides current solution values for the Lagrange multipliers μ, λ, which upon termination are also valid for the original formulation Eq. (6.9).

9.2 The structure of GO algorithms

Most of the algorithms for globally solving NLPs (and MINLPs) have two phases: a global one and a local one [265]. During the global phase, calls are made to other possibly complex algorithms which belong to the local phase. The function of the global phase is to survey the whole of the search space, while local phase algorithms identify local optima. The global phase is also tasked with selecting the best local optima to return to the user as "global".

9.2.1 Deterministic vs. stochastic

GO algorithms are usually partitioned into two main categories: deterministic and stochastic. While this partition mostly concerns the global phase, it may occasionally also refer to the local phase. An algorithm is stochastic when it involves an element of random choice, whereas a deterministic algorithm does not. We do not enter into philosophical discussions of what exactly is a "random choice": for our purposes, a pseudo-random number generator initialized with the current exact time as a seed is "random enough" to warrant the qualification of "stochastic" to the calling algorithm.

Stochastic algorithms typically offer a theoretical guarantee of global optimality only in infinite time with probability 1 (or none at all). Deterministic algorithm typically offer theoretical guarantees of either local or global optimality in finite time under certain conditions on their input (such as, e.g., Slater's constraint qualification in Sect. 6.3.3). Moreover, "global optimality" for NLP incurs the issue of solution representability (Sect. 4.2). The approach encountered most often in GO is sketched in Sect. 4.2.2, and is "the easiest approach" in the sense that one uses floating point numbers instead of reals, and hopes everything will turn out well in the end. In any case, all general-purpose deterministic GO algorithms require an $\epsilon > 0$ tolerance in input, which is used to approximate with floating point numbers some computational tasks that should in theory be performed with real numbers (such as, e.g., whether two real numbers are equal or not).

9.2.2 Algorithmic reliability

As we shall see below, GO algorithms are usually quite complicated: either because the global phase is complicated (this is the case for BB), or because the local phase is complicated, or both. In practice, complicated algorithms require complicated implementations, and these in turn make it more difficult for programmers to avoid or even remove bugs. Therefore, the overall (practical) reliability of many implementations of GO algorithms might not be perfect.

While this is a point that could be made for any sufficiently complicated algorithm, there is however an inherent unreliability in all local optimization algorithms for NLPs which affects the reliability of the calling algorithm. In other words, GO algorithms are only as reliable as their most unreliable component.

Let us consider the SQP algorithm of Sect. 9.1 as an example. The theory of SQP guarantees convergence subject to conditions: in general, these conditions are that the starting point should be feasible, and that some constraint qualification conditions should hold. In practice, SQP implementations are often called from global phases of GO algorithms with either of these conditions not holding. There are good reasons for this: (a) verifying feasibility in NLP is just as hard, for computational complexity purposes, as verifying optimality (see Sections 4.3.1, 4.3.2, 4.3.2.2, 5.1.2.3); (b) verifying constraint qualification conditions might in general require precise real number computation, which may be impossible. This implies that the local phase might fail: either in a controlled fashion (reporting some proof of failure), or, more spectacularly, by not converging or simply crashing.

The following issues provide the most popular reasons why SQP implementations might fail on even the most innocent-looking NLPs:

- the linearized constraints of (9.1) may be infeasible, even though the original constraints of (6.9) are feasible;
- the Jacobian of the constraints may not have full rank (see Thm. 6.2.1).

Tracing a definite cause for such occurrences is generally very difficult. Subproblems may be defective locally (because the approximation is poor) or because the original formulation is ill-posed. Some of these issues may be overcome by simply changing the starting point.

9.2.3 Stochastic global phase

Stochastic global phases may be based on sampling and/or escaping.

9.2.3.1 Sampling approaches

In sampling approaches, a local phase is deployed from each of many different starting points, sampled according to various rules (Fig. 9.1). Typical examples are provided by *multistart* algorithms, such as Multi-Level Single Linkage (MLSL) [197]. The best local optimum is returned as putative global optimum One of the most common artificial termination conditions is the Bayesian stopping rule, which is based on the expected number of local minima in the sampling set. CPU time limit may also be employed.

Sampling stochastic algorithms (without artificial termination) are guaranteed to converge in infinite time with probability 1, but in practice there is no guarantee that the returned point will actually be the global optimum. These algorithms work reasonably well for small- and medium-sized instances. The chance of finding global optima worsens considerably as the number of dimensions of the search space increases.

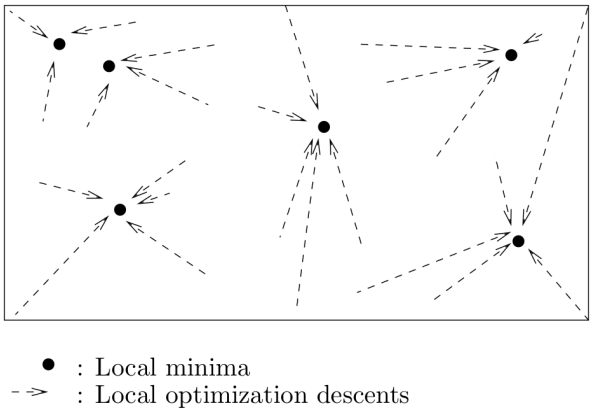


Figure 9.1: Sampling approach of the stochastic global phase. The arrows indicate a local optimization descent path from a starting point to the target local minimum.

9.2.3.2 Escaping approaches

In the escaping approach (see e.g. Tabu Search [161], Simulated Annealing [195] and tunneling methods [201]), various strategies are devised in order to "escape from local optima", so that after the local phase terminates, the global phase is able to reach another feasible point, from which a new local phase can be deployed (Fig. 9.2). Termination conditions for these methods are usually based on the number of calls to the local phase, or on a CPU time limit. As in the sampling approach, there are no definite guarantees on global optimality apart from convergence in infinite time with probability 1.

9.2.1 Remark
In Tabu Search, successive candidate points are located by performing some "moves" in the search space (e.g. $x \leftarrow x + \alpha d$ for some scalar α and vector d). The inverse of the move is then put into a tabu list and forbidden from being applied for as long as it stays in the list. This makes it unlikely to fall back on the same local minimum we are escaping from. In Simulated Annealing, the search procedure is allowed to escape from local optima with a decreasing probability. Tunneling methods are inspired by the quantum-mechanical idea of a particle in a potential well which escapes from the well with a certain probability.

9.2.3.3 Mixing sampling and escaping

There are also some stochastic methods, such as e.g. Variable Neighbourhood Search (VNS), which use both approaches at the same time. VNS is structured in a major and a minor iteration loop. Each major iteration is as follows: from the current local optimum x' loop over increasingly larger neighbourhoods centered at x'; run a multistart in each neighbourhood (minor iteration) until an improved local optimum $\bar{x}$ is found (this concludes a major iteration); if $\bar{x}$ improves the incumbent, update it with $\bar{x}$, then update x' with $\bar{x}$ and repeat. If no improving optimum is found during the minor iterations, terminate.

9.2.3.4 Clustering starting points

In both approaches, a nontrivial issue is that of reducing the number of local phase deployments converging to the same local optimum, which is a waste of computational resources. The idea of *clustering* for the stochastic global phase suggests that the sampling of starting points should be grouped into clusters of nearby points: only one local phase from each cluster should be performed [265]. The MLSL algorithm

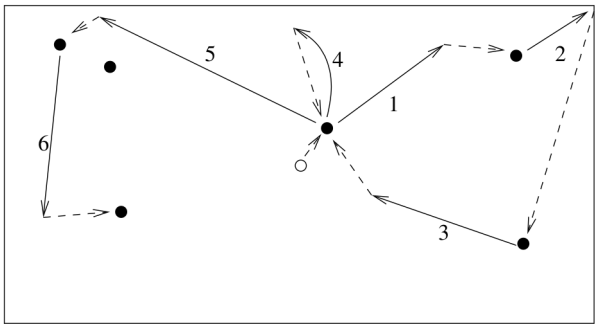


Figure 9.2: Escaping approach of the stochastic global phase. The trail consists of local optimization descents alternating with random escaping paths. Observe that certain escaping paths (e.g. arc 4) might position the new starting point in the same basin of attraction as the previous local optimum.

proposes a clustering based on linkage: a point x is clustered with a point y if x is "not too far" from y and the objective function value at y is better than that at x. The clusters are represented by a directed tree, and the local phase is deployed with the root of this tree (Fig. 9.3).

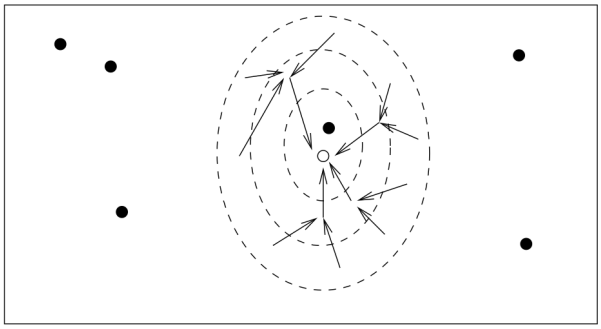


Figure 9.3: Linkage clustering in the stochastic global phase. The points in the cluster are those incident to the arcs; each arc (x, y) expresses the relation "x is clustered to y". The arcs point in the direction of objective function descent. The root of the tree isthe "best starting point" in the cluster.

9.2.4 Deterministic global phase

This section contains a very general treatment (mostly taken from [289, §5.5]) of a class of algorithms, called Branch-and-Select (B&S), which is an abstract model for BB, B&C and sBB algorithms. Although B&S is not the only deterministic global phase approach to GO, it is nonetheless the most widely used.

B&S algorithms can be used to solve Eq. (2.2) in full generality to global optimality. Without any "acceleration devices" (such as e.g. pre-processing) they often show their exponential worst-case complexity even on very small-sized instances, and tend to be very inefficient in practice. Furthermore, their performance may depend strongly on the formulation: a different algebraic form of the same equations

and inequalities might lead to a different algorithmic performance.

Let $\Omega \subseteq \mathbb{R}^n$. A finite family of sets $\mathcal{S}$ is a *net* for Ω if it is pairwise disjoint and it covers Ω, that is, $\forall s, t \in \mathcal{S}$ $(s \cap t = \emptyset)$ and $\Omega \subseteq \bigcup_{s\in\mathcal{S}} s$. A net $\mathcal{S}'$ is a *refinement* of the net $\mathcal{S}$ if $\mathcal{S}'$ has been obtained from $\mathcal{S}$ by finitely partitioning some set s in $\mathcal{S}$ and then replacing s by its partition; that is, given $s \in \mathcal{S}$ with $s = \bigcup_{i\in I} s_i$ where I is an index subset of $\{1, \ldots, |\mathcal{S}'|\}$ and each $s_i \in \mathcal{S}'$, we have

$$\mathcal{S}' = (\mathcal{S}\backslash s) \cup \{s_i \mid i \in I\}.$$

Let $\langle \mathcal{S}_n \rangle$ be an infinite sequence of nets for Ω such that, for all $i \in \mathbb{N}$, $\mathcal{S}_i$ is a refinement of $\mathcal{S}_{i-1}$; let $\langle M_n \rangle$ be an infinite sequence of subsets of Ω such that $M_i \in \mathcal{S}_i$. Then $\langle M_n \rangle$ is a *filter* for $\langle \mathcal{S}_n \rangle$ if $\forall i \in \mathbb{N}$ we have $(M_i \subseteq M_{i-1})$. We call $M_\infty = \bigcap_{i\in\mathbb{N}} M_i$ the *limit* of the filter.

We will now present a general algorithmic framework (Fig. 9.4) to solve the generic problem $\min\{f(x) \mid x \in \Omega\}$.

Given any net $\mathcal{S}$ for Ω and any objective function value $\gamma \in \mathbb{R}$, we consider a selection rule which determines:

(a) a distinguished point x^*_M for every $M \in \mathcal{S}$;

(b) a subfamily of qualified sets $\mathcal{R} \subset \mathcal{S}$ such that, for all $x \in \bigcup_{s\in\mathcal{S}\backslash\mathcal{R}} s$, we have $f(x) \geq \gamma$;

(c) a distinguished set $B_\mathcal{R}$ of $\mathcal{R}$.

For each set of $\mathcal{R}$, B&S iteratively calculates a distinguished point (for example, by deploying a local phase within the specified region); $B_\mathcal{R}$ is then picked for further net refinement.

BRANCH-AND-SELECT:

1. (Initialization) Start with a net $\mathcal{S}_1$ for Ω, set $x_0 = \emptyset$ and let γ_0 be any strict upper bound for $f(\Omega)$. Set $\mathcal{P}_1 = \mathcal{S}_1$, $k = 1$, $\sigma_0 = \emptyset$.
2. (Evaluation) Let $\sigma_k = \{x^*_M \mid M \in \mathcal{P}_k\}$ be the set of all distinguished points.
3. (Incumbent) Let x_k be the point in $\{x_{k-1}\} \cup \sigma_k$ such that $\gamma_k = f(x_k)$ is lowest; set $x^* = x_k$.
4. (Screening) Determine the family $\mathcal{R}_k$ of qualified sets of $\mathcal{S}_k$ (in other words, from now on ignore those sets that can be shown not to contain a solution with objective value lower than γ_k).
5. (Termination) If $\mathcal{R}_k = \emptyset$, terminate. The problem is infeasible if $\gamma_k \geq \gamma_0$; otherwise x^* is the global optimum.
6. (Selection) Select the distinguished set $M_k = B_{\mathcal{R}_k} \in \mathcal{R}_k$ and partition it according to a prespecified branching rule. Let $\mathcal{P}_{k+1}$ be a partition of $B_{\mathcal{R}_k}$. In $\mathcal{R}_k$, replace M_k by $\mathcal{P}_{k+1}$, thus obtaining a new refinement net $\mathcal{S}_{k+1}$. Set $k \leftarrow k + 1$ and go back to Step 2.

Figure 9.4: The B&S algorithm.

A B&S algorithm is *convergent* if γ^*, defined as $\lim_{k\to\infty} \gamma_k$, is such that $\gamma^* = \inf f(\Omega)$. A selection rule is *exact* if:

(i) $\inf(\Omega \cap M) \geq \gamma^*$ for all M such that $M \in \mathcal{R}_k$ for all k (i.e. each region that remains qualified forever in the solution process is such that $\inf(\Omega \cap M) \geq \gamma^*$);

(ii) the limit M_∞ of any filter $\langle M_k \rangle$ is such that $\inf f(\Omega \cap M_\infty) \geq \gamma^*$.

9.2.2 Theorem
A B&S algorithm using an exact selection rule converges.

Proof. Suppose, to get a contradiction, that there is $x \in \Omega$ with $f(x) < \gamma^*$. Let $x \in M$ with $M \in \mathcal{R}_n$ for some $n \in \mathbb{N}$. Because of condition (i) above, M cannot remain qualified forever; furthermore, unqualified regions may not, by hypothesis, include points with better objective function values than the current incumbent γ_k. Hence M must necessarily be split at some iteration $n' > n$. So x belongs to every M_n in some filter $\{M_n\}$, thus $x \in \Omega \cap M_\infty$. By condition (2) above, $f(x) \geq \inf f(\Omega \cap M_\infty) \geq \gamma^*$. The result follows. □

We remark that this algorithmic framework does not provide a guarantee of convergence in a finite number of steps. Consequently, most B&S implementations make use of the concept of ε-optimality, (Sect. 4.2.2). By employing this notion and finding lower and upper bound sequences converging to the incumbent at each step of the algorithm, finite termination can be ensured. For finite termination with $\varepsilon = 0$, some additional regularity assumptions are needed (see e.g. [272, 12]). While ε-optimality (with $\varepsilon > 0$) is sufficient for most practical purposes, it is worth pointing out that a ε-optimum x' is such that $f(x')$ is at most ε away from the true globally optimal objective function value f^*: this says nothing at all about the distance between x' and a true global optimum x^*; in fact, $\|x' - x^*\|_2$ might be arbitrarily large.

9.2.4.1 Fathoming

Let $\mathcal{S}_k$ be the net at iteration k. For each region $M \in \mathcal{S}_k$, find a lower bounding solution x^*_M for $f(x)$ defined on $\Omega \cap M$ and the corresponding lower bounding objective function value $\ell(M)$. A set $M \in \mathcal{S}_k$ is qualified if $\ell(M) \leq \gamma_k$. The distinguished region is usually selected as the one with the lowest $\ell(M)$.

The algorithm is accelerated if one also computes an upper bound $u(M)$ for $\inf f(x)$ on $\Omega \cap M$ and uses it to bound the problem from above by rejecting any M for which $\ell(M)$ exceeds the best upper bound $u(M)$ encountered so far. In this case, we say that the rejected regions have been *fathomed* (Fig. 9.5). This acceleration device has become part of all BB implementations. Usually, the upper bound $u(M)$ is computed by solving the problem to local optimality in the current region: this also represents a practical alternative to the implementation of the evaluation step (Step (2) of the B&S algorithm), since we can take the distinguished points $\omega(M)$ to be the local solutions $u(M)$ of the problem in the current regions. With a numerically precise local phase, the accuracy of ε-global optima is likely to be better than if we simply use $\ell(M)$ to define the distinguished points.

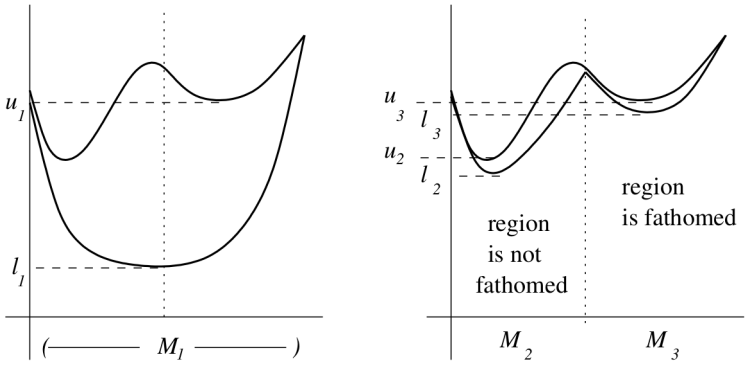


Figure 9.5: Fathoming via upper bound computation.

The convergence of B&S with fathoming is ensured if every filter $\{M_k | k \in K\}$ contains an infinite

nested sequence $\{M_k | k \in K_1 \subseteq K\}$ such that:

$$\lim_{\substack{k\to\infty \\ k\in K_1}} \ell(M_k) = \gamma^*. \tag{9.2}$$

To establish this, we will show that under such conditions the selection procedure is exact, and then invoke Thm. 9.2.2. Let $\{M_k | k \in K\}$ be any filter and M_∞ its limit. Because of equation (9.2), $\inf f(\Omega \cap M_k) \geq \ell(M_k) \to \gamma^*$, hence $\inf f(\Omega \cap M_\infty) \geq \gamma^*$. Furthermore, if $M \in \mathcal{R}_k$ for all k then $\inf f(\Omega \cap M) \geq \ell(M) \geq \ell(M_k) \to \gamma^*$ as $k \to \infty$, i.e. $\inf f(\Omega \cap M) \geq \gamma^*$. Thus the selection procedure is exact and, by theorem 9.2.2, the B&S algorithm with fathoming converges.

9.2.5 Example of solution by B&S

The formulation

$$\min\{f(x) = \frac{1}{4}x + \sin(x) \mid x \geq -3, x \leq 6\}$$

describes a simple NLP with two minima, only one of which is global (Fig. 9.6). We solve it (graphically) with a sBB approach: at each iteration we find lower and upper bounds to the optimum within a subregion of the search space; if these bounds are not sufficiently close, say to within a $\varepsilon > 0$ tolerance, we split the subregion in two, repeating the process on each subregion until all the regions have been examined). In this example we set ε to 0.15, but note that this is not realistic: in practice ε is set to values between 1×10^{-6} and 1×10^{-3}. At the first iteration we consider the region consisting of the whole x variable

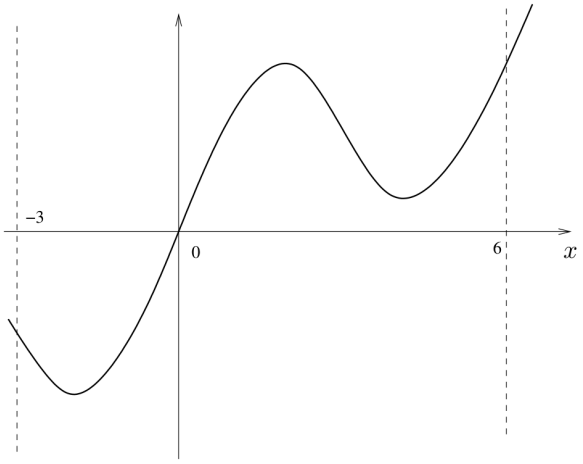


Figure 9.6: The problem $\min\{\frac{1}{4}x + \sin(x) \mid x \geq -3, x \leq 6\}$.

range $-3 \leq x \leq 6$. We calculate a lower bound by underestimating the objective function with a convex function: since for all x we have $-1 \leq \sin(x) \leq 1$, the function $\frac{1}{4}x - 1$ is a convex underestimator of the objective function. Limited to this example only, We can draw the tangents to $f(x)$ at the range endpoints $x = -3$ and $x = 6$ to make the underestimator tighter, as in Fig. 9.7. The whole underestimator is the pointwise maximum of the three linear functions emphasized in Fig. 9.7. The tangent to the objective function in the point $x = -3$ is the line $y = -0.74x - 3.11$, whereas in the point $x = 6$ it is $y = 1.21x - 6.04$. Thus the convex underestimator is $\bar{f}(x) = \max\{-0.74x - 3.11, \frac{1}{4}x - 1, 1.21x - 6.04\}$. Finding the minimum of $\bar{f}(x)$ in the region $-3 \leq x \leq 6$ yields a solution $\bar{x} = -2.13$ with objective function value $l = -1.53$ which is a valid lower bound for the original problem (indeed, the value of the original objective function at $\bar{x}$ is -1.42). Now we find an upper bound to the original objective function by locally solving the original problem. Suppose we pick the range endpoint $x = 6$ as a starting point and we employ Newton's method in one dimension: we find a solution $\tilde{x} = 4.46$ with objective function value $u = 0.147$. Since

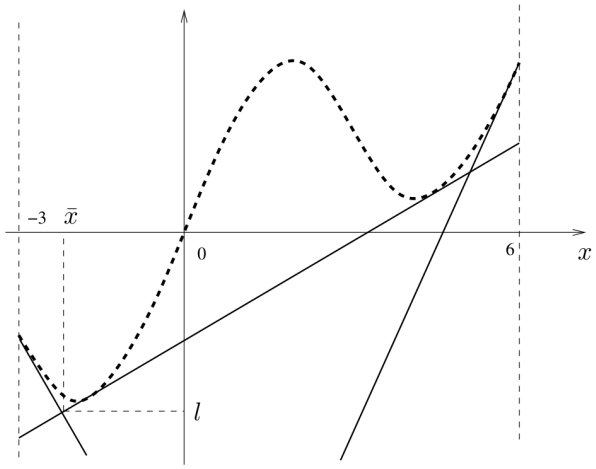


Figure 9.7: Convex underestimation of the objective function in Sect. 9.2.5 and optimal solution of the convex underestimating problem.

$|u - l| = |0.147 - (-1.53)| = 1.67 > \varepsilon$ shows that u and l are not sufficiently close to be reasonably sure that $\tilde{x}$ is the global optimum of the region under examination, we split this region in two. We choose the current $\tilde{x} = 4.46$ as the branching point, we add two regions $-3 \leq x \leq 4.46$, $4.46 \leq x \leq 6$ to a list of "unexamined regions" and we discard the original region $-3 \leq x \leq 6$. At the second iteration we pick the region $-3 \leq x \leq 4.46$ for examination and compute lower and upper bounds as in Fig. 9.8. We find

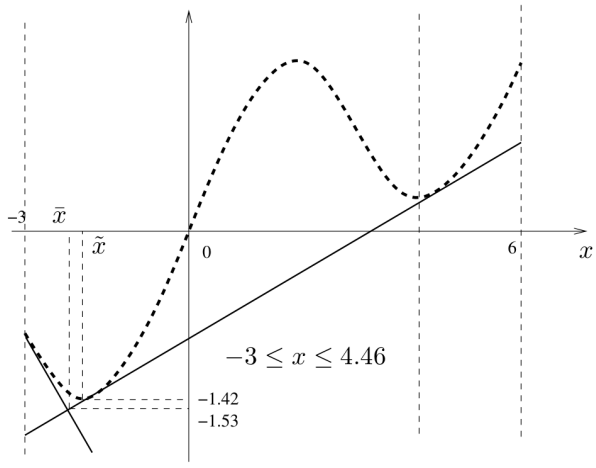


Figure 9.8: Second iteration of the sBB algorithm.

the same underestimating solution $\bar{x}$ as before (with $l = -1.53$) and by deploying Newton's method from the endpoint $x = -3$ we locate the local optimum $\tilde{x} = -1.823$ with $u = -1.42$. Since $|u - l| = 0.11 < \varepsilon$, we accept $\tilde{x}$ as the global optimum for the region under examination, and we update the "best solution found" $x^* = -1.823$ with associated function value $U = -1.42$. Since we have located the global optimum for this region, no branching occurs. This leaves us with just one region to examine, namely $4.46 \leq x \leq 6$. In the third iteration, we select this region and we compute lower and upper bounds (Fig. 9.9) to find $\bar{x} = \tilde{x} = 4.46$ with $l = 1.11$ and $u = 1.147$. Since $|u - l| = 0.04 < \varepsilon$, we have located the global optimum for this region, so there is no need for branching. Since $U = -1.42 < 1.147 = u$, we do not update the "best solution found", so on discarding this region from the list, the list is empty. The algorithm

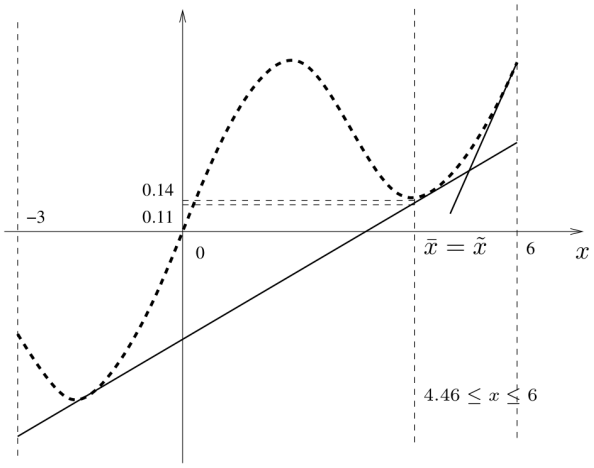


Figure 9.9: Third iteration of the sBB algorithm.

terminates with global optimum $x^* = -1.823$.

This example shows most of the important features of a typical sBB algorithm run, save three:

(a) the convex underestimation can (and usually is) modified in each region, so that it is tighter;

(b) in formulations with many variables, the choice of the branching variable is a crucial task. This will be discussed in detail in Sect. 9.4;

(c) in practice, pruning of region occurs when the lower bound in a region is higher than the current best solution.

In this example there was no pruning, as the region $4.46 \leq x \leq 6$ did not need to be branched upon. Had there been need for branching in that region, at the subsequent iterations two new regions would have been created with an associated lower bound $l = 1.11$. Since the current best solution has an objective function value of -1.42, which is strictly smaller than 1.11, both regions would have been pruned before being processed.

9.3 Variable Neighbourhood Search

The VNS approach was briefly sketched in Sect. 9.2.3.3. Here we describe a version of VNS which addresses nonconvex NLP formulations. An extension to MINLP was proposed in [189].

As mentioned in Sect. 9.2.3.3, VNS rests on sampling starting points, a local phase, and a neighbourhood structure designed both to delimit sampling, and to make it easier to escape from current local minima towards improved ones. The pseudocode for the VNS algorithm is given in Alg. 1

We adapt Alg. 1 to Eq. (6.9) in the following fashion.

- We employ a local NLP solver, such as SQP (Sect. 9.1), as the local phase delivering the current local minma x^*, x'.
- We consider a neighbourhood structure formed by a set of embedded hyper-rectangles, each side of which has length proportional to the corresponding variable range length and scaled by $r_k = \frac{k}{k_{\max}}$ all "centered" at the incumbent x^*.

Algorithm 1 The VNS algorithm.

0: *Input:* maximum number of neighbourhoods $k_{\max}$, number of local searches in each neighbourhood L
0: *Output:* a putative global optimum x^*.
loop
Set $k \leftarrow 1$, pick a random point $\tilde{x}$, perform a local search to find a local minimum x^*.
while $k \leq k_{\max}$ **do**
Consider a neighbourhood $N_k(x^*)$ of x^* s.t. $\forall\, k > 1$ $(N_k(x^*) \supset N_{k-1}(x^*))$.
for $i = 1$ to L **do**
Sample a random point $\tilde{x}$ from $N_k(x^*)$.
Perform a local search from $\tilde{x}$ to find a local minimum x'.
If x' is better than x^*, set $x^* \leftarrow x'$, $k \leftarrow 0$ and exit the FOR loop.
end for
Set $k \leftarrow k + 1$.
Verify termination condition; if true, exit.
end while
end loop

- Termination conditions based on maximum allowable CPU time.

More formally, let $H_k(x^*)$ be the hyper-rectangle $y^L \leq x \leq y^U$ where, for all $i \leq n$,

$$\begin{aligned} y_i^L &= x_i^* - \frac{k}{k_{\max}}(x_i^* - x_i^L) \\ y_i^U &= x_i^* + \frac{k}{k_{\max}}(x_i^U - x_i^*). \end{aligned}$$

This construction forms a set of hyper-rectangles "centered" at x^* and proportional to $x^L \leq x \leq x^U$. Two strategies are possible to define each neighbourhood $N_k(x^*)$:

- $N_k(x^*) = H_k(x^*)$;
- $N_k(x^*) = H_k(x^*) \setminus H_{k-1}(x^*)$,

for all $k \leq k_{\max}$.

This extremely simple algorithm has a very small number of configurable parameters: the maximum neighbourhood size $k_{\max}$, the number of sampling points and local searches started in each neighbourhood (L in Alg. 1), a ε tolerance to allow accepting a new optimum, and the maximum CPU time allowed for the search. Notwithstanding its simplicity, this was found to be a very successful algorithm for nonconvex NLP [179].

9.4 Spatial Branch-and-Bound

In this section we shall give a detailed description of the sBB algorithm, which belongs to the B&S class (Sect. 9.2.4). This provides an example of a deterministic global phase. Although it is designed to address nonconvex NLP as in Eq. (6.9), an extension to address Eq. (2.2) in full generality can be provided quite simply by adding the possibility of branching on integer variables, as is done by the BB algorithm of Sect. 8.3. The convergence proof theorem 9.2.2 also covers the sBB algorithm described in this section.

The history of sBB algorithms for nonconvex NLP starts in the 1990s [256, 20, 5, 6, 4, 106, 158]. Here, we follow a particular type of sBB algorithm called *spatial Branch-and-Bound with symbolic reformulation* [276, 277, 170].

sBB algorithms all rely on the concept of a *convex relaxation* of the original nonconvex NLP. This is a convex NLP formulation, the solution of which provides a guaranteed lower bound to the optimal objective function value of the original NLP. At each iteration, the sBB algorithm solves restrictions to particular subregions of space of the original formulation and of its convex relaxation, in order to obtain upper and lower bounds to the optimal value of the objective function value in the subregion. If the bounds are very close, a global optimum relative to the subregion has been identified. The selection rule choosing next subregions makes it possible to exhaustively explore the search space. When subregions are defined by hyper-rectangles (i.e. simple variable ranges), starting with the smallest possible ranges speeds up the sBB considerably. This issue is addressed in Sect. 9.4.1.

Most sBB algorithms have the following form:

1. (*Initialization*) Initialize a list of regions to a single region comprising the entire set of variable ranges. Set the convergence tolerance $\varepsilon > 0$; set the best objective function value found up so far $U := \infty$ and the corresponding solution $x^* := (\infty, \ldots, \infty)$. Optionally, perform *Optimization-Based Bounds Tigthening* (OBBT), see 9.4.1.1.
2. (*Choice of Region*) If the region list is empty, terminate the algorithm with solution x^* and objective function value U. Otherwise, choose a region R (the "current region") from the list (see Sect. 9.4.2). Remove R from the list. Optionally, perform *Feasibility-Based Bounds Tightening* on R (see Sect. 9.4.1.2).
3. (*Lower Bound*) Generate a convex relaxation from the original formulation in the selected region R (see Sect. 9.4.3), and solve it to obtain an underestimation ℓ of the objective function, with corresponding solution $\bar{x}$. If $\ell > U$ or the relaxation is infeasible, go back to Step 2.
4. (*Upper Bound*) Attempt to solve the original formulation restricted to the selected region to obtain a (locally optimal) solution $\tilde{x}$ with objective function value u (see Sect. 9.4.4). If this fails, set $u := +\infty$ and $\tilde{x} = (\infty, \ldots, \infty)$.
5. (*Pruning*) If $U > u$, set $x^* = \tilde{x}$ and $U := u$. Remove all regions from the list having lower bounds greater than U, since they cannot possibly contain the global minimum.
6. (*Check Region*) If $u - \ell \leq \varepsilon$, accept u as the global minimum for this region and return to Step 2. Otherwise, we may not have located the global minimum for the current region yet, so we proceed to the next step.
7. (*Branching*) Apply a branching rule to the current region, in order to split it into subregions (see Sect. 9.4.5). Append these to the list of regions, assigning to them ℓ as an (initial) lower bound. Go back to Step 2.

The convex relaxation is generated automatically by means of a symbolic analysis of the mathematical expressions occurring in the original formulation. Below, we consider some of the key steps of the algorithm in more detail.

9.4.1 Bounds tightening

Bounds tightening procedures appear in steps 1 and 2. They are optional in the sense that the algorithm will converge even without them. In practice, however, bounds tightening is essential. Two major bounds tightening schemes have been proposed in the literature: OBBT and FBBT.

9.4.1.1 Optimization-based bounds tightening

OBBT identifies the smallest range of each variable, subject to feasibility of the convex relaxation. This prevents the sBB algorithm from exploring subregions that can be proved infeasible *a priori*.

OBBT requires the solution of at least $2n$ convex relaxations, where n is the number of variables. Let $\alpha \leq \bar{g}(x) \leq \beta$ be the set of constraints in the convex relaxation: the OBBT procedure, given below, constructs sequences $x^{L,k}, x^{U,k}$ of possibly tighter lower and upper variable bounds.

1. Set $x^{L,0} \leftarrow x^L$, $x^{U,0} \leftarrow x^U$, $k \leftarrow 0$.

2. Repeat

$$\begin{aligned} x_i^{L,k} &\leftarrow \min\{x_i \mid \alpha \leq \bar{g}(x) \leq \beta \wedge x^{L,k-1} \leq x \leq x^{U,k-1}\}, \quad \forall i \leq n; \\ x_i^{U,k} &\leftarrow \max\{x_i \mid \alpha \leq \bar{g}(x) \leq \beta \wedge x^{L,k-1} \leq x \leq x^{U,k-1}\}, \quad \forall i \leq n; \\ k &\leftarrow k+1. \end{aligned}$$

until $x^{L,k} = x^{L,k-1}$ and $x^{U,k} = x^{U,k-1}$.

Because of its computational cost, OBBT is usually only performed at the root node of the sBB.

9.4.1.2 Feasibility-based bounds tightening

FBBT is computationally cheaper than OBBT, and as such it can be applied at each region. Variable ranges are tightened using the constraints, which are exploited in order to calculate extremal values attainable by the variables. This is done by isolating a variable on the LHS of a constraint and evaluating the extremal values of the RHS by means of interval arithmetic.

FBBT is easiest for the case of linear constraints. Given constraints $l \leq Ax \leq u$ where A is an $m \times n$ matrix, interval analysis immediately yields:

$$\begin{aligned} x_j \in & \left[\max\left(x_j^L, \min_i\left(\tfrac{1}{a_{ij}}\left(l_i - \sum_{k\neq j}\max(a_{ik}x_k^L, a_{ik}x_k^U)\right)\right)\right) \right. \\ & \left. \min\left(x_j^U, \max_i\left(\tfrac{1}{a_{ij}}\left(u_i - \sum_{k\neq j}\min(a_{ik}x_k^L, a_{ik}x_k^U)\right)\right)\right)\right] \quad \text{if } a_{ij} > 0 \\ x_j \in & \left[\max\left(x_j^L, \min_i\left(\tfrac{1}{a_{ij}}\left(l_i - \sum_{k\neq j}\min(a_{ik}x_k^L, a_{ik}x_k^U)\right)\right)\right) \right. \\ & \left. \min\left(x_j^U, \max_i\left(\tfrac{1}{a_{ij}}\left(u_i - \sum_{k\neq j}\max(a_{ik}x_k^L, a_{ik}x_k^U)\right)\right)\right)\right] \quad \text{if } a_{ij} < 0 \end{aligned}$$

for all $1 \leq j \leq n$. As pointed out in [276, p. 202], FBBT can also be carried out for *factorable* nonlinear constraints.

9.4.1 Remark (Factorable expressions)
A constraint is factorable if it involves factorable expressions. Although a precise definition of factorability is provided in [215], a more useful characterization is "an expression which can be parsed using an arithmetic expression language" (see Sect. 2.1.1). In this sense, all expressions occurring in this book are factorable.

9.4.2 Choice of region

The region selection at Step 2 follows the simple policy of choosing the region in the list with the smallest lower objective function bound ℓ. Intuitively, this is the most promising region for further analysis.

9.4.3 Convex relaxation

The convex relaxation solved at Step 3 of the sBB algorithm aims at finding a guaranteed lower bound to the objective function value in the current region. It is generated automatically for Eq. (2.2) in two stages.

The formulation is first reduced to a standard form where each nonlinear term is linearized, i.e. replaced by an additional variable. Each linearization is then encoded in a new constraint of the form:

$$\text{additional variable} = \text{linearized nonlinear term}.$$

These constraints, called *defining constraints*, are all adjoined to the formulation: this yields a formulation where nonlinear terms only occur in defining constraints.

In the second stage, each defining constraint is replaced by pair of constraints, each defining a convex portion of space. One provides a convex underestimator, and the other a concave overestimator. This pair of convex estimating constraints is known as *convexification* (the union of all such pairs is also known as "convexification").

9.4.3.1 Reformulation to standard form

The symbolic reformulation algorithm scans the expression trees (see Sect. 2.1.1 recursively from the leaves. If the current node is a nonlinear operator $\otimes$ with arity k, it will have k subnodes $s_1^{\otimes}, \ldots, s_k^{\otimes}$, each of which is either a variable x_j in the original formulation or an additional variable w_h (by recursion) for some index h. We then create a new additional variable w_p, where p is the next available unused additional variable index, and adjoin the defining constraint

$$w_p = \otimes(x, w)$$

to the standard form reformulation, where (x, w) is an appropriate sequence of k variables (some of which may be original and some additional). The new additional variable w_p replaces the subexpression represented by the subtree rooted at the current node $\otimes$. See [276, 277] for more details.

9.4.2 Remark
Note that the linearization process is not well-defined, as there may be different linearizations corresponding to a single mathematical expression, depending on associativity. For example, the expression $x_1x_2x_3$ might be linearized as $w_1 = x_1x_2 \wedge w_2 = w_1x_3$, $w_1 = x_1x_3 \wedge w_2 = w_1x_2$ or as $w_1 = x_2x_3 \wedge w_2 = w_2x_3$, but also simply as $w_1 = x_1x_2x_3$, as long as the symbolic algorithm is able to recognize the product operator as an arbitrary k-ary operator (rather than just binary).

In general, if a tight convexification is available for a complicated subexpression, it is advisable to generate a single additional variable linearizing the whole subexpression [37, 60].

A standard form reformulation based on a symbolic parser which recognizes nonlinear operators of arity $k \in \{1, 2\}$ is as follows:

$$\min v_\omega \tag{9.3}$$
$$a \leq Av \leq b \tag{9.4}$$
$$v_k = v_i v_j \qquad \forall (i,j,k) \in \mathcal{M} \tag{9.5}$$
$$v_k = \frac{v_i}{v_j} \qquad \forall (i,j,k) \in \mathcal{D} \tag{9.6}$$
$$v_k = v_i^\nu \qquad \forall (i,k,\nu) \in \mathcal{P} \tag{9.7}$$
$$v_k = \phi_\mu(v_i) \qquad \forall (i,k,\mu) \in \mathcal{U} \tag{9.8}$$
$$v^L \leq v \leq v^U \tag{9.9}$$

where $v = (v_1, \ldots, v_q)$ are the decision variables (including both original and additional variables), $\omega \leq q$, A is the constraint matrix, a, b are the linear constraint bound vectors, v^L, v^U are the variable bounds vectors, and ϕ_μ is a given sequence of labels denoting unary functions such as log, exp, sin, cos, which the parser is aware of.

The defining constraints (9.5)-(9.8) encode all of the nonlinearities occurring in the original formulation: $\mathcal{M}, \mathcal{D}$ are sets of variable index triplets defining bilinear and linear fractional terms respectively. $\mathcal{P}$ is a set of variable index triplets defining power terms (each triplet consists of two variable indices i, k and the corresponding power exponent $\nu \in \mathbb{R}$). $\mathcal{U}$ is a set of triplets which define terms involving the univariate function labels ϕ_μ (each triplet consists of two variable indices i, k and a label index μ).

Note that we also replace the objective function with the additional variable x_ω, corresponding to a defining constraint of the form $x_\omega =$ objective function.

9.4.3 Example

In this example we show the reformulation obtained for Sect. 9.2.5, together with its graphical representation. Since there is only one nonconvex term in the problem (namely, $\sin(x)$*), the reformulation is immediate:*

$$\left.\begin{array}{ll} \min\limits_{x,y} & \frac{1}{4}x + y \\ s.t. & y = \sin(x) \\ & -3 \leq x \leq 6 \\ & -1 \leq y \leq 1. \end{array}\right\}$$

The standard form reformulation is a lifting, since introducing new variables lifts the formulation into a higher-dimensional space. This is immediately apparent in this simple example. Compare Fig. 9.6 (the original formulation) and its reformulation shown in Fig. 9.10.

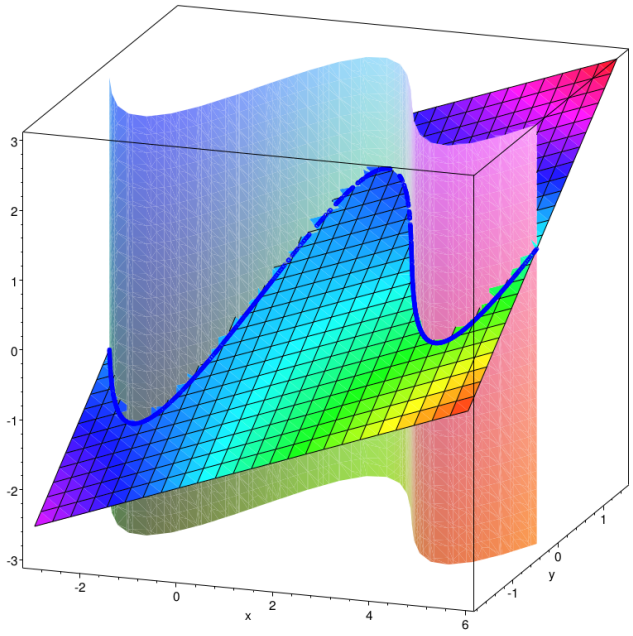


Figure 9.10: The problem $\min\{\frac{1}{4}x + \sin(x) \mid x \geq -3, x \leq 6\}$ reformulated to standard form.

9.4.4 Example (Reformulation of the HPP)
The reformulation obtained for the HPP (see Sect. 2.2.7.7) is as follows:

$$
\begin{array}{lll}
\min\limits_{x,y,p,w} & 6x_{11} + 16x_{21} + 10x_{12} - 9(y_{11} + y_{21}) - 15(y_{12} + y_{22}) & \textit{cost} \\
s.t. & x_{11} + x_{21} - y_{11} - y_{12} = 0 & \textit{mass balance} \\
& x_{12} - y_{21} - y_{22} = 0 & \textit{mass balance} \\
& y_{11} + y_{21} \leq 100 & \textit{demands} \\
& y_{12} + y_{22} \leq 200 & \textit{demands} \\
& 3x_{11} + x_{21} - w_1 = 0 & \textit{sulphur balance} \\
& w_1 = pw_2 & \textit{defining constraint} \\
& w_2 = y_{11} + y_{12} & \textit{(linear) defining constraint} \\
& w_3 + 2y_{21} \leq 2.5(y_{11} + y_{21}) & \textit{quality requirement} \\
& w_3 = py_{11} & \textit{defining constraint} \\
& w_4 + 2y_{22} \leq 1.5(y_{12} + y_{22}) & \textit{quality requirement} \\
& w_4 = py_{12}. & \textit{defining constraint}
\end{array}
$$

Note that linear terms do not undergo any linearization. Among the defining constraints, some are nonlinear (hence nonconvex, since they are equality constraints) and some are linear. This conveniently splits the problem into a linear and a nonlinear part.

9.4.3.2 Convexification

The convex relaxation of the original formulation consists of the convexification of the defining constraints in the standard form. The convexification depends on the current region $[v^L, v^U]$, and is generated as follows:

1. The defining constraint $v_i = v_j v_k$ is replaced by four linear inequalities:

$$
\begin{aligned}
v_i &\geq v_j^L v_k + v_k^L v_j - v_j^L v_k^L && (9.10)\\
v_i &\geq v_j^U v_k + v_k^U v_j - v_j^U v_k^U && (9.11)\\
v_i &\leq v_j^L v_k + v_k^U v_j - v_j^L v_k^U && (9.12)\\
v_i &\leq v_j^U v_k + v_k^L v_j - v_j^U v_k^L. && (9.13)
\end{aligned}
$$

   Note that Eq. (9.10)-(9.13) provide the *convex envelope* of the set $B(i,j,k) = \{(v_i, v_j, v_k) \mid v_i = v_j v_k \wedge v \in [v^L, v^U]\}$ (i.e. the smallest convex set containing $B(i,j,k)$).

2. The defining constraint $v_i = v_j / v_k$ is reformulated to $v_i v_k = v_j$, and the convexification rules for bilinear terms (9.10)-(9.13) are applied. Note that this introduces the possibility that $v_k = 0$ is feasible in the convex relaxation, whereas the original constraint where v_k appears in the denominator forbids it.

3. The defining constraints $v_i = \phi_\mu(v_j)$ where ϕ_μ is concave univariate is replaced by two inequalities: the function itself and the secant:

$$
\begin{aligned}
v_i &\leq f_\mu(v_j) && (9.14)\\
v_i &\geq f_\mu(v_j^L) + \frac{f_\mu(v_j^U) - f_\mu(v_j^L)}{v_j^U - v_j^L}(v_j - v_j^L). && (9.15)
\end{aligned}
$$

4. The defining constraint $v_i = \phi_\mu(v_j)$ where ϕ_μ is convex univariate is replaced by:

$$
\begin{aligned}
v_i &\leq f_\mu(v_j^L) + \frac{f_\mu(v_j^U) - f_\mu(v_j^L)}{v_j^U - v_j^L}(v_j - v_j^L) && (9.16)\\
v_i &\geq f_\mu(v_j). && (9.17)
\end{aligned}
$$

5. The defining constraint $v_i = v_j^{\nu}$, where $0 < \nu < 1$, is treated as a concave univariate function in the manner described in Step 3 above.

6. The defining constraint $v_i = v_j^{2m}$ for any $m \in \mathbb{N}$ is treated as a convex univariate function in the manner described in Step 4 above.

7. The defining constraint $v_i = v_j^{2m+1}$ for any $m \in \mathbb{N}$ may be convex, concave, or piecewise convex and concave with a turning point at 0. If the range of v_j does not include 0, the function is convex or concave and falls into a category described above. A complete discussion of convex underestimators and concave overestimators for this term when the range includes 0 can be found in [190].

8. Other defining constraints involving trigonometric functions need to be analysed on a case-by-case basis, similarly to Step 7.

9.4.4 Local solution of the original problem

This is usually obtained by deploying a local phase on the original formulation restricted to the current region (see Sect. 9.1). This is also typically the most computationally expensive step in each iteration of the sBB algorithm.

9.4.4.1 Branching on additional variables

An issue arises when branching on additional variables (say w_h): when this happens, the current region is partitioned into two subregions along the w_h axis, the convex relaxation is modified to take the new variable ranges into account, and lower bounds are found in each subregion.

The upper bounds, however, are found by solving the original formulation, which is not dependent on the additional variables. Thus the same original formulation is solved at least three times in the course of the algorithm (i.e. once for the original region and once for each of its two sub-regions).

We address this issue by storing the upper bounds to each region. Whenever the branching variable is an additional variable, we use the stored bounds rather than deploying the local phase.

9.4.4.2 Branching on original variables

Even when branching occurs on an original variable, there are some considerations that help avoid solving local optimization problems unnecessarily. Suppose that the original variable x is selected for branching in a certain region. Then its range $[x^L, x^U]$ is partitioned into $[x^L, x']$ and $[x', x^U]$. If the solution of the original formulation restricted to $[x^L, x^U]$ is x^*, and $x^* \in [x^L, x']$, then deploying the local phase in the subregion $[x^L, x']$ is unnecessary. Of course, the lolca phase still needs to be deployed on the other subregion $[x', x^U]$ (see Fig. 9.11).

9.4.5 Branching

There are many branching strategies [105, 38] available for use in sBB algorithms, most of which are based on branching along coordinate directions, i.e. by partitioning the variable ranges. It is known that best results on some particularly difficult MILP instances (e.g. [75]) are achieved by branching along constraints [76], but such branching schemes have apparently not been used in sBB algorithms yet. Other approaches include [196, 79, 80, 251]. Henceforth, we shall assume that "branching" implies "along coordinate directions".

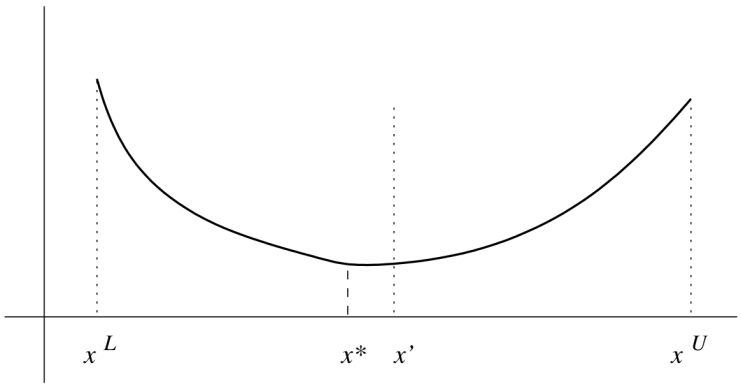


Figure 9.11: If the locally optimal solution in $[x^L, x^U]$ has already been determined to be at x^*, solving in $[x^L, x']$ is unnecessary.

Generally, branching involves two steps: determining the index of the variable on which to branch (called *branching variable*), and determining the point (called *branching point*) on which to partition the branching variable range. Several heuristics exist (see [276, p. 205-207] and [38]). Here, we use the upper bounding solution $\tilde{x}$ (obtained in Step 4 of the sBB algorithm) as the branching point, if such a solution is found; otherwise the lower bounding solutoin $\bar{v}$ (Step 3) is used. Recall that $v = (x, w)$, where x are the original variables and w the additional ones.

We then employ the standard form to identify the nonlinear term with the largest error with respect to its convex relaxation. By definition of the standard form Eq. (9.3) (see Sect. 9.4.3.1), this is equivalent to evaluating the defining constraints (9.5)-(9.8) at $\bar{v}$ and choosing the one giving rise to the largest error. In case the chosen defining constraint represents a unary operator, the only variable operand is chosen as the branching variable; if it represents a binary operator, the branching variable is chosen as the one having branching point value closest to the range midpoint.

Part IV

Advanced Topics

Chapter 10

Distance Geometry

Distance Geometry (DG) is the study of geometry with the basic entity being distance (instead of lines, planes, circles, polyhedra, conics, surfaces and varieties). Its interest in this context is that it provides an a good problem for studying certain current conic techniques in MP. The problem of interest is the reverse of the following direct problem: given a set V of points in $\mathbb{R}^K$, find some of their pairwise distances, and use them to define and weigh the edges of a graph defined on V. The reverse problem is

> Distance Geometry Problem (DGP). Given a simple edge-weighted graph $G = (V, E)$ where $d : E \to \mathbb{R}_+$ is the edge weight function, and an integer K, determine whether there exists a mapping $x : V \to \mathbb{R}^K$ such that:
>
> $$\forall \{i, j\} \in E \quad \|x_i - x_j\| = d_{ij}. \tag{10.1}$$

We remark that we denote $x(i), x(j)$ by x_i, x_j and $d(\{i, j\})$ by d_{ij}. If a mapping x satisfies Eq. (10.1), it is called a *realization*. Sometimes we refer to "invalid realizations" for mappings that do not satisfy Eq. (10.1), and to "approximate realizations" for mappings that satisfy Eq. (10.1) with an approximation error. In Eq. (10.1), the norm can be arbitrary, but we mostly use ℓ norms with $\ell \in \{1, 2, \infty\}$. When the norm has $\ell = 2$, we usually use the definition

$$\forall \{i, j\} \in E \quad \|x_i - x_j\|_2^2 = d_{ij}^2. \tag{10.2}$$

10.1 Some applications of DG

We review some important DG applications briefly.

10.1.1 Clock synchronization

Alice just told you her watch is wrong by 5 minutes, Bob's is wrong by 7 with respect to Alice's, Charles' by 3 with respect to Alice's and 4 with respect to Bob's. Exasperated, you check the atomic clock public website to find that it is precisely 16:27. Can you find out the time on Alice's, Bob's and Charles' watches? Is there only one solution, or can there be many?

When $K = 1$, a typical application of DG is that of clock synchronization [275]. Network protocols for wireless sensor networks are designed so as to save power in communication. When synchronization and battery usage are key, the peer-to-peer communications needed to exchange the timestamp can be

limited to the exchange of a single scalar, i.e. the time (or phase) difference. The problem is then to retrieve the absolute times of all of the clocks, given some of the phase differences. This is equivalent to a DGP on the time line, i.e. in a single dimension.

More precisely, let $K = 1$ and V be a set of clocks. An edge $\{i, j\}$ is in E if the time difference between clocks i and j is d_{ij}. Suppose that each clock i knows its own time as well as the weighted graph G. Then a solution $x^* \in \mathbb{R}^1 = \mathbb{R}$ is a consistent evaluation of the absolute times of each clock in V.

10.1.2 Sensor network localization

WIFI-enabled smartphones can create what is known as an "ad-hoc network", i.e. they can create a WIFI network where each communication is peer-to-peer, as long as the distance between two smartphones is not excessive. Also, smartphones can estimate pairwise distances (with close enough peers) by measuring how much battery they use to send/receive a data packet: the higher the battery consumption, the larger the distance. The network administrator at headquarters must locate each person in the building at any time: (s)he defines a protocol that has every corporate smartphone send out the distance to each of its neighbours to a central server. Using these distances, the problem is that of finding positions for each smartphone (and hopefully its owner), see Fig. 10.1.

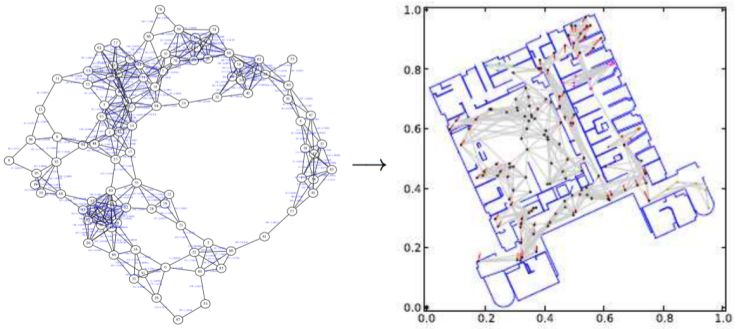


Figure 10.1: Realizing the graph in $\mathbb{R}^2$.

Let $K = 2$ and V be a set of mobile devices in a wireless network at a certain given time t. An edge $\{i, j\}$ is in E if the distance between i and j is known to be d_{ij} at time t. Such distances can be measured up to a certain threshold, for example by estimating how much battery power a peer-to-peer communication between i and j takes. By exploiting network communication, the weighted graph G can be transmitted to a central server. A solution $x^* \in \mathbb{R}^2$ gives a consistent position of the devices at time t. This application arises naturally in the localization of sensors in wireless networks [312, 46, 27, 102, 81].

This, by the way, appears to be the application that first motivated the study of the DGP. Originally, the DGP was only studied on complete graphs, i.e. there were no missing distances. As we shall see later, arbitrarily good approximations of realizations of complete graphs can be found in polynomial time. The DGP was then seen as the inverse problem to "compute the distance matrix of a set of n points in $\mathbb{R}^K$". To the best of my knowledge, the first mention of the DGP on complete graphs is [312].[1]

[1]The paper is titled "The positioning problem: a draft of an intermediate summary". Rarely was such a cautiously titled paper accepted by a conference for inclusion in the corresponding proceedings. One can almost imagine the young Yemini being pressured into submitting an incomplete work and putting up a fierce but ultimately futile resistance to taking responsibility for it.

10.1.3 Molecular structure from distance data

Let $K = 3$ and V be a set of atoms in a molecule such as e.g. a protein. It is well known that proteins interact with cells because of their geometrical (as well as chemical) properties: in other words, their shape matters. Being so small, we cannot directly observe their shape. We can, however, measure inter-atomic distances up to a certain threshold, usually 5Å to 6Å [264], using Nuclear Magnetic Resonance (NMR) [311]. Although NMR only really gives a frequency for each triplet (*atom type*, *atom type*, *length*), which means that we can only know how frequent it is to observe a certain distance between, say, a hydrogen and a carbon, using these data we can reconstruct a weighted graph where $\{i, j\}$ is in E if the distance d_{ij} between atom i and atom j is known. A solution $x^* \in \mathbb{R}^3$ gives a consistent conformation of the protein [231, 252, 310, 82, 167, 96, 63], see Fig. 10.2.

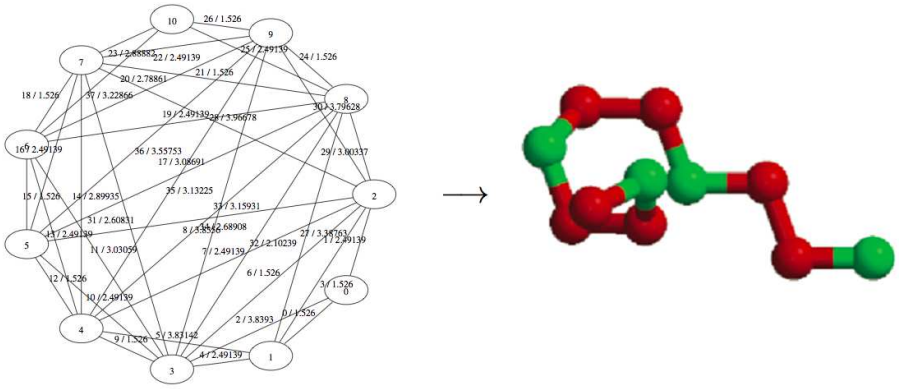


Figure 10.2: Realizing the graph in $\mathbb{R}^3$.

10.1.4 Autonomous underwater vehicles

Let $K = 3$ and V be a set of unmanned submarines, such as e.g. those used by Shell when trying to fix the oil spill in the Gulf of Mexico. Since GPS cannot be used underwater, the position of each submarine must be inferred. An edge $\{i, j\}$ is in E if the distance between submarines i and j can be obtained using the sonars. A solution $x^* \in \mathbb{R}^3$ gives a consistent snapshot of the positions of the submarines at a certain time instant.

10.1.5 Finding graph embeddings for deep learning

Artificial Neural Networks (ANN) are supervised Machine Learning (ML) models: once trained, they encode a function mapping an input vector to an output vector (usually having different dimensions and different domains). In order to use ANNs with discrete input (such as graphs), it is necessary to define a vector representation. Obvious ones, such as support vectors of incidence or adjacency structures, are usually very high-dimensional. Contemporary deep learning wisdom endows ANNs with auto-encoder modules: basically a preliminary layer which maps the very high dimensional support vectors to lower-dimensional vectors. An alternative to this view is to solve a DGP on the given graph [177, 176].

10.2 A short summary of DG history

Distance Geometry (DG) became fashionable around about 50AD, when Heron of Alexa ndria published a book called *Metrica* [137]. At the time, the Alexa ndria library was the most prominent place of learning and research in civilizat ion. It employed people who were paid to just sit and think, and solve the world 's problems. Among them, Heron was concerned with applied geometry, and came up with what is now known as Heron's theorem: a formula for computing the area of a ny triangle given the length of its sides — the first result of DG in history. It is at least plausible that a motivation for his work might have been to decr ease fights among farmers trying to enlarge their own triangular fields at the e xpense of their neighbors, e.g., by surreptitiously moving milestones a few cubi ts within a bordering field in the dead of night, hoping no-one would notice. Previously known formulæ for the area of triangles would require the measure of a height of the triangle. But walking in a straight line down from a vertex (marked by a milestone) hoping you would reach the opposite side (exactly perpendicularly) is an endeavor prone to mistakes (accidental or else). Instead, walking around the triangle touching all milestones is an easier feat. Heron made sure this was enough to measure the plowable area: if a farmer suddenly found his area smaller by a fraction, he would be able to prove it and claim justice.

DG was again fashionable throughout history: in 1766 in Saint Petersburg, in 1813 in Paris, in 1841 in Cambridge, in 1864 in London, in 1928 and in 1933 in Vienna, in 1953 at Columbia, Missouri, and in 1978 in Ithaca, New York (see Figure 10.3). After these events the activity became too frantic to be able to list it by place/date stamps, but the events themselves are important in the history of mathematics, and deserve some more attention.

The year 1766 is believed to be the time when Leonhard Euler wrote his well-known rigidity conjecture, stated in the form of a problem to be solved: *find two [triangulated] surfaces such that it is possible to [continuously] transform one into the other, in such a way that corresponding [pairs of] points on either keep the same pairwise distance.* Towards the end of this fragment [109], Euler writes *as soon as the shape is everywhere closed, it can no longer be continuously transformed.* In modern terms, we would ask: are closed polyhedra rigid? Euler simply wrote that they are, without proving the claim. The first progress on this famous conjecture was made by Augustin Cauchy in 1813, who was at the time a professor at École Polytechnique in Paris: he proved that Euler's conjecture held for strictly convex polyhedra [64]. His proof, admired for its elegance in mixing geometry and topology, contained two errors, which were fixed later. We should also explain that we mention "convex polyhedra" since we take polyhedra to be defined by their face incidence lattice and metric (which can give rise to nonconvex polyhedral surfaces) rather than as intersections of half-spaces (which are always convex). The story of this conjecture is interesting and gave rise to a considerable amount of mathematical research, see e.g., [182] for more information. Here we will only say that the conjecture was disproved by Robert Connelly, who provided in 1978 [72] an example of a closed nonconvex three-dimensional polyhedron with a hinge and a flexible movement.

Another historical trend is given by the events which occurred in 1841, 1928, and 1953. The earliest refers to Arthur Cayley publishing a paper about the geometry of position [65]. In this paper, Cayley proved that the four-dimensional volume of a four-dimensional simplex that can be embedded in three dimensions is zero. It would perhaps be tempting to react to this statement with supreme indifference. But Cayley's result is obtained as an algebraic relation on the *side lengths only*, which makes the achievement much less trivial (and in fact entirely nontrivial for Cayley's times). This result is related to the 1928 work of Karl Menger on the foundations of geometry through distances [219] (translated in [220]): in fact the determinant appearing in Cayley's paper for 5-simplices was generalized by Menger to arbitrary $(K+1)$-simplices, and is now known as the Cayley-Menger determinant. Menger's extensive work on metric spaces was organized and clearly laid out by one of his students, Leonard Blumenthal, who published his work in 1953 [48]. In a sense, this is the completion of the line of work started by Heron almost 2000 years before, as the simplex volumes in terms of the sides are direct generaliziations of Heron's theorem.

In carrying out his work, Menger, then a young professor at the University of Vienna, was following another mathematical fashion: that of axiomatization. Hilbert's influence on foundations and axiomatization was very strong in the 1930s *Mitteleuropa* [139]. This pushed many people towards axiomatizing

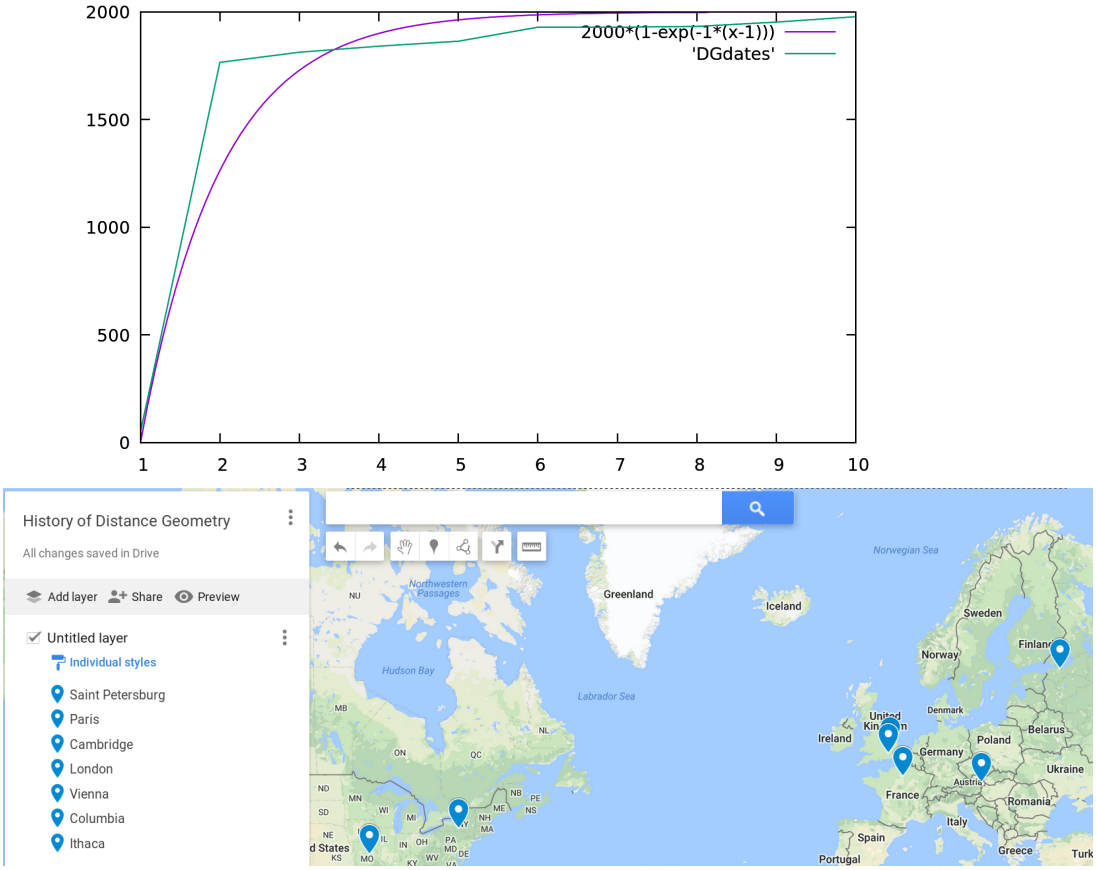


Figure 10.3: Above: the frequency of DG landmark events is reasonably well approximated by the function $2000(1-e^{-2(x-1)})$ (a "fun fact" according to a certain Sheldon Cooper). Below: geography of the history of distance geometry.

existing mathematical theories [136]. The Vienna Circle was a group of philosophers and mathematicians which convened in Vienna's *Reichsrat café* around the nineteen-thirties to discuss philosophy, mathematics and, presumably, drink coffee. When the meetings became excessively politicized, Menger distanced himself from it, and organized instead a seminar series, which ran from 1929 to 1937 [221]. A notable name crops up in the intersection of Menger's geometry students, the Vienna Circle participants, and the speakers at Menger's *Kolloquium*: Kurt Gödel. Most of the papers Gödel published in the Kolloquium's proceedings are about logic and foundations, but two, dated 1933, are about the geometry of distances on spheres and surfaces. The first [221, 18 Feb. 1932, p. 198] answers a question posed at a previous seminar by Laura Klanfer,[2] and shows that a set X of four points in any metric space, congruent to four non-coplanar points in $\mathbb{R}^3$, can be realized on the surface of a three-dimensional sphere using geodesic distances. The second [221, 17 May 1933, p. 252] shows that Cayley's relationship hold locally on certain surfaces which behave locally like Euclidean spaces.

The pace quickens: in 1935, Isaac Schoenberg published some remarks on a paper [266] by Fréchet on the *Annals of Mathematics*, and gave, among other things, an algebraic proof of equivalence between Euclidean Distance Matrices (EDM) and Gram matrices. This is almost the same proof which is nowadays given to show the validity of the classic Multidimensional Scaling (MDS) technique [53, § 12.1].

This brings us to the computer era, where the historical account ends and the contemporary treatment begins. Computers allow the efficient treatment of masses of data, some of which are incomplete and

[2] See Appendix A.

noisy. Many of these data concern, or can be reduced to, distances, and DG techniques are the subject of an application-oriented renaissance [185, 233]. Motivated by the Global Positioning System (GPS), for example, the old geographical concept of *trilateration* (a system for computing the position of a point given its distances from three known points) makes its way into DG in wireless sensor networks [107]. Wüthrich's Nobel Prize for using Nuclear Magnetic Resonance (NMR) techniques in the study of proteins brings DG to the forefront of structural bioinformatics research [133]. The massive use of robotics in mechanical production lines requires mathematical methods based on DG [255].

DG is also tightly connected with graph rigidity [126]. This is an abstract mathematical formulation of statics, the study of structures under the action of balanced forces [214], which is at the basis of architecture [291]. Rigidity of polyhedra gave rise to a conjecture of Euler's [108] about closed polyhedral surfaces, which was proved correct only for some polyhedra: strictly convex [64], convex and higher-dimensional [13], and generic (a polyhedron is generic if no algebraic relations on $\mathbb{Q}$ hold on the components of the vectors which represent its vertices) [121]. It was however disproved in general by means of a very special, non-generic nonconvex polyhedron [72].

10.2.1 A proof of Heron's theorem

Although this section is strictly speaking unnecessary to the flow of this book, I found this proof so elegant that I cannot refrain from presenting it here. This proof is due to a certain Miles Dillon Edwards, while he was a high school student at Lassiter High School, Marietta, Georgia (USA) in 2007. Further attempts to trace him and contact him unfortunately failed.

10.2.1 Theorem (Heron)
The area of any triangle with side lengths a, b, c and semiperimeter $s = \frac{1}{2}(a + b + c)$ is equal to $\sqrt{s(s-a)(s-b)(s-c)}$.

Proof. We refer to the picture in Fig. 10.4 to define symbols used in the proofs. First, we note that

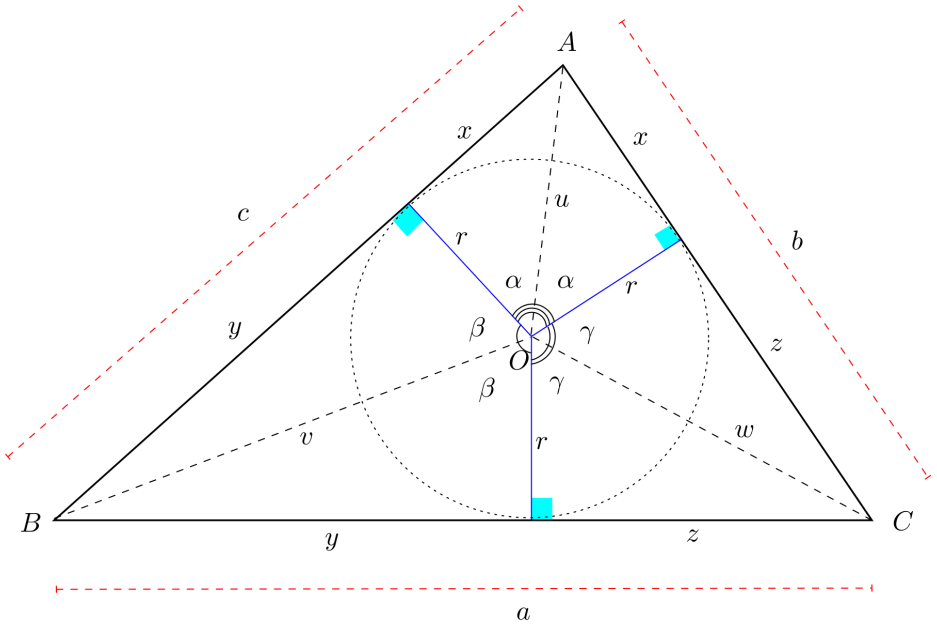


Figure 10.4: Proof of Heron's theorem.

$2\alpha + 2\beta + 2\gamma = 2\pi$, which implies $\alpha + \beta + \gamma = \pi$. We then exploit these angles to represent orthogonal vectors as complex numbers:

$$\begin{aligned} r + ix &= ue^{i\alpha} \\ r + iy &= ve^{i\beta} \\ r + iz &= we^{i\gamma}. \end{aligned}$$

Multiplying these together, we obtain $(r + ix)(r + iy)(r + iz) = (uvw)e^{i(\alpha+\beta+\gamma)} = uvw\, e^{i\pi} = -uvw$. We remark that $-uvw \in \mathbb{R}$, from which we infer that $\mathsf{Im}((r + ix)(r + iy)(r + iz)) = 0$, and hence that $r^2(x + y + z) = xyz \Rightarrow r = \sqrt{\frac{xyz}{x+y+z}}$.

We can now use this definition of r to prove the theorem as follows. We remark that $s = \frac{1}{2}(a+b+c) = x + y + z$, and that

$$\begin{aligned} s - a &= x + y + z - y - z = x \\ s - b &= x + y + z - x - z = y \\ s - c &= x + y + z - x - y = z. \end{aligned}$$

Hence the area is:

$$\frac{1}{2}(ra + rb + rc) = r\frac{a + b + c}{2} = rs = \sqrt{s(s - a)(s - b)(s - c)}$$

as claimed. □

10.3 The universal isometric embedding

Let (X, d) be a finite metric space with distance matrix $D = (d_{ij})$. An *embedding* of X into a Euclidean space of some dimension K is a mapping $\eta : X \to \mathbb{R}^K$. An embedding is *isometric* for some norm $\|\cdot\|$ if the distance matrix constructed on $\eta(X)$ using $\|\cdot\|$ is equal to D.

We now look at the ℓ_∞ norm, and ask whether there exists an isometric embedding valid for all finite metric spaces X. Surprisingly, the answer is yes. If $X = \{1, \ldots, n\}$ we let $U : X \to \mathbb{R}^n$ be the embedding defined as follows:

$$\forall i \in X \quad U(i) = (d_{i1}, \ldots, d_{in})^\top.$$

We call U the *universal isometric embedding* (UIE).

10.3.1 Theorem
The embedding U is isometric for any finite metric space X.

Proof. Denote $U(i) = (d_{i1}, \ldots, d_{in})^\top = x_i$ for each $i \in X$. We must show that for all $i \leq j$ we have $\|x_i - x_j\|_\infty = d_{ij}$. We first note that, by the triangular inequalities due to the axioms of a matrice space, we have

$$\begin{aligned} \forall i, j \in X \qquad & d_{ik} \leq d_{ij} + d_{jk} \quad \wedge \quad d_{jk} \leq d_{ij} + d_{ik} \\ \Rightarrow \quad & d_{ik} - d_{jk} \leq d_{ij} \quad \wedge \quad d_{jk} - d_{ik} \leq d_{ij} \\ \Rightarrow \quad & |d_{ik} - d_{jk}| \leq d_{ij}. \end{aligned}$$

Since these hold for all i, j, k, they also hold for the maximum over k, hence

$$\|x_i - x_j\|_\infty = \max_{k \leq n} |d_{ik} - d_{jk}| \leq \max_{k \leq n} |d_{ij}| = d_{ij}.$$

Now we note that $\max_{k \le n} |d_{ik} - d_{jk}|$ is achieved when $k \in \{i, j\}$. Wlog we take $k = j$, and write $\max_k |d_{ik} - d_{jk}| = \max_k |d_{ij} - d_{jj}| = \max_k |d_{ij} - 0| = |d_{ij}| = d_{ij}$ (since every entry of a distance matrix is non-negative by definition). □

We are now in a position where, given any symmetric matrix D, we can verify if it is a metric under ℓ_∞ norm by simply letting x_i be its i-th row (or column), and checking whether the distance matrix $(\|x_i - x_j\|_\infty)$ is equal to D.

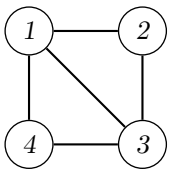
Isometric embeddings do not provide valid solution algorithms for the DGP, since (a) the DGP input graph G is usually not a complete graph (this is equivalent to saying that the distance matrix D has some "holes", some unknown values — corresponding to the edges missing from G); (b) usually we solve DGPs using the ℓ_2 norm rather than ℓ_∞.

The UIE, on the other hand, provides an amazingly easy construction. We can exploit it to devise a heuristic method: we can try filling in the holes of the given partial distance matrix, and then we can simply ignore Thm. 10.3.1, and hope that ℓ_2 and ℓ_∞ norms are not all that different. In fact, it is known that

$$\forall x \in \mathbb{R}^K \quad \|x\|_\infty \le \|x\|_2 \le n\|x\|_\infty, \tag{10.3}$$

so this can be used to derive some (slack) error bounds on the application of the UIE in norm ℓ_2.

10.3.2 Example
The incomplete graph G on the left is realized in $\mathbb{R}^2$, where 4 is at the origin, 1 is at $(0, 1)$, 3 is at $(1, 0)$, and 2 is at $(1, 1)$. G corresponds to the partial distance matrix on the right — the question marks indicate "holes", i.e. missing values.



$$D = \begin{pmatrix} 0 & 1 & \sqrt{2} & 1 \\ 1 & 0 & 1 & ? \\ \sqrt{2} & 1 & 0 & 1 \\ 1 & ? & 1 & 0 \end{pmatrix}$$

10.3.1 Matrix completion problems

Filling in the holes of a partially defined matrix gives rise to a class of problems called Matrix Completion Problems (MCP), see [164] or ieee-focs.org/focs2012/workshops/RandomNLA/Recht_RandNLA@FOCS_2012.pdf.

We look at a specific MCP problem, namely completing a partial square symmetric matrix to a EDM. An EDM is an $n \times n$ symmetric matrix $D = (d_{ij}^2)$ of nonnegative values such that there exists an integer K and a set of n points $x_1, \ldots, x_n$ in $\mathbb{R}^K$ for which

$$\forall i, j \le n \quad d_{ij}^2 = \|x_i - x_j\|_2^2.$$

> Euclidean Distance Matrix Completion Problem (EDMCP). Given a partial square symmetric matrix $\bar{D} = (d_{ij})$, determine whether it can be completed to a Euclidean distance matrix D.

Note that $\bar{D}$ can be seen as the adjacency matrix of a simple undirected graph $G = (V, E)$ with an edge weight function $d : E \to \mathbb{R}_+$. The EDMCP asks to determine whether there exists an integer $K > 0$ and a realization $x : V \to \mathbb{R}^K$ such that Eq. (10.2) holds for G: this suffices to fill in the unspecified components of $\bar{D}$ with the values $\|x_i - x_j\|_2^2$ for all $\{i, j\} \notin E$.

10.3.3 Remark
The difference between EDMCP and DGP may appear diminutive, but it is in fact very important. In the DGP the integer K is part of the input, whereas in the EDMCP it is part of the output. This has a large effect on worst-case complexity: while the DGP is **NP***-hard even when only an ε-approximate realization is sought [263, §5], ε-approximate realizations of EDMCPs can be found in polynomial time by solving an SDP [14]. See [181, 261] for more information about the relationship between EDMCP and DGP.*

In this context we consider the *shortest-path metric* method. As above, let $G = (V, E)$ be the graph defined by using the partially defined matrix $\bar{D} = (d_{ij})$ as its adjacency matrix. We assume that G is connected. Then, for each $\{i, j\} \notin E$, we fill the missing value d_{ij} in D with the squared values of the shortest path length on G from i to j:

$$\forall \{i, j\} \notin E \quad d_{ij} = (\mathsf{shortest\_path\_length}_G(i, j))^2. \tag{10.4}$$

We shall see next how to compute the length of all shortest paths in a graph.

10.3.2 Floyd-Warshall algorithm

We can compute all shortest paths in a graph $G = (V, E)$ in $O(|V|^3)$ worst-case time using the Floyd-Warshall algorithm [217]. This algorithm can be easily modified so it does not change edge weights when they are already given.

The algorithm works by checking each triplet of vertices u, v, z in V: is it cheaper to go from u to v via z or not? If so, then update d_{uv} with $d_{uz} + d_{zv}$, see Fig. 10.5. The algorithm is correct since it loops

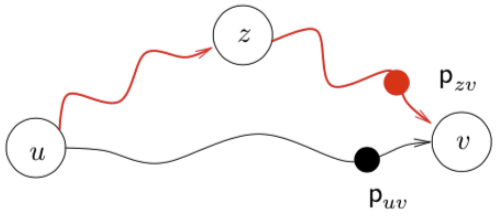


Figure 10.5: The idea behind the Floyd-Warshall algorithm.

over all triplets. The order of the triplets in the loop is important (z first). As we can see in Alg. 2, we can adapt the Floyd-Warshall algorithm to complete the given matrix by means of only updating the entry d_{uv} if it was originally missing (i.e., a "hole").

The technique consisting of: (i) completing the distance matrix of the given graph, then (ii) using an UIE to find an approximate realization, provides an approximate solution to the DGP.

There is a serious issue with this technique, however, namely that $K = n$. In other words, the embedding is in $\mathbb{R}^n$. Since n is usually very large, this technique is not very useful. We shall next look at a method for reducing this dimension without losing too much information.

10.3.3 Multidimensional scaling

The literature on Multidimensional Scaling (MDS) is extensive [77, 53], and many variants exist. The basic version, called *classic MDS*, aims at finding an approximate realization of a partial distance matrix. In other words, it is a heuristic solution method for the EDMCP.

Algorithm 2 The Floyd-Warshall algorithm (input: a weighted graph $G = (V, E, d)$).

```
# initialization
for u ≤ n, v ≤ n do
  if d_ij =? then
    d_uv ← ∞
  end if
end for
# main loop
for z ≤ n do
  for u ≤ n do
    for v ≤ n do
      if d_uv > d_uz + d_zv then
        d_uv ← d_uz + d_zv # only execute this if d_uv is a "hole"
      end if
    end for
  end for
end for
```

The *Gram matrix* of any set of n vectors $x_1, \ldots, x_n$ in $\mathbb{R}^K$ is the square symmetric matrix G such that:

$$\forall i, j \leq n \quad G_{ij} = \langle v_i, v_j \rangle. \tag{10.5}$$

If $x \in \mathbb{R}^{n \times K}$ is a realization of a graph, its Gram matrix G is obtained by setting its (i, j)-th component to $x_i^\top x_j$. The special case where $K = 1$ yields the Gram matrix of a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, where $G_{ij} = x_i x_j$. We also introduce the following constant matrix for later reference:

$$J = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top = \begin{pmatrix} 1 - \frac{1}{n} & -\frac{1}{n} & \cdots & -\frac{1}{n} \\ -\frac{1}{n} & 1 - \frac{1}{n} & \cdots & -\frac{1}{n} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{n} & -\frac{1}{n} & \cdots & 1 - \frac{1}{n} \end{pmatrix}.$$

The MDS method is based on three results:

1. there is a linear relation between EDMs and Gram matrices;
2. the class of Gram matrices is the same as the class of PSD matrices;
3. PSD matrices A have a spectral decomposition $P\Lambda P^\top$ where Λ is a diagonal matrix with non-negative diagonal components.

The method works by completing a partial EDM $\bar{D}$ using the shortest-path metric to obtain an approximate EDM D', then deriving the corresponding approximate Gram matrix G', then using spectral decomposition in order to zero the negative elements of the diagonal matrix Λ. This yields a corrected matrix G which is PSD by construction, and hence Gram by Item 2 above; the matrix G can be factored (e.g. by again using spectral decomposition) into a product $xx^\top$, where x is an approximate realization of the graph $G = (V, E)$ having $\bar{D}$ as adjacency matrix. Specifically, x satisfies Eq. (10.2) approximately.

10.3.4 Remark

Originally, the relation between EDMs and PSD matrices was exhibited and proved by the geometer I. Schoenberg [266], who showed that $D = (d_{ij})$ is EDM iff $\frac{1}{2}(d_{1i}^2 + d_{1j}^2 - d_{ij}^2 | 2 \leq i, j \leq n)$ is PSD.

We begin by proving Item 1.

10.3.5 Theorem
For any integer $K > 0$ and any set $x_1, \dots, x_n$ of n vectors in $\mathbb{R}^K$ yielding an EDM D and Gram matrix G, we have:

$$G = -\frac{1}{2} JDJ. \tag{10.6}$$

Proof. This proof is technical, but not difficult. To make the steps clearer, they are presented by bullet points. To improve readability, we denote the scalar product $x_i^\top x_j$ simply by $x_i x_j$.

- Assume that the set $x_1, \dots, x_n \in \mathbb{R}^K$ has zero centroid, i.e. $\frac{1}{n}\sum_{k\leq K} x_{ik} = 0$ for all $i \leq n$. This can be achieved wlog since EDMs are invariant w.r.t. translations.
- Expand d_{ij}^2 into $\|x_i - x_j\|_2^2 = (x_i - x_j)(x_i - x_j) = x_i x_i + x_j x_j - 2x_i x_j$. We denote the equation $d_{ij}^2 = x_i x_i + x_j x_j - 2x_i x_j$ by $(*)$.
- We aim at "inverting" the equation $(*)$ in order to express $x_i x_j$ in function of d_{ij}^2
- We sum $(*)$ over $i \leq n$: we obtain

$$\sum_{i\leq n} d_{ij}^2 = \sum_{i\leq n} x_i x_i + n x_j x_j - 2x_j \cancelto{0}{\sum_{i\leq n} x_i}$$

(0 by zero centroid)

- Similarly, we sum $(*)$ over $j \leq n$, obtaining a similar equation.
- We divide both equations (over i and j) by n, and obtain:

$$\begin{aligned} \frac{1}{n}\sum_{i\leq n} d_{ij}^2 &= \frac{1}{n}\sum_{i\leq n} x_i x_i + x_j x_j \quad (\dagger) \\ \frac{1}{n}\sum_{j\leq n} d_{ij}^2 &= x_i x_i + \frac{1}{n}\sum_{j\leq n} x_j x_j \quad (\ddagger) \end{aligned}$$

- We sum $(\dagger)$ over $j \leq n$, and get:

$$\frac{1}{n}\sum_{i,j\leq n} d_{ij}^2 = n\frac{1}{n}\sum_{i\leq n} x_i x_i + \sum_{j\leq n} x_j x_j = 2\sum_{i\leq n} x_i x_i$$

- We divide by n, and get:

$$\frac{1}{n^2}\sum_{i,j\leq n} d_{ij}^2 = \frac{2}{n}\sum_{i\leq n} x_i x_i \quad (**)$$

- We rearrange $(*)$, $(\dagger)$, $(\ddagger)$ as follows:

$$2x_i x_j = x_i x_i + x_j x_j - d_{ij}^2 \tag{10.7}$$

$$x_i x_i = \frac{1}{n}\sum_{j\leq n} d_{ij}^2 - \frac{1}{n}\sum_{j\leq n} x_j x_j \tag{10.8}$$

$$x_j x_j = \frac{1}{n}\sum_{i\leq n} d_{ij}^2 - \frac{1}{n}\sum_{i\leq n} x_i x_i \tag{10.9}$$

- We replace the LHS of Eq. (10.8)-(10.9) in the RHS of Eq. (10.7), and obtain:

$$2x_i x_j = \frac{1}{n}\sum_{k\leq n} d_{ik}^2 + \frac{1}{n}\sum_{k\leq n} d_{kj}^2 - d_{ij}^2 - \frac{2}{n}\sum_{k\leq n} x_k x_k.$$

- By $(**)$, we replace $\frac{2}{n}\sum_{i\leq n} x_i x_i$ with $\frac{1}{n^2}\sum_{i,j\leq n} d_{ij}^2$, and get

$$2x_i x_j = \frac{1}{n}\sum_{k\leq n}(d_{ik}^2 + d_{kj}^2) - d_{ij}^2 - \frac{1}{n^2}\sum_{h,k\leq n} d_{hk}^2 \quad (\S),$$

  which expresses $x_i x_j$ in function of D.

At this point, we only need to explicitly compute the expression $-\frac{1}{2}JDJ$, and verify its relation to (§). Again, this is long and technical but poses no logical difficulty: it is all computation. We first evaluate the (i,j)-th component of the matrix product JD. The i-th row of J is $(-1/n,\ldots,(1-1/n)_i,\ldots,-1/n)$, and the j-th column of D is $(d_{1j}^2,\ldots,0_j,\ldots,d_{nj}^2)$. The scalar product is $-(1/n)\sum_{k\notin\{i,j\}} d_{kj}^2 + (1-1/n)d_{ij}^2 = d_{ij}^2 - (1/n)\sum_k d_{kj}^2$. Now the i-th row of JD is

$$(d_{i1}^2 - \frac{1}{n}\sum_k d_{k1}^2, \ldots, -\frac{1}{n}\sum_k d_{ki}^2, \ldots, d_{in}^2 - \frac{1}{n}\sum_k d_{kn}^2)$$

and the j-th column of J is $(-1/n,\ldots,(1-1/n)_j,\ldots,-1/n)$. Their scalar product is:

$$\begin{aligned}
& -\frac{1}{n}\sum_{h\notin\{i,j\}}(d_{ih}^2 - \frac{1}{n}\sum_k d_{kh}^2) + \frac{1}{n^2}\sum_k d_{ki}^2 + (1-\frac{1}{n})(d_{ij}^2 - \frac{1}{n}\sum_k d_{kj}^2) \\
= & -\frac{1}{n}\sum_{h\neq i}(d_{ih}^2 - \frac{1}{n}\sum_k d_{kh}^2) - \frac{1}{n}\sum_k d_{kj}^2 + \frac{1}{n^2}\sum_k d_{ki}^2 + d_{ij}^2 \\
= & \frac{1}{n}\sum_{h\neq i} d_{ih}^2 + \frac{1}{n^2}\sum_{\substack{k\leq n\\ h\neq i}} d_{kh}^2 - \frac{1}{n}\sum_k d_{kj}^2 + \frac{1}{n^2}\sum_k d_{ki}^2 + d_{ij}^2 \\
= & -\frac{1}{n}\sum_h d_{ih}^2 - \frac{1}{n}\sum_k d_{kj}^2 + \frac{1}{n^2}\sum_{k,h} d_{kh}^2 + d_{ij}^2 \\
= & -\frac{1}{n}\sum_k (d_{ik}^2 + d_{kj}^2) + \frac{1}{n^2}\sum_{k,h} d_{kh}^2 + d_{ij}^2.
\end{aligned}$$

We note that multiplying the last quantity by -1 we obtain the RHS of (§). Now the claim holds because, after multiplying (§) by $\frac{1}{2}$, we note that the LHS is the (i,j)-th entry of the Gram matrix G, whereas the RHS is the (i,j)-th entry of the matrix $-\frac{1}{2}JDJ$. □

Now we prove Item 2.

10.3.6 Lemma
The set of Gram matrices is equal to the set of PSD matrices.

Proof. First we prove that any Gram matrix G is PSD. Since G is Gram, it arises from the product of a realization $x \in \mathbb{R}^{n\times K}$ by its transpose, i.e. $G = xx^\top$. So, for each $y \in \mathbb{R}^n$ we have:

$$y^\top G y = y^\top(xx^\top)y = (y^\top x)(x^\top y) = (x^\top y)(x^\top y)^\top = \|x^\top y\|_2^2 \geq 0,$$

which proves that G is PSD. Now we establish the converse. Let G by any PSD matrix. Since it is PSD, it has a spectral decomposition $G = P\Lambda P^\top$ where P is a matrix of eigenvectors and $\Lambda = \mathsf{diag}(()\lambda)$ is a diagonal matrix with diagonal vector $0 \leq \lambda \in \mathbb{R}^n$ of eigenvalues. Since $\lambda \geq 0$, the vector $\sqrt{\lambda} = (\sqrt{\lambda_i} \mid i \leq n)$ is real. Hence

$$P\Lambda P^\top = P(\mathsf{diag}(\sqrt{\lambda})\mathsf{diag}(\sqrt{\lambda})^\top)P^\top = (P\mathsf{diag}(\sqrt{\lambda}))(\mathsf{diag}(\sqrt{\lambda})^\top P^\top) = P\mathsf{diag}(\sqrt{\lambda})(P\mathsf{diag}(\sqrt{\lambda}))^\top,$$

which shows that G is the Gram matrix of $x = P\mathsf{diag}(\sqrt{\lambda})$. □

Note that Item 3 is a well-known linear algebra property, which was also used in the second part of Lemma 10.3.6.

Finally, we give a more detailed explanation of the MDS algorithm. Given a partial EDM D', MDS consists of the following steps:

1. complete $\bar{D}$ with the shortest-path metric as in Eq. (10.4), obtaining a matrix D';
2. compute $G' = -\frac{1}{2}JD'J$
3. let $P\Lambda'P^\top$ be the spectral decomposition of G' into an eigenvector matrix P and a diagonal matrix $\Lambda' = \mathsf{diag}(\lambda')$;
4. if $\lambda' \geq 0$ then, by Eq. (10.6), D' is a EDM, with corresponding (exact) realization $x = P\sqrt{\Lambda'}$ with $K = n$;
5. otherwise, let $\lambda^+ = \mathsf{diag}((\max(\lambda'_i, 0) \mid i \leq n))$ and $\Lambda^+ = \mathsf{diag}(\lambda^+)$: then $x' = P\sqrt{\Lambda^+}$ is an approximate realization of D'.

Note that the last step reduces the dimensionality of the realization: instead of yielding a realization in $\mathbb{R}^n$, it achieves a realization in a lower dimensional space $\mathbb{R}^K$ with $K < n$. The trade-off is between dimensional reduction and approximation quality is as follows: the more negative components of λ' are discarded, the worse the approximation of the realization x' is.

We also note that the input matrix D' does not, strictly speaking, need to be a partial EDM. Any symmetric matrix D' can provide an input to the MDS algorithm. Naturally, the farther D' is from an EDM, the worse the approximate realization x' will be. On the other hand, MDS is often used in order to provide a graphical representation of "difference matrices" that have nothing to do with Euclidean distances [53].

10.3.4 Principal component analysis

Principal Component Analysis (PCA) is one of the best known dimensional reduction techniques. It was first proposed by Harold Hotelling[3] [144].

Consider an $n \times m$ matrix X consisting of n data row vectors in $\mathbb{R}^m$, and let $K < m$ be a given integer. We want to find a change of coordinates for X such that the first component has largest variance over the transformed vectors, the second component has second-largest variance, and so on, until the K-th component. The other components can be neglected, as the variance of the data in those directions is low.

The usual geometric interpretation of PCA is to take the smallest enclosing ellipsoid $\mathcal{E}$ for X: then the required coordinate change maps component 1 to the line parallel to the largest radius of $\mathcal{E}$, component 2 to the line parallel to the second-largest radius of $\mathcal{E}$, and so on until component K (see Fig. 10.6). The statistical interpretation of PCA looks for the change of coordinates which makes the data vectors be uncorrelated in their components. Fig. 10.6 should give an intuitive idea about why this interpretation corresponds with the ellipsoid of the geometric interpretation. The cartesian coordinates in Fig. 10.6 are certainly correlated, while the rotated coordinates look far less (linearly) correlated. The zero correlation

[3] A young and unknown George Dantzig had just finished his presentation of LP to an audience of "big shots", including Koopmans and Von Neumann. Harold Hotelling raised his hand, and stated: "but we all know that the world is nonlinear!", thereby obliterating the simplex method as a mathematical curiosity. Luckily, Von Neumann answered on Dantzig's behalf and in his defence [87].

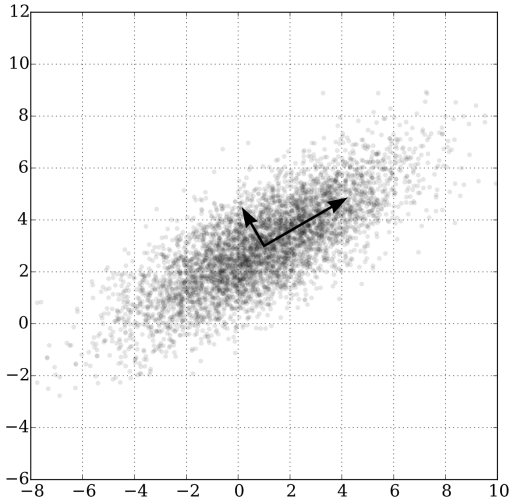


Figure 10.6: Geometric interpretation of PCA (image from [304]).

situation corresponds to a perfect ellipsoid. An ellipsoid is described by the equation $\sum_{j\leq n}\left(\frac{x_j}{r_j}\right)^2 = 1$, which has no mixed terms $x_i x_j$ contributing to correlation. Both interpretations are well (and formally) argued in [299, §2.1].

The interpretation we give here is motivated by DG, and related to MDS (Sect. 10.3.3). PCA can be seen as a modification of MDS which only takes into account K (nonnegative) principal components. Instead of Λ^+ (step 5 of the MDS algorithm), PCA uses a different diagonal matrix Λ^{pca}: the i-th diagonal component is

$$\Lambda^{\mathsf{pca}}_{ii} = \begin{cases} \max(\Lambda_{ii}, 0) & \text{if } i \leq K \\ 0 & \text{otherwise,} \end{cases} \tag{10.10}$$

where $P\Lambda P^\top$ is the spectral decomposition of $G' = -\frac{1}{2}JD'J$ (using the same notation as in the MDS algorithm in Sect. 10.3.3). In this interpretation, when given a partial EDM and the integer K as input, PCA can be used as an approximate solution method for the DGP.

On the other hand, the PCA algorithm is most usually considered as a method for dimensionality reduction, so it has a data matrix X and an integer K as input. It is as follows:

1. let $G' = XX^\top$ be the $n \times n$ Gram matrix of the data matrix X;
2. let $P\Lambda P^\top$ be the spectral decomposition of G';
3. return $\tilde{x} = P\sqrt{\Lambda^{\mathsf{pca}}}$.

Then $\tilde{x}$ is an $n \times K$ matrix, where $K < n$. The i-th row vector in $\tilde{x}$ is a dimensionally reduced representation of the i-th row vector in X.

There is an extensive literature on PCA, ranging over many research papers, dedicated monographs and textbooks [304, 153, 299]. Among the variants and extensions, see [99, 257, 91, 15, 100].

10.3.7 Example

Consider the Mathematical Genealogy Project (`genealogy.math.ndsu.nodak.edu/`). We extract a subset of the vertices from the tree shown in Fig. 10.7. We evaluate their distances using shortest paths on the tree. This yields:

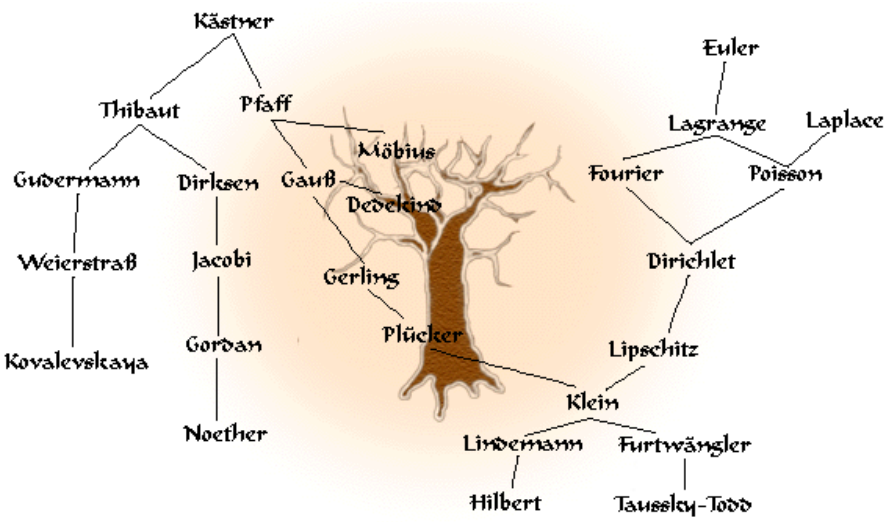


Figure 10.7: A subtree of the Mathematical Genealogy Project graph.

	Euler	*Thibaut*	*Pfaff*	*Lagrange*	*Laplace*	*Möbius*	*Gudermann*	*Dirksen*	*Gauss*
Kästner	*10*	*1*	*1*	*9*	*8*	*2*	*2*	*2*	*2*
Euler		*11*	*9*	*1*	*3*	*10*	*12*	*12*	*8*
Thibaut			*2*	*10*	*10*	*3*	*1*	*1*	*3*
Pfaff				*8*	*8*	*1*	*3*	*3*	*1*
Lagrange					*2*	*9*	*11*	*11*	*7*
Laplace						*9*	*11*	*11*	*7*
Möbius							*4*	*4*	*2*
Gudermann								*2*	*4*
Dirksen									*4*

corresponding to the distance matrix

$$D' = \begin{pmatrix} 0 & 10 & 1 & 1 & 9 & 8 & 2 & 2 & 2 & 2 \\ 10 & 0 & 11 & 9 & 1 & 3 & 10 & 12 & 12 & 8 \\ 1 & 11 & 0 & 2 & 10 & 10 & 3 & 1 & 1 & 3 \\ 1 & 9 & 2 & 0 & 8 & 8 & 1 & 3 & 3 & 1 \\ 9 & 1 & 10 & 8 & 0 & 2 & 9 & 11 & 11 & 7 \\ 8 & 3 & 10 & 8 & 2 & 0 & 9 & 11 & 11 & 7 \\ 2 & 10 & 3 & 1 & 9 & 9 & 0 & 4 & 4 & 2 \\ 2 & 12 & 1 & 3 & 11 & 11 & 4 & 0 & 2 & 4 \\ 2 & 12 & 1 & 3 & 11 & 11 & 4 & 2 & 0 & 4 \\ 2 & 8 & 3 & 1 & 7 & 7 & 2 & 4 & 4 & 0 \end{pmatrix}$$

which is not necessarily Euclidean. By using PCA, we obtain the representations of these data in 2D and 3D shown in Fig. 10.8.

10.4 Complexity

We show that the DGP is **NP**-hard. The best known proof, given in [263], reduces from PARTITION (see Sect. 5).

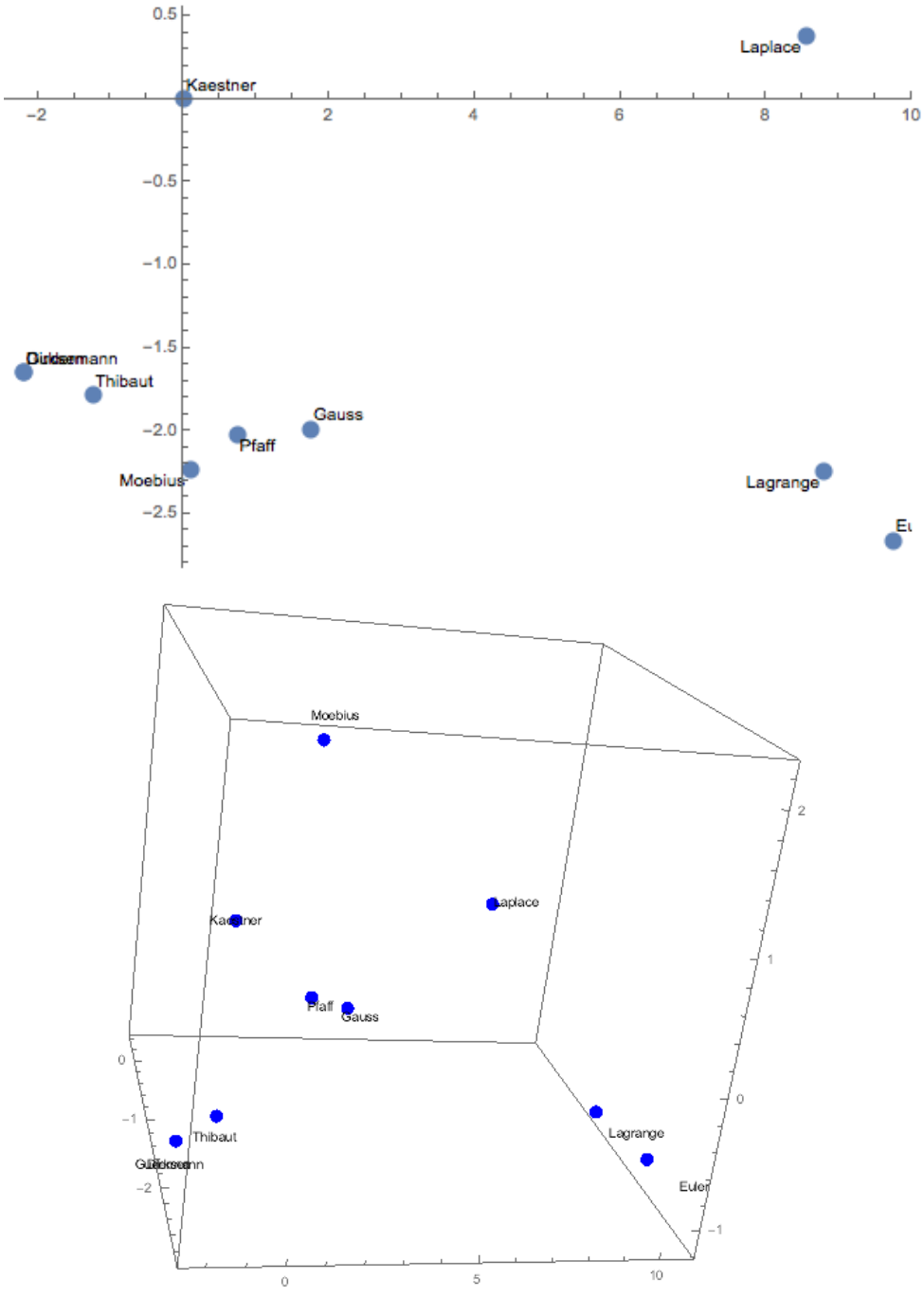


Figure 10.8: PCA-derived representations in 2D and 3D.

10.4.1 Reduction proof

More precisely, Saxe's first proof in [263] proves that the DGP is weakly **NP**-hard by inclusion of the DGP with $K = 1$.

PARTITION. Given a sequence $A = (a_1, \ldots, a_n)$ of non-negative integers, is there a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

For a given instance $(a_1, \ldots, a_n)$ of PARTITION, consider a simple cycle $C = (V, E)$ with $|V| = |E| = n$ and an edge weight function d such that $d_{i,i+1} = a_i$ for $i \leq n-1$, and $d_{n1} = a_n$: by arbitrarily setting $K = 1$, we make this weighted cycle graph into a DGP instance.

Assume that the given PARTITION instance is a YES instance: we show that the corresponding DGP instance is also a YES instance, i.e. there is a realization x of C in 1D (i.e. on a line). We construct a realization x of C in $\mathbb{R}$ inductively, as follows:

1. we fix $x_1 = 0$;
2. if we know the position x_i for $i < n$ and $i \in I$, we let $x_{i+1} = x_i + d_{i,i+1}$ (right placement), else $x_{i+1} = x_i - d_{i,i+1}$ (left placement).

Following the same induction on i, we easily prove that x is a valid realization for C: it suffices to assume it is correct as far as vertex i and conclude, by the induction hypothesis, that it is also correct for vertex $i+1$, since we place x_{i+1} at distance $d_{i,i+1}$ from i. This induction holds as far as we are applying rule 2 above, i.e. until $i = n-1$, which yields position for $i+1 = n$. In order for the realization to be valid, however, we also need to ensure that the distance d_{1n} is preserved. To this end, we define a dummy index $n+1$ to be equivalent to index 1: the position x_{n+1}, computed according to rule 2 where $d_{n,n+1} = d_{n1} = d_{1n}$, should turn out to be equal to x_1, which we still need to prove.

We have:

$$\begin{aligned}
\sum_{i \in I}(x_{i+1} - x_i) &= \sum_{i \in I} d_{i,i+1} \quad \text{[by defn. of the cycle graph]} \\
&= \sum_{i \in I} a_i = \sum_{i \notin I} a_i \quad \text{[PARTITION instance is YES]} \\
&= \sum_{i \notin I} d_{i,i+1} = \sum_{i \notin I}(x_i - x_{i+1}).
\end{aligned}$$

This implies that the first and the last terms are equal, hence:

$$\begin{aligned}
0 &= \sum_{i \in I}(x_{i+1} - x_i) - \sum_{i \notin I}(x_i - x_{i+1}) \\
&= \sum_{i \in I}(x_{i+1} - x_i) + \sum_{i \notin I}(x_{i+1} - x_i) \quad \text{[sign change in 2nd term]} \\
&= \sum_{i \leq n}(x_{i+1} - x_i) \quad \text{[grouping terms from both sums]} \\
&= (x_{n+1} - x_n) + (x_n - x_{n-1}) + \ldots + (x_2 - x_1) \\
&= x_{n+1} + (x_n - x_n) + \ldots + (x_2 - x_2) - x_1 = x_{n+1} - x_1,
\end{aligned}$$

implying $x_{n+1} = x_1$ as claimed. So the DGP instance is YES.

Now assume that the given PARTITION instance is a NO instance, and suppose, to get a contradiction, that the corresponding DGP instance is a YES instance. So we suppose there is a realization x of C in 1D. Since we are realizing on a line, for any two points x_u, x_v, we either have $x_u \leq x_v$ or $x_u > x_v$. Let $F = \{\{u, v\} \in E \mid x_u \leq x_v\}$, so that $E \setminus F$ will only contain edges $\{u, v\}$ for which $x_u > x_v$. Because C is a cycle, starting with any vertex v, we must be able to walk the cycle from v back to itself passing through every vertex: for this to hold, the walk must have one direction over all edges in F and

the opposite direction for all edges in $E \setminus F$. Since there is a unique position x_v for each vertex v, the distance walked in one direction must be equal to the distance walked in the opposite direction. Hence:

$$\begin{aligned} \sum_{\{u,v\}\in F} (x_v - x_u) &= \sum_{\{u,v\}\in E\setminus F} (x_u - x_v) \\ \Rightarrow \sum_{\{u,v\}\in F} |x_u - x_v| &= \sum_{\{u,v\}\in E\setminus F} |x_u - x_v| \\ \Rightarrow \sum_{\{u,v\}\in F} d_{uv} &= \sum_{\{u,v\}\in E\setminus F} d_{uv}. \end{aligned} \tag{10.11}$$

By definition, every edge in E has the form $\{i, i+1\}$ for $i < n$ or $\{n, 1\}$: let J be the set of all $i < n$ such that $\{i, i+1\}$ is in F, and let it also contain n if $\{n, 1\}$ is in F. Then Eq. (10.11) becomes:

$$\sum_{i\in J} a_i = \sum_{i\notin J} a_i,$$

which implies that J is a solution for the given PARTITION instance, against the assumption that it was a NO instance. Hence the corresponding DGP instance must also be a NO instance, as claimed.

Lastly, it is easy to note that the transformation of a PARTITION instance into the corresponding DGP instance can be carried out in time bounded by a polynomial in n, since for each $i \leq n$ we construct a vertex and an edge in the cycle graph.

This means that we have a polytime transformation to turn PARTITION instances to DGP instances with $K = 1$ so that YES instances map to YES instances and NO instances map to NO instances. In other words, if we could solve the DGP in polytime then we could exploit this polytime transformation to solve PARTITION in polytime too. But since PARTITION is **NP**-hard [116] then DGP must also be **NP**-hard with $K = 1$. And since the case $K = 1$ determines a subset of instances of the DGP, the DGP itself must be **NP**-hard.

10.4.1 Example
Consider the PARTITION *instance* $a_1 = 2$, $a_2 = 1$, $a_3 = 4$, $a_4 = 1$, $a_5 = 2$. *We construct the cycle* C *over* $\{1, 2, 3, 4, 5\}$ *with edges* $\{i, i+1\}$ *for* $i \leq 4$ *and* $\{5, 1\}$ *(which closes the cycle), weighted by* a_i *for each* $i \leq 4$ *and* $d_{51} = a_5$. *We realize* C *with* $x_1 = 0$, $x_2 = -2$, $x_3 = -3$, $x_4 = 1$, $x_5 = 2$, *as shown in Fig. 10.9. Now the set* F *of edges* $\{u, v\}$ *with* $u < v$ *and* $x_u \leq x_v$ *is* $\{3, 4\}$ *and* $\{4, 5\}$, *so* $J = \{3, 4\}$, *and it is easy*

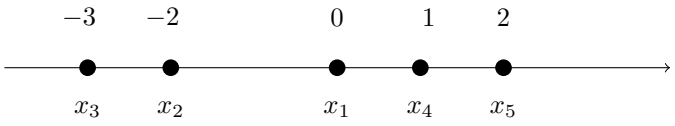


Figure 10.9: The realization of a cycle in 1D.

to verify that $\sum_{i\in J} a_i = 4 + 1 = 5 = 2 + 1 + 2 = \sum_{i\notin J} a_i$, *showing that* (a_3, a_4) *and* (a_1, a_2, a_5) *is the desired partition.*

Saxe also proved that the DGP is **NP**-hard for any fixed value of K, using a more complicated reduction from a different **NP**-hard problem.

10.4.2 Membership in NP

In Sect. 10.4.1 we proved hardness of the DGP. Since the DGP is a decision problem, it makes sense to ask whether it is in **NP** (and hence whether it is not only **NP**-hard but also **NP**-complete)..

The issue here is that, in general, even a graph having edges weighted with integers might yield a realization with irrational components (e.g. a triangle graph with unit weights has a realization

$$(0,0),(1,0),(0.5,\cos\pi/6),$$

and there is no congruence that can simultaneously make all realization components rational). So the problem is one of being able to represent a certificate in a polynomial amount of space, necessary to membership in **NP**.

If $K = 1$, however, as long as the graph has rational edge weights, the realization must also have rational components. It can therefore be proved that the DGP with $= 1$ is indeed in **NP**. Let $G = (V, E)$ be a graph and $x \in \mathbb{R}^{n\times 1}$ be a valid realization. Suppose there is a component $x_i \notin \mathbb{Q}$. Then, since every edge weight is in $\mathbb{Q}$, every neighbour j of i is realized at an irrational position x_j, and so on for every vertex in V. Take any translation t moving x_i to a rational value, i.e. $t : p \to p - x_i$, so that $t(x_i) = 0$. Again since every edge weight is rational, every vertex must be realized at a rational position, i.e. $t(x) \in \mathbb{Q}$. This argument shows that if G can be realized in $\mathbb{R}^1$, then there is a rational realization x. Lastly, we need to prove that x can be written using a polynomial amount of space. This follows because every component can be obtained as a sum or difference of edge weights from any chosen component, say x_1.

If $K > 1$, as mentioned above, it is unlikely that the DGP is in **NP**. Since realizations are solutions of a system of polynomial equations of degree 2 (Eq. (10.2)), we can at least rule out the need for transcendental realizations: it suffices to consider algebraic numbers. Although there exist finitary representations of algebraic numbers, [35] shows that the simplest such representation is not enough to prove membership in **NP** of the DGP.

10.5 Number of solutions

A DGP instance may have no solutions if the given distances do not define a metric, a finite number of solutions if the graph is rigid, or uncountably many solutions if the graph is flexible.

Restricted to the ℓ_2 norm, there are several different notions of rigidity. We only define the simplest, which is easiest to explain intuitively: if we consider the graph as a representation of a bar-and-joint framework, a graph is flexible if the framework can move (excluding translations and rotations) and rigid otherwise. The formal definition of rigidity of a graph $G = (V, E)$ involves: (a) a mapping D from a realization $x \in \mathbb{R}^{nK}$ to the partial distance matrix

$$\mathsf{D}(x) = (\|x_u - x_v\| \mid \{u, v\} \in E);$$

and (b) the completion $\mathsf{K}(G)$ of G, defined as the complete graph on V. We want to say that G is rigid if, were we to move x ever so slightly (excluding translations and rotations), $\mathsf{D}(x)$ would also vary accordingly. We formalize this idea indirectly: a graph is *rigid* if the realizations in a neighbourhood χ of x corresponding to changes in $\mathsf{D}(x)$ are equal to those in the neighbourhood $\bar{\chi}$ of a realization $\bar{x}$ of $\mathsf{K}(G)$ [183, Ch. 7]. We note that realizations $\bar{x} \in \bar{\chi}$ correspond to small variations in $\mathsf{D}(\mathsf{K}(G))$: this definition makes sense because $\mathsf{K}(G)$ is a complete graph, which implies that its distance matrix is invariant, and hence $\bar{\chi}$ may only contain congruences.

We thus obtain the following formal characterization of rigidity [24]:

$$\mathsf{D}^{-1}(\mathsf{D}(x)) \cap \chi = \mathsf{D}^{-1}(\mathsf{D}(\bar{x})) \cap \bar{\chi}. \tag{10.12}$$

Let us parse Eq. (10.12): for a partial distance matrix Y, $\mathsf{D}^{-1}(Y)$ corresponds to all of the realizations that give rise to Y (which are uncountably many because of congruences). Now, let x be a realization of the partial distance matrix Y, and $\bar{x}$ a realization of the metric completion $\bar{Y}$ of Y (if it exists). Moreover, χ is a neighbourhood of x and $\bar{\chi}$ is a neighbourhood of $\bar{x}$ (in the vector space $\mathbb{R}^{nK}$). Since we know

that $\bar{Y}$ corresponds to a realizable complete graph, its framework is rigid. So the set $\mathsf{D}^{-1}(\mathsf{D}(\bar{x})) \cap \bar{\chi}$ only contains realizations obtained from $\bar{x}$ by means of congruences. Eq. (10.12) states that the framework realized by x is rigid if the realizations of the partial distance matrix of x can be obtained from x only from congruences: in other words, if it "behaves like" the framework of a complete graph.

Uniqueness of solution (modulo congruences) is sometimes a necessary feature in applications. Many different sufficient conditions to uniqueness have been found [185, §4.1.1]. By way of example as concerns the number of DGP solutions in graphs, a complete graph has at most one solution modulo congruences, as remarked above. It was proved in [184] that protein backbone graphs have a realization set having power of two cardinality with probability 1. As shown in Fig. 10.10 (bottom row), a cycle graph on four vertices has uncountably many solutions.

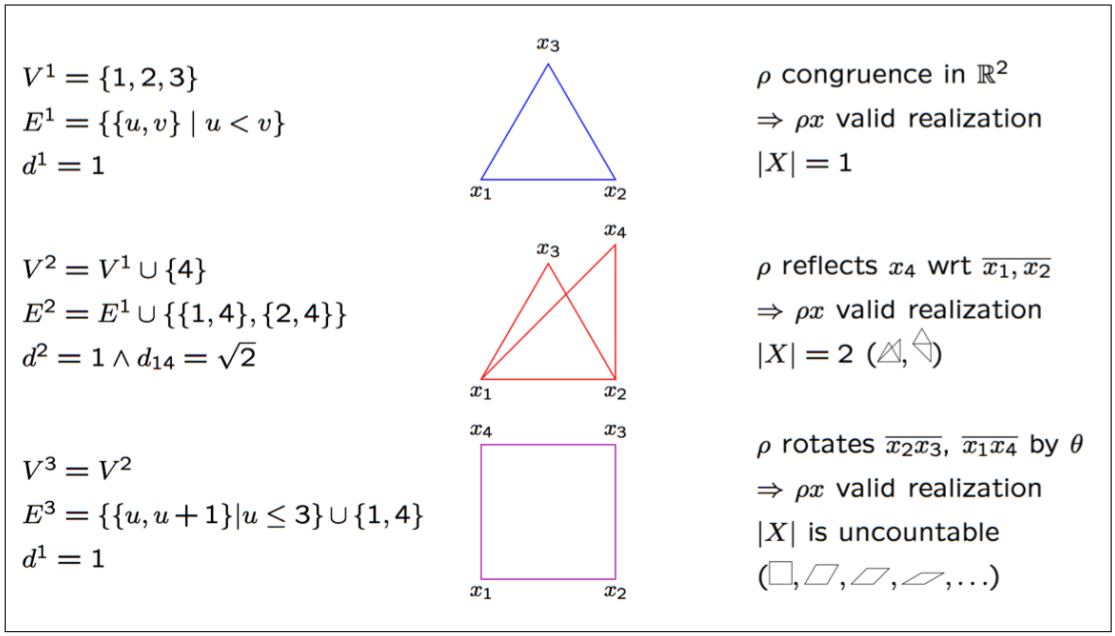


Figure 10.10: Instances with one, two, and uncountably many realizations.

On the other hand, the remaining possibility of a countably infinite set of realizations of a DGP instance cannot happen, as shown in Thm. 10.5.1. This result is a simple corollary of a well-known theorem of Milnor [222]. It was noted informally in [185, p. 27] without details; we provide a proof here.

10.5.1 Theorem
No DGP instance may have an infinite but countable number of solutions.

Proof. Eq. (10.2) is a system of m quadratic equations associated with the instance graph G. Let $X \subseteq \mathbb{R}^{nK}$ be the variety associated to Eq. (10.2). Now suppose X is countable: then no connected component of X may contain uncountably many elements. By the notion of connectedness, this implies that every connected component is an isolated point in X. Since X is countable, it must contain a countable numbers of connected components. By [222], the number of connected components of X is finite; in particular, it is bounded by $O(3^{nK})$. Hence the number of connected components of X is finite. Since each is an isolated point, i.e. a single realization of G, $|X|$ is finite. □

10.6 Formulation-based solution methods

DGP solution methods based on MP can be tailored to handle noisy or wrong data because MP allows for modification of the objective and constraints, as well as adjoining of further side constraints. Moreover, although we do not review these here, there are MP-based methodologies for ensuring robustness of solutions [39], probabilistic constraints [245], and scenario-based stochasticity [45], which can be applied to the formulations in this section.

10.6.1 Unconstrained quartic formulation

A system of equations such as Eq. (10.2) is itself a MP formulation with objective function identically equal to zero, and $X = \mathbb{R}^{nK}$. It therefore belongs to the Quadratically Constrained Programming (QCP) class. In practice, solvers for this class perform rather poorly when given Eq. (10.2) as input [165]. Much better performances can be obtained by solving the following unconstrained formulation:

$$\min \sum_{\{u,v\}\in E} \left(\|x_u - x_v\|_2^2 - d_{uv}^2\right)^2. \tag{10.13}$$

We note that Eq. (10.13) consists in the minimization of a polynomial of degree four. It belongs to the class of nonconvex NLP formulations. In general, this is an **NP**-hard class [175], which is not surprising, as it formulates the DGP which is itself an **NP**-hard problem (see Sect. 10.4. Very good empirical results can be obtained on the DGP by solving Eq. (10.13) with a local NLP solver (such as e.g. IPOPT [70] or SNOPT[119] [120]) from a good starting point [165]. This is the reason why Eq. (10.13) is very important: it can be used to improve approximate solutions obtained with other methods, as it suffices to let such solutions be starting points given to a local solver acting on Eq. (10.13).

Even if the distances d_{uv} are noisy or wrong, optimizing Eq. (10.13) can yield good approximate realizations. If the uncertainty on the distance values is modelled using an interval $[d_{uv}^L, d_{uv}^U]$ for each edge $\{u, v\}$, the following function [186] can be optimized instead of Eq. (10.13):

$$\min \sum_{\{u,v\}\in E} \left(\max(0, (d_{uv}^L)^2 - \|x_u - x_v\|_2^2) + \max(0, \|x_u - x_v\|_2^2 - (d_{uv}^U)^2)\right). \tag{10.14}$$

The DGP variant where distances are intervals instead of values is known as the INTERVAL DGP (iDGP) [125, 166]. We remark that, with interval distances, the formultions proposed in this section are no longer exact reformulations of Eq. (10.2).

Note that Eq. (10.14) involves max functions with two arguments. Relatively few MP user interfaces/solvers would accept this function. To overcome this issue, we replace the two max terms by two sets of added decision variables y, z, and obtain

$$\left.\begin{array}{rrcl} \min & \sum_{\{u,v\}\in E} (y_{uv} + z_{uv}) & & \\ \forall \{u,v\} \in E & y_{uv} & = & \max(0, (d_{uv}^L)^2 - \|x_u - x_v\|_2^2) \\ \forall \{u,v\} \in E & z_{uv} & = & \max(0, \|x_u - x_v\|_2^2 - (d_{uv}^U)^2). \end{array}\right\}$$

Now we observe that, because of the objective function direction, we can replace the equality sense in the constraint by an inequality of type $\geq$:

$$\left.\begin{array}{rrcl} \min & \sum_{\{u,v\}\in E} (y_{uv} + z_{uv}) & & \\ \forall \{u,v\} \in E & y_{uv} & \geq & \max(0, (d_{uv}^L)^2 - \|x_u - x_v\|_2^2) \\ \forall \{u,v\} \in E & z_{uv} & \geq & \max(0, \|x_u - x_v\|_2^2 - (d_{uv}^U)^2). \end{array}\right\}$$

Finally, we note that $a \geq \max(b, c)$ is equivalent to $a \geq b \wedge a \geq c$. This yields:

$$\left.\begin{array}{rrcl} \min & \sum\limits_{\{u,v\}\in E} (y_{uv} + z_{uv}) & & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \geq & (d_{uv}^L)^2 - y_{uv} \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \leq & (d_{uv}^U)^2 + z_{uv} \\ & y, z & \geq & 0, \end{array}\right\} \tag{10.15}$$

which follows from Eq. (10.14) because of the objective function direction, and because $a \geq \max(b, c)$ is equivalent to $a \geq b \wedge a \geq c$. We note that Eq. (10.15) is no longer an unconstrained quartic, however, but a QCP. It expresses a minimization of penalty variables to the quadratic inequality system

$$\forall \{u,v\} \in E \quad (d_{uv}^L)^2 \leq \|x_u - x_v\|_2^2 \leq (d_{uv}^U)^2. \tag{10.16}$$

We also note that many local NLP solvers take rather arbitrary functions in input (such as functions expressed by computer code), so the reformulation Eq. (10.15) may be unnecessary when only locally optimal solutions of Eq. (10.14) are needed.

10.6.2 Constrained quadratic formulations

We propose two formulations in this section. The first is derived directly from Eq. (10.2):

$$\left.\begin{array}{rrcl} \min & \sum\limits_{\{u,v\}\in E} s_{uv}^2 & & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & = & d_{uv}^2 + s_{uv}. \end{array}\right\} \tag{10.17}$$

We note that Eq. (10.17) is a Quadratically Constrained Quadratic Programming (QCQP) formulation. Similarly to Eq. (10.15) it uses additional variables to penalize feasibility errors w.r.t. (10.2). Differently from Eq. (10.15), however, it removes the need for two separate variables to model slack and surplus errors. Instead, s_{uv} is unconstrained, and can therefore take any value. The objective, however, minimizes the sum of the squares of the components of s. In practice, Eq. (10.17) performs much better than Eq. (10.2); on average, the performance is comparable to that of Eq. (10.13). We remark that Eq. (10.17) has a convex objective function but nonconvex constraints.

The second formulation we propose is an exact reformulation of Eq. (10.13). First, we replace the minimization of squared errors by absolute values, yielding

$$\min \sum_{\{u,v\}\in E} \left| \|x_u - x_v\|_2^2 - d_{uv}^2 \right|,$$

which clearly has the same set of global optima as Eq. (10.13). We then rewrite this similarly to Eq. (10.15) as follows:

$$\left.\begin{array}{rrcl} \min & \sum\limits_{\{u,v\}\in E} (y_{uv} + z_{uv}) & & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \geq & d_{uv}^2 - y_{uv} \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \leq & d_{uv}^2 + z_{uv} \\ & y, z & \geq & 0, \end{array}\right\}$$

which, again, does not change the global optima. Next, we note that we can fix $z_{uv} = 0$ without changing global optima, since they all have the property that $z_{uv} = 0$. Now we replace y_{uv} in the objective function by $d_{uv}^2 - \|x_u - x_v\|_2^2$, which we can do without changing the optima since the first set of constraints reads $y_{uv} \geq d_{uv}^2 - \|x_u - x_v\|_2^2$. We can discard the constant d_{uv}^2 from the objective, since adding constants to the objective does not change optima, and change $\min -f$ to $-\max f$, yielding:

$$\left.\begin{array}{rrcl} \max & \sum\limits_{\{u,v\}\in E} \|x_u - x_v\|_2^2 & & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \leq & d_{uv}^2, \end{array}\right\} \tag{10.18}$$

which is a QCQP known as the "push-and-pull" formulation of the DGP, since the constraints ensure that x_u, x_v are pushed closer together, while the objective attempts to pull them apart [218, §2.2.1].

Contrariwise to Eq. (10.17), Eq. (10.18) has a nonconvex (in fact, concave) objective function and convex constraints. Empirically, this often turns out to be somewhat easier than tackling the reverse situation. The theoretical justification is that finding a feasible solution in a nonconvex set is a hard task in general, whereas finding local optima of a nonconvex function in a convex set is tractable: the same cannot be said for global optima, but in practice one is often satisfied with "good" local optima.

10.6.3 Semidefinite programming

SDP is linear optimization over the cone of PSD matrices, which is convex: if A, B are two PSD matrices, $C = \alpha A + (1-\alpha)B$ is PSD for $\alpha \in [0,1]$. Suppose there is $x \in \mathbb{R}^n$ such that $x^\top C x < 0$. Then $\alpha x^\top A x + (1-\alpha) x^\top B x < 0$, so $0 \leq \alpha x^\top A x < -(1-\alpha) x^\top B x \leq 0$, i.e. $0 < 0$, which is a contradiction, hence C is also PSD, as claimed. Therefore, SDP is a subclass of cNLP.

The SDP formulation we propose is a relaxation of Eq. (10.2). First, we write $\|x_u - x_v\|_2^2 = \langle x_u, x_u\rangle + \langle x_v, x_v\rangle - 2\langle x_u, x_v\rangle$. Then we linearize all of the scalar products by means of additional variables X_{uv}:

$$\begin{array}{rrcl} \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = & d_{uv}^2 \\ & X & = & xx^\top. \end{array}$$

We note that $X = xx^\top$ constitutes the whole set of defining constraints $X_{uv} = \langle x_u, x_v\rangle$ (for each $u, v \leq n$) introduced by the linearization procedure (Sect. 2.2.3.5).

The relaxation we envisage does not entirely drop the defining constraints, as in Sect. 2.2.3.5. Instead, it relaxes them from $X - xx^\top = 0$ to $X - xx^\top \succeq 0$. In other words, instead of requiring that all of the eigenvalues of the matrix $X - xx^\top$ are zero, we simply require that they should be ≥ 0. Moreover, since the original variables x do not appear anywhere else, we can simply require $X \succeq 0$, obtaining:

$$\left.\begin{array}{rrcl} \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = & d_{uv}^2 \\ & X & \succeq & 0. \end{array}\right\} \tag{10.19}$$

The SDP relaxation in Eq. (10.19) has the property that it provides a solution $\bar{X}$, which is an $n \times n$ symmetric matrix. Spectral decomposition of $\bar{X}$ yields $P\Lambda P^\top$, where P is a matrix of eigenvectors and $\Lambda = \mathsf{diag}(\lambda)$ where λ is a vector of eigenvalues of $\bar{X}$. Since $\bar{X}$ is PSD, $\lambda \geq 0$, which means that $\sqrt{\Lambda}$ is a real matrix. Therefore, by setting $Y = P\sqrt{\Lambda}$ we have that

$$YY^\top = (P\sqrt{\Lambda})(P\sqrt{\Lambda})^\top = P\sqrt{\Lambda}\sqrt{\Lambda}P^\top = P\Lambda P^\top = \bar{X},$$

which implies that $\bar{X}$ is the Gram matrix of Y. Thus we can take Y to be a realization satisfying Eq. (10.2). The only issue is that Y, as an $n \times n$ matrix, is a realization in n dimensions rather than K. Naturally, $\mathsf{rk}(Y) = \mathsf{rk}(\bar{X})$ need not be equal to n, but could be lower; in fact, in order to find a realization of the given graph, we would like to find a solution $\bar{X}$ with rank at most K. Imposing this constraint is equivalent to asking that $X = xx^\top$ (which have been relaxed in Eq. (10.19)).

We note that Eq. (10.19) is a pure feasibility problem. Every SDP solver, however, also accepts an objective function as input. In absence of a "natural" objective in a pure feasibility problem, we can devise one to heuristically direct the search towards parts of the PSD cone which we believe might contain "good" solutions. A popular choice is

$$\begin{array}{rcl} \min \mathsf{tr}(X) & = & \min \mathsf{tr}(P\Lambda P^\top) = \min \mathsf{tr}(PP^\top\Lambda) = \\ & = & \min \mathsf{tr}(PP^{-1}\Lambda) = \min \lambda_1 + \cdots + \lambda_n, \end{array}$$

where tr is the trace, the first equality follows by spectral decomposition (with P a matrix of eigenvectors and Λ a diagonal matrix of eigenvalues of X), the second by commutativity of matrix products under

the trace, the third by orthogonality of eigenvectors, and the last by definition of trace. This aims at minimizing the sum of the eigenvalues of X, hoping this will decrease the rank of $\bar{X}$.

For the DGP applied to protein conformation (Sect. 10.1.3), the objective function

$$\min \sum_{\{u,v\}\in E} (X_{uu} + X_{vv} - 2X_{uv})$$

was empirically found to be a good choice [101, §2.1]. We remark that the equality constraints in Eq. (10.19) can be used to reformulate the function in Eq. (10.6.3) to the constant $\sum_{\{u,v\}\in E} d_{ij}^2$. The reason why Eq. (10.6.3) did not behave like a constant function in empirical tests must be related to the fact the current iterate is not precisely feasible at every step of the solution algorithm. More experimentation showed that the scalarization of the two objectives:

$$\min \sum_{\{u,v\}\in E} (X_{uu} + X_{vv} - 2X_{uv}) + \gamma \mathsf{tr}(X), \tag{10.20}$$

with γ in the range $O(10^{-2})$-$O(10^{-3})$, is a good objective function for solving Eq. (10.19) when it is applied to protein conformation.

In the majority of cases, solving SDP relaxations does not yield solution matrices with rank K, even with objective functions such as Eq. (10.20). We discuss methods for constructing an approximate rank K realization from $\bar{X}$ in Sect. 10.6.5.

SDP is one of those problems which is not known to be in **P** (nor **NP**-complete) in the Turing machine model. It is, however, known that SDPs can be solved in polytime up to a desired error tolerance $\epsilon > 0$, with the complexity depending on $\frac{1}{\epsilon}$ as well as the instance size. Currently, however, the main issue with SDP is technological: state-of-the art solvers do not scale all that well with size. One of the reasons is that K is usually fixed (and small) with respect to n, so the while the original problem has $O(n)$ variables, the SDP relaxation has $O(n^2)$. Another reason is that the Interior Point Method (IPM), which often features as a "state of the art" SDP solver, has a relatively high computational complexity [248]: a "big oh" notation estimate of $O(\max(m,n)mn^{2.5})$ is given in Bubeck's blog at ORFE, Princeton.[4]

10.6.4 Diagonally dominant programming

In order to address the size limitations of SDP, we employ some interesting linear approximations of the PSD cone proposed in [205, 8]. An $n \times n$ real symmetric matrix X is Diagonally Dominant (DD) if

$$\forall i \leq n \quad \sum_{j\neq i} |X_{ij}| \leq X_{ii}. \tag{10.21}$$

It is well known that every DD matrix is also PSD [124], while the converse may not hold. Specifically, the set of DD matrices form a sub-cone of the cone of PSD matrices [29]. This follows from Gershgorin's circle theorem, given below.

10.6.1 Theorem (Gershgorin's Circle Theorem)
Let A be a symmetric $n \times n$ matrix. For each $i \leq n$ let $R_i = \sum_{j\neq i} |A_{ij}|$, and let I_i be the interval $[A_{ii} - R_i, A_{ii} + R_i]$. Then for every eigenvalue λ of A there is an $i \leq n$ such that $\lambda \in I_i$.

Proof. Let λ be an eigenvalue of A with corresponding eigenvector x. Since eigenvectors can be rescaled, we let $i = \arg\max_j |x_j|$ and divide x by $\mathrm{sgn}(x_i)|x_i|$. This gives x the property that $x_i = 1$ and, for each $j \neq i$, we have $|x_j| \leq 1$. Thus,

$$A_i x = \sum_{j\neq i} A_{ij}x_j + A_{ii}x_i = \sum_{j\neq i} A_{ij}x_j + A_{ii}.$$

[4] blogs.princeton.edu/imabandit/2013/02/19/orf523-ipms-for-lps-and-sdps/

But $Ax = \lambda x$ because λ, x are an eigenvalue of A and its corresponding eigenvector: hence $A_i x = \lambda x_i$, which yields

$$\sum_{j \neq i} A_{ij} x_j + A_{ii} = \lambda x_i = \lambda.$$

Thus, we obtain $\sum_{j \neq i} A_{ij} x_j = \lambda - A_{ii}$. Therefore,

$$|\lambda - A_{ii}| = \big|\sum_{j \neq i} A_{ij} x_j\big| \leq \sum_{j \neq i} |A_{ij}|\, |x_j|$$

by the triangle inequality. Moreover, since $|x_j| \leq 1$, we have

$$|\lambda - A_{ii}| \leq \sum_{j \neq i} |A_{ij}| = R_i.$$

This implies that $\lambda \in I_i$ as claimed. □

10.6.2 Corollary
If A is DD, then A is PSD.

Proof. Assume the $n \times n$ symmetric matrix A is DD. Let λ be any eigenvalue of A. For each $i \leq n$ we let $R_i = \sum_{j \neq i} |A_{ij}|$. Then for each $i \leq n$ we have $A_{ii} - R_i \geq 0$. By Thm. 10.6.1, this implies $\lambda \geq 0$, which in turns proves that A is PSD. □

The interest of DD matrices is that, by linearization of the absolute value terms, Eq. (10.21) can be reformulated so it becomes linear: we introduce an added matrix T of decision variables, then write:

$$\forall i \leq n \quad \sum_{j \neq i} T_{ij} \leq X_{ii} \tag{10.22}$$

$$-T \leq X \leq T, \tag{10.23}$$

which are linear constraints equivalent to Eq. (10.21) [8, Thm. 10]. One can see this easily whenever $X \geq 0$ or $X \leq 0$. Note that

$$\forall i \leq n \quad X_{ii} \geq \sum_{j \neq i} T_{ij} \geq \sum_{j \neq i} X_{ij}$$

$$\forall i \leq n \quad X_{ii} \geq \sum_{j \neq i} T_{ij} \geq \sum_{j \neq i} -X_{ij}$$

follow directly from Eq. (10.22)-(10.23). Now one of the RHSs is equal to $\sum_{j \neq i} |X_{ij}|$, which implies Eq. (10.21). For the general case, the argument uses the extreme points of Eq. (10.22)-(10.23) and elimination of T by projection.

We can now approximate Eq. (10.19) by the pure feasibility LP:

$$\left.\begin{array}{rrcl} \forall \{u, v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = & d_{uv}^2 \\ \forall i \leq n & \sum_{j \neq i} T_{ij} & \leq & X_{ii} \\ & -T \leq X & \leq & T, \end{array}\right\} \tag{10.24}$$

which we call a *diagonally dominant program* (DDP). As in Eq. (10.19), we do not explicitly give an objective function, since it depends on the application. Since the DDP in Eq. (10.24) is an inner approximation of the corresponding SDP in Eq. (10.19), the DDP feasible set is a subset of that of the SDP. This situation yields both an advantage and a disadvantage: any solution $\tilde{X}$ of the DDP is PSD, and can be

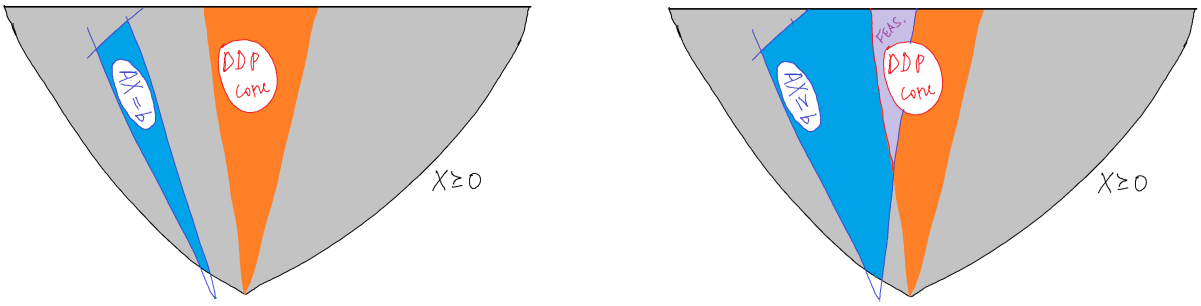


Figure 10.11: On the left, the DDP is infeasible even if the SDP is not; on the right, a relaxed set of constraints makes the DDP feasible.

obtained at a smaller computational cost; however, the DDP might be infeasible even if the corresponding SDP is feasible (see Fig. 10.11, left). In order to decrease the risk of infeasibility of Eq. (10.24), we relax the equation constraints to inequality, and impose an objective as in the push-and-pull formulation Eq. (10.18):

$$\left.\begin{array}{rrcl} \max & \sum\limits_{\{u,v\}\in E} (X_{uu} + X_{vv} - 2X_{uv}) & & \\ \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & \leq & d_{uv}^2 \\ \forall i \leq n & \sum\limits_{j\neq i} T_{ij} & \leq & X_{ii} \\ & -T \leq X & \leq & T. \end{array}\right\} \tag{10.25}$$

This makes the DDP feasible set larger, which means it is more likely to be feasible (see Fig. 10.11, right). Eq. (10.25) was successfully tested on protein graphs in [101].

If C is any cone in $\mathbb{R}^n$, the *dual cone* C^* is defined as:

$$C^* = \{y \in \mathbb{R}^n \mid \forall x \in C \ \langle x, y\rangle \geq 0\}.$$

Note that the dual cone contains the set of vectors making a non-obtuse angle with all of the vectors in the original (primal) cone. We can exploit the dual DD cone in order to provide another DDP formulation for the DGP which turns out to be an outer approximation. Outer approximations have symmetric advantages and disadvantages w.r.t. inner ones: if the original SDP is feasible, than the outer DDP approximation is also feasible; however, the solution $\tilde{X}$ we obtain from the outer DDP need not be a PSD matrix. Some computational experience related to [260] showed that it often happens that more or less half of the eigenvalues of $\tilde{X}$ are negative.

We now turn to the actual DDP formulation related to the dual DD cone. A cone C of $n \times n$ real symmetric matrices is *finitely generated* by a set $\mathcal{X}$ of matrices if:

$$\forall X \in C \ \exists \delta \in \mathbb{R}_+^{|\mathcal{X}|} \quad X = \sum_{x\in\mathcal{X}} \delta_x x x^\top.$$

It turns out [29] that the DD cone is finitely generated by

$$\mathcal{X}_{\mathsf{dd}} = \{e_i \mid i \leq n\} \cup \{e_i \pm e_j \mid i < j \leq n\},$$

where $e_1, \ldots, e_n$ is the standard orthogonal basis of $\mathbb{R}^n$. This is proved in [29] by showing that the following rank-one matrices are extreme rays of the DD cone:

- $E_{ii} = \mathsf{diag}(e_i)$, where $e_i = (0, \ldots, 0, 1_i, 0, \ldots, 0)^\top$;
- E_{ij}^+ has a minor $\begin{pmatrix} 1_{ii} & 1_{ij} \\ 1_{ji} & 1_{jj} \end{pmatrix}$ and is zero elsewhere;

- E_{ij}^{-} has a minor $\begin{pmatrix} 1_{ii} & -1_{ij} \\ -1_{ji} & 1_{jj} \end{pmatrix}$ and is zero elsewhere,

and, moreover, that the extreme rays are generated by the standard basis vectors as follows:

$$\begin{array}{rrcl} \forall i \leq n & E_{ii} & = & e_i e_i^\top \\ \forall i < j \leq n & E_{ij}^{+} & = & (e_i + e_j)(e_i + e_j)^\top \\ \forall i < j \leq n & E_{ij}^{-} & = & (e_i - e_j)(e_i - e_j)^\top. \end{array}$$

This observation allowed Ahmadi and his co-authors to write the DDP formulation Eq. (10.25) in terms of the extreme rays $E_{ii}, E_{ij}^{\pm}$ [8], and also to define a column generation algorithms over them [7].

If a matrix cone is finitely generated, the dual cone has the same property. Let $\mathbb{S}_n$ be the set of real symmetric $n \times n$ matrices; for $A, B \in \mathbb{S}_n$ we define an inner product $\langle A, B \rangle = A \bullet B \triangleq \mathsf{tr}(AB^\top)$.

10.6.3 Theorem
Assume C is finitely generated by $\mathcal{X}$. Then C^* is also finitely generated. Specifically, $C^* = \{Y \in \mathbb{S}_n \mid \forall x \in \mathcal{X}\ (Y \bullet xx^\top \geq 0)\}$.

Proof. By assumption, $C = \{X \in \mathbb{S}_n \mid \exists \delta \in \mathbb{R}_+^{|\mathcal{X}|}\ X = \sum_{x \in \mathcal{X}} \delta_x xx^\top\}$.
($\Rightarrow$) Let $Y \in \mathbb{S}_n$ be such that, for each $x \in \mathcal{X}$, we have $Y \bullet xx^\top \geq 0$. We are going to show that $Y \in C^*$, which, by definition, consists of all matrices Y such that for all $X \in C$, $Y \bullet X \geq 0$. Note that, for all $X \in C$, we have $X = \sum_{x \in \mathcal{X}} \delta_x xx^\top$ (by finite generation). Hence $Y \bullet X = \sum_x \delta_x Y \bullet xx^\top \geq 0$ (by definition of Y), whence $Y \in C^*$.
($\Leftarrow$) Suppose $Z \in C^* \setminus \{Y \mid \forall x \in \mathcal{X}\ (Y \bullet xx^\top \geq 0)\}$. Then there is $\mathcal{X}' \subset \mathcal{X}$ such that for any $x \in \mathcal{X}'$ we have $Z \bullet xx^\top < 0$. Consider any $Y = \sum_{x \in \mathcal{X}'} \delta_x xx^\top \in C$ with $\delta \geq 0$. Then $Z \bullet Y = \sum_{x \in \mathcal{X}'} \delta_x Z \bullet xx^\top < 0$, so $Z \notin C^*$, which is a contradiction. Therefore $C^* = \{Y \mid \forall x \in \mathcal{X}\ (Y \bullet xx^\top \geq 0)\}$ as claimed. □ □

We are going to exploit Thm. 10.6.3 in order to derive an explicit formulation of the following DDP formulation based on the dual cone C_{dd}^* of the DD cone C_{dd} finitely generated by $\mathcal{X}_{\mathsf{dd}}$:

$$\left.\begin{array}{rrcl} \forall \{u, v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = & d_{uv}^2 \\ & X & \in & C_{\mathsf{dd}}^*. \end{array}\right\}$$

We remark that $X \bullet vv^\top = v^\top X v$ for each $v \in \mathbb{R}^n$. By Thm. 10.6.3, $X \in C_{\mathsf{dd}}^*$ can be restated as $\forall v \in \mathcal{X}_{\mathsf{dd}}\ v^\top X v \geq 0$. We obtain the following LP formulation:

$$\left.\begin{array}{rrcl} \max & \sum_{\{u,v\} \in E} (X_{uu} + X_{vv} - 2X_{uv}) & & \\ \forall \{u, v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = & d_{uv}^2 \\ \forall v \in \mathcal{X}_{\mathsf{dd}} & v^\top X v & \geq & 0. \end{array}\right\} \tag{10.26}$$

Note that the constraints $v^\top X v \geq 0$ for $v \in \mathcal{X}_{\mathsf{dd}}$ are equivalent to the following constraints:

$$\begin{array}{rrcl} \forall i \leq n & X_{ii} & \geq & 0 \\ \forall \{i, j\} \notin E & X_{ii} + X_{jj} - 2X_{ij} & \geq & 0 \\ \forall i < j & X_{ii} + X_{jj} + 2X_{ij} & \geq & 0 \end{array}$$

With respect to the primal DDP, the dual DDP formulation in Eq. (10.26) provides a very tight bound to the objective function value of the push-and-pull SDP formulation Eq. (10.18). On the other hand, the solution $\bar{X}$ is usually far from being a PSD matrix.

10.6.5 Barvinok's naive algorithm

By Eq. (10.19), we can solve an SDP relaxation of the DGP and obtain an $n \times n$ PSD matrix solution $\bar{X}$ which, in general, will not have rank K (i.e., it will not yield an $n \times K$ realization matrix, but rather an $n \times n$ one). In this section we shall derive a dimensionality reduction algorithm to obtain an approximation of $\bar{X}$ which has the correct rank K.

10.6.5.1 Quadratic Programming feasibility

Barvinok's naive algorithm [31, §5.3] is a probabilistic algorithm which finds an approximate vector solution $x' \in \mathbb{R}^n$ to a system of quadratic equations

$$\forall i \leq m \quad x^\top Q^i x = a_i, \tag{10.27}$$

where the Q^i are $n \times n$ symmetric matrices, $a \in \mathbb{R}^m$, $x \in \mathbb{R}^n$, and m is polynomial in n. The analysis of this algorithm provides a probabilistic bound on the maximum distance that x' can have from the set of solutions of Eq. (10.27). Thereafter, one can run a local NLP solver with x' as a starting point, and obtain a hopefully good (approximate) solution to Eq. (10.27). We note that this algorithm is still not immediately applicable to our setting where K might be different from 1: we shall address this issue in Sect. 10.6.5.4.

Barvinok's naive algorithm solves an SDP relaxation of Eq. (10.27), and then retrieves a certain randomized vector from the solution:

1. form the SDP relaxation
$$\forall i \leq m \; (Q^i \bullet X = a_i) \wedge X \succeq 0 \tag{10.28}$$
of Eq. (10.27) and solve it to obtain $\bar{X} \in \mathbb{R}^{n \times n}$;

2. let $T = \sqrt{\bar{X}}$, which is a real matrix since $\bar{X} \succeq 0$ (T can be obtained by spectral decomposition, i.e. $\bar{X} = P \Lambda P^\top$ and $T = P\sqrt{\Lambda}$);

3. let y be a vector sampled from the multivariate normal distribution $\mathsf{N}^n(0, 1)$;

4. compute and return $x' = Ty$.

The analysis provided in [31] shows that $\exists c > 0$ and an integer $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$

$$\mathsf{P}\left(\forall i \leq m \quad \mathsf{dist}(x', \mathcal{X}_i) \leq c\sqrt{\|\bar{X}\|_2 \ln n}\right) \geq 0.9. \tag{10.29}$$

In Eq. (10.29), $\mathsf{dist}(b, B) = \inf_{\beta \in B} \|b - \beta\|_2$ is the Euclidean distance between the point b and the set B, and c is a constant that only depends on $\log_n m$. We recall that $\mathsf{P}(\cdot)$ denotes the probability of an event. We note that the term $\sqrt{\|\bar{X}\|_2}$ in Eq. (10.29) arises from T being a factor of $\bar{X}$. We note also that 0.9 follows from assigning some arbitrary value to some parameter — i.e. the constant 0.9 can be increased as long as the problem size is large enough.

For cases of Eq. (10.27) where one of the quadratic equations is $\|x\|_2^2 = 1$ (namely, the solutions of Eq. (10.27) must belong to the unit sphere), it is noted in [31, Eg. 5.5] that, if $\bar{X}$ is "sufficiently generic", then $\|\bar{X}\|_2 = O(1/n)$, which implies that the bounding function $c\sqrt{\bar{X}_2 \ln n} \to 0$ as $n \to \infty$. This, in turn, means that x' converges towards a feasible solution of the original problem in the limit.

10.6.5.2 Concentration of measure

The term $\ln n$ in Eq. (10.29) arises from a phenomenon of high-dimensional geometry called "concentration of measure".

We first give an example of concentration of measure around the median value of a Lipschitz function. We recall that a function $f : \mathcal{X} \to \mathbb{R}$ is *Lipschitz* if there is a constant $M > 0$ s.t. for any $x, y \in \mathcal{X}$ we have $|f(x) - f(y)| < M\|x - y\|_2$. A measure space $(\mathcal{X}, \mu)$ has the *concentration of measure* property if for any Lipschitz function f, there are constants $C, c > 0$ such that:

$$\forall \varepsilon > 0 \quad \mathsf{P}(|f(x) - \mathsf{M}_\mu(f)| > \varepsilon \mid x \in \mathcal{X}) \leq C\, e^{-c\varepsilon^2} \tag{10.30}$$

where $\mathsf{M}_\mu(f)$ is the median value of f w.r.t. μ. In other words, $\mathcal{X}$ has measure concentration if for any Lipschitz function f, its discrepancy from its median value is small with arbitrarily high probability. It turns out that the Euclidean space $\mathbb{R}^n$ with the Gaussian density measure $\phi(x) = (2\pi)^{n/2} e^{-\|x\|_2^2/2}$ has measure concentration around the mean [32, §5.3].

Measure concentration is interesting in view of applications since, given any large enough closed subset A of $\mathcal{X}$, its ε-neighbourhood

$$A(\varepsilon) = \{x \in \mathcal{X} \mid \mathsf{dist}(x, A) \leq \varepsilon\} \tag{10.31}$$

contains almost the whole measure of $\mathcal{X}$. More precisely, if $(\mathcal{X}, \mu)$ has measure concentration and $A \subset \mathcal{X}$ is closed, then for any $p \in (0, 1)$ there is a $\varepsilon_0(p) > 0$ such that [193, Prop. 2]:

$$\forall \varepsilon \geq \varepsilon_0(p) \quad \mu(A(\varepsilon)) > 1 - p. \tag{10.32}$$

Eq. (10.32) is useful for applications because it defines a way to analyse probabilistic algorithms. For a random point sampled in $(\mathcal{X}, \mu)$ that happens to be in A on average, Eq. (10.32) ensures that it is unlikely that it should be far from A. This can be used to bound errors, as Barvinok did with his naive algorithm. Concentration of measure is fundamental in data science, insofar as it may provide algorithmic analyses to the effect that some approximation errors decrease in function of the increasing instance size.

10.6.5.3 Analysis of Barvinok's algorithm

We sketch the main lines of the analysis of Barvinok's algorithm (see [30, Thm. 5.4] or [193, §3.2] for a more detailed proof). We let $\mathcal{X} = \mathbb{R}^n$ and $\mu(x) = \phi(x)$ be the Gaussian density measure. It is easy to show that

$$\mathsf{E}_\mu(x^\top Q^i x \mid x \in \mathbb{R}^n) = \mathsf{tr}(Q^i)$$

for each $i \leq m$. From this fact and the factorization $\bar{X} = TT^\top$, one obtains

$$\mathsf{E}_\mu(x^\top T^\top Q^i\, Tx \mid x \in \mathcal{X}) = \mathsf{tr}(T^\top Q^i\, T) = \mathsf{tr}(Q^i \bar{X}) = Q^i \bullet \bar{X} = a_i.$$

This shows that, for any $y \sim \mathsf{N}^n(0, 1)$, the average of $y^\top T^\top Q^i\, Ty$ is a_i.

The analysis then goes on to show that, for some $y \sim \mathsf{N}^n(0, 1)$, it is unlikely that $y^\top T^\top Q^i\, Ty$ should be far from a_i. It achieves this result by defining the sets $A_i^+ = \{x \in \mathbb{R}^n \mid x^\top Q^i x \geq a_i\}$, $A_i^- = \{x \in \mathbb{R}^n \mid x^\top Q^i x \leq a_i\}$, and their respective neighbourhoods $A_i^+(\varepsilon)$, $A_i^-(\varepsilon)$. Using a technical lemma [193, Lemma 4] it is possible to apply Eq. (10.32) to $A_i^+(\varepsilon)$ and $A_i^-(\varepsilon)$ to argue for concentration of measure. Applying the union bound it can be shown that their intersection $A_i(\varepsilon)$ is the neighbourhood of $A_i = \{x \in \mathbb{R}^n \mid x^\top Q^i x = a_i\}$. Another application of the union bound to all the sets $A_i(\varepsilon)$ yields the result [193, Thm. 5].

We note that concentration of measure proofs often have this structure: (a) prove that a certain event holds on average; (b) prove that the discrepancy from average gets smaller and/or more unlikely with increasing size. Usually proving (a) is easier than proving (b).

10.6.5.4 Applicability to the DGP

The issue with trying to apply Barvinok's naive algorithm to the DGP is that we should always assume $K = 1$ by Eq. (10.27). To circumvent this issue, we might represent an $n \times K$ realization matrix as a vector in $\mathbb{R}^{nK}$ by stacking its columns (or concatenating its rows). This, on the other hand, would require solving SDPs with $nK \times nK$ matrices, which is prohibitive because of size.

Luckily, Barvinok's naive algorithm can be very easily extended to arbitrary values of K. We replace Step 3 by:

3*b*. let y be an $n \times K$ matrix sampled from $\mathsf{N}^{n \times K}(0, 1)$.

The corresponding analysis needs some technical changes [193], but the overall structure is the same as the case $K = 1$. The obtained bound replaces $\sqrt{\ln n}$ in Eq. (10.29) with $\sqrt{\ln nK}$.

In the DGP case, the special structure of the matrices Q^i (for i ranging over the edge set E) makes it possible to remove the factor K, so we retrieve the exact bound of Eq. (10.29). As noted in Sect. 10.6.5.1, if the DGP instance is on a sphere [192], this means that $x' = Ty$ converges to an exact realization with probability 1 in the limit of $n \to \infty$. Similar bounds to Eq. (10.29) were also derived for the iDGP case [193].

Barvinok also described concentration of measure based techniques for finding low-ranking solutions of the SDP in Eq. (10.28) (see [30] and [32, §6.2]), but these do not allow the user to specify an arbitrary rank K, so they only apply to the EDMCP.

10.6.6 Isomap

One of the most interesting applications of PCA to the DGP is possibly the Isomap algorithm [287]. Isomap uses PCA to perform a nonlinear dimensional reduction from the original dimension m to a given target dimension K, as follows.

1. Form a connected graph $H = (V, E)$ with the column indices $1, \ldots, n$ of X as vertex set V: determine a threshold value τ such that, for each column vector x_i in X (for $i \leq n$), and for each x_j in X such that $\|x_i - x_j\|_2 \leq \tau$, the edge $\{i, j\}$ is in the edge set E; the graph H should be as sparse as possible but also connected.
2. Complete H using the shortest-path metric (Eq. (10.4)).
3. Use PCA in the MDS interpretation mentioned above. The completion of (V, E) is a metric space; we construct its (approximate) EDM $\tilde{D}$, compute the corresponding (approximate) Gram matrix $\tilde{G}$, compute the spectral decomposition of $\tilde{G}$, replace its diagonal eigenvalue matrix Λ as in Eq. (10.10), and return the corresponding K-dimensional vectors.

Intuitively, Isomap works well because in many practical situations where a set X of points in $\mathbb{R}^m$ are close to a (lower) K-dimensional manifold, the shortest-path metric is likely to be a better estimation of the Euclidean distance in $\mathbb{R}^K$ than the Euclidean distance in $\mathbb{R}^m$, see [287, Fig. 3] (reproduced in Fig. 10.12).

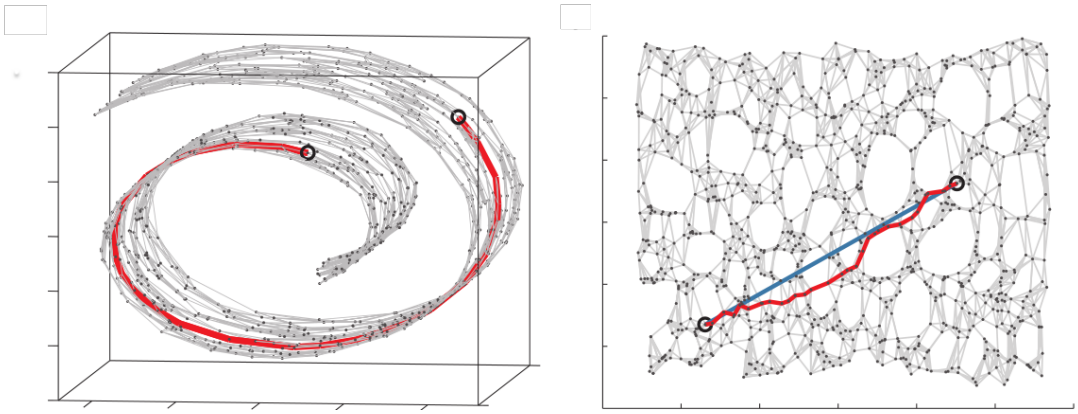


Figure 10.12: The shortest-path distance gives a better approximation than the Euclidean one to the natural distance defined on the manifold interpolating the data.

Chapter 11

Quantile regression

Quantile Regression (QR) is similar to linear regression, but gives more information. In fact a better analogy would be to median regression, since the median is the 0.5-quantile. More information about QR can be found in [159, 313].

In general, quantile regression plots in 2D give a better visual information about a linear trend of the data than linear regression does (see Fig. 11.1).

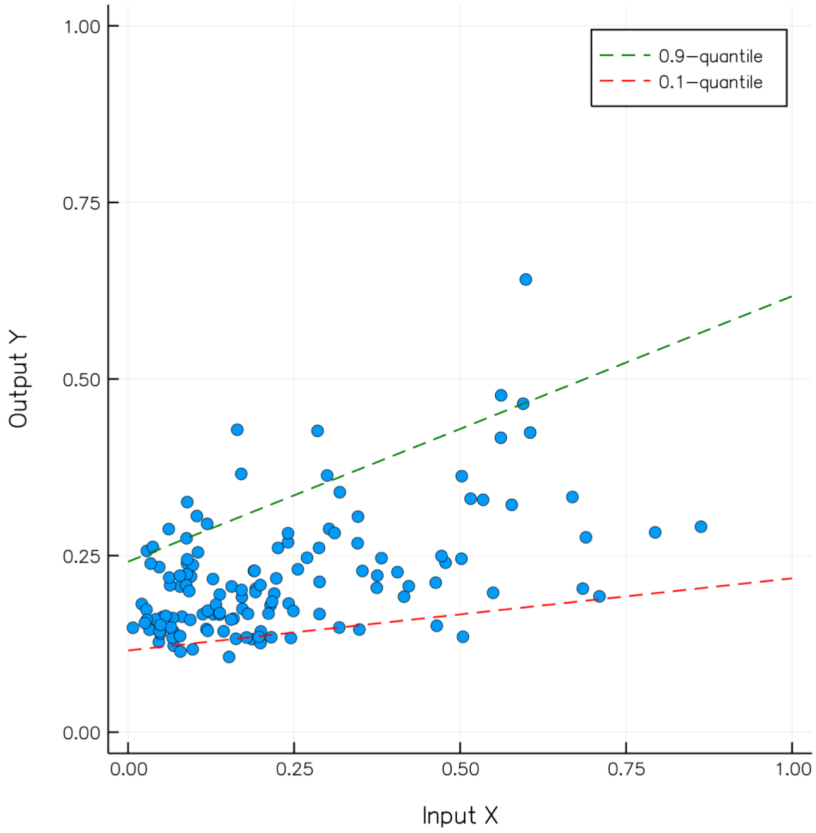


Figure 11.1: The two quantiles for 0.1 and 0.9 of a random dataset.

11.1 Quantiles

Consider a random variable (r.v.) X with cumulative distribution function $F(x) = \mathsf{Prob}(X \leq x)$. The *median* is the value m such that $F(m) = 0.5$. The τ-*quantile* is the value q such that $F(q) = \tau$.

11.1.1 Example
Let X be a r.v. taking values in $\{2, 3, 5, 10, 11\}$ with uniform distribution. Then the 0.8-quantile is the value 10, since

$$F(10) = \mathsf{Prob}(X \leq 10) = \sum_{x \in \{2,3,5,10\}} \mathsf{Prob}(X = x) = \sum_{x \in \{2,3,5,10\}} 0.2 = 0.8.$$

11.2 Regression

The term *regression* refers to a particular (linear) form of conditional probability. Let B be a r.v. conditional on the r.v.s $A_1, \ldots, A_p$. Assume further that B depends linearly on $A_1, \ldots, A_p$. Regression methods find coefficients $x_1, \ldots, x_p$ such that $B = \sum_{j \leq p} x_j A$ according to various criteria. In particular, the components of the x vector $(x_1, \ldots, x_p)$ are estimated using data samples $b, a_1, \ldots, a_p \in \mathbb{R}^m$ (each sample is a vector of m components, meaning that we randomly sample m values of each r.v. according to the corresponding distribution). We obtain a data matrix $A = (a^i_j \mid i \leq m \wedge j \leq p)$, where a^i denotes the i-th row and a_j the j-th column of A (a stacking of the row vectors $a_1, \ldots, a_p$).

There are multiple forms of regression. The best known is regression to the mean, also known as *linear regression*. It is best understood using the definition of the sample mean of B:

$$\hat{\mu} = \arg\min_{\mu \in \mathbb{R}} \sum_{i \leq m} (b_i - \mu)^2.$$

Now, since we know that B is a linear combination of the r.v.s A_j, we can replace the single decision variable μ in the above QP with νa^i, where $\nu \in \mathbb{R}^p$ is a row vector of decision variables:

$$\hat{\nu} = \arg\min_{\nu \in \mathbb{R}^p} \sum_{i \leq m} (b_i - \nu a^i)^2,$$

which yield a (multivariate) QP.

We can proceed similarly for the median. The median is obtained by solving the following single-variable MP formulation:

$$\begin{aligned} \hat{\xi} &= \arg\min_{\xi \in \mathbb{R}} \sum_{i \leq m} |b_i - \xi| \\ &= \arg\min_{\xi \in \mathbb{R}} \sum_{i \leq m} \left(\frac{1}{2} \max(b_i - \xi, 0) - \frac{1}{2} \min(b_i - \xi, 0) \right). \end{aligned}$$

After replacement of ξ by ζa^i, we obtain:

$$\hat{\zeta} = \arg\min_{\zeta \in \mathbb{R}^p} \sum_{i \leq m} \left(\frac{1}{2} \max(b_i - \zeta a^i, 0) - \frac{1}{2} \min(b_i - \zeta a^i, 0) \right).$$

Finally, we generalize the median, which is a 0.5-quantile, to a τ-quantile for any $\tau \in [0, 1]$. From

$$\hat{\xi} = \arg\min_{\xi \in \mathbb{R}} \sum_{i \leq m} \left(\tau \max(b_i - \xi, 0) - (1 - \tau) \min(b_i - \xi, 0) \right),$$

after replacement of ξ by βa^i, we obtain:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i \leq m} \left(\tau \max(b_i - \beta a^i, 0) - (1 - \tau) \min(b_i - \beta a^i, 0) \right). \tag{11.1}$$

11.3 LP formulation

In this section we propose an LP formulation for performing QR. The formulation is:

$$\left.\begin{array}{rll} \hat{\beta} = \underset{\substack{u^+,u^- \in \mathbb{R}^m \\ \beta \in \mathbb{R}^p}}{\arg\min} & \tau \mathbf{1}^\top u^+ + (1-\tau)\mathbf{1}^\top u^- & \\ & A\beta + u^+ - u^- &= b \\ & u^+, u^- &\geq 0, \end{array}\right\} \tag{11.2}$$

where the parameters are the $m \times p$ matrix A, the vector $b \in \mathbb{R}^m$, and the scalar $\tau \in \mathbb{R}$. The decision variables are $\beta \in \mathbb{R}^p$, and $u^+, u^- \in \mathbb{R}^m_+$.

11.3.1 Theorem
The LP in Eq. (11.2) has the same optima $\hat{\beta}$ as the QR problem in Eq. (11.1).

Proof. The proof applies a linearization reformulation (Sect. 2.2.3.5) to Eq. (11.1). We then argue that this yields Eq. (11.2). For all $i \leq m$ carry out the following replacement:

$$u_i^+ = \max(b_i - \beta a^i, 0) \quad u_i^- = -\min(b_i - \beta a^i, 0).$$

It is easy to see that we have $u^+, u^- \geq 0$ by definition. We claim that either u_i^+ or u_i^- or both are going to be zero at any optimum: otherwise, we could assign to $u_i^\pm$ the value $u_i^\pm - \min(u_i^+, u_i^-)$, which immediately implies that $\min(u_i^+, u_i^-) = 0$: this contradicts optimality since $u_i^+ + u_i^-$ decreases. Next, we prove that:

$$\forall i \leq m \quad u_i^+ - u_i^- = b_i - \beta a^i. \tag{11.3}$$

By the claim above, at most one of u_i^+ and u_i^- is nonzero at the optimum. If $u_i^+ \neq 0$ then $u_i^+ - u_i^- = u_i^+ = b_i - \beta a^i$. If $u_i^- \neq 0$ then $u_i^+ - u_i^- = -u_i^- = b_i - \beta a^i$. Now it is simply a matter of notation: recall $A = (a_j^i \mid i \leq m \wedge j \leq p)$. This means that

$$\big(\forall i \leq m \quad b_i - \beta a^i\big) \equiv b - A\beta,$$

whence $A\beta + u^+ - u^- = b$ as claimed. □

We note that Eq. (11.2) is an LP having a form which is close to the standard form

$$\left.\begin{array}{rll} \min\limits_{x \in \mathbb{R}^n} & c^\top x & \\ & \bar{A}x &= b \\ & x &\geq 0. \end{array}\right\}$$

This is apparent if we write $x = (\beta, u^+, u^-) \in \mathbb{R}^{p+2m}$,

$$c = (\underbrace{0, \ldots, 0}_{p}, \underbrace{\tau, \ldots, \tau}_{m}, \underbrace{1-\tau, \ldots, 1-\tau}_{m}),$$

and $\bar{A} = (A, I_m, -I_m)$ (where I_m is the $m \times m$ identity matrix). The only difference is that only $u^+, u^- \geq 0$, but β need not be constrained. It would be easy to achieve a standard form by replacing β by $\beta^+ - \beta^-$, where $\beta^+, \beta^- \in \mathbb{R}^p_+$.

11.3.1 Density

Eq. (11.2) in standard form has a $m \times (p + 2m)$ constraint matrix $\bar{A}$ with $\frac{p}{p+2m}$ density. Moreover, the size of $\bar{A}$ depends on the size of A, which can grow as large as the database table it represents. Compared with most large-sized LPs, the density of Eq. (11.2) for typical values of p, m is considerably high (see Fig. 11.2). So much, in fact, that QR is known to be a numerically challenging problem (a few experiments with R's [250] QR module with a reasonably large dataset should convince you of this).

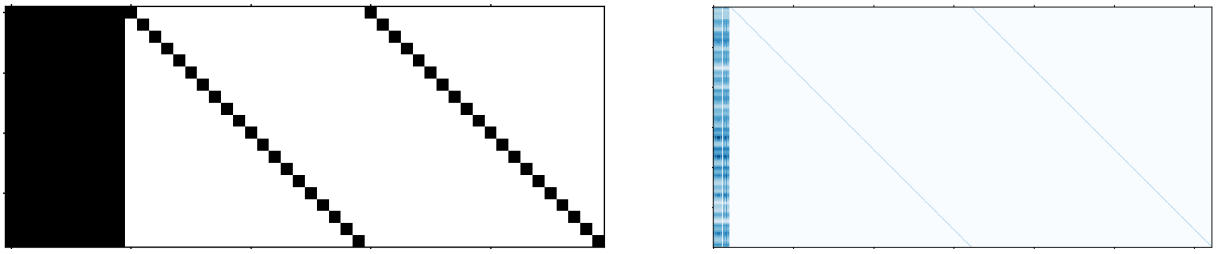


Figure 11.2: QR LP constraint matrices: left, with density 0.2; right, with density 0.0012. Both are large densities w.r.t. typical LPs arising from combinatorial problems.

11.4 Solution properties

The QR problem can be expressed as $\min q(u) = \tau u^+ + (1-\tau)u^-$ such that $u^+ - u^- = A\beta - b$, where $u = (u^+, u^-)$. In other words, it minimizes a solution error to the linear system $A\beta = b$, where A is $m \times p$. We assume for now that A has full rank.

Let u^* is a solution of the QR LP. The average solution error is $\frac{1}{m}q(u^*)$. We are going to consider a solution of Eq. (11.2) "good" if its average error is small with respect to the average data value in A. If $m < p$, $A\beta = b$ is underdetermined. If $m = p$, $A\beta = b$ is a square system having a unique solution. If $m > p$, $A\beta = b$ is an overdetermined system with no solutions. Since we are minimizing the error $q(u)$ from solving $A\beta = b$ exactly, if $m \leq p$ then $\frac{1}{m}q(u^*) = 0$, otherwise $\frac{1}{m}q(u^*) > 0$.

We now review the assumption that A should have full rank. This assumption may be unwarranted, since A is realistically a numerical database table, which might have empty columns, empty rows, and values outside our control. More precisely, real-world databases often have:

- large/negative values to indicate "invalid entry";
- zero columns (unfilled database fields);
- different columns with the same values (for example, in European energy price databases it often happens that different countries within the Schengen treaty might apply exactly the same prices in some context).

These common occurrences worsen the degeneracy of the QR LP. Some of these effects can be lightened using pre-processing techniques. It is easy, for example, to replace "invalid entry" with some scalar which is not used elsewhere in the table. This might create a bias, but at least makes the computation possible. Zero or equal columns can also be dealt with easily using pre-processing, as long as the table is static. This ceases to be the case if the table is an aggregation of data from multiple sources arriving in real-time.

Post-processing can also help us. Assuming we could solve the QR LP and obtain a solution β^*, if the j-th column is empty we can ignore β^*_j. If columns j, h have the same values, the QR LP has a solution symmetry group [172, 173] including the swap (j, h).

In general, the IPM algorithm (a.k.a. "barrier method") (see Sect. 7.5) is better suited to solve degenerate LPs than the simplex method. Note that most IPM implementations run some iterations of the simplex method to achieve an exact LP solution (this is known as the *crossover* phase). It is important to disable crossover whenever the LP input to the IPM-based solver is very degenerate.

11.5 Prediction and visualization

Fig. 11.1 shows the quantiles as lines in the plane, which do not necessarily pass through the origin. On the other hand, we defined QR as the determination of the best linear subspace satisfying some criteria. Since all linear subspaces of the Euclidean space must necessarily pass through the origin, there seems to be an apparent contradiction. To dispel this, it suffices to add one r.v. A_0 to the list $A_1, \ldots, A_p$, and set A_0 to take the value 1 with certainty. This will yield a sample $a_0 = (1, \ldots, 1) \in \mathbb{R}^m$, which, in turn, means that the linear system $A\beta = b$ becomes

$$(\mathbf{1}|A) \begin{pmatrix} \beta_0 \\ \beta \end{pmatrix} = b,$$

or, more explicitly,

$$\forall i \leq m \quad \beta_0 + \sum_{j \leq p} A_{ij}\beta_j = b_i,$$

which defines m affine subspaces in $\mathbb{R}^p$. We note that this "trick" to interpret linear as affine subspaces is independent of QR.

The affine subspaces provided by QR can be used to predict trends based on data. Such predictions can be based on purely quantitative analyses of the QR solution, but it sometimes help to visualize the affine subspaces. The issue is that visualization can only occur in 2D or 3D. We focus here on the 2D case, which means that the linear model for QR is about the dependence of the r.v. B from a single r.v. from the set $A_1, \ldots, A_p$. We assume wlog that we look at the dependence of B from A_1, which yields:

$$\forall i \leq m \quad a_{i1}\beta_1 + \beta_2 = b_i. \tag{11.4}$$

The QR LP in 2D therefore becomes:

$$\left.\begin{array}{lrcl} \min_{\beta,u} & \tau \sum\limits_{i \leq m} u_i^+ + (1-\tau) \sum\limits_{i \leq m} u_i^- & & \\ \forall i \leq m & \beta_1 a_{i1} + \beta_2 + u_i^+ - u_i^- & = & b_i \\ \forall i \leq m & u_i^+, u_i^- & \geq & 0. \end{array}\right\} \tag{11.5}$$

11.6 Constrained QR

In this section we denote the QR computation problem on data A, b for a given value of τ as $\mathsf{QR}_\tau(A, b)$. Usually, one needs to solve $\mathsf{QR}_\tau(A, b)$ for different values of τ ranging in a discretization $T = \{\tau_1, \ldots, \tau_L\}$ of $(0, 1)$, where we assume that $\tau_h \leq \tau_\ell$ for all $h < \ell$ (e.g. $T = \{0.1, 0.2, \ldots, 0.9\}$). This can be achieved by separately solving $\mathsf{QR}_\tau(A, b)$ for each $\tau \in T$. Each solution $\beta^*(\tau)$ gives rise to a different subspace in $\mathbb{R}^p$.

On the other hand, prior knowledge about the data may influence the relative positions of these subspaces. In [247] we find requirements such as "quantile subspaces must be parallel" and "quantile subspaces cannot cross over the domain of the data points". In order to satisfy side constraints of this type, we leverage on the flexibility of MP. We arrange each constraint matrix corresponding to $\mathsf{QR}_{\tau_\ell}(A, b)$ (for $\ell \leq L$) in a block structure, as shown in Fig. 11.3, and sum the objectives for each quantile:

$$\min \sum_{\ell \leq L} \sum_{i \leq m} (\tau_\ell u_{i\ell}^+ + (1-\tau_\ell) u_{i\ell}^-). \tag{11.6}$$

Note that each decision variable needs to be indexed by $\ell \leq L$ too: in this setting, β is a $(p+1) \times L$ matrix, and u^+, u^- are $m \times L$ matrices. The block-structured formulation is:

$$\left.\begin{array}{lrcl} \min\limits_{\substack{u^+,u^- \in \mathbb{R}^{m \times L} \\ \beta \in \mathbb{R}^{(p+1) \times L}}} & \sum\limits_{\ell \leq L} \tau_\ell \mathbf{1}^\top u_\ell^+ + (1-\tau_\ell)\mathbf{1}^\top u_\ell^- & & \\ \forall \ell \leq L & A\beta_\ell + \beta_{0\ell} + u_\ell^+ - u_\ell^- & = & b \\ & u^+, u^- & \geq & 0. \end{array}\right\} \tag{11.7}$$

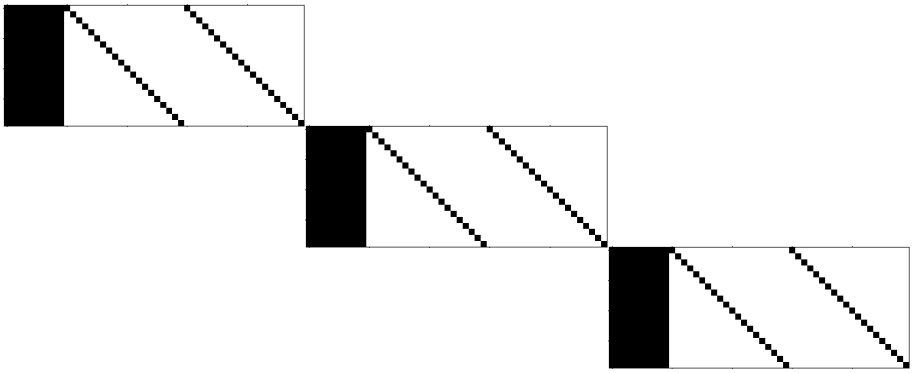


Figure 11.3: Multiple QR problems in a single LP.

We can now adjoin side constraints to Eq. (11.7) in order to enforce the geometric requirements on β.

- Parallel quantile subspaces:

$$\forall h < \ell \leq L, 1 \leq j \leq p \quad \beta_{jh} = \beta j \ell,$$

  i.e. only the affine constant β_0 may vary between subspaces h and ℓ.

- Non-crossing quantile subspaces:

$$\forall h < \ell \leq L \quad A\beta_h + \beta_{0h} \leq A\beta_\ell + \beta_{0\ell}.$$

Fig. 11.4 shows how the quantiles vary with these side constraints.

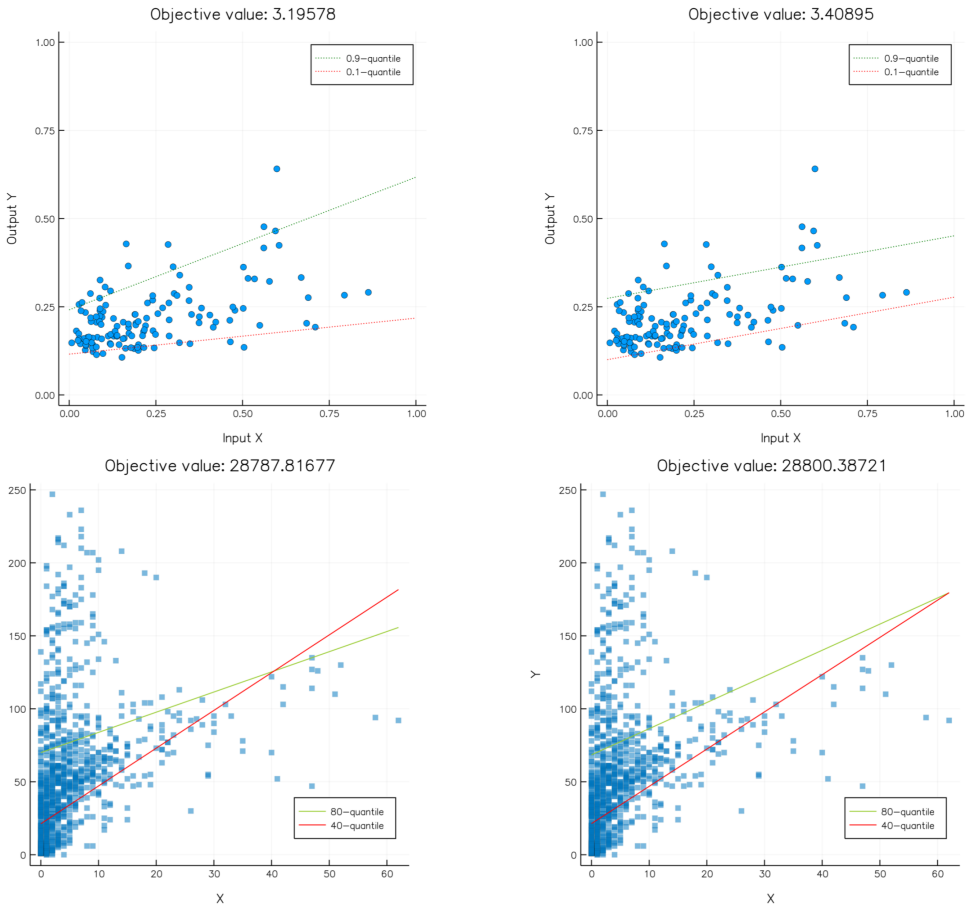


Figure 11.4: Parallel and non-crossing quantiles.

Chapter 12

Sparsity and ℓ_1 minimization

In this chapter we look at another type of LP which is usually even denser than the LPs arising in QR (see Sect. 11.3.1). This LP arises originally from decoding messages received over a communication line.

12.1 Motivation

We present two applications where a dense LP is used to decode a received message.

12.1.1 Coding problem for costly channels

Suppose you need to send data over a costly communication line, e.g. a satellite link. Consider a digitized wave representing a human voice over a time interval. Such a signal is usually easy to transform to a reasonably sparse vector using some efficient pre-processing of patterns of consecutive equal data (this might occur because of silences in conversation, or vowel sounds). Let $y \in \mathbb{R}^n$ be a long ($n \gg 1$) sparse vector representing the signal to be sent over the line.

We assume that the sender and the receiver both know a full rank $m \times n$ matrix A with $m \leq n$ (in fact $m \ll n$). Such a matrix can be randomly sampled, and can be used for every communication of a similarly sparse vector in $\mathbb{R}^n$. The density of the signal vector is related with m, as we shall see later. Communicating the matrix itself is not an issue in this context — for example the two parties can exchange the seed of a pseudorandom number generator.

The communication protocol is as follows:

1. the sender computes $b = Ay \in \mathbb{R}^m$;
2. the sender puts b over the costly line (recall $m \ll n$);
3. the receiver retrieves y as the sparsest solution of the system $Ax = b$.

Step 3 makes two assumptions: (i) there is a unique sparsest solution of the linear system $Ax = b$; (ii) the sparsest solution can be found efficiently. These assumptions do not hold in general, but, surprisingly, they do hold with high probability depending on the relationship between the density of y and m, as we shall see below.

12.1.2 Coding problems for noisy channels

Suppose you need to send a message $w \in \mathbb{R}^d$ (for some $d \in \mathbb{N}$) over a noisy communication line with a known error rate $e \in [0, 1]$, i.e. a fraction e of the signal packets sent over the line may be corrupt in unforeseeable ways. We assume that $e \ll 1$. We also assume (again) that both sender and receiver know a certain $n \times d$ matrix Q, with $n > d$.

The communication protocol is as follows:

1. the sender computes $z = Qw$, which is a vector in $\mathbb{R}^n$;
2. the receiver receives a vector $\bar{z} = z + y$, where y is the error vector: the density of y is the same as the error rate e of the line;
3. the receiver chooses an $m \times n$ matrix A such that $m = n - d$ (so $m < n$) and $AQ = 0$ (this can be done efficiently in a number of ways);
4. the receiver computes $b = A\bar{z}$: note that
$$b = A\bar{z} = A(z + y) = A(Qw + y) = AQw + Ax = Ay$$
since $AQ = 0$ by construction;
5. the receiver retrieves y as the sparsest solution of the system $Ax = b$.
6. the receiver recovers z as $\bar{z} - x$, and w as $(Q^\top Q)^{-1}Q^\top z$.

Note that Step 5 is the same as Step 3 in Sect. 12.1.1, and therefore makes the same assumptions.

12.2 Sparsest solution of a linear system

We define the zero-norm (which is not actually a norm) $\|x\|_0$ of a vector $x \in \mathbb{R}^n$ as the number of zero components of x. The sparsest solution of a linear system $Ax = b$, where A is $m \times n$ with $m < n$, is defined as the globally optimal solution of the following MP:

$$\min\{\|x\|_0 \mid Ax = b\}. \tag{12.1}$$

We note first that such a linear system is underdetermined, and if A has the same rank as (A, b), then it has uncountably many solutions (otherwise it has no solution).

We note that Eq. (12.1) is **NP**-hard by reduction from EXACT COVER BY 3-SETS [116, A6(MP5)]. On the other hand, it was empirically observed in the signal processing literature over many years that solving the following relaxation

$$\min\{\|x\|_1 \mid Ax = b\} \tag{12.2}$$

of Eq. (12.1) very often yielded the sparsest solution of $Ax = b$. Note that Eq. (12.2) is a convex NLP which is easy to turn into an LP, as we shall see below. Although Eq. (12.2) fails to give the correct optima in some cases, the coincidences became too many to be thought of as such. The theoretical explanation behind such occurrences were given in [62, 278], which spawned a considerable body of subsequent research. The best known collective names for this body of knowledge is *compressed sensing*.

12.3 MILP formulation and LP relaxation

Eq. (12.1) can be formulated as the following MILP:

$$\left.\begin{array}{rrcl} \min & \sum_{j\leq n} s_j & & \\ \forall j \leq n & -Ms_j \leq & x_j & \leq Ms_j \\ & Ax & = & b \\ & s & \in & \{0,1\}^n, \end{array}\right\} \tag{12.3}$$

where M is a large enough constant (see Sect. 2.2.7.5.1).

The LP formulation of Eq. (12.2) is:

$$\left.\begin{array}{rrcl} \min & \sum_{j\leq n} s_j & & \\ \forall j \leq n & -Ms_j \leq & x_j & \leq Ms_j \\ & Ax & = & b. \end{array}\right\} \tag{12.4}$$

Note that Eq. (12.4) is basically the continuous relaxation of Eq. (12.3) (bar the constraints $s \in [0,1]^n$, which, however, can be adjoined to Eq. (12.4) wlog). The method for finding sparsest solutions of linear systems based on solving the LP in Eq. (12.4) is known as the *basis pursuit* method.

12.3.1 Exercise
Why can the constraints $x \in [0,1]^n$ be adjoined to Eq. (12.4) wlog?

We recall the two assumptions in Sect. 12.1.1: (i) there is a unique sparsest solution of the linear system $Ax = b$; (ii) the sparsest solution can be found efficiently. With respect to Eq. (12.3)-(12.4), assumption (ii) can be restated as follows:

solving the LP in Eq. (12.4) yields a solution to the MILP in Eq. (12.3).	$(*)$

Finally, we note that, if the application warrants x to be in $[-1,1]$, then we can set $M = 1$. We shall make this assumption in the rest of this section to simplify notation.

12.4 Intuitive explanations

Before the formal arguments, we supply some intuitive explanation as to why the assumptions (i) and (ii) in Sect. 12.1.1 hold. These explanations are borrowed from [61], and they come with the same warning: they are not a proof, and there are cases they cannot explain. Let us look at Fig. 12.1.

- Simplistic intuition: the minimum ℓ_1 norm vector in the $Ax = b$ subspace is represented as an extreme vertex of the ℓ_1 ball polytope. This vertex is sparse (since it is one of the standard basis elements); and there is high probability for a subspace to have this property.
- More convoluted intuition: the descent cone of the ℓ_1 norm is the cone pointed at x, i.e. moving within that cone would reduce the ℓ_1 norm. In order for x not to have minimum ℓ_1 norm on $Ax = b$ would be that the $Ax = b$ subspace intersects the descent cone (which is not the case in Fig. 12.1). Moreover, the descent cone is narrow at sparse vectors, so if the dimensionality of the subspace $Ax = b$ is large enough, it will likely miss the descent cone.

Fig. 12.2 suggests the same intuition, but in 2D. It also explains why the ℓ_1 norm is the only ℓ_p norm to ensure assumptions (i) and (ii), and suggests that some nonconvex pseudonorms also have the same property.

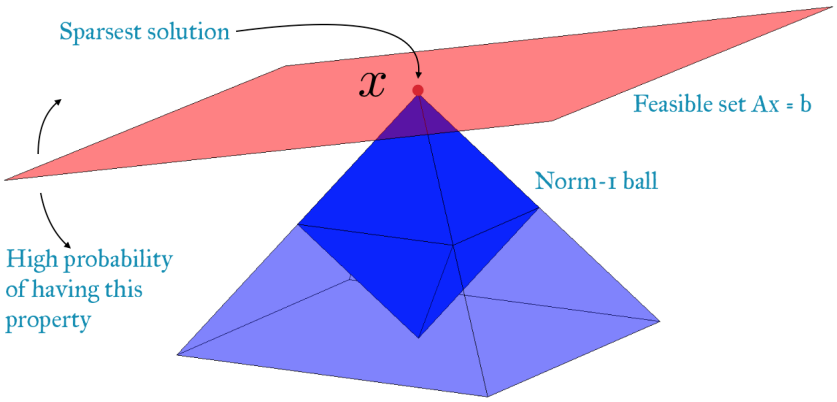


Figure 12.1: An intuitive explanation of compressed sensing (from [61]).

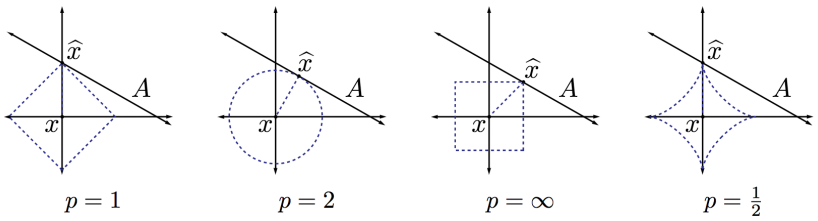


Figure 12.2: Some norms satisfy the assumptions with high probability, others do not (from [92]).

12.5 The phase transition

In this section we informally discuss the relationship between the density of x and m in Eq. (12.2). Fig. 12.3 shows two empirical tests with A being $m \times n$, and s being the number of nonzeroes in the solution x^* of Eq. (12.2). The pixels in the pictures are colored in grayscale: their intensity ranges in the interval $[0, 1]$, with 0 being black and 1 being white. Therefore, each pixel can represent, by means of its color, a probability. More precisely, each pixel represents the frequency with which 10 solutions of Eq. (12.2) with randomly sampled A matrices (with varying m) yield an optimal solution x^* with at most s nonzeroes. For example, if you want to find a unique sparse solution with up to 50 nonzeroes in $\mathbb{R}^{100}$ with very high probability, you need $m \geq 90$. If you limit the definition of "sparse" to 10 nonzeroes, m can be around 40.

12.6 Theoretical results

In this section we give an analysis of the theory of compressed sensing. We mainly follow the treatment in [86]. We denote $P^0(A, b)$ the problem defined in Eq. (12.1), and by $P^1(A, b)$ the problem defined in Eq. (12.2).

12.6.1 Main theorem

We shall present a proof sketch of the following main result. Assume the following hold:

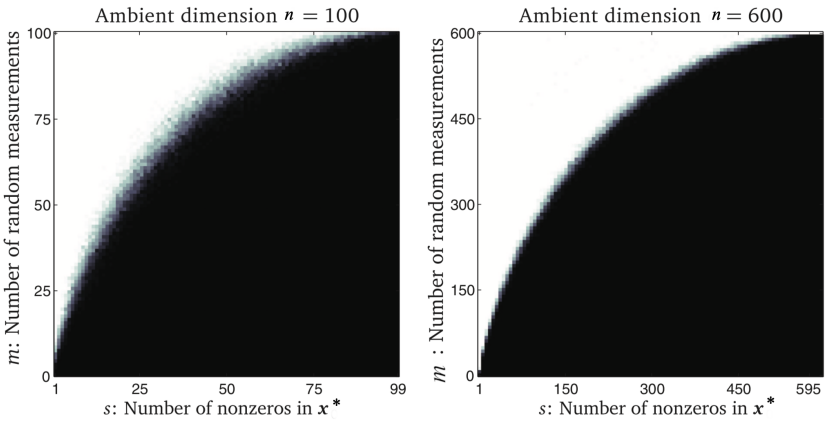


Figure 12.3: The phase transition of compressed sensing, with $n = 100$ (left) and $n = 600$ (right) [19].

- $\hat{x} \in \mathbb{R}^n$ has s nonzeros and $n - s$ near-zeros or zeros
- $\bar{x}$ is the closest vector (in ℓ_2 norm) to $\hat{x}$ with exactly s nonzeros
- A is sampled from $\mathsf{N}(0,1)^{mn}$ (a multivariate standard normal distribution) with $m < n$, but such that m is "not too small"
- $\hat{b} = A\hat{x}$ and x^* is the unique s-sparse minimum of $P^1(A, \hat{b})$

then

x^* is a "good approximation" of $\bar{x}$ $\quad (\star)$

By "good approximation" we mean that we provide reasonably tight (theoretical) bounds to some reasonable form of distance between x^* and $\bar{x}$, as detailed below.

In order to illustrate at least a part of the proof of this theorem, we shall need two notions: the *null space property* (NSP), and the *restricted isometry property* (RIP), which we will formally introduce later. Then $(*)$ will hold by the following propositions.

1. If A has the NSP, then $(*)$ holds.
2. If A has the RIP, then it has the NSP.
3. If A is sampled from $\mathsf{N}(0,1)^{mn}$, then A has the RIP.

12.6.2 The null space property

We consider the system $Ax = b$ where A is $m \times n$ and $m < n$. Let $x \in \mathbb{R}^n$ be such that $Ax = b$, Let $N_A = \mathsf{null}(A)$ be the null space of A, and let $N_A^0 = N_A \setminus \{0\}$. Note that this implies that:

$$\forall y \in N_A \quad A(x + y) = Ax + Ay = Ax + 0 = b.$$

We now look at the indices of the components of vectors in $\mathbb{R}^n$. Let $[n] = \{1, \ldots, n\}$. For any subset $S \subseteq [n]$ let $\bar{S} = [n] \setminus S$. We also define a vector restriction of any $z \in \mathbb{R}^n$ given a subset S of its component indices, namely

$$z[S] = ((z_j \text{ iff } j \in S) \text{ xor } 0 \mid j \leq n).$$

We call $z[S]$ the *restriction* of z to S. Note that this restriction does not remove components, it just zeroes those that are outside of S. that we have

$$z = z[S] + z[\bar{S}].$$

The NSP is a matrix property parametrized on an integer s. For a matrix A we denote it $\mathsf{NSP}_s(A)$. The formal definition of the NSP is as follows:

$$\mathsf{NSP}_s(A) \quad \equiv \quad \big[\, \forall S \subseteq [n] \ \big(|S| = s \ \rightarrow \ \forall y \in N_A^0 \ \|y[S]\|_1 < \|y[\bar{S}]\|_1 \big) \,\big]. \tag{12.5}$$

Basically, if A has the NSP property of order s, every nonzero solution of $Ax = b$ have the property that each restrictions on s components has smaller ℓ_1 norm than the corresponding complementary restriction on the other $n - s$ components.

12.6.1 Lemma
For each $m \times n$ matrix A and for every pair of integers t, s such that $t < s \leq n$ we have that $\mathsf{NSP}_s(A) \Rightarrow \mathsf{NSP}_t(A)$.

Proof. Let T, U be two disjoint non-empty proper subsets of $[n]$ such that $|T| = t$ and $|T \cup U| = s$. By Eq. (12.6.2) we have:

$$\begin{array}{rll} \forall y \in N_A^0 & \|y[T \cup U]\|_1 < \|y[T \,\bar{\cup}\, U]\|_1 = \|y[[n] \setminus (T \cup U)]\|_1 \\ \Rightarrow & \|y[T]\|_1 + \|y[U]\|_1 < \|y\|_1 - \|y[T]\|_1 - \|y[U]\|_1 \\ \Rightarrow & \|y[T]\|_1 < \|y[\bar{T}]\|_1 - 2\|y[U]\|_1. \end{array}$$

since $\|y[U]\|_1 > 0$, for each $T \subseteq [n]$ with $|T| = t$ we have that $\forall y \in N_A^0 \ \|y[T]\|_1 < \|y[\bar{T}]\|_1$, which implies $\mathsf{NSP}_t(A)$. □

We are now ready to prove the proposition in Item 1 above.

12.6.2 Proposition
For each $x^* \in \mathbb{R}^n$ such that (i) $|\mathsf{supp}(x^*)| \leq s$ and (ii) $b = Ax^*$, x^* is the unique minimum of $P^1(A, b)$ iff $\mathsf{NSP}_s(A)$.

Proof. We first note that, by Lemma 12.6.1, we can replace condition (i) with $|\mathsf{supp}(x^*)| = s$.

We prove the ($\Rightarrow$) direction, i.e.

$$\forall x \in \mathbb{R}^n \ (x \text{ is a unique minimum of } P^1(A, Ax) \wedge |\mathsf{supp}(x)| = s) \Rightarrow \mathsf{NSP}_s(A).$$

Let $y \in N_A^0$ and $S \subseteq [n]$ with $|S| = s$. Assume that $A\,y[S] \neq 0$ (this assumption is wlog since if there did not exist any $S \subseteq [n]$ with $y[S] \neq 0$, then $y = 0$, which contradicts $y \in N_A^0$). Note that $|S| = s$ implies $|\mathsf{supp}(y(S))| = s$, so $y[S]$ is the unique minimum of $P^1(A, Ay[S])$ by hypothesis. Now

$$\begin{array}{rcl} y & = & y[S] + y[\bar{S}] \in N_A^0 \\ \Rightarrow \ 0 = Ay = Ay[S] + Ay[\bar{S}] & & \\ \Rightarrow \ A(-y[\bar{S}]) = Ay[S] \neq 0. & & \end{array}$$

We also have that $y[S] \neq -y[\bar{S}]$, since otherwise, by $y = y[S] + y[\bar{S}]$, both would be scalings of y, and hence both in N_A^0, which cannot happen since we assumed that $Ay[S] \neq 0$. Therefore we have that $y[S]$ is the unique minimum of $P^1(A, Ay[S])$ and that $-y[\bar{S}]$ is feasible in $P^1(A, Ay[S])$, which implies that $\| - y[\bar{S}]\|_1 > \|y[S]\|_1$. Since $\| - y[\bar{S}]\|_1 = \|y[\bar{S}]\|_1$, $\mathsf{NSP}_s(A)$ holds.

Now we prove the ($\Leftarrow$) direction, i.e.

$$\mathsf{NSP}_s(A) \Rightarrow \forall x^* \in \mathbb{R}^N \ (x^* \text{ is a unique minimum of } P^1(A, Ax^*) \wedge |\mathsf{supp}(x^*)| = s).$$

Let $x^* \in \mathbb{R}^n$, $b = Ax^*$, $S = \mathsf{supp}(x^*)$ and $|S| = s$. Let $\bar{x}$ be a solution of $Ax = b$, then $\bar{x} = x^* - y$ for some $y \in N_A$. We have:

$$\begin{aligned}
\|x^*\|_1 = \|(x^* - \bar{x}[S]) + \bar{x}[S]\|_1 &\leq && \text{[by triangle inequality]} \\
\leq \|x^* - \bar{x}[S]\|_1 + \|\bar{x}[S]\|_1 &= && \text{[since } S = \mathsf{supp}(x^*)\text{]} \\
= \|x^*[S] - \bar{x}[S]\|_1 + \|\bar{x}[S]\|_1 &= && \text{[since } x^* - \bar{x} = y\text{]} \\
= \|y[S]\|_1 + \|\bar{x}[S]\|_1 &< && \text{[by } \mathsf{NSP}_s(A)\text{]} \\
< \|y[\bar{S}]\|_1 + \|\bar{x}[S]\|_1 &= && \text{[since } x^*[\bar{S}] = 0 \wedge y = x^* - \bar{x}\text{]} \\
= \| - \bar{x}[\bar{S}]\|_1 + \|\bar{x}[S]\|_1 &= && \text{[since } \| - z\|_1 = \|z\|_1 \wedge z[S] + z[\bar{S}] = z\text{]} \\
&= && \|\bar{x}\|_1
\end{aligned}$$

Since the ℓ_1 norm is strictly convex, we have that $x^* = \bar{x}$, which implies that x^* is the unique minimum of $P^1(A, Ax^*)$. □

12.6.2.1 A realistic variant of the NSP

The issue with the NSP is that it requires sparsity to be exact: in other words, the zero components must be exactly zero. In computations, it often happens that there may be values that are close to zero, but not exactly zero; and, worse, that one cannot decide whether rounding to zero is a valid operation. In this setting we propose a variant of the NSP, as follows.

Given $\hat{x} \in \mathbb{R}^n$ with $|\mathsf{supp}(\hat{x})| \geq s$ and $b = A\hat{x}$, we define a "sparsification" operation yielding the closest s-sparse vector $\bar{x}$ to $\hat{x}$: let $S = \underset{T \subseteq [n]:|T|=s}{\arg\max} \|\hat{x}\|_1$ and $\bar{x} = \hat{x}[S]$. The maximum componentwise error of $\bar{x}$ w.r.t. $\hat{x}$ is $\epsilon = \max_{j \in \bar{S}} |\hat{x}_j|$. We say that $\hat{x}$ almost has support size s up to an ϵ error. We want to find the solution x^* of $P^1(A, A\hat{x})$ closest to $\hat{x}$ with $|\mathsf{supp}(x^*)| = s$. To this end, we adapt the NSP by endowing it with a further parameter ρ:

$$\mathsf{NSP}_s^\rho(A) \quad \equiv \quad \big[\exists \rho \in [0,1]\ \forall S \subseteq [n]\ (|S| = s\ \rightarrow\ \forall y \in N_A^0\ \|y[S]\|_1 \leq \rho \|y[\bar{S}]\|_1)\big]. \tag{12.6}$$

The difference of $\mathsf{NSP}_s^\rho(A)$ with respect to $\mathsf{NSP}_s(A)$ is that, instead of requiring that $\|y[S]\|_1 < \|y[\bar{S}]\|_1$, we weaken the requirement to $\|y[S]\|_1 \leq \rho\|y[\bar{S}]\|_1$.

A proposition similar to (but weaker than) Prop. 12.6.2 also holds for $\mathsf{NSP}_s^\rho(A)$.

12.6.3 Proposition
Let $\hat{x}$ almost have support size s, let $b = A\hat{x}$, and assume $\mathsf{NSP}_s^\rho(A)$ holds. Then, if x^* is a minimum of $P^1(A, b)$ we have

$$\|x^* - \hat{x}\|_1 \leq 2\frac{1+\rho}{1-\rho}\|\bar{x} - \hat{x}\|_1 \leq (n-s)\epsilon. \tag{12.7}$$

We remark that Eq. (12.7) is the precise meaning of "good approximation" in Sect. 12.6.1.

Proof. First, we remark that for any two scalars μ, ν the triangle inequality immplies that $|\mu + \nu| \leq |\mu| + |\nu|$, and hence that $|\mu + \nu| - |\nu| \leq |\mu|$. We now carry out the change of variables $\alpha = \mu + \nu$ and $\beta = -\nu$, which implies $|\alpha| - |-\beta| \leq |\alpha + \beta|$, whence $|\alpha + \beta| \geq |\alpha| - |\beta|$. Below, we apply the triangle inequality in this form.

Note that x^* is feasible in $Ax = b = A\hat{x}$. So there is a unique $y \in N_A$ such that $x^* = \hat{x} + y$. This implies $\|x^*\|_1 = \|\hat{x} + y\|_1 \leq \|\hat{x}\|_1$ by optimality of x^* in $P^1(A, A\hat{x})$. We look at the term $\|\hat{x} + y\|_1$. We

have:

$$\begin{aligned}
\|\hat{x}+y\|_1 &= \sum_{j\in S}|\hat{x}_j+y_j| + \sum_{j\in\bar{S}}|\hat{x}_j+y_j| \geq \sum_{j\in S}(|\hat{x}_j|-|y_j|) + \sum_{j\in\bar{S}}(|y_j|-|\hat{x}_j|) \quad \text{by triangle ineq.} \\
&= \|\hat{x}[S]\|_1 - \|y[S]\|_1 + \|y[\bar{S}]\|_1 - \|\hat{x}[\bar{S}]\|_1 \\
&= \|\hat{x}\|_1 + \|y[\bar{S}]\|_1 - 2\|\hat{x}[\bar{S}]\|_1 - \|y[S]\|_1 \\
&= \|x\|_1 - 2\|\hat{x}-\bar{x}\|_1 + \|y[\bar{S}]\|_1 - \|y[S]\|_1, \quad (*)
\end{aligned}$$

since by definition of $\bar{x}$ we have $\bar{x}[\bar{S}] = 0$ and $\|\hat{x}-\bar{x}\|_1 \geq \|\hat{x}[\bar{S}]\|_1$, implying $(*) \leq \|\hat{x}+y\|_1 \leq \|\hat{x}\|_1$. Therefore,

$$\begin{aligned}
\|\hat{x}\|_1 &\geq \|\hat{x}\|_1 - 2\|\hat{x}-\bar{x}\|_1 + \|y[\bar{S}]\|_1 - \|y[S]\|_1 \\
\Rightarrow 2\|\hat{x}-\bar{x}\|_1 &\geq \|y[\bar{S}]\|_1 - \|y[S]\|_1
\end{aligned}$$

By NSP_s^ρ we have $-\|y[S]\|_1 \geq -\rho\|y[\bar{S}]\|_1$, so by the above we obtain $2\|\hat{x}-\bar{x}\|_1 \geq (1-\rho)\|y[\bar{S}]\|_1$, whence $\|y[\bar{S}]\|_1 \leq \frac{2}{1-\rho}\|\hat{x}-\bar{x}\|_1 \quad (\dagger)$. Now $x^* = \hat{x}+y$ implies

$$\|x^*-\hat{x}\|_1 = \|y\|_1 = \|y[S]\|_1 + \|y[\bar{S}]\|_1.$$

By NSP_s^ρ we have $\|y[S]\|_1 \leq \rho\|y[\bar{S}]\|_1$, hence $\|x^*-\hat{x}\|_1 \leq (1+\rho)\|y[\bar{S}]\|_1$. Therefore, by $(\dagger)$, we have

$$\|x^*-\hat{x}\|_1 \leq 2\frac{1+\rho}{1-\rho}\|\hat{x}-\bar{x}\|_1.$$

Lastly, we have

$$\|\hat{x}-\bar{x}\|_1 = \|\hat{x}-\hat{x}[S]\|_1 = \|\hat{x}[\bar{S}]\| \leq |\bar{S}|\epsilon = (n-s)\epsilon$$

as claimed. □

12.6.4 Corollary
With the assumptions of Prop. 12.6.3, if $|\mathsf{supp}(\hat{x})| = s$, *then* $x^* = \hat{x} = \bar{x}$.

Proof. This follows because ϵ can be taken to be zero, which implies $\|x^*-\hat{x}\|_1 \leq 2\frac{1+\rho}{1-\rho}\|\bar{x}-\hat{x}\|_1 \leq 0$, which in turn implies the result. □

12.6.3 Restricted isometry property

We now examine Item 2 in Sect. 12.6.1. For an $m \times n$ matrix A, we define the *restricted isometry property* (RIP) as follows:

$$\mathsf{RIP}_s^\delta(A) \quad \equiv \quad \big[\forall x \in \mathbb{R}^n \ (|\mathsf{supp}(x)| = s \ \rightarrow \ (1-\delta)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1-\delta)\|x\|_2^2\big]. \tag{12.8}$$

If the matrix A has the RIP, then the vector Ax has more or less the same ℓ_2 norm as x, up to a factor of δ.

The exact form of the proposition in Item 2 is as follows.

12.6.5 Proposition
Let A *be an* $m \times n$ *matrix with* $m < n$, $s < \frac{n}{2}$ *be an integer, and* $\delta \geq 0$. *If* $\mathsf{RIP}_{2s}^\delta(A)$ *holds, and* $\rho = \frac{\sqrt{2}\delta}{1-\delta} < 1$, *then* $\mathsf{NSP}_s^\rho(A)$ *holds.*

For the proof, see [86, Thm. 5.12]. We note that it suffices that $\delta < \frac{1}{1+\sqrt{2}} \approx 0.4142$ in order for $\rho < 1$ to hold in Prop. 12.6.5.

12.6.3.1 Applicability to Eq. (12.1)

Recall that Eq. (12.1), i.e. $P^0(A,b) \equiv \min\{\|x\|_0 \mid Ax = b\}$, is an **NP**-hard problem that requires to find the solution x to $Ax = b$ with smallest support size.

We report a result which illustrates the effect of the RIP on the original MILP P^0 rather than the LP relaxation P^1 [270, Thm. 23.6].

12.6.6 Theorem
Let $\hat{x} \in \mathbb{R}^n$ with $|\mathsf{supp}(\hat{x})| = s$, let $\delta \in (0,1)$, let A such that $\mathsf{RIP}_{2s}^{\delta}(A)$ holds, and let x^* be an optimum of $P^0(A, A\hat{x})$. Then $x^* = \hat{x}$.

Proof. Suppose the theorem does not hold. Then $y = x^* - \hat{x}$ is a nonzero vector. By definition of x^* we have $\|x^*\|_0 \leq \|\hat{x}\|_0 \leq s$, hence $\|y\|_0 \leq 2s$. Since A has RIP, we obtain $\|Ay\|_2^2 \in (1 \pm \delta)\|y\|_2^2$. However, $Ay = Ax^* - A\hat{x} = 0$ while $y \neq 0$, and $\delta \in (0,1)$ implies that $1 \pm \delta > 0$, hence $0 \in (\alpha, \beta)$ where $\alpha, \beta > 0$, which is a contradiction. □

This result is of limited applicability, since we do not know whether $P^0(A,b)$ can be solved efficiently if A has the RIP.

12.6.3.2 RIP and eigenvalues

In this section we give a sufficient eigenvalue condition for the RIP to hold.

Let A be an $m \times n$ matrix, let $S \subset [n]$ with $|S| = s < n$, let A_S be the $m \times s$ matrix consisting of the columns of A indexed by S, and consider the $s \times s$ PSD matrix $B(S) = A_S^\top A_S$. Let $\lambda^L = \min_{|S|=s} \lambda_{\mathsf{min}}(B(S))$ and $\lambda^U = \min_{|S|=s} \lambda_{\mathsf{max}}(B(S))$.

12.6.7 Theorem
If, for all $S \subset [n]$ with $|S| = s$, there is a $\delta \in (0,1)$ such that

$$1 - \delta \leq \lambda^L \leq \lambda^U \leq 1 + \delta, \tag{12.9}$$

then $\mathsf{RIP}_s^{\delta}(A)$ holds.

Proof. In this proof we consider $x[S]$ as a vector in $\mathbb{R}^s$, i.e. we remove the zero components in $x[\bar{S}]$.

Let $S \subset [n]$ with $|S| = s$, and let $x \in \mathbb{R}^n$ with $\mathsf{supp}(x) = S$. We define diagonal matrices $\Lambda^L = \mathsf{diag}(\mathbf{1}\lambda^L)$ and $\Lambda^U = \mathsf{diag}(\mathbf{1}\lambda^U)$. By definition of λ^L, λ^U we have

$$\begin{aligned} & \langle \Lambda^L x[S], x[S] \rangle \leq \langle B(S)x[S], x[S] \rangle \leq \langle \Lambda^U x[S], x[S] \rangle \\ \Rightarrow \quad & \lambda^L \|x[S]\|_2^2 \leq \langle B(S)x[S], x[S] \rangle \leq \lambda^U \|x[S]\|_2^2. \end{aligned}$$

We note that, since $\mathsf{supp}(x) = S$, $\|x[S]\|_2^2 = \|x\|_2^2$. By Eq. (12.9) we have:

$$(1-\delta)\|x\|_2^2 \leq \lambda^L \|x\|_2^2 \leq \langle B(S)x[S], x[S] \rangle \leq \lambda^U \|x\|_2^2 \leq (1+\delta)\|x\|_2^2.$$

Now we note that

$$\langle B(S)x[S], x[S] \rangle = \langle A_S^\top A_S x[S], x[S] \rangle = \langle A_S x[S], A_S x[S] \rangle = \|A_S x[S]\|_2^2.$$

Again since $\mathsf{supp}(x) = S$ we have $\|A_S x[S]\|_2^2 = \|Ax\|_2^2$. Hence

$$(1-\delta)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1+\delta)\|x\|_2^2$$

holds for any $S \subset [n]$ with $|S| = s$ and x having support S. Therefore $\mathsf{RIP}_s^\delta(A)$ holds, as claimed. □

Hence, in order to construct a matrix A with the RIP, we can focus on matrices A such that $B(S) = A_S^\top A_S$ has eigenvalues close to one for any $S \subset [n]$.

12.6.4 Normally sampled matrices

We come to the last Item 3 of Sect. 12.6.1. We shall sketch the proof that a normally sampled matrix A has the RIP with high probability.

We exploit the sufficient condition in Thm. 12.6.7: we show that normally sampled matrices A have the conditions

$$\forall i < j \leq n \quad A_i^\top A_j \approx 0 \tag{12.10}$$
$$\forall i \leq n \quad A_i^\top A_i = \|A_i\|_2^2 \approx 1, \tag{12.11}$$

which means that $A^\top A$ is approximately an identity matrix, which satisfies Eq. (12.9) (see Sect. 13.1.3). More precisely, we argue that a sufficient condition to the orthonormality of the columns of A is that each component of A is sampled independently from $\mathsf{N}(0, \frac{1}{\sqrt{m}})$.

We now offer a proof sketch of Eq. (12.10)-(12.11). A random vector $A_i \in \mathbb{R}^m$ is *isotropic* iff $\mathsf{cov}(A_i) = I_m$. We recall that $\mathsf{cov}(X) = \mathsf{E}(XX^\top)$ for any matrix X, and remark that if $A_i \sim \mathsf{N}(0,1)^m$, then A_i isotropic.

12.6.8 Lemma
An isotropic random vector A_i has the property that

$$\forall x \in \mathbb{R}^m \quad \mathsf{E}(\langle A_i, x\rangle^2) = \|x\|_2^2.$$

Proof. For two square symmetric matrices B, C we have $B = C$ iff

$$\forall x \quad (x^\top B x = x^\top C x);$$

hence $x^\top \mathsf{E}(A_i A_i^\top) x = x^\top I_m x$. Note that the LHS is $\mathsf{E}(\langle A_i, x\rangle^2)$ and the RHS is $\|x\|_2^2$. □

12.6.9 Lemma
An isotropic random vector x in $\mathbb{R}^m$ is such that $\mathsf{E}(\|x\|_2^2) = m$.

Proof. We simply note that

$$\mathsf{E}(\|x\|_2^2) = \mathsf{E}(x^\top x) = \mathsf{E}(\mathsf{tr}(x^\top x)) = \mathsf{E}(\mathsf{tr}(xx^\top)) = \mathsf{tr}(\mathsf{E}(xx^\top)) = \mathsf{tr}(I_m) = m,$$

which proves the lemma. □

12.6.10 Lemma
Two independent isotropic random vectors A_i, A_j in $\mathbb{R}^m$ are such that $\mathsf{E}(\langle A_i, A_j\rangle^2) = m$.

Proof. By definition of conditional expectation we have:

$$\mathsf{E}(\langle A_i, A_j\rangle^2) = \mathsf{E}_{A_j}(\mathsf{E}_{A_i}(\langle A_i, A_j\rangle^2 \mid A_j)).$$

By Lemma 12.6.8, the inner expectation is $\|A_j\|_2^2$. By Lemma 12.6.9, the outer expectation is m, as claimed. □

Moreover, by [298, Thm. 3.1.1], if A_i is sampled from $\mathsf{N}(0,1)^m$, then $\|A_i\|_2 \approx \sqrt{m}$ with high probability. We can now prove that independent random vectors are almost orthogonal. By the above results, $\|A_i\|_2$, $\|A_j\|_2$, $\langle A_i, A_j\rangle$ are all approximately equal to $\sqrt{m}$. We now scale A_i to $\bar{A}_i = A_i/\|A_i\|_2$ and A_j to $\bar{A}_j = A_j/\|A_j\|_2$, obtaining

$$\langle \bar{A}_i, \bar{A}_j\rangle = \frac{1}{m}\langle A_i, A_j\rangle = \frac{\sqrt{m}}{m} = \frac{1}{\sqrt{m}}.$$

Thus, for m large, $\langle \bar{A}_i, \bar{A}_j\rangle$ tends to zero.

The reasoning so far explains why sampling A from $\mathsf{N}(0, \frac{1}{\sqrt{m}})$ suffices to solve $P^1(A,b)$ to find a solution for $P^0(A,b)$. While we made some parts of this argument precise, some pieces have been left sketchy to avoid too many technical details. The precise statement of the theorem is as follows.

12.6.11 Theorem (Thm. 5.17 in [86])
Let A be sampled from $\mathsf{N}(0,1)^{m\times n}$ and $\delta \in (0,1)$. Then there exist constants c_1, c_2 depending only on δ such that:

$$\forall s < m \quad \frac{s \ln n/s}{c_1} \leq m \;\rightarrow\; \mathsf{Prob}(\mathsf{RIP}_s^\delta(A)) \geq 1 - e^{-c_2 m}. \tag{12.12}$$

We note several discrepancies of this precise result with respect to our imprecise argument.

- The sparsity integer parameter s should be smaller than m (rather than n, which was usually the case in our treatment). This is not an issue, since $m < n$.
- The distribution whence we sample A is $\mathsf{N}(0,1)$ rather than $\mathsf{N}(0, \frac{1}{\sqrt{m}})$. This extra $\sqrt{m}$ factor in A comes from the well-known norm inequality $\|\cdot\|_2 \leq \|\cdot\|_1 \leq \sqrt{m}\|\cdot\|_2$, which is necessary in order to reason about the ℓ_1 norm while using the ℓ_2 norm.
- The informal terms "approximate", "with high probability", and the symbol "$\approx$", are interpreted formally in Eq. (12.12), namely: the probability that our sampled matrix has the RIP exceeds the value of a function which tends asymptotically to 1 exponentially fast. Such a statement is more qualitative than quantitative because of the two (unknown) constants c_1, c_2. This is a common feature of theoretical results holding with high probability.

We make more remarks to help with the application of these results in practice.

- We find that $\mathsf{Prob}(\mathsf{RIP}_s^\delta(A)) = 0$ for m too small for a fixed s.
- As m increases, $\mathsf{Prob}(\mathsf{RIP}_s^\delta(A))$ becomes nonzero.
- As m increases even more, $\mathsf{Prob}(\mathsf{RIP}_s^\delta(A))$ tends to 1 very fast.
- The three previous points are consistent with the phase transition phenomenon in Sect. 12.5.
- In practice, Thm. 12.6.11 allows the achievement of logarithmic compression of data for large n and fixed s: a sparse data vector x is encoded into a vector Ax where A is $m \times n$ and m is logarithmic in n. A more precise statement to this effect can be find in [227, Lem 5.5.2]:

$$A \sim \mathsf{N}(0,1)^{mn} \wedge m \geq 10 s \ln \frac{n}{s} \Rightarrow \mathsf{RIP}_s^{1/3}(A) \text{ with high probability.}$$

- This compression technique actually works better than the worst-case bounds ensured by the corresponding theory.

Chapter 13

Random projections in MP

Random projections (RPs) are another dimensionality reduction technique exploiting high-dimensional geometry properties and, in particular, the concentration of measure phenomenon (Sect. 10.6.5.2). They are more general than Barvinok's naive algorithm (Sect. 10.6.5) in that they apply to sets of vectors in some high-dimensional Euclidean space $\mathbb{R}^n$ (with $n \gg 1$). These sets are usually finite and growing polynomially with instance sizes [296], but they may also be infinite [309], in which case the technical name used is *subspace embeddings*. In this chapter we introduce random projections and their application to MP.

13.1 The Johnson-Lindenstrauss Lemma

The foremost result in RPs is the Johnson-Lindenstrauss Lemma (JLL) [152]. For a set of vectors $\mathcal{X} \subset \mathbb{R}^n$ with $|\mathcal{X}| = m$, and an $\varepsilon \in (0,1)$ there is a $k = O(\frac{1}{\varepsilon^2} \ln m)$ and a mapping $f : \mathcal{X} \to \mathbb{R}^k$ such that:

$$\forall x, y \in \mathcal{X} \quad (1-\varepsilon)\|x-y\|_2 \leq \|f(x)-f(y)\|_2 \leq (1+\varepsilon)\|x-y\|_2. \tag{13.1}$$

The proof of this result [152, Lemma 1] is probabilistic: it shows that an f satisfying Eq. (13.1) exists with some nonzero probability.

Later and more modern proofs (e.g. [90]) clearly point out that f can be a linear operator represented by a $k \times n$ matrix T, each component of which can be sampled from a *subgaussian distribution*. This term refers to a r.v. $\mathfrak{V}$ for which there are constants C, c s.t. for each $t > 0$ we have

$$\mathsf{P}(|\mathfrak{V}| > t) \leq C\, e^{-ct^2}.$$

In particular, the Gaussian distribution is also subgaussian. Then the probability that a randomly sampled T satisfies Eq. (13.1) can be shown to exceed $1/m$. The union bound then provides an estimate on the number of samplings of T necessary to guarantee Eq. (13.1) with a desired probability.

Some remarks are in order.

1. In practice, Eq. (13.1) is applied to some given data as follows: given a set $\mathcal{X}$ of m vectors in $\mathbb{R}^n$ and some error tolerance $\varepsilon \in (0,1)$, find an appropriate $k = O(\frac{1}{\varepsilon^2} \ln m)$, construct the $k \times n$ RP T by sampling each of its components from $\mathsf{N}(0, \frac{1}{\sqrt{k}})$, then define the set $T\mathcal{X} = \{Tx \mid x \in X\}$. By the JLL, $T\mathcal{X}$ is *approximately congruent* to $\mathcal{X}$ in the sense of Eq. (13.1); however, $T\mathcal{X} \subset \mathbb{R}^k$ whereas $\mathcal{X} \subset \mathbb{R}^n$, and, typically, $k \ll n$.
2. The computation of an appropriate k would appear to require an estimation of the constant in the expression $O(\frac{1}{\varepsilon^2} \ln m)$. Values computed theoretically are often so large as to make the technique

useless in practice. As far as we know, this constant has only been computed empirically in some cases [297], ending up with an estimation of the constant at 1.8.

3. The term $\frac{1}{\sqrt{k}}$ is the standard deviation of the normal distribution from which the components of T must be sampled. It corresponds to a scaling of the vectors in $T\mathcal{X}$ induced by the loss in dimensions (see Prop. 13.1.4).

4. In the expression $O(\frac{1}{\varepsilon^2}\ln m)$, the logarithmic term is the one that counts for analysis purposes, but in practice ε^{-2} can be large. Our advice is to take $\varepsilon \in (0.1, 0.2)$ and then fine-tune ε according to results.

5. Surprisingly, the target dimension k is independent of the original dimension n.

6. Even if the data in $\mathcal{X}$ is sparse, $T\mathcal{X}$ ends up being dense. Different classes of sparse RPs have been investigated [2, 155] in order to tackle this issue. A simple algorithm [85, §5.1] consists in initializing T as the $k \times n$ zero matrix, and then only fill components using samples from $\mathsf{N}(0, \frac{1}{\sqrt{kp}})$ with some given probability p. The value of p corresponds to the density of T. In general, and empirically, it appears that the larger n and m are, the sparser T can be.

7. Obviously, a Euclidean space of dimension k can embed at most k orthogonal vectors. An easy, but surprising corollary of the JLL is that as many as $O(2^k)$ approximately orthogonal vectors can fit in $\mathbb{R}^k$. This follows by [302, Prop. 1] applied to the standard basis $S = \{e_1, \ldots, e_n\}$ of $\mathbb{R}^n$: we obtain $\forall i < j \leq n$ $(-\varepsilon \leq \langle Te_i, Te_j\rangle - e_i e_j \leq \varepsilon)$, which implies $|\langle Te_i, Te_j\rangle| \leq \varepsilon$ with $TS \subset \mathbb{R}^k$ and $k = O(\ln n)$. Therefore TS is a set of $O(2^k)$ almost orthogonal vectors in $\mathbb{R}^k$, as claimed.

8. Typical applications of RPs arise in clustering databases of large files (e.g. e-mails, images, songs, videos), performing basic tasks in ML (e.g. k-means [54], k-nearest neighbors (k-NN) [149], robust learning [23] and more [147]), and approximating large MP formulations (e.g. LP, QP, see Sect. 13.2).

9. The JLL seems to suggest that most of the information encoded by the congruence of a set of vectors can be maintained up to an ε tolerance in much smaller dimensional spaces. This is not true for sets of vectors in low dimensions. For example, with $n \in \{2, 3\}$ a few attempts immediately show that RPs yield sets of projected vectors which are necessarily incongruent with the original vectors.

13.1.1 Union and intersection bounds

In this section prove one of the most fundamental results in probability theory. Scott Aaronson writes "Despite its triviality, the union bound is probably the most useful fact in all of theoretical computer science. I use it maybe 200 times in every paper I write." [1, Ch. 7]

13.1.1 Lemma (Union bound)
Let $t \in [0, 1]$ and consider events $E_1, \ldots, E_k$ such that $\mathsf{Prob}(E_i) \geq t$ for each $i \leq k$. Then

$$\mathsf{Prob}(\exists i \leq k\ E_i) \leq kt$$

holds.

Proof. The claim follows by a straightforward induction applied to: (i) $\mathsf{Prob}(\exists i \leq k\ E_i) = \mathsf{Prob}\left(\bigcup_{i \leq k} E_i\right)$; (ii) $\mathsf{Prob}(E_1 \cup E_2) = \mathsf{Prob}(E_1) + \mathsf{Prob}(E_2) - \mathsf{Prob}(E_1 \cap E_2)$. □

The union bound is a trivial but very useful result, so it is worth giving it an official "lemma" status.

It is also worth stating a related result (which is in fact a corollary of the union bound) about intersections. It is sometimes also called "union bound", although, to distinguish it from Lemma 13.1.1, we shall call it the "intersection bound".

13.1.2 Lemma (Intersection bound)
Let $t \in [0,1]$ and consider events $E_1, \dots, E_k$ such that $\mathsf{Prob}(E_i) \geq 1 - t$ for each $i \leq k$. Then

$$\mathsf{Prob}(\forall i \leq k\ E_i) \geq 1 - kt$$

holds.

Proof. We observe that $\mathsf{Prob}(\forall i \leq k\ E_i) = 1 - \mathsf{Prob}(\exists i \leq k\ \neg E_i)$. Therefore,

$$\mathsf{Prob}\Big(\bigwedge_{i\leq k} E_i\Big) = 1 - \mathsf{Prob}\Big(\bigvee_{i\leq k} (\neg E_i)\Big) \geq 1 - \sum_{i\leq k} \mathsf{Prob}(\neg E_i) = 1 - \sum_{i\leq k}(1-(1-t)) = 1 - kt,$$

as claimed. □

13.1.2 Proving the JLL

We prove the "squared version" of the JLL.

13.1.3 Theorem
For any $\varepsilon \in (0,1)$, any $n \in \mathbb{N}$, any $k \in \mathbb{N}$ such that

$$k \geq \frac{4 \ln n}{\varepsilon^2/2 - \varepsilon^3/3}, \tag{13.2}$$

and any set $\mathcal{X} \subset \mathbb{R}^m$ with $|\mathcal{X}| = n$, there is a map $f : \mathbb{R}^m \to \mathbb{R}^k$ such that:

$$\forall x, y \in \mathcal{X} \quad (1-\varepsilon)\|x-y\|_2^2 \leq \|f(x) - f(y)\|_2^2 \leq (1+\varepsilon)\|x-y\|_2^2. \tag{13.3}$$

We follow the treatment in [90]. Other proofs can be found in [152, 148, 11, 155, 211, 16, 296, 212, 156, 298].

First, we focus on a special subclass of maps f, namely we let T be a $k \times n$ RP sampled from $\mathsf{N}(0, \frac{1}{\sqrt{k}})$. Then we prove that this map preserves Euclidean norms on average.

13.1.4 Proposition
Let T be a $k \times n$ RP sampled from $\mathsf{N}(0, \frac{1}{\sqrt{k}})$, and $u \in \mathbb{R}^n$; then $\mathsf{E}(\|Tu\|_2^2) = \|u\|_2^2$.

Proof. We prove the claim for $\|u\|_2 = 1$; the result will follow by scaling. For each $i \leq k$ we define $v_i = \sum_{j\leq n} T_{ij} u_j$. Then $\mathsf{E}(v_i) = \mathsf{E}\big(\sum_{j\leq m} T_{ij} u_j\big) = \sum_{j\leq m} \mathsf{E}(T_{ij}) u_j = 0$. Moreover,

$$\mathsf{Var}(v_i) = \sum_{j\leq m} \mathsf{Var}(T_{ij} u_j) = \sum_{j\leq m} \mathsf{Var}(T_{ij}) u_j^2 = \sum_{j\leq m} \frac{u_j^2}{k} = \frac{1}{k}\|u\|^2 = \frac{1}{k}.$$

Now, $\frac{1}{k} = \mathsf{Var}(v_i) = \mathsf{E}(v_i^2 - (\mathsf{E}(v_i))^2) = \mathsf{E}(v_i^2 - 0) = \mathsf{E}(v_i^2)$. Hence

$$\mathsf{E}(\|Tu\|^2) = \mathsf{E}(\|v\|^2) = \mathsf{E}\big(\sum_{i\leq k} v_i^2\big) = \sum_{i\leq k} \mathsf{E}(v_i^2) = \sum_{i\leq k} \frac{1}{k} = 1,$$

as claimed. □

While Prop. 13.1.4 is not strictly essential to prove Thm. 13.1.3, we believe it clarifies the most basic reason why RPs work as advertised.

Having shown that lengths are preserved by T on average, we now prove that the errors decrease rapidly with their size. Let $X_1, \ldots, X_m$ be m independent normal r.v. distributed like $\mathsf{N}(0,1)$. Let X be the random vector $(X_1, \ldots, X_m)$, and $Y = X/\|X\|_2$. It is well known that Y is a point chosen uniformly at random from the surface of the m-dimensional unit sphere $\mathbb{S}^{m-1}$ [234]. The proof of this rests on the fact that m independent normal r.v. define a multivariate normal distribution, having constant distribution function value over concentric spheres [78, §24.2].

Let $Z = (X_1, \ldots, X_k)$ and $L = \|Z\|_2^2$. Since $\|Y\|_2 = 1$ by definition, $\mu = \mathsf{E}(L) = \mathsf{E}(\|Z\|_2^2) = \frac{k}{n}$.

13.1.5 Lemma
Let $k < n$ and $\beta > 0$. If $\beta < 1$, then

$$\mathsf{Prob}(L \leq \beta k/n) \leq \beta^{k/2}\big(1 + \frac{(1-\beta)k}{n-k}\big)^{\frac{n-k}{2}} \leq \exp(k(1-\beta+\ln\beta)/2); \tag{13.4}$$

if $\beta > 1$, then

$$\mathsf{Prob}(L \geq \beta k/n) \leq \beta^{k/2}\big(1 + \frac{(1-\beta)k}{n-k}\big)^{\frac{n-k}{2}} \leq \exp(k(1-\beta+\ln\beta)/2). \tag{13.5}$$

Proof. It can be shown[1] that, if V is a r.v. distributed as $\mathsf{N}(0,1)$, then $\mathsf{E}(\exp(sV^2)) = \frac{1}{\sqrt{1-2s}}$, for $s \leq \frac{1}{2}$. Now:

$$\begin{aligned}
\mathsf{Prob}(L \leq \beta k/n) &= \mathsf{Prob}(n(X_1^2 + \cdots + X_k^2) \leq k\beta(X_1^2 + \cdots + X_n^2)) \\
&= \mathsf{Prob}(k\beta(X_1^2 + \cdots + X_n^2) - n(X_1^2 + \cdots + X_k^2) \geq 0) \\
&= \mathsf{Prob}(e^{t(k\beta(X_1^2+\cdots+X_n^2)-n(X_1^2+\cdots+X_k^2))} \geq 1) \quad \text{for } t > 0 \\
&\leq \mathsf{E}(e^{t(k\beta(X_1^2+\cdots+X_n^2)-n(X_1^2+\cdots+X_k^2))}) \quad \text{by Markov's inequality} \\
&= \mathsf{E}(e^{tk\beta V^2})^{n-k}\mathsf{E}(e^{t(k\beta-n)V^2})^k \quad \text{where } V \sim \mathsf{N}(0,1) \\
&= (1-2tk\beta)^{\frac{k-d}{2}}(1-2t(k\beta-d))^{-\frac{k}{2}} = g(t).
\end{aligned}$$

We note that the expression $g(t)$ requires $t \in \bar{T} = \big(0, \frac{1}{2k\beta}\big)$. We minimize[2] $g(t)$ over $\bar{T}$, and find $t' = \frac{1-\beta}{2\beta(d-k\beta)} \in \bar{T}$, yielding

$$\min_{t\in\bar{T}} g(t) = \left(\frac{d-k}{d-k\beta}\right)^{\frac{k-d}{2}}\left(\frac{1}{\beta}\right)^{-\frac{k}{2}} = \beta^{k/2}\left(1 + \frac{k(1-\beta)}{d-k}\right)^{\frac{d-k}{2}}.$$

Finally, we note that $\beta^{k/2} = e^{\frac{k\ln\beta}{2}}$, and that, since $e^x = \lim_{p\to\infty}(1+x/p)^p$ and $(1+x/p)^p$ is monotonically increasing w.r.t. p, we have

$$\left(1 + \frac{k(1-\beta)}{d-k}\right)^{\frac{d-k}{2}} \leq e^{(1-\beta)k/2}.$$

This establishes that $\mathsf{Prob}(L \leq \beta k/d) \leq e^{k(1-\beta+\ln\beta)/2}$, as claimed. The case for $\beta > 1$ is similar[3]. □

Proof of Thm. 13.1.3. We note that the theorem states that "for each n, there exists a k greater than a certain linear function of $\ln n$". Obviously, if $k \geq d$ then the theorem is trivial, since it would consists of a dimensionality increase (it would suffice it to consider a function f that appends sufficiently many zeros to its vector argument in order to satisfy Eq. (13.3)). Hence, we assume $k \leq d$. For $x, y \in \mathcal{X}$ we let

[1] Left as an exercise: you can for example read the section "Functions of random variables" in [305], then note that the function $\phi(v) = \exp(sv^2)$ can be inverted, then use a symbolic integration package to compute the expectation.
[2] Again left as an exercise: again, you can use a symbolic algebra package to solve for the derivative equal to zero.
[3] Left as an exercise, too. Watch out for the different interval $\bar{T}$.

$L = \|Tx - Ty\|_2^2$. Following the proof of Prop. 13.1.4 we let $\mu = \frac{k}{n}\|x-y\|_2^2$. By Lemma 13.1.5 (Eq. (13.4)) we obtain:

$$\begin{aligned}\mathsf{Prob}(L \le (1-\varepsilon)\mu) &\le \exp(k(1-(1-\varepsilon)+\ln(1-\varepsilon))/2) \quad \text{by } \ln(1-x) \le -x - x^2/2 \text{ for all } x \in [0,1)\\ &\le \exp(k(\varepsilon - (\varepsilon + \varepsilon^2/2))/2) = \exp(-k\varepsilon^2/4) \quad \text{by Eq. (13.2)}\\ &\le \exp(-2\ln n) = 1/n^2.\end{aligned}$$

By Lemma 13.1.5 (Eq. (13.5)) we obtain:

$$\begin{aligned}\mathsf{Prob}(L \ge (1+\varepsilon)\mu) &\le \exp(k(1-(1+\varepsilon)+\ln(1+\varepsilon))/2) \quad \text{by } \ln(1-x) \le x - x^2/2 + x^3/3 \text{ for all } x \ge 0\\ &\le \exp(k(-\varepsilon + (\varepsilon - \varepsilon^2/2 + \varepsilon^3/3))/2) = \exp(-k(\varepsilon^2/2 - \varepsilon^3/3)/2) \quad \text{by Eq. (13.2)}\\ &\le \exp(-2\ln n) = 1/n^2.\end{aligned}$$

Thus, the probability that

$$\frac{\|Tx - Ty\|_2^2}{\|x-y\|_2^2} \notin [1-\varepsilon, 1+\varepsilon] \tag{13.6}$$

is at most $\frac{2}{n^2}$. Hence, by the union bound Lemma 13.1.1, the probability that there exists at least one pair of vectors $x, y \in \mathcal{X}$ satisfying Eq. (13.6) is bounded above by $\binom{n}{2}\frac{2}{n^2} = 1 - \frac{1}{n}$. Therefore, T satisfies Eq. (13.3) with probability at least $\frac{1}{n}$. The result follows by the probabilistic method [18]. □

13.1.6 Corollary
There is a randomized polynomial time algorithm for finding a T satisfying the hypotheses of Thm. 13.1.3 with probability 0.99.

Proof. The algorithm consists in independently sampling a RP T at most t times, and testing whether it satisfies Eq. (13.3). The probability of failure of a single sample is, by the proof of Thm. (13.1.3), $1 - \frac{1}{n}$. The probability of failure of every sample is $(1-1/n)^t$. We want to find t such that $(1-1/n)^t \le 0.01$: this implies $t \ge \frac{\ln 0.01}{\ln(1-1/n)}$. □

In practice, it suffices to take $t \approx 4.7n$.

We note that Cor. 13.1.6 clarifies another fact, namely that the probability of failure can be made arbitrarily small. We make this notion more precise as follows.

13.1.7 Corollary
Let $c, \mathcal{C}$ be sufficiently large constants, $\varepsilon \in (0,1)$, $\mathcal{X} \subset \mathbb{R}^m$ with $|\mathcal{X}| = n$, and T be a $k \times m$ RP, where $k \ge c\varepsilon^{-2}\ln n$. Then

$$\mathsf{Prob}\left(\forall x, y \in \mathcal{X}(1-\varepsilon)\|x-y\|_2^2 \le \|Tx - Ty\|_2^2 \le (1+\varepsilon)\|x-y\|_2^2\right) \ge 1 - 2e^{-\mathcal{C}(\varepsilon^2 - \varepsilon^3)k} \tag{13.7}$$

holds.

The "universal constant" $\mathcal{C}$ originates from estimates of the error between certain functions (notably the exponential one) and their approximations using Taylor series expansions (see e.g. [212, Claim 2.3.2]).

For later reference, we state a different variant of Lemma 13.1.5.

13.1.8 Lemma
For any $x \in \mathbb{R}^n$ and $\varepsilon \in (0,1)$,

$$\mathsf{Prob}((1-\varepsilon)\|x\|_2^2 \le \|Tx\|_2^2 \le (1+\varepsilon)\|x\|_2^2) \ge 1 - 2e^{-\mathcal{C}\varepsilon^2 k} \tag{13.8}$$

holds, where $\mathcal{C}$ is a universal constant.

13.1.3 Approximating the identity

If T is a $k \times n$ RP where $k = O(\varepsilon^{-2} \ln n)$, both $TT^\top$ and $T^\top T$ have some relation with the identity matrices I_k and I_n. This is a lesser known phenomenon, so it is worth discussing it here in some detail.

We look at $TT^\top$ first. By [314, Cor. 7] for any $\epsilon \in (0, \frac{1}{2})$ we have

$$\|\frac{1}{n} T T^\top - I_k\|_2 \leq \varepsilon$$

with probability at least $1 - \delta$ as long as $n \geq \frac{(k+1)\ln(2k/\delta)}{\mathcal{C}\varepsilon^2}$, where $\mathcal{C} \geq \frac{1}{4}$ is a constant.

In Table 13.1 we give values of $\|s\, TT^\top - I_d\|_2$ for $s \in \{1/n, 1/d, 1\}$, $n \in \{1000, 2000, \ldots, 10000\}$ and $d = \lceil \ln(n)/\epsilon^2 \rceil$ where $\epsilon = 0.15$. It is clear that the error decreases as the size increases only in the case

	n									
s	1e3	2e3	3e3	4e3	5e3	e3	7e3	8e3	9e3	1e4
$1/n$	9.72	7.53	6.55	5.85	5.36	5.01	4.71	4.44	4.26	4.09
$1/d$	5e1	1e2	1.5e2	2e2	2.5e2	3e2	3.5e2	3.9e2	4.4e2	4.8e2
1	2e5	4e5	6e5	8e5	1e6	1.2e6	1.4e6	1.6e6	1.8e6	2e6

Table 13.1: Values of $\|sTT^\top - I_d\|$ in function of s, n.

$s = \frac{1}{n}$. This seems to indicate that the scaling is a key parameter in approximating the identity.

Let us now consider the product $T^\top T$. It turns out that, for each fixed vector x not depending on T, the matrix $T^\top T$ behaves like the identity w.r.t. x.

13.1.9 Theorem
Given any fixed $x \in \mathbb{R}^n$, $\varepsilon \in (0,1)$ and a RP $T \in \mathbb{R}^{d \times n}$, there is a universal constant $\mathcal{C}$ such that

$$-\mathbf{1}\varepsilon \leq T^\top T x - x \leq \mathbf{1}\varepsilon \tag{13.9}$$

with probability at least $1 - 4e^{\mathcal{C}\varepsilon^2 d}$.

Proof. By definition, for each $i \leq n$ we have $x_i = \langle e_i, x\rangle$, where e_i is the i-th unit coordinate vector. By elementary linear algebra we have $\langle e_i, T^\top T x\rangle = \langle Te_i, Tx\rangle$. By [84, Lemma 3.1], for $i \leq n$ we have

$$\langle e_i, x\rangle - \varepsilon\|x\|_2 \leq \langle Te_i, Tx\rangle \leq \langle e_i, x\rangle + \varepsilon\|x\|$$

with high probability, which implies the result. □ □

One might be tempted to infer from Thm. 13.1.9 that $T^\top T$ "behaves like the identity matrix" (independently of x). This is generally false: Thm. 13.1.9 only holds for a given (fixed) x.

In fact, since T is a $k \times n$ matrix with $k < n$, $T^\top T$ is a PSD $n \times n$ matrix with rank k, hence $n - k$ of its eigenvalues are zero — and the nonzero eigenvalues need not have value one. On the other hand, $T^\top T$ looks very much like a slightly perturbed identity, on average, as shown in Table 13.2.

13.2 Random projections in mathematical programming

RPs have mostly been applied to probabilistic approximation algorithms. By randomly projecting their (vector) input, one can execute algorithms with lower-dimensional vector more efficiently. The approximation guarantee is usually derived from the JLL or similar results.

n	diagonal	off-diag
500	1.00085	0.00014
1000	1.00069	0.00008
1500	0.99991	-0.00006
2000	1.00194	0.00005
2500	0.99920	-0.00004
3000	0.99986	-0.00000
3500	1.00044	0.00000
4000	0.99693	0.00000

Table 13.2: Average values of diagonal and off-diagonal components of $T^\top T$ in function of n, where T is a $k \times n$ RP with $k = O(\varepsilon^{-2} \ln n)$ and $\varepsilon = 0.15$.

A line of research about applying RPs to MP formulations was started in [303, 302, 301, 84]. Whichever algorithm one may choose in order to solve the MP, the RP properties guarantee an approximation on optimality and/or feasibility. Thus, this approach leads to stronger/more robust results with respect to applying RPs to algorithmic input.

Linear and integer feasibility problems (i.e. LP and MILP formulations without objective function) are investigated in [303] from a purely theoretical point of view. The effect of RPs on LPs (with nonzero objective) are investigated in [302], both theoretically and computationally. Specifically, the randomly projected LP formulation is shown to have bounded feasibility error and an approximation guarantee on optimality. The computational results suggest that the range of practical application of this technique starts with relatively small LPs (thousands of variables/constraints). In both [303, 302] we start from a (MI)LP in standard form

$$\mathcal{P} \equiv \min\{c^\top x \mid Ax = b \wedge x \geq 0 \wedge x \in X\} \tag{13.10}$$

(where $X = \mathbb{R}^n$ or $\mathbb{Z}^n$ respectively), and obtain a randomly projected formulation under the RP $T \sim \mathsf{N}^{k \times n}(0, \frac{1}{\sqrt{k}})$ with the form

$$T\mathcal{P} \equiv \min\{c^\top x \mid TAx = Tb \wedge x \geq 0 \wedge x \in X\}, \tag{13.11}$$

i.e. T reduces the number of constraints in $\mathcal{P}$ to $O(\ln n)$, which can therefore be solved more efficiently.

The RP technique in [301, 84] is different, insofar as it targets the number of variables. In [84] we consider a QP of the form:

$$\mathcal{Q} \equiv \max\{x^\top Q x + c^\top x \mid Ax \leq b\}, \tag{13.12}$$

where Q is $n \times n$, $c \in \mathbb{R}^n$, A is $m \times n$, and $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$. This is projected by a $k \times n$ RP T as follows:

$$T\mathcal{Q} \equiv \max\{u^\top \bar{Q} x + \bar{c}^\top u \mid \bar{A}u \leq b\}, \tag{13.13}$$

where $\bar{Q} = TQT^\top$ is $k \times k$, $\bar{A} = AT^\top$ is $m \times k$, $\bar{c} = Tc$ is in $\mathbb{R}^k$, and $u \in \mathbb{R}^k$. In [301] we consider a QCQP $\mathcal{Q}'$ like $\mathcal{Q}$ but subject to a ball constraint $\|x\|_2 \leq 1$. In the projected problem $T\mathcal{Q}'$, this is replaced by a ball constraint $\|u\|_2 \leq 1$. Both [84, 301] are both theoretical and computational. In both cases, the number of variables of the projected problem is $O(\ln n)$.

In applying RPs to MPs, one solves the smaller projected problems in order to obtain an answer concerning the corresponding original problems. In most cases one has to devise a way to retrieve a solution for the original problem using the solution of the projected problem. This may be easy or difficult depending on the structure of the formulation and the nature of the RP.

In the rest of this section we shall take an informal approach to proofs, and give reference to the precise versions published in papers. The issue is that many proofs are very technical, and the details end up hiding the intuitive reason why certain statements hold. We shall remind the informal nature of proofs by denoting them by the word "sketch".

13.2.1 Linear feasibility

We shall look at linear feasibility first. Given a matrix A, we denote by A_j the j-th column of A. We consider the pure feasibility problem consisting in finding an x' in the set

$$S = \{x \in X \mid Ax = b\}, \tag{13.14}$$

or determining that S is empty, where A is an $m \times n$ matrix, $b \in \mathbb{R}^m$, and X is any subset of $\mathbb{R}^n$. We mean to apply a RP $k \times m$ matrix T to S so that we obtain:

$$TS = \{x \in X \mid TAx = Tb\}, \tag{13.15}$$

where TS indicates that the set in the RHS of Eq. (13.15) is the result of applying T to S.

We first look at cases where X is finite [302, Thm. 2].

13.2.1 Theorem
Assume $S \neq \varnothing$, and $x' \in S$. Then there exists a constant $\mathcal{C} > 0$ (independent of n, k) such that the following three statements:

(a) if $b = \sum_{j \leq n} x'_j A_j$ then $Tb = \sum_{j \leq n} x'_j TA_j$;

(b) if $b \neq \sum_{j \leq n} x'_j A_j$ then $\mathsf{Prob}\left(Tb = \sum_{j \leq n} x'_j TA_j\right) \geq 1 - 2\,e^{-\mathcal{C}k}$;

(c) if $|X|$ is finite and $b \neq \sum_{j \leq n} y_j A_j$ for all $y \in X$, then

$$\mathsf{Prob}\left(\forall y \in X\ Tb \neq \sum_{j \leq n} y_j TA_j\right) \geq 1 - 2|X|\,e^{-\mathcal{C}k}; \tag{13.16}$$

all hold.

Proof. (*Sketch*) Statement (a) holds by linearity of T. On the other hand, if x' is not in S, then $\|Ax - b\|_2 > 0$. By the JLL, Euclidean distances are well projected by T with high probability, which proves Statement (b). Statement (c) follows by the intersection bound Lemma 13.1.2 applied to the $|X|$ independent events that each $y \in X$ is infeasible in $Ax = b$ and hence, by Statement (b), also in $TAx = Tb$ with the given probability. □

By Thm. 13.2.1, we conclude that $TS \neq \varnothing$ if $S \neq \varnothing$, and that $TS = \varnothing$ if $S = \varnothing$ if certain other conditions hold. In general, Thm. 13.2.1 is not sufficient to grant an "if and only if" feasibility relationship between S and TS for any X, which is what we would like.

13.2.2 Corollary
If $|X|$ grows polynomially fast with the storage size of (A, b), then $TS = \varnothing$ if $S = \varnothing$ with high probability.

Proof. (*Sketch*) We observe that "with high probability" stands in this context for an event $E(t)$ such that $\mathsf{Prob}(E) \geq 1 - \alpha e^{-\beta t}$ for some positive α, β. The corollary follows by remarking that, if $|X|$ grows polynomially with the instance size, then $|X|e^{-\mathcal{C}k}$ tends to zero as k tends to infinity, even as $|X|$ also grows polynomially with k. In particular, for any small probability p of failure, there is a possibly large integer $K > 0$ such that $|X|e^{-\mathcal{C}k} \leq p$ for all $k > K$, which implies that we can make the probability

of success $1 - |X|e^{-\mathcal{C}k}$ as large as we want. Here, by "failure" we mean the occurrence that $TS \neq \varnothing$ whenever $S = \varnothing$. □

By Cor. 13.2.2, it follows that for S such that $|X|$ grows polynomially with the size of (A, b), we have $S = \varnothing$ iff $TS = \varnothing$ with high probability, which implies that we can solve the smaller-sized projected problem, decide whether TS is empty or not, and correspondingly infer the emptiness of S with high probability, without having solved the larger-sized original problem. The issue is given by the requirement that $|X|$ should increase polynomially: this is not easy to prove *a priori* for some given S. Moreover, the requirement is certainly not satisfied whenever X is infinite, which happens e.g. if X is the non-negative orthant (the feasibility problem corresponding to LP).

In order to address LP feasibility, we recall that the ellipsoid method (Sect. 7.3) works by iteratively separating an infeasible point from a polyhedron [127]. Specifically, we can re-state the separating hyperplane Prop. 6.2.5 in this setting as follows: if $A_1, \ldots, A_n, b$ are unit column vectors in $\mathbb{R}^m$ such that $C = \mathsf{cone}(A_1, \ldots, A_n)$ is a pointedcone and $b \notin C$, then there is an $\varepsilon > 0$ and a vector $c \in \mathbb{R}^m$ such that $\langle c, b\rangle < -\varepsilon$ and $\langle c, A_j\rangle \geq \varepsilon$ for each $j \leq n$. We now prove that if T is a RP, then a similar statement holds for $\{TA_j \mid j \leq n\}$ and Tb with high probability [302, Thm. 3].

13.2.3 Theorem
Given (a) $c, b, A_1, \ldots, A_n \in \mathbb{R}^m$ of unit norm such that $C = \mathsf{cone}(A_1, \ldots, A_n)$ is pointed and $b \notin C$; (b) an $\varepsilon > 0$ such that $\langle c, b\rangle < -\varepsilon$ and $c^\top, A_j \geq \varepsilon$ for all $j \leq n$; (c) a $k \times m$ RP matrix T such that Eq. (13.2) holds,

$$\mathsf{Prob}(Tb \notin \mathsf{cone}(TA_1, \ldots, TA_n)) \geq 1 - 4(n+1)e^{-\mathcal{C}(\varepsilon^2 - \varepsilon^3)k}, \tag{13.17}$$

where $\mathcal{C}$ is a universal constant (i.e. not depending on the problem data).

Proof. Let $\mathscr{A}$ be the event that T approximately preserves $\|c-\chi\|_2^2$ and $\|c+\chi\|_2^2$ for all $\chi \in \{b, A_1, \ldots, A_n\}$. Since $\mathscr{A}$ consists of $2(n+1)$ events, by Cor. 13.1.7 and the intersection bound Lemma 13.1.2, we obtain

$$\mathsf{Prob}(\mathscr{A}) \geq 1 - 4(n+1)e^{-\mathcal{C}(\varepsilon^2 - \varepsilon^3)k}.$$

Now consider $\chi = b$. We have:

$$\begin{aligned}
\langle Tc, Tb\rangle &= \frac{1}{4}(\|T(c+b)\|^2 - \|T(c-b)\|^2) \\
&\leq \frac{1}{4}(\|c+b\|^2 - \|c-b\|^2) + \frac{\varepsilon}{4}(\|c+b\|^2 + \|c-b\|^2) \quad \text{by the JLL} \\
&= \langle c, b\rangle + \varepsilon < 0 \quad \text{by hypothesis.}
\end{aligned}$$

Similarly, we obtain $\langle Tc, TA_i\rangle \geq \varepsilon$, as claimed. □

Thm. 13.2.3 allows us to claim that if $S = \varnothing$ then $TS = \varnothing$ with high probability even when X is the nonnegative orthant.

13.2.2 Linear optimization

In Sect. 13.2.1 we have argued that separation is sufficient to establish linear feasibility, which is the essential ingredient of the ellipsoid method. Thus, in a certain sense, we have already dealt with RPs applied to LP. On the other hand, we would like to quantify the error that solving a projected LP yields w.r.t. the original LP.

We recall the notation $\mathcal{P}$ for the original LP formulation and $T\mathcal{P}$ for the projected LP formulation, introduced in Eq. (13.10)-(13.11).

13.2.4 Theorem
Assume that $\mathsf{feas}(\mathcal{P})$ *is non-empty and bounded, and that there is a* $\theta > 0$ *such that all of the optima* x^* *of* $\mathcal{P}$ *satisfy* $\sum_j x_j^* \leq \theta$. *Given* $\varepsilon \in (0,1)$, *a* $k \times m$ *RP matrix* T *(where the constraint matrix of* $\mathcal{P}$ *is* $m \times n$ *and* $k = O(\varepsilon^{-2} \ln n)$*), we have*

$$\mathsf{Prob}\left(\mathsf{val}(\mathcal{P}) - 2(\theta+1)\varepsilon\eta \leq \mathsf{val}(T\mathcal{P}) \leq \mathsf{val}(\mathcal{P})\right) \geq 1 - 4ne^{\mathcal{C}(\varepsilon^2-\varepsilon^3)k}, \tag{13.18}$$

where $\eta = O(\|y^*\|_2)$, y *is a dual optimal solution of* $\mathcal{P}$ *having minimum* ℓ_2 *norm, and* $\mathcal{C}$ *is a universal constant.*

The full and detailed proof of Thm. 13.2.4 is given in [302, §4]. Here we give a very brief proof sketch.
Proof. (*Sketch*) The easy part is showing that $\mathsf{val}(T\mathcal{P}) \leq \mathsf{val}(\mathcal{P})$: the constraints of $\mathcal{P}$ are $Ax = b \wedge x \geq 0$; those of $T\mathcal{P}$ are $TAx = Tb \wedge x \geq 0$. By definition, this implies that each constraint of $T\mathcal{P}$ is a random linear combination of constraints of $\mathcal{P}$. Thus, every feasible solution of $\mathcal{P}$ is also feasible in $T\mathcal{P}$. Moreover, $\mathcal{P}$ and $T\mathcal{P}$ have the same objective functions, which makes $T\mathcal{P}$ a relaxation of $\mathcal{P}$, which in turn proves the claim. The difficult part, which consists in showing that $\mathsf{val}(P) - \gamma \leq \mathsf{val}(T\mathcal{P})$ for $\gamma = 2(\theta+1)\varepsilon\eta$, is only summarily sketched. We reformulate $\mathcal{P}$ to the following pure feasibility system:

$$\left.\begin{array}{rcl} cx & \leq & \mathsf{val}(\mathcal{P}) - \gamma \\ Ax & = & b \\ x & \geq & 0, \end{array}\right\} \tag{13.19}$$

and note that the reformulation is exact iff $\gamma = 0$; in particular, Eq. (13.19) is infeasible for $\gamma > 0$. By Sect. 13.2.1, the projected linear feasibility system

$$\left.\begin{array}{rcl} cx & \leq & \mathsf{val}(T\mathcal{P}) - \gamma \\ TAx & = & Tb \\ x & \geq & 0 \end{array}\right\} \tag{13.20}$$

is also infeasible with high probability for $\gamma > 0$. Therefore, $cx < \mathsf{val}(T\mathcal{P}) - \gamma$ is infeasible for $x \in \mathsf{feas}(T\mathcal{P})$ with high probability, implying that $cx \geq \mathsf{val}(T\mathcal{P}) - \gamma$ is feasible for $x \in \mathsf{feas}(T\mathcal{P})$ with high probability. Thus $\mathsf{val}(\mathcal{P}) - \gamma \leq cx = \mathsf{val}(T\mathcal{P})$, as claimed. □

Unfortunately, the approximation guarantee afforded by Thm. 13.2.4 rests on the value of dual optimum y^* of the original LP having the smallest ℓ_2 norm. As such, it cannot be evaluated *a priori* on general LPs before having actually solved the LP itself (which would make the projected LP moot). If the LP structure provides a bound on the smallest ℓ_2 norm of a dual optimum, then solving the projected LP provides an additive approximation algorithm for the original LP.

13.2.3 Solution retrieval

Most often we are not only interested in $\mathsf{val}(P)$, but also need an optimal solution x^* of an LP formulation $\mathcal{P}$. By the easy part of the proof of Thm. 13.2.4, $T\mathcal{P}$ is a relaxation of $\mathcal{P}$. Since we usually want $k < m$ (otherwise the RP would not make the LP smaller), a solution $\bar{x}$ of $T\mathcal{P}$ is going to be in $\mathsf{feas}(\mathcal{P})$ with probability zero (this holds because $\mathsf{rk}(TA, Tb) = k < m = \mathsf{rk}(A, b)$, assuming that the constraint matrix (A, b) of $\mathcal{P}$ has full rank). Given a set of column indices J from a matrix M, we denote by M_J the submatrix of M consisting of the columns indexed by J.

Let $\bar{x}$ be an optimum of $T\mathcal{P}$, corresponding to a set H of column indices of TA in a basis corresponding to $\bar{x}$ (see Defn. 7.1.1). Note that $|H| = k$, that that $(TA)_H \bar{x} = b$, and that, by definition of basis we know that $(TA)_H$ is nonsingular, so that we can write $\bar{x} = (TA)_H^{-1} b$.

Since A and TA have the same column indices, we let A_H be the $m \times k$ matrix formed by the columns of A indexed by H. We would like to find x satisfying $A_H x' = b$, but, unfortunately, A_H is not even

square, let alone invertible. We therefore resort to a heuristic method based on the pseudoinverse: we form the system

$$A_H^\top A_H x' = A_H^\top b, \tag{13.21}$$

obtained by pre-multiplying $A_H x' = b$ by $A_H^\top$, and observe that the matrix $A_H^\top A^\top$ is $k \times k$ and nonsingular since A_H has linearly independent columns by construction. Therefore Eq. (13.21) can be solved for x' as follows:

$$x' = (A_H^\top A_H)^{-1} A_H^\top b. \tag{13.22}$$

Now x' is a vector in $\mathbb{R}^k$. In order to construct a feasible solution $\tilde{x}$ for the original system $Ax = b$, we make a heuristic choice, and arbitrarily decide to fill in with zeros all of the components corresponding to indices not in H. Reindexing the columns of A so that $H = \{1, \ldots, k\}$, we obtain $\tilde{x} = (x', \mathbf{0})$, where $\mathbf{0}$ is a zero vector in $\mathbb{R}^{n-k}$. By construction, we know that $A\tilde{x} = b$. On the other hand, our heuristic choice means that some feasibility error with respect to $\mathsf{feas}(\mathcal{P})$ is likely to occur. By exclusion, we expect $\tilde{x} \not\geq 0$.

It was empirically shown in [302] that $\min(0, \tilde{x}) \to 0$ as $k \to \infty$, which is reassuring. On the whole, however, a method for solving LPs which does not respect the nonnegativity constraints is perplexing and unusual. Luckily, there are some applications which "forgive" such errors by design. When basis pursuit LPs (see Eq. (12.4)) are used for encoding/decoding digital messages on noisy channels, any value of $\tilde{x}$ outside of $[0, 1]$ is rounded to $\{0, 1\}$ (since these are the only possible bit values): if $\tilde{x}_j < 0$ then $\tilde{x}_j$ is set to 0, and if $\tilde{x}_j > 1$ then $\tilde{x}_j$ is set to 1; values in $[0, 1]$ are rounded to the closest binary value.

13.2.4 Quadratic optimization

We recall the original QP formulation $\mathcal{Q}$ and its projected version $T\mathcal{Q}$ in Eq. (13.12)-(13.13). We note that while our LP formulation $\mathcal{P}$ was in minimization form, $\mathcal{Q}$ and $T\mathcal{Q}$ are maximization problems.

13.2.4.1 Feasibility and retrieval

As for the linear case (Sect. 13.2.1-13.2.3, we must account for feasibility, error w.r.t. optimality, and solution retrieval. In the QP case, however, the first and last issue are much simpler to deal with than in the LP case.

13.2.5 Proposition
Let T be a $k \times n$ RP and u^* be an optimal solution of $T\mathcal{Q}$. Then $\tilde{x} = T^\top u^*$ is in $\mathsf{feas}(\mathcal{Q})$.

Proof. We have

$$A\tilde{x} = A(T^\top u^*) = (AT^\top)u^* = \bar{A}u^* \leq b$$

since $u^* \in \mathsf{feas}(T\mathcal{Q})$ and $\bar{A} = AT^\top$. □

Note that Prop. 13.2.5 handles both feasibility and solution retrieval. For the latter, given an optimum u^* of $T\mathcal{Q}$, a solution $\tilde{x}$ for $\mathcal{Q}$ is constructed by simply setting $\tilde{x} = T^\top u^*$ (we shall see the approximation error of $\tilde{x}$ later). For the former, the feasibility of $\tilde{x}$ is very easy to prove. It is also very easy to see that $T\mathcal{Q}$ is a restriction of $\mathcal{Q}$ (or, equivalently, that $\mathcal{Q}$ is a relaxation of $T\mathcal{Q}$).

13.2.6 Proposition
$\mathsf{val}(T\mathcal{Q}) \leq \mathsf{val}(\mathcal{Q})$.

Proof. Let u^* be an optimum of $T\mathcal{Q}$, and $\tilde{x} = T^\top u^*$. By Prop. 13.2.5, $\tilde{x}$ is feasible in $\mathcal{Q}$. Now the objective function value of $\mathcal{Q}$ at $\tilde{x}$, we have

$$\tilde{x}^\top Q x + c^\top x = {u^*}^\top T Q T^\top u^* + c^\top T^\top u^* = {u^*}^\top \bar{Q} u^* = \bar{c}^\top u^* = \mathsf{val}(T\mathcal{P}).$$

Moreover, since $\tilde{x}$ is feasible in $\mathcal{Q}$, $\mathsf{val}(\mathcal{Q}) \geq \tilde{x}^\top Q x + c^\top x$. Since $\mathcal{Q}$ and $T\mathcal{Q}$ are maximization problems, the result follows. □

13.2.4.2 Approximation error

By Prop. 13.2.6, the main issue in applying RPs to QP is to bound the approximation error. We make the following assumptions:

1. all the rows of A are unit vectors;
2. $Ax \leq b$ is a full-dimensional (bounded) polytope $\mathscr{P}$ with non-empty interior;
3. there is a sphere circumscribing $\mathscr{P}$ and centered at the origin with given radius R;
4. there exists a sphere inscribed in $\mathscr{P}$ with known radius r.

We note that Assumption 1 is wlog: it suffices to rescale $Ax \leq b$ to the system $\forall i \leq m\ \langle A^i/\|A^i\|_2, x\rangle \leq b_i/\|A^i\|_2$, where A^i is the i-th row of A.

We introduce the following perturbed QP

$$\mathcal{Q}_\varepsilon \equiv \max\{x^\top Q x + c^\top x \mid Ax + R\varepsilon \mathbf{1} \leq b\}$$

and its projected version

$$T\mathcal{Q}_\varepsilon \equiv \max\{u^\top \bar{Q} x + \bar{c}^\top u \mid \bar{A}u + \mathbf{1} \leq b\}, \tag{13.23}$$

where $\mathbf{1}$ is the all-one vector in $\mathbb{R}^m$.

The following approximation theorems are proved in [85]. The first theorem expresses the error of $T\mathcal{Q}$ in terms of $\mathsf{val}(\mathcal{Q})$ being between $\mathsf{val}(T\mathcal{Q})$ and a perturbation plus an error depending on the objective function data.

13.2.7 Theorem
There is a universal constant $\mathcal{C} > 1$ such that:

$$\mathsf{Prob}(\mathsf{val}(T\mathcal{Q}) \leq \mathsf{val}(\mathcal{Q}) \leq \mathsf{val}(T\mathcal{Q}_\varepsilon)) \geq 1 - O(m)e^{-\mathcal{C}\varepsilon^2 k}, \tag{13.24}$$

where $\eta = 3R^2\varepsilon\|Q\|_{\mathrm{F}} + R\varepsilon\|c\|_2$.

The second theorem shows that $\mathsf{val}(T\mathcal{Q})$ is "not too far" from $\mathsf{val}(T\mathcal{Q}_\varepsilon)$ (multiplicative approximation).

13.2.8 Theorem
There is a universal constant $\mathcal{C} > 1$ such that, if $\varepsilon \in (0, r/R)$, then

$$\mathsf{Prob}(\sigma^2\mathsf{val}(T\mathcal{Q}_\varepsilon) \leq \mathsf{val}(T\mathcal{Q}) \leq \mathsf{val}(T\mathcal{Q}_\varepsilon)) \geq 1 - O(m)e^{-\mathcal{C}\varepsilon^2 k}, \tag{13.25}$$

where $\sigma = 1 - \frac{R\varepsilon}{r(1-\varepsilon)^2} > 0$.

The third theorem shows that, if $\mathcal{Q}$ is convex, $\mathsf{val}(T\mathcal{Q})$ is "not too far" from $\mathsf{val}(T\mathcal{Q}_\varepsilon)$ (additive approximation).

13.2.9 Theorem
There is a universal constant $\mathcal{C} > 1$ such that, if $\mathcal{Q}$ is a cQP, we have:

$$\mathsf{Prob}(\mathsf{val}(T\mathcal{Q}_\varepsilon) \leq \mathsf{val}(T\mathcal{Q}) \leq \mathsf{val}(T\mathcal{Q}_\varepsilon) + E) \geq 1 - O(m+\rho)e^{-\mathcal{C}\varepsilon^2 k}, \tag{13.26}$$

where $\rho = \mathsf{rk}(Q)$, $E = \eta + \varepsilon\|x^*\|_2\|y^*\|_1$, x^* is a primal global optimum of $\mathcal{Q}$ having minimum ℓ_2 norm, and y^* is a dual global optimum of $T\mathcal{Q}$ having minimum ℓ_1 norm.

We note that none of the above theorems gives a tight approximation error bound, unfortunately. Moreover, the bound for the convex case cannot be computed *a priori* because it is in terms of the solution of $\mathcal{Q}$.

In the following, we shall only give a sketch of the proof of Thm. 13.2.7. We first present three lemmata about additive errors. We write $x \in y \pm \alpha$ to mean $x \in [y - \alpha, y + \alpha]$.

13.2.10 Lemma
For any $x, y \in \mathbb{R}^n$ there is a $k \times n$ RP T such that:

$$\mathsf{Prob}(\langle Tx, Ty\rangle \in \langle x, y\rangle \pm \varepsilon\|x\|_2\|y\|_2) \geq 1 - 4e^{-\mathcal{C}\varepsilon^2 k}, \tag{13.27}$$

where $\mathcal{C}$ is a universal constant.

Proof. Let $\mathcal{C}$ be the same universal constant as in Lemma 13.1.8. By Eq. (13.8), for any two vectors $u + v$ and $u - v$, we have

$$\begin{aligned} |\langle Pu, Pv\rangle - \langle u, v\rangle| &= \frac{1}{4}\big|\|P(u+v)\|^2 - \|P(u-v)\|^2 - \|u+v\|^2 + \|u-v\|^2\big| \\ &\leq \frac{1}{4}\big|\|P(u+v)\|^2 - \|u+v\|^2\big| + \frac{1}{4}\big|\|P(u-v)\|^2 - \|u-v\|^2\big| \\ &\leq \frac{\varepsilon}{4}(\|u+v\|^2 + \|u-v\|^2) = \frac{\varepsilon}{2}(\|u\|^2 + \|v\|^2), \end{aligned}$$

with probability at least $1 - 4e^{-\mathcal{C}\varepsilon^2 k}$ (the coefficient 4 is due to having invoked the event in Eq. (13.8) twice, and the intersection bound Lemma 13.1.2). Applying the above derivation to $u = x/\|x\|_2$ and $v = y/\|y\|_2$ yields the result. □

13.2.11 Lemma
For any $\varepsilon \in (0, 1)$, $x \in \mathbb{R}^n$, $m \times n$ matrix A with unit row vectors, there is a $k \times n$ RP matrix T such that:

$$\mathsf{Prob}(AT^\top Tx \in Ax \pm \varepsilon\|x\|_2\mathbf{1}) \geq 1 - 4me^{-\mathcal{C}\varepsilon^2 k},$$

where $\mathcal{C}$ is a universal constant.

Proof. Let $A_1, \ldots, A_m$ be the unit row vectors of A. We have:

$$AT^\top Tx - Ax = \begin{pmatrix} A_1^\top T^\top Tx - A_1^\top x \\ \cdots \\ A_m^\top T^\top Tx - A_m^\top x \end{pmatrix} = \begin{pmatrix} (TA_1)^\top Tx - {A_1}^\top x \\ \cdots \\ (TA_m)^\top Tx - {A_m}^\top x \end{pmatrix}.$$

The result follows by repeatedly applying Lemma 13.2.10 and the intersection bound Lemma 13.1.2. □

13.2.12 Lemma
For any $\varepsilon \in (0, 1)$, $x, y \in \mathbb{R}^n$, $n \times n$ symmetric matrix Q having rank ρ, there is a $k \times n$ RP matrix T such that:

$$\mathsf{Prob}(x^\top T^\top TQT^\top Ty \in x^\top Qy \pm 3\varepsilon\|x\|_2\|y\|_2\|Q\|_{\mathrm{F}}) \geq 1 - 8\rho e^{-\mathcal{C}\varepsilon^2 k},$$

where $\mathcal{C}$ is a universal constant.

Proof. (*Sketch*) Let $U\Sigma V^\top$ be the Singular Value Decomposition (SVD) of Q

$$x^\top T^\top TQT^\top Ty = \big(U^\top x + U^\top(T^\top T - I_n)x\big)^\top \Sigma\big(V^\top y + V^\top(T^\top T - I_n)y\big)$$

$$\begin{aligned}\Rightarrow |x^\top T^\top TQT^\top Ty - x^\top Qy| &\leq |(U^\top x)^\top \Sigma V^\top (T^\top T - I_n)y| \\ &+ |(U^\top (T^\top T - I_n)x)^\top \Sigma V^\top y| \\ &+ |(U^\top (T^\top T - I_n)x)^\top \Sigma V^\top (T^\top T - I_n)y|. \quad (*)\end{aligned}$$

Lemma 13.2.10 applied to $\langle (V\Sigma)_i, y\rangle$, the intersection bound Lemma 13.1.2, and the symmetry of Σ implies that, with probability $1 - 8\rho e^{-\mathcal{C}\varepsilon^2 k}$, we have:

$$\forall i \leq n \quad |(\Sigma V^\top (T^\top T - I_n)y)_i| \leq \varepsilon\sigma_i\|y\| \tag{13.28}$$

$$\forall i \leq n \quad |((U^\top (T^\top T - I_n)x)^\top \Sigma)_i| \leq \varepsilon\sigma_i\|x\| \tag{13.29}$$

The Cauchy-Schwartz inequality applied to the RHS of $(*)$ yields bounds on the RHS terms in function of their norms. Using Eq. (13.28)-(13.29), $\|U^\top x\| \leq \|x\|$, and $\|V^\top y\| \leq \|y\|$ (since $UU^\top$ and $VV^\top$ are projection matrices), we have:

$$\begin{aligned}\langle U^\top x\,, \Sigma V^\top (T^\top T - I_n)y\rangle &\leq \varepsilon\|x\|\,\|y\|\,\|\sigma\| \\ \langle (U^\top (T^\top T - I_n)x)^\top \Sigma\,, V^\top y\rangle &\leq \varepsilon\|y\|\,\|x\|\,\|\sigma\| \\ \langle (U^\top (T^\top T - I_n)x)^\top \Sigma\,, V^\top (T^\top T - I_n)y\rangle &\leq \varepsilon^2\|x\|\,\|y\|\,\|\sigma\| \\ \Rightarrow |x^\top T^\top TQT^\top Ty - x^\top Qy| &\leq 3\varepsilon\|x\|\,\|y\|\,\|\sigma\|,\end{aligned}$$

as claimed. □

Proof of Thm. 13.2.7 (Sketch). Let x^* be a global optimum of the original problem $\mathcal{Q}$. Its feasibility follows by Prop. 13.2.5. The bounds on the linear part of the optimal objective function value $\bar{c}^\top Tx^*$ follow by Lemma 13.2.11. The bounds on the quadratic part $(Tx^*)^\top \bar{Q}Tx^*$ follow by Lemma 13.2.12. Combine the bounds so the slackest is accommodated. The probability follows by the intersection bound Lemma 13.1.2. □

13.2.4.3 Relations of QP results to LP

All of the results above, about applying RPs to QPs, hold whenever Q is the zero matrix, i.e. whenever the original QP is in fact an LP.

The relationship between these results and those about directly applying RPs to LP (Sect. 13.2.1-13.2.3) is as follows. Setting $Q = 0$ in $\mathcal{Q}$ yields an LP $\mathcal{D} = \max\{c^\top x \mid Ax \leq b\}$ with more constraints than variables, which is essentially the LP dual of $\mathcal{P}$ (note that the symbol x in $\mathcal{D}$ is not the same as the symbol x in $\mathcal{P}$). The projected LP $T\mathcal{D} = \max\{\bar{c}^\top u \mid \bar{A}u \leq b\}$ reduces the number of variables and keeps the number of constraints the same. Naturally, the analysis carries over if we want to solve the dual of $\mathcal{D}$, i.e. an LP with the form of $\mathcal{P}$.

In this sense, the results on QPs provide a different analysis for $\mathcal{P}$ than the one derived in Sect. 13.2.1-13.2.3.

13.3 Minimum sum-of-squares clustering

In this section we discuss the application of RPs to some MP formulations of the MINLP class. Differently from Sect. 13.2, we do not apply RPs to a MP subfamily (such as LP or QP) but to a specific application related to clustering.

Given a set P of n entities and some pairwise similarity function $P \times P \to \mathbb{R}$, cluster analysis aims at finding a set of k subsets $C_1, \ldots, C_k \subseteq P$ such that each cluster contains as many similar entities, and as few dissimilar entities, as possible. Cluster analysis — as a field — grew out of statistics in the course

of the second half of the 20th century, encouraged by the advances in computing power. But some early forms of cluster analysis may also be attributed to earlier scientists (e.g. Aristotle, Buffon, Cuvier, Linné [131]).

One of the most studied cluster analysis problems occurs over Euclidean spaces.

MINIMUM SUM-OF-SQUARES CLUSTERING (MSSC). Given an integer $k > 0$ and a set $P \subset \mathbb{R}^m$ of n vectors, find a set $\mathcal{C} = \{C_1, \ldots, C_k\}$ of subsets of P such that the function

$$f(\mathcal{C}) = \sum_{j \leq k} \sum_{p \in C_j} \|p - \mathsf{centroid}(C_j)\|_2^2 \tag{13.30}$$

is minimum, where

$$\mathsf{centroid}(C_j) = \frac{1}{|C_j|} \sum_{p \in C_j} p \tag{13.31}$$

and $p \in P$.

The MSSC is the problem which k-means [203] aims at solving. The k-means algorithm is one of the most famous algorithms in cluster analysis. It improves a given initial clustering $\mathcal{C}$ by means of the two following operations:

1. compute centroids $c_j = \mathsf{centroid}(C_j)$ for each $j \leq k$;
2. for any pair of clusters $C_h, C_j \in \mathcal{C}$ and any point $x \in C_h$, if x is closest to c_j than to c_h, move x from C_h to C_j.

These two operations are repeated until $\mathcal{C}$ no longer changes. Since the only decision operation (i.e. operation 2) carries out a change only if it decreases $f(\mathcal{C})$, it follows that k-means is a local descent algorithm. As stated, it offers no guarantee on the approximation of the objective function. It is interesting to note that the MSSC problem can also be seen as a discrete analogue of the problem of partitioning a body into smaller bodies having minimum sum of moments of inertia [279].

The MSSC can be formulated by means of two arrays of decision variables: $x_{ij} \in \{0, 1\} = 1$ if point $p_i \in P$ (for $i \leq n$) is assigned to cluster $j \leq k$ and zero otherwise, and $y_j \in \mathbb{R}^m$ denoting $\mathsf{centroid}(C_j)$, the centroid of cluster j. This immediately gives a formulation of Eq. (13.30) in terms of x, y:

$$f(x, y) = \sum_{j \leq k} \sum_{\substack{i \leq n \\ x_{ij} = 1}} \|p_i - y_j\|_2^2. \tag{13.32}$$

The function $f(x, y)$ cannot be employed "as is" as an objective function to be minimized, since the decision variables x appear in a sum quantifier, which makes $\min f(x, y)$ an invalid sentence in the MP language. We shall see in Sect. 13.3.1 that there is an easy reformulation of $f(x, y)$ so that it becomes valid.

The data science oriented survey [280] presents some interesting matrix formulations of the MSSC. In particular, [280, Eq. (10)] shows that the MSSC is equivalent to finding the projection matrix $x(x^\top x)^{-1}x^\top$ (where $x \in \{0, 1\}^{n \times k}$ is the point-cluster assignment matrix, as in Eq. (13.32)) minimizing

$$f(x) = \mathsf{tr}(P^\top (I - x(x^\top x)^{-1} x^\top) P). \tag{13.33}$$

13.3.1 cMINLP formulation

In this section, we shall exhibit a cMINLP reformulation of the following (nonconvex) MINLP formulation of the MSSC (with optional side constraints).

$$\left.\begin{array}{lrcl} \min\limits_{x,y,s,\gamma} & \sum\limits_{i\leq n}\sum\limits_{j\leq k} \|p_i - y_j\|_2^2\, x_{ij} & & \\ \forall j \leq k & \frac{1}{s_j}\sum\limits_{i\leq n} p_i x_{ij} & = & y_j \\ \forall j \leq k & \sum\limits_{i\leq n} x_{ij} & = & s_j \\ \forall i \leq n & \sum\limits_{j\leq k} x_{ij} & = & 1 \\ \forall j \leq k & y_j & \in & \mathbb{R}^m \\ & G(x,y,\gamma) & \leq & 0 \\ & x & \in & \{0,1\}^{nk} \\ & s & \in & \mathbb{N}^k. \end{array}\right\} \quad \text{(MSSC)} \tag{13.34}$$

The parameters of formulation Eq. (13.34) are the given set $P = \{p_1, \ldots, p_n\}$ of vectors in $\mathbb{R}^m$, and the number of clusters k. The decision variables are:

- for each $i \leq n$ and $j \leq k$, the binary assignment variables x_{ij}, set to 1 iff vector i is assigned to cluster j and to 0 otherwise;
- for each $j \leq k$, the integer variables s_j, equal to the cardinality of cluster j;
- for each $j \leq k$, the vector $y_j \in \mathbb{R}^m$, containing the centroid of cluster j.

The sentence $G(x, y, \gamma) \leq 0$ encodes some optional side constraints (possibly with additional decision variables γ). Note that the side constraints and additional variables endow Eq. (13.34) with a considerable generality. Insofar as we aim at obtaining a cMINLP reformulation, we require that the continuous relaxation of the set $\{(x, y, \gamma) \mid G(x, y, \gamma) \leq 0\}$ should be convex. Since our reformulations are not going to exploit the structure of these constraints, we no longer list them in the reformulations below.

We note that Eq. (13.34) imposes a restriction on the MSSC solution, namely that each cluster must be non-empty, since the cardinality s_j appears in a denominator. This restriction can be relaxed by multiplying both sides of the equation constraint by s_j.

Solving Eq. (13.34) directly is unadvisable for several reasons. It has a mixture of continuous, binary and general integer variables; some decision variables appear in the denominator of a fraction; while the objective function consists of sums of products of convex terms, the products makes it nonconvex. We shall address all of these issues by using elementary reformulation steps, and construct a cMINLP reformulation of the MSSC.

13.3.1.1 Removing centroid constraints

The first reformulation of MSSC consists in eliminating the centroid constraints. We shall see that this is an *exact reformulation* (i.e. preserving global optima).

$$\left.\begin{array}{lrcl} \min\limits_{x,y} & \sum\limits_{i\leq n}\sum\limits_{j\leq k} \|p_i - y_j\|_2^2\, x_{ij} & & \\ \forall j \leq k & \sum\limits_{i\leq n} x_{ij} & \geq & 1 \\ \forall i \leq n & \sum\limits_{j\leq k} x_{ij} & = & 1 \\ \forall j \leq k & y_j & \in & \mathbb{R}^m \\ & x & \in & \{0,1\}^{nk}. \end{array}\right\} \tag{13.35}$$

13.3.1 Lemma
For any $j \leq k$, $v = \mathsf{centroid}(C_j)$ iff $\sum_{p \in C_j} \|p - v\|_2^2$ is minimum over all $v \in \mathbb{R}^m$.

Proof. By [17, p. 199]. □ □

13.3.2 Proposition
$\mathcal{C}^*$ is a global optimum of Eq. (13.35) iff it is also a global optimum of MSSC.

Proof. Note that any clustering $\mathcal{C}$ is completely defined by the binary assignment variables x, since once the values of x are known, one can easily compute centroids y and their cardinalities s. Moreover, because of the constraint $\sum_i x_{ij} \geq 1$ in Eq. (13.35), no cluster in $\mathcal{C}^*$ may be empty; and, since the term $1/s_j$ in MSSC forces $s_j \geq 1$ for all $j \leq k$, the same must also hold for an optimum of MSSC. Let x^* be the solution for the binary variables x determined by $\mathcal{C}^*$: since the objective functions of the two formulations (MSSC and Eq. (13.35)) are identical, when x is fixed at x^*, their optimal values w.r.t. y must match, whence x^* yields the same optimal objective function values in both. If we denote $f^1(x)$ the objective function of MSSC and $f^2(x)$ that of Eq. (13.35), we have

$$f^1(x^*) = f^2(x^*). \qquad (\dagger)$$

We also observe that Eq. (13.35) was obtained by MSSC by removing the constraints $\frac{1}{s_j} \sum_i p_i x_{ij} = y_j$ and $\sum_i x_{ij} = s_j$, and adjoining the constraint $\sum_i x_{ij} \geq 1$. But the latter is always satisfied in MSSC as observed above, which implies that Eq. (13.35) is a relaxation of MSSC, i.e.

$$\min_x f^1(x) \geq \min_x f^2(x). \qquad (\ddagger)$$

($\Rightarrow$) Eq. ($\dagger$) implies $\min_x f^1(x) \leq \min_x f^2(x)$ since x^* is a global optimum of Eq. (13.35). By Eq. ($\ddagger$), we have $\min_x f^1(x) = \min_x f^2(x)$, showing that x^* is a global optimum of MSSC.
($\Leftarrow$) Let $\mathcal{C}^*$ be a global optimum for MSSC corresponding to x^* and suppose x^* is not a global optimum for Eq. (13.35). By Eq. ($\dagger$)-($\ddagger$), x^* is feasible in Eq. (13.35) and has the same objective function value, so the only way it can fail to be a global optimum is that $\mathcal{C}^*$ is not an optimal clustering for Eq. (13.35). So let $\mathcal{C}'$ be an optimal clustering for Eq. (13.35) with corresponding variables (x', y', s'): the only way it can be better than $\mathcal{C}^*$ is that it should be infeasible for MSSC. By Lemma 13.3.1, y' defines the centroids of the clusters of $\mathcal{C}'$. Moreover, $s'_j = \sum_i x'_{ij} \geq 1$ makes $1/s'_j$ well defined. Hence (x', y', s') must be feasible in MSSC, against the assumption. □ □

The advantage of Eq. (13.35) w.r.t. MSSC is that the former has fewer nonlinear terms as well as fewer constraints.

13.3.1.2 Linearization of products

In this section we reformulate products of terms involving decision variables in an exact way. While this is not generally possible, when at least one of the terms in the product takes a finite set of values and the other can be constrained to lie in a set of variable ranges, it becomes possible [178, §3.3]. In the case of the formulation in Eq. (13.35), we aim at linearizing all products $\alpha_{ij}(x, y) = \|p_i - y_j\|_2^2 x_{ij}$ over all $i \leq n$ and $j \leq k$.

We first remark that, even though the centroid variables y are unconstrained in Eq. (13.35), no centroid may ever lie outside the hyper-rectangle

$$[y^L, y^U] = [\min_{i \leq n} p_i, \max_{i \leq n} p_i], \qquad (13.36)$$

where the min and max operators are applied componentwise to the vectors. This means that $\|p_i - y_j\|_2^2$ lies in $[0, P^U]$ for any $i \le n$, $j \le k$, where

$$P^U = \max_{i<h\le n} \|p_i - p_h\|_2^2. \tag{13.37}$$

Next, we replace $\alpha_{ij}(x,y)$ by additional variables $\chi_{ij} \in [0, P^U]$ in the objective function, and adjoin the defining constraints $\chi_{ij} = \alpha(x,y)$ for each i,j. Now we exploit the fact that $x_{ij} \in \{0,1\}$, which yields $\chi_{ij} \in [0, P^U]$ for all i,j. Again because the x variables are binary,

$$\chi_{ij} = \begin{cases} \|p_i - y_j\|_2^2 & \text{if} \quad x_{ij} = 1 \\ 0 & \text{if} \quad x_{ij} = 0. \end{cases}$$

For each $i \le n$, $j \le k$, let $D = \{0,1\} \times [y^L, y^U]$, and

$$\begin{aligned} A_{ij} &= \{(\chi_{ij}, x_{ij}, y_j) \mid \chi_{ij} = \|p_i - y_j\|_2^2 x_{ij} \wedge (x_{ij}, y_j) \in D\} \\ B_{ij} &= \{(\chi_{ij}, x_{ij}, y_j) \mid 0 \le \chi_{ij} \le P^U x_{ij} \wedge \|p_i - y_j\|_2^2 \le \chi_{ij} + P^U(1 - x_{ij}) \wedge (x_{ij}, y_j) \in D\}. \end{aligned}$$

It is easy to verify by inspection that $A_{ij} \subseteq B_{ij}$ by checking the two cases $x_{ij} = 0$ and $x_{ij} = 1$. Now let

$$\bar{B}_{ij} = \arg\min_{\chi_{ij}} B_{ij}.$$

We claim that $A_{ij} = \bar{B}_{ij}$. Let $\beta' = (\chi'_{ij}, x'_{ij}, y'_j) \in \bar{B}_{ij}$, then χ'_{ij} must be minimal in B_{ij}. If $x'_{ij} = 0$ then $\chi'_{ij} = 0$ which yields $\beta' \in A_{ij}$. If $x'_{ij} = 1$ then $\chi'_{ij} = \|p_i - y'_j\|_2^2$, which again implies that $\beta' \in A_{ij}$. Therefore $A_{ij} \supseteq \bar{B}_{ij}$. Conversely, if $\beta' \in A_{ij}$ then $\beta' \in B_{ij}$ as shown above. In order to obtain $\chi'_{ij} = 0$ if $x'_{ij} = 0$ and $\chi'_{ij} = \|p_i - y_j\|_2^2$ if $x'_{ij} = 1$ then χ'_{ij} must be minimal in B_{ij}, i.e. $\beta' \in \bar{B}_{ij}$ as claimed.

Thus, we can reformulate Eq. (13.35) as follows:

$$\left.\begin{array}{rrcl} \min\limits_{\chi,x,y} & \sum\limits_{i\le n}\sum\limits_{j\le k} \chi_{ij} & & \\ \forall i \le n, j \le k & \chi_{ij} & \le & P^U x_{ij} \\ \forall i \le n, j \le k & \|p_i - y_j\|_2^2 & \le & \chi_{ij} + P^U(1 - x_{ij}) \\ \forall j \le k & \sum\limits_{i\le n} x_{ij} & \ge & 1 \\ \forall i \le n & \sum\limits_{j\le k} x_{ij} & = & 1 \\ \forall i \le n, j \le k & \chi_{ij} & \ge & 0 \\ \forall j \le k & y_j & \in & [y^L, y^U] \\ & x & \in & \{0,1\}^{nk}. \end{array}\right\} \tag{13.38}$$

By the above discussion, Eq. (13.38) is an exact reformulation of Eq. (13.35) and hence, by transitivity, also of MSSC. The advantage of Eq. (13.38) w.r.t. Eq. (13.35) is that the former is a cMINLP instead of a (nonconvex) MINLP.

13.3.2 Approximating reformulations

In this section we introduce two types of approximating reformulations: one based on replacing the ℓ_2 norm with ℓ_1 or ℓ_∞ (the so-called "linearizable norms"), and the other obtained by applying a RP to the point set P.

13.3.2.1 Linearizable norms

In this section we exploit the well-known inequalities

$$\|\zeta\|_\infty^2 \le \|\zeta\|_2^2 \le \|\zeta\|_1^2 \tag{13.39}$$

$$\frac{1}{m}\|\zeta\|_1^2 \le \|\zeta\|_2^2 \le m\|\zeta\|_\infty^2 \tag{13.40}$$

which hold for every $\zeta \in \mathbb{R}^m$. We replace the ℓ_2 norm in MSSC with the ℓ_1 and ℓ_∞ norms, for which we provide approximate MILP reformulations of Eq. (13.38). Since MILP solvers (such as e.g. [146]) are technologically more advanced than cMINLP solvers (such as e.g. [52]), we can hope to solve larger instances of MSSC with the MILP formulations: these will derive bounds on the optimal objective values by means of Eq. (13.39)-(13.40).

13.3.2.1.1 The ℓ_∞ norm We are going to replace $\|p_i - y_j\|_2^2$ in MSSC by $\|p_i - y_j\|_\infty$, and compute P^U with the ℓ_∞ norm. The same reformulations follow through, and we get to a variant of Eq. (13.38) where the (convex) nonlinear constraints are:

$$\|p_i - y_j\|_\infty \leq \chi_{ij} + P^U(1 - x_{ij}) \tag{13.41}$$

for all $i \leq n, j \leq k$. This is equivalent to

$$\max_{\ell \leq m} |p_{i\ell} - y_{j\ell}| \leq \chi_{ij} + P^U(1 - x_{ij})$$

which can be reformulated to

$$\forall \ell \leq m \quad |p_{i\ell} - y_{j\ell}| \leq \chi_{ij} + P^U(1 - x_{ij}),$$

whence

$$\begin{array}{rrcl} \forall \ell \leq m & p_{i\ell} - y_{j\ell} & \leq & \chi_{ij} + P^U(1 - x_{ij}) \\ \forall \ell \leq m & y_{j\ell} - p_{i\ell} & \leq & \chi_{ij} + P^U(1 - x_{ij}). \end{array}$$

Note that the replacement of a square ℓ_2 norm with a non-square ℓ_∞ norm makes χ_{ij} take a linear, rather than squared, value at the optimum (as long as $x_{ij} = 1$). Thus the terms χ_{ij} on the objective function should be squared.

This yields the following approximating reformulation:

$$\left.\begin{array}{rrcl} \min\limits_{\chi,x,y} & \sum\limits_{i \leq n} \sum\limits_{j \leq k} \chi_{ij}^2 & & \\ \forall i \leq n, j \leq k & \chi_{ij} & \leq & P^U x_{ij} \\ \forall i \leq n, j \leq k, \ell \leq m & p_{i\ell} - y_{j\ell} & \leq & \chi_{ij} + P^U(1 - x_{ij}) \\ \forall i \leq n, j \leq k, \ell \leq m & y_{j\ell} - p_{i\ell} & \leq & \chi_{ij} + P^U(1 - x_{ij}) \\ \forall j \leq k & \sum\limits_{i \leq n} x_{ij} & \geq & 1 \\ \forall i \leq n & \sum\limits_{j \leq k} x_{ij} & = & 1 \\ \forall i \leq n, j \leq k & \chi_{ij} & \geq & 0 \\ \forall j \leq k & y_j & \in & [y^L, y^U] \\ & x & \in & \{0,1\}^{nk}, \end{array}\right\} \tag{13.42}$$

which is again a cMINLP, where the only nonlinearities are in the objective function (the constraints are wholly linear).

Lastly, we propose to optimize the following linear objective function:

$$\min \sum_{i \leq n} \sum_{j \leq k} \chi_{ij} \tag{13.43}$$

instead of the quadratic form in Eq. (13.42), so as to be able to use a MILP solver. We remark that this last step is wholly heuristic. In the worst case, it may find clusterings which are arbitrarily different from the optima of the MSSC problem.

13.3.2.1.2 The ℓ_1 norm Similarly to Sect. 13.3.2.1.1, we are going to replace $\|p_i - y_j\|_2^2$ in MSSC by $\|p_i - y_j\|_1$ for each $i \leq n$, $j \leq k$, and compute P^U with the ℓ_1 norm. Again, we get to a variant of Eq. (13.38) where the (convex) nonlinear constraints are:

$$\|p_i - y_j\|_1 \leq \chi_{ij} + P^U(1 - x_{ij}) \tag{13.44}$$

for all $i \leq n, j \leq k$. This is equivalent to

$$\sum_{\ell \leq m} |p_{i\ell} - y_{j\ell}| \leq \chi_{ij} + P^U(1 - x_{ij})$$

which, by means of some additional variables $\eta \in \mathbb{R}^{nkm}$ can be reformulated to

$$\begin{array}{rrcl} \forall \ell \leq m & p_{i\ell} - y_{j\ell} & \leq & \eta_{ij\ell} \\ \forall \ell \leq m & p_{i\ell} - y_{j\ell} & \geq & -\eta_{ij\ell} \\ & \sum_{\ell \leq m} \eta_{ij\ell} & \leq & \chi_{ij} + P^U(1 - x_{ij}). \end{array}$$

As in Sect. 13.3.2.1.1, the terms χ_{ij} on the objective function should be squared.

This yields the following approximating reformulation:

$$\left.\begin{array}{rrcl} \min_{\chi,x,y} & \sum_{i \leq n} \sum_{j \leq k} \chi_{ij}^2 & & \\ \forall i \leq n, j \leq k & \chi_{ij} & \leq & P^U x_{ij} \\ \forall i \leq n, j \leq k, \ell \leq m & p_{i\ell} - y_{j\ell} & \leq & \eta_{ij\ell} \\ \forall i \leq n, j \leq k, \ell \leq m & p_{i\ell} - y_{j\ell} & \geq & -\eta_{ij\ell} \\ \forall i \leq n, j \leq k & \sum_{\ell \leq m} \eta_{ij\ell} & \leq & \chi_{ij} + P^U(1 - x_{ij}) \\ \forall j \leq k & \sum_{i \leq n} x_{ij} & \geq & 1 \\ \forall i \leq n & \sum_{j \leq k} x_{ij} & = & 1 \\ \forall i \leq n, j \leq k & \chi_{ij} & \geq & 0 \\ \forall j \leq k & y_j & \in & [y^L, y^U] \\ & x & \in & \{0,1\}^{nk}. \end{array}\right\} \tag{13.45}$$

We again obtain a cMINLP where the only nonlinearities are in the objective function. The same comment about heuristically optimizing Eq. (13.43) with a MILP solver holds.

13.3.2.2 Approximation guarantees

13.3.3 Proposition
If χ^*, x^*, y^* is a global optimum of Eq. (13.42) (resp. Eq. (13.45)), then the globally optimal objective function value $\sum_{i,j}(\chi^*_{ij})^2$ is equal to the globally minimal value of $f(\mathcal{C})$ (see Eq. (13.30)) with the ℓ_2 norm replaced by the ℓ_∞ norm (resp. the ℓ_1 norm).

Proof. If p_i is not assigned to the cluster C_j in Eq. (13.30), then $x_{ij} = 0$ and $\chi^*_{ij} = 0$ by the optimization direction and Eq. (13.41) (resp. Eq. (13.44)). For the same reasons, if p_i is assigned to C_j then $x_{ij} = 1$ and $\chi^*_{ij} = \|p_i - y_j\|_\infty$ (resp. $\chi^*_{ij} = \|p_i - y_j\|_1$). □ □

By Eq. (13.39)-(13.40) and Prop. 13.3.3, we have:

$$\begin{array}{rrcl} & \mathsf{val}(13.42) \leq & \mathsf{val}(\mathsf{MSSC}) & \leq \mathsf{val}(13.45) \\ \frac{1}{m}\mathsf{val}(13.45) \leq & \mathsf{val}(\mathsf{MSSC}) & \leq m\,\mathsf{val}(13.42), \end{array} \tag{13.46}$$

whence

$$\max\left(\mathsf{val}(13.42), \frac{1}{m}\mathsf{val}(13.45)\right) \leq \mathsf{val}(\mathsf{MSSC}) \leq \min\left(\mathsf{val}(13.45), m\,\mathsf{val}(13.42)\right) \tag{13.47}$$

and

$$\frac{1}{m}\mathsf{val}(\mathsf{MSSC}) \leq \mathsf{val}(13.42) \leq \mathsf{val}(13.45) \leq m\,\mathsf{val}(\mathsf{MSSC}). \tag{13.48}$$

Eq. (13.47)-(13.48) will be useful mostly for cases where m is fixed and small (e.g. clustering in the plane).

13.3.3 Randomly projected formulations

In describing the application of RPs to the MSSC, we note that we do not use k, as usual, for the projected dimension, since k is used in k-means to indicate the number of clusters of the MSSC. Instead, we denote by d the projected dimension.

Assume m is large. We pre-multiply the vector set P (seen as an $m \times n$ matrix) by a $d \times m$ RP matrix $T = (T_{h\ell})$, where $d = O(\frac{1}{\epsilon^2}\ln n)$ and $\epsilon > 0$ appears in the approximation guarantee in Eq. (13.1). We obtain the same formulations and reformulations as above, with p_i replaces by Tp_i, as well as y_j replaced by Ty_j. While P is given, so TP can be computed, Ty_j depends on a decision variable. More precisely,

$$\forall j \leq k \quad Ty_j = \left(\sum_{\ell \leq m} T_{h\ell} y_{j\ell} \mid h \leq d\right)^\top.$$

We employ an additional variable matrix $z \in \mathbb{R}^{kd}$, replace Ty_j by z_j, and relax the defining constraints $z_j = Ty_j$, yielding projected versions of MSSC, Eq. (13.38), Eq. (13.42), Eq. (13.45) where $p_i - y_j \in \mathbb{R}^m$ is replaced by $Tp_i - z_j \in \mathbb{R}^d$. If $m \gg 1$ and $d = O(\ln n)$, these projected reformulations have considerably fewer variables than their original counterparts.

In the rest of this section we only consider the MSSC formulation (i.e. Eq. (13.34)), since the others follow by exact or approximate reformulation operations, whether we use P or TP as data. We denote the randomly projected MSSC formulation by TMSSC.

13.3.3.1 Applicability of the JLL

An alternative to invoking [54] would be to simply rely on the approximation of Euclidean distances given by the JLL [152], which would directly apply to Eq. (13.32). We pursue a different critique for this alternative.

The symbols y_j appearing in Eq. (13.32) are decision variables ranging in continuous space. As such, they represent a potentially uncountable infinity of vectors. The JLL, however, only applies to finite subsets of vectors.

Observe, however, that there are only as many centroids as there are possible different partitions of n entities in k clusters — so the number of vectors assigned to the y variables may not be known in advance, but it is not infinite. In the worst case, there are B_n partitions of n elements, where B_n is the n-th Bell number [36]. It turns out that B_n grows like a product of two exponentials in n [200, §1.14, Problem 9], and that $\ln B_n$ behaves asymptotically like $O(n \ln n)$ (plus smaller terms) [97, p. 108]. Thus, the JLL would yield $d = O(\ln B_n/\varepsilon^2) = O(n \ln n/\varepsilon^2)$. This would only be useful in cases where $n \ll m$, which are a minority.

We also note that the k-means algorithm is not expected to run into the worst case from complete enumeration: since k-means is a heuristic, it is expected to reach termination reasonably quickly, after having examined only a few indicator matrices. This also holds for heuristic utilizations of exact algorithms, such as e.g. BB algorithms with a time or iteration limit. The JLL therefore provides a simple and valid analysis for such cases, which are the cases we actually test in practice in this paper.

13.3.3.2 The additive JLL for infinite sets

In this section we give a formulation-based analysis of the applicability of the JLL to the MSSC which is independent of the algorithm used (a similar result was also proved in [54, Thm. 1], but the proofs therein are kept very short, at the expense of clarity). In view of Sect. 13.3.3.1 we do not employ the JLL, but a similar result that also applies to infinite sets, namely the *additive JLL* (see Thm. 13.3.4 below).

We first introduce two quantities, the sub-Gaussian norm of a sub-Gaussian r.v., and the Gaussian width of a set. A r.v. X is *sub-Gaussian* iff there exists a constant K such that:

$$\forall t \geq 0 \quad \mathsf{Prob}(|X| \geq t) \leq 2e^{-(t/K)^2}.$$

We denote the *sub-Gaussian norm* of X by

$$\|X\|_{\psi_2} = \inf\{t > 0 \mid \mathsf{E}(e^{(X/t)^2}) \leq 2\}.$$

Given a set $S \subseteq \mathbb{R}^m$, the *Gaussian width* of S is

$$w(T) = \mathsf{E}\{\sup_{x \in S} \langle g, x\rangle \mid g \sim \mathsf{N}(0, I_m)\},$$

where the expectation is computed over all multivariate standard normal samples g.

13.3.4 Theorem
Let $S \subset \mathbb{R}^m$, and consider a $d \times m$ matrix T' having independent isotropic sub-Gaussian random vectors T'_i for rows, $K = \max_{h \leq d} \|T'_i\|_{\psi_2}$, and $T = \frac{1}{\sqrt{d}} T'$. Then, with high probability, there exists a universal constant κ such that:

$$\forall x, y \in S \quad \|x - y\|_2 - \delta \leq \|Tx - Ty\|_2 \leq \|x - y\|_2 + \delta, \tag{13.49}$$

where $\delta = \frac{\kappa K^2 w(S)}{\sqrt{d}}$.

Proof. See [298, Prop. 9.3.2]. □

Our treatment follows the same logical order as the results in [262, 54]. Each of the results below corresponds to a result in [262, 54]. We adapted the proofs to employ the additive RP T satisfying Thm. 13.3.4 instead of the standard JLL. We first show that T preserves inner products approximately.

13.3.5 Corollary
For every $x, y \in \mathbb{R}^m$, the following holds with high probability:

$$|\langle Tx, Ty\rangle - \langle x, y\rangle| \leq \frac{\delta}{2} \|x\|_2 \|y\|_2.$$

Proof. We observe that the parallelogram rule and Eq. (13.49) imply that, with high probability, for any $x, y \in \mathbb{R}^m$,

$$\begin{aligned} 4\langle Tx, Ty\rangle &= \|Tx + Ty\|_2^2 - \|Tx - Ty\|_2^2 \\ &\geq \|x + y\|_2^2 - \delta - \delta - \|x - y\|_2^2 \\ &= 4\langle x, y\rangle - 2\delta. \end{aligned}$$

Therefore, $\langle Tx, Ty\rangle - \langle x, y\rangle \geq -\frac{\delta}{2}$. Analogously, we obtain $\langle Tx, Ty\rangle - \langle x, y\rangle \leq \frac{\delta}{2}$. Thus,

$$|\langle Tx, Ty\rangle - \langle x, y\rangle| \leq \frac{\delta}{2}. \tag{13.50}$$

Since T is linear, we have

$$\langle Tx, Ty\rangle = \|x\|_2 \|y\|_2 \langle T(x/\|x\|_2), T(y/\|y\|_2)\rangle. \tag{13.51}$$

The result follows by replacement of Eq. (13.51) in Eq. (13.50). □

13.3.6 Lemma
For every $x, y \in \mathbb{R}^m$, the following hold:

(i) $\mathsf{E}(\langle Tx, Ty\rangle) = \langle x, y\rangle$;

(ii) $\mathsf{Var}(\langle Tx, Ty\rangle) \leq \frac{\delta^2+4\delta}{4}\|x\|_2^2\|y\|_2^2$.

Proof. We recall that $T = \frac{1}{\sqrt{d}}T'$, thus

$$\langle Tx, Ty\rangle = \frac{1}{d}\langle T'x, T'y\rangle. \tag{13.52}$$

Moreover,

$$\left.\begin{aligned} T'x &= \big(\langle T'_1, x\rangle, \ldots, \langle T'_d, x\rangle\big) \\ T'y &= \big(\langle T'_1, y\rangle, \ldots, \langle T'_d, y\rangle\big). \end{aligned}\right\}$$

Thus, we can write

$$\langle T'x, T'y\rangle = \langle T'_1, x\rangle\langle T'_1, y\rangle + \cdots + \langle T'_d, x\rangle\langle T'_d, y\rangle. \tag{13.53}$$

We first prove (i) for $x = y$. In this case Eq. (13.53) becomes

$$\langle T'x, T'x\rangle = \langle T'_1, x\rangle^2 + \cdots + \langle T'_d, x\rangle^2.$$

Therefore,

$$\begin{aligned} \mathsf{E}(\langle T'x, T'x\rangle) &= \mathsf{E}(\langle T'_1, x\rangle^2) + \cdots + \mathsf{E}(\langle T'_d, x\rangle^2) \\ &= \|x\|_2^2 + \cdots + \|x\|_2^2 \\ &= d\|x\|_2^2, \end{aligned}$$

where we have used the isotropy (see Sect. 12.6.4) of the random vectors T'_i. From Eq. (13.52) we obtain

$$\mathsf{E}(\langle Tx, Tx\rangle) = \langle x, x\rangle. \tag{13.54}$$

In order to prove (i) for generic $x, y \in \mathbb{R}^m$, it is enough to apply Eq. (13.54) to $x - y$ and use the linearity of expectation. For (ii), we first set $X = \langle Tx, Ty\rangle - \mathsf{E}(\langle Tx, Ty\rangle) = \langle Tx, Ty\rangle - \langle x, y\rangle$. The properties of the variance and (i) give

$$\mathsf{E}(X^2) = \mathsf{Var}(X) + \mathsf{E}(X)^2 = \mathsf{Var}(\langle Tx, Ty\rangle).$$

Moreover,

$$\begin{aligned} \mathsf{E}(X^2) &= \mathsf{E}(\langle Tx, Ty\rangle^2 + \langle x, y\rangle^2 - 2\langle Tx, Ty\rangle\langle x, y\rangle) \\ &= \mathsf{E}(\langle Tx, Ty\rangle^2) + \langle x, y\rangle^2 - 2\langle x, y\rangle^2 \\ &= \mathsf{E}(\langle Tx, Ty\rangle^2) - \langle x, y\rangle^2. \end{aligned}$$

From Cor. 13.3.5 we have

$$\langle Tx, Ty\rangle^2 \leq \langle x, y\rangle^2 + \frac{\delta}{4}\|x\|_2^2\|y\|_2^2 + \delta\langle x, y\rangle\|x\|_2\|y\|_2. \tag{13.55}$$

Exploiting Eq. (13.55) and $\langle x, y\rangle \leq \|x\|_2\|y\|_2$ we obtain

$$\begin{aligned} \mathsf{Var}(\langle Tx, Ty\rangle) &= \mathsf{E}(X^2) = \mathsf{E}(\langle Tx, Ty\rangle^2) - \langle x, y\rangle^2 \\ &\leq \mathsf{E}(\langle x, y\rangle^2 + \tfrac{\delta}{4}\|x\|_2^2\|y\|_2^2 + \delta\langle x, y\rangle\|x\|_2\|y\|_2) - \langle x, y\rangle \\ &= \langle x, y\rangle^2 + \tfrac{\delta}{4}\|x\|_2^2\|y\|_2^2 + \delta\langle x, y\rangle\|x\|_2\|y\|_2 - \langle x, y\rangle \\ &\leq \tfrac{\delta^2}{4}\|x\|_2^2\|y\|_2^2 + \delta\|x\|_2^2\|y\|_2^2 \\ &= \tfrac{\delta^2+4\delta}{4}\|x\|_2^2\|y\|_2^2, \end{aligned}$$

which concludes the proof. □

Following [262], we can use Cor. 13.3.5 and Lemma 13.3.6 to prove that the RP T can be used to approximate the product of two matrices.

13.3.7 Lemma
Let $A \in \mathbb{R}^{r\times m}$ and $B \in \mathbb{R}^{m\times s}$ be two matrices. Then the following hold:

(a) $\|AB - AT^\top TB\|_F \le \frac{\delta}{2}\|A\|_F\|B\|_F$ *whp;*

(b) $\mathsf{E}(AT^\top TB) = AB$;

(c) $\mathsf{E}(\|AB - AT^\top TB\|_F^2) \le \frac{\delta^2+4\delta}{4}\|A\|_F^2\|B\|_F^2$.

Proof. Set $a_i = A_i$ and $b_j = B^j$. Then,

$$(AT^\top)_i = Ta_i, \quad (TB)^j = TB_j.$$

Let $Y_{ij} = (AB)_{ij} - (AT^\top TB)_{ij} = \langle a_i, b_j\rangle - \langle Ta_i, Tb_j\rangle$, then from Cor. 13.3.5 we have that, whp,

$$|Y_{ij}| \le \frac{\delta}{2}\|a_i\|_2\|b_j\|_2.$$

Hence,

$$\|AB - AT^\top TB\|_F^2 = \sum_{i,j} Y_{ij}^2 \le \sum_{ij}\frac{\delta^2}{4}\|a_i\|_2^2\|b_j\|_2^2 = \frac{\delta^2}{4}\|A\|_F^2\|B\|_F^2,$$

which proves (a). Applying Lemma 13.3.6 on the random variable Y_{ij} we obtain $\mathsf{E}(Y_{ij}) = 0$ for all i, j and thus (b). Moreover, $\mathsf{E}(Y_{ij}^2) = \mathsf{Var}(\langle Ta_i, Tb_j\rangle)$ and, from Lemma 13.3.6 we obtain

$$\mathsf{E}(\|AB - AT^\top TB\|_F^2) = \sum_{i,j}\mathsf{E}(Y_{ij}^2) \le \frac{\delta^2+4\delta}{4}\|A\|_F^2\|B\|_F^2,$$

which concludes the proof. □

Moreover, the RP T almost preserves the Frobenius norm of any matrix, in the sense of the next result.

13.3.8 Lemma
Let $C \in \mathbb{R}^{m\times n}$ be a matrix and $X = \|TC\|_F^2$. Then we have (i) $\mathsf{E}(X) = \|C\|_F^2$ and (ii) $\mathsf{Var}(X) \le \frac{2}{d}\|C\|_F^4$.

Proof. We prove (i) first. We recall that $T = \frac{1}{\sqrt{d}}T'$, where the rows of T' are isotropic sub-Gaussian random vectors. Let $Y_i = \|T_i'C\|_2^2$, then

$$X = \sum_{i=1}^{d}\frac{1}{d}Y_i.$$

Using the isotropy of T_i', we obtain

$$\begin{aligned}\mathsf{E}(Y_i) = \mathsf{E}(\|T_i'C\|_2^2) &= \mathsf{E}(\textstyle\sum_j \langle T_i', C^j\rangle^2)\\ &= \textstyle\sum_j \mathsf{E}(\langle T_i', C^j\rangle^2)\\ &= \textstyle\sum_j \|C^j\|_2^2\\ &= \|C\|_F^2.\end{aligned}$$

Thus,

$$\mathsf{E}(X) = \mathsf{E}(\sum_{i=1}^{d}\frac{1}{d}Y_i) = \sum_{i=1}^{d}\frac{1}{d}\mathsf{E}(Y_i) = \|C\|_F^2.$$

As for (ii), from Lemma 13.3.6 and $\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$ we have

$$\mathsf{E}((\langle T_i', x\rangle\langle T_i', y\rangle)^2) \le 3\|x\|_2^2\|y\|_2^2 \quad \forall x, y \in \mathbb{R}^m.$$

Thus,

$$\begin{aligned}\mathsf{E}(Y_i^2) = \mathsf{E}((\textstyle\sum_j \langle T_i', C^j\rangle^2)^2) &= \textstyle\sum_{j,k} \mathsf{E}((\langle T_i', C^j\rangle\langle T_i', C^k\rangle)^2) \\ &\leq 3\textstyle\sum_{j,k} \|C^j\|_2^2\|C^k\|_2^2 = 3\|C\|_F^4\end{aligned}$$

Thus $\mathsf{Var}(Y_i) = \mathsf{E}(Y_i^2) - \mathsf{E}(Y_i)^2 \leq 2\|C\|_F^4$ and we obtain

$$\mathsf{Var}(X) = \mathsf{Var}(\sum_{i=1}^{d} \frac{1}{d} Y_i) = \sum_{i=1}^{d} \frac{1}{d^2}\mathsf{Var}(Y_i) \leq \frac{2}{d}\|C\|_F^4,$$

which concludes the proof. □

The next result corresponds to [262, Cor. 11], where T, however, is the RP in Thm. 13.3.4, which can be applied to possibly infinite sets $S \subset \mathbb{R}^m$.

13.3.9 Corollary
Let $U \in \mathbb{R}^{m\times k}$ be a unitary matrix. Then, the following holds with high probability:

$$|1 - \sigma_i(TU)| \leq \delta \quad \forall i \leq k. \tag{13.56}$$

Proof. The singular values $\sigma_i(TU)$ are the square roots of the eigenvalues λ_i of the matrix $U^\top T^\top TU$. Let x be the unitary eigenvector corresponding to λ_i for a generic $i \in \{1, \ldots, k\}$. Then,

$$\begin{aligned}\|TUx\|_2 &= \sqrt{\langle TUx, TUx\rangle} = \sqrt{(TUx)^\top (TUx)} \\ &= \sqrt{x^\top U^\top T^\top TUx} = \sqrt{x^\top \lambda_i x} \\ &= \sqrt{\lambda_i}\|x\|_2 = \sqrt{\lambda_i}.\end{aligned}$$

Therefore, Eq. (13.56) is equivalent to

$$|1 - \|TUx\|_2| \leq \delta. \tag{13.57}$$

In order to prove Eq. (13.57), since $\|x\|_2 = 1$ and U is a unitary matrix, we have $\|Ux\|_2 = 1$. Thus, if the set S in Thm. 13.3.4 contains the points

$$\{Ux \mid x \text{ unitary eigenvector of } U^\top T^\top TU\},$$

we have, with high probability,

$$|1 - \|TUx\|_2| = |\|Ux\|_2 - \|TUx\|_2| \leq \delta,$$

which concludes the proof. □

We can now prove results corresponding to [54, Lemmata 3-5] for the RP T satisfying Eq. (13.49).

13.3.10 Lemma
Let $P \in \mathbb{R}^{m\times n}$ be the matrix representing the points we want to cluster and $P_k = U_k\Sigma_k {V_k}^\top$ its SVD of rank k. Then, whp,

$$\|(TU_k)^+ - (TU_k)^\top\|_2 \leq \frac{2-\delta^2}{1-\delta}.$$

Proof. Let $\Phi = TU_k$ and $\Phi = U_\Phi \Sigma_\Phi {V_\Phi}^\top$ be its SVD. If we consider the SVD of Φ^+ and $\Phi^\top$ and the fact that the spectral norm of a matrix is invariant with respect to unitary matrices, we obtain

$$\begin{aligned}\|(TU_k)^+ - (TU_k)^\top\|_2 &= \|V_\Phi \Sigma_\Phi^{-1} {U_\Phi}^\top - V_\Phi \Sigma_\Phi {U_\Phi}^\top\|_2 \\ &= \|V_\Phi(\Sigma_\Phi^{-1} - \Sigma_\Phi){U_\Phi}^\top\|_2 \\ &= \|\Sigma_\Phi^{-1} - \Sigma_\Phi\|_2.\end{aligned}$$

Now, let $\Psi = \Sigma_\Phi^{-1} - \Sigma_\Phi$, σ_i be the i-th singular value of Φ and τ_i the i-th entry of Ψ. Then, a simple computation shows that

$$\tau_i = \frac{1 - \sigma_i \sigma_{k+1-i}}{\sigma_{k+1-i}}.$$

From Cor. 13.3.9 we know that $1 - \delta \leq \sigma_i \leq 1 + \delta$, for $i = 1, \ldots, k$. Therefore,

$$\tau_1 = \frac{1}{\sigma_{k+1-i}} - \sigma_i \leq \frac{1}{1-\delta} + 1 + \delta = \frac{2 - \delta^2}{1 - \delta}.$$

Since Ψ is diagonal we have $\|\psi\|_2 = \max_i \tau_i$, which concludes the proof. □

13.3.11 Lemma
Let $C \in \mathbb{R}^{m \times n}$, then, whp,

$$\|TC\|_F \leq \sqrt{1+\delta}\|C\|_F.$$

Proof. Let $Z = \|TC\|_F^2$, from Lemma 13.3.8 we have $\mathsf{E}(Z) = \|C\|_F^2$ and $\mathsf{Var}(Z) \leq \frac{2}{d}\|C\|_F^4$. Applying the Chebyshev inequality to the r.v. Z we obtain

$$\mathsf{Prob}\Big(\big(\big)|Z - \mathsf{E}(Z)| \geq \delta\|C\|_F^2\Big) \leq \frac{\mathsf{Var}(Z)}{\delta^2\|C\|_F^4} \leq \frac{2}{d\delta}.$$

Hence,

$$\mathsf{Prob}\Big(\big(\big)Z \geq (1+\delta)\|C\|_F^2\Big) \leq \frac{2}{d\delta} \Leftrightarrow \mathsf{Prob}\Big(\big(\big)\sqrt{Z} \geq \sqrt{1+\delta}\|C\|_F\Big) \leq \frac{2}{d\delta},$$

which concludes the proof. □

13.3.12 Lemma
The following holds with high probability:

$$P_k = U_k(TU_k)^+TP + E,$$

where $E \in \mathbb{R}^{m \times n}$ satisfies $\|E\|_F \leq f_1(\delta)\|P - P_k\|$, $f_1(\delta) = \frac{\delta}{2} + \sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}$.

Proof. Define $E = P_k - U_k(TU_k)^+TP$. We want to prove that

$$\|E\|_F \leq f_1(\delta)\|P - P_k\|.$$

Let $\rho = \mathsf{rk}(\big)P)$, then we can write $P = P_k + P_{\rho-k}$, where $P_{\rho-k} = P - P_k$. Replacing P with $P_k + P_{\rho-k}$ in the definition of E we have

$$\begin{aligned} E &= P_k - U_k(TU_k)^+T(P_k + P_{\rho-k}) \\ &= (P_k - U_k(TU_k)^+TP_k) + (U_k(TU_k)^+TP_{\rho-k}). \end{aligned}$$

If we consider the Frobenius norm and we use the triangular inequality we get

$$\|E\|_F \leq \|P_k - U_k(TU_k)^+TP_k\|_F + \|U_k(TU_k)^+TP_{\rho-k}\|_F. \tag{13.58}$$

In the first term appearing in the second member of Eq. (13.58) we can replace $P_k = U_k\Sigma_k{V_k}^\top$ and exploit the fact that $(TU_k)^+(TU_k) = I$, to obtain

$$\begin{aligned} \|P_k - U_k(TU_k)^+TP_k\|_F &= \|U_k\Sigma_k{V_k}^\top - U_k(TU_k)^+TU_k\Sigma_k{V_k}^\top\| \\ &= \|U_k\Sigma_k{V_k}^\top - U_k\Sigma_k{V_k}^\top\| = 0. \end{aligned}$$

It remains to bound the second term in the second member of Eq. (13.58). We add and subtract $U_k(TU_k)^\top TP_{\rho-k}$, we use the fact that the Frobenius norm is invariant with respect to unitary matrices,

as well as the strong sub-multiplicativity of every pair of matrices X, Y of appropriate dimensions, namely that $\|XY\|_F \leq \|X\|_F \|Y\|_2$. We then obtain

$$\left.\begin{array}{rl} & \|U_k(TU_k)^+ TP_{\rho-k} - U_k(TU_k)^\top TP_{\rho-k} + U_k(TU_k)^\top TP_{\rho-k}\|_F \\ \leq & \|U_k(TU_k)^\top TP_{\rho-k}\|_F + \|U_k((TU_k)^+ - (TU_k)^\top)TP_{\rho-k}\|_F \\ \leq & \|(TU_k)^\top TP_{\rho-k}\|_F + \|TP_{\rho-k}\|_F \|(TU_k)^+ - (TU_k)^\top\|_2 \\ = & \|U_k{}^\top T^\top TP_{\rho-k}\|_F + \|TP_{\rho-k}\|_F \|(TU_k)^+ - (TU_k)^\top\|_2. \end{array}\right\} \tag{13.59}$$

We bound the last three terms in Eq. (13.59) separately.

1. For $\|U_k{}^\top T^\top TP_{\rho-k}\|_F$, we observe that

$$U_k{}^\top P_{\rho-k} = U_k{}^\top U_{\rho-k}\Sigma_{\rho-k}V_{\rho-k}{}^\top = \mathbf{0}\Sigma_{\rho-k}V_{\rho-k}{}^\top = \mathbf{0},$$

since U_k and $U_{\rho-k}$ have orthogonal columns by definition. Therefore, we can apply Lemma 13.3.7 (a) and obtain, whp,

$$\|U_k{}^\top T^\top TP_{\rho-k}\|_F = \|U_k{}^\top P_{\rho-k} - U^\top T^\top TP_{\rho-k}\|_F \leq \frac{\delta}{2}\|U_k{}^\top\|_F \|P_{\rho-k}\|_F = \frac{\delta}{2}\|P_{\rho-k}\|_F.$$

2. In order to bound $\|TP_{\rho-k}\|_F$, it is enough to apply Lemma 13.3.11 with $C = P_{\rho-k}$. This yields

$$\|TP_{\rho-k}\|_F \leq \sqrt{1+\delta}\|P_{\rho-k}\|_F.$$

3. As concerns $\|(TU_k)^+ - (TU_k)^\top\|_2$, we can apply Lemma 13.3.10 and obtain, with high probability,

$$\|(TU_k)^+ - (TU_k)^\top\|_2 \leq \frac{2-\delta^2}{1-\delta}.$$

Putting these bounds together we obtain:

$$\begin{array}{rl} \|E\|_F & \leq \|U_k{}^\top T^\top TP_{\rho-k}\|_F + \|TP_{\rho-k}\|_F \|(TU_k)^+ - (TU_k)^\top\|_2 \\ & \leq \frac{\delta}{2}\|P_{\rho-k}\|_F + \sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\|P_{\rho-k}\|_F \\ & = \left(\frac{\delta}{2} + \sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\right)\|P_{\rho-k}\|_F \\ & = \left(\frac{\delta}{2} + \sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\right)\|P - P_k\|_F, \end{array}$$

which concludes the proof. □

13.3.13 Remark
Let $X_{\mathsf{opt}} = \underset{X\in\mathcal{X}}{\arg\min}\|P^\top - XX^\top P^\top\|_2^2$, *where* $\mathcal{X}$ *is the set of all cluster indicator matrices. Since* $X_{\mathsf{opt}}X_{\mathsf{opt}}{}^\top P^\top$ *is a matrix with rank at most* k, *the Eckart-Young Theorem implies*

$$\|P_{\rho-k}{}^\top\|_F^2 = \|P^\top - P_k^\top\|_F^2 \leq \|P^\top - X_{\mathsf{opt}}X_{\mathsf{opt}}{}^\top P^\top\|_F^2.$$

We are finally ready to state and prove the equivalent of [54, Thm. 1] for the RP T satisfying Eq. (13.49).

13.3.14 Theorem
Let $P \in \mathbb{R}^{m\times n}$ *be the matrix representing the points we want to cluster,* k *the number of clusters and* $T \in \mathbb{R}^{d\times m}$ *the RP satisfying Thm. 13.3.4. Assume that we have access to an algorithm which takes as input* P, k, T *and returns a cluster indicator matrix* X_γ *which satisfies, with high probability*

$$\|(TP)^\top - X_\gamma X_\gamma{}^\top (TP)^\top\|_F^2 \leq \gamma \min_{X\in\mathcal{X}}\|(TP)^\top - XX^\top(TP)^\top\|_F^2,$$

where $\mathcal{X}$ is the set of all cluster indicator matrices and $\gamma \geq 1$. Then, with high probability, we have

$$\|P^\top - X_\gamma {X_\gamma}^\top P^\top\|_F^2 \leq f(\delta,\gamma)\|P^\top - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top P^\top\|_F^2, \tag{13.60}$$

where $X_{\mathsf{opt}} = \underset{X\in\mathcal{X}}{\arg\min}\, \|P^\top - XX^\top P^\top\|_F^2$ *and*

$$f(\delta,\gamma) = \left(1 + \frac{(2-\delta^2)(\sqrt{\gamma}+1)\sqrt{1+\delta}}{1-\delta} + \frac{\delta}{2}\right)^2.$$

Proof. In Eq. (13.60) we replace $P = P_k + P_{\rho-k}$ and we use the fact that ${P_k}^\top - X_\gamma {X_\gamma}^\top {P_k}^\top$ and ${P_{\rho-k}}^\top - X_\gamma {X_\gamma}^\top {P_{\rho-k}}^\top$ generate orthogonal subspaces to obtain

$$\left.\begin{array}{rl} & \|P^\top - X_\gamma {X_\gamma}^\top P^\top\|_F^2 \\ = & \|{P_k}^\top - X_\gamma {X_\gamma}^\top {P_k}^\top\|_F^2 + \|{P_{\rho-k}}^\top - X_\gamma {X_\gamma}^\top {P_{\rho-k}}^\top\|_F^2 \\ = & \|(I - X_\gamma {X_\gamma}^\top){P_k}^\top\|_F^2 + \|(I - X_\gamma {X_\gamma}^\top){P_{\rho-k}}^\top\|_F^2. \end{array}\right\} \tag{13.61}$$

We can bound the second term in the second member of Eq. (13.61) using the fact that $I - X_\gamma {X_\gamma}^\top$ is a projection matrix and the Frobenius norm does not increase if we drop a projection matrix

$$\|(I - X_\gamma {X_\gamma}^\top){P_{\rho-k}}^\top\|_F^2 \leq \|{P_{\rho-k}}^\top\|_F^2 \leq \|P^\top - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top P^\top\|_F^2, \tag{13.62}$$

where we used Remark 13.3.13 in the last inequality. We now bound the term $\|(I - X_\gamma {X_\gamma}^\top){P_k}^\top\|_F^2$. Using Lemma 13.3.12, the triangular inequality and the fact that $I - X_\gamma {X_\gamma}^\top$ is a projection matrix we get

$$\left.\begin{array}{rl} \|(I - X_\gamma {X_\gamma}^\top){P_k}^\top\|_F & \leq \|(I - X_\gamma {X_\gamma}^\top)(U_k(TU_k)^+TP)^\top\|_F + \|E^\top\|_F \\ & = \|(I - X_\gamma {X_\gamma}^\top)(TP)^\top((TU_k)^+)^\top\|_F + \|E\|_F, \end{array}\right\}$$

where the last member is obtained from the fact that the Frobenius norm is invariant with respect to unitary matrices and that $\|A^\top\|_F = \|A\|_F$ holds for every matrix A. Now we can apply strong sub-multiplicativity:

$$\left.\begin{array}{r} \|(I - X_\gamma {X_\gamma}^\top)(TP)^\top((TU_k)^+)^\top\|_F + \|E\|_F \\ \leq \|(I - X_\gamma {X_\gamma}^\top)(TP)^\top\|_F\|(TU_k)^+\|_2 + \|E\|_F. \end{array}\right\}$$

From the construction of X_γ we have that

$$\|(I - X_\gamma {X_\gamma}^\top)(TP)^\top\|_F \leq \sqrt{\gamma}\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)(TP)^\top\|_F.$$

Thus, applying Lemma 13.3.10, Lemma 13.3.12 and Remark 13.3.13 we obtain

$$\left.\begin{array}{rl} & \|(I - X_\gamma {X_\gamma}^\top)(TP)^\top\|_F\|(TU_k)^+\|_2 + \|E\|_F \\ \leq & \sqrt{\gamma}\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)(TP)^\top\|_F\frac{2-\delta^2}{1-\delta} \\ + & \left(\frac{\delta}{2} + \sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\right)\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\|_F. \end{array}\right\} \tag{13.63}$$

Now we observe that

$$\begin{array}{rl} & \|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)(TP)^\top\|_F \\ = & \|\left((I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top T^\top\right)^\top\|_F \\ = & \|T\left((I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\right)^\top\|_F. \end{array}$$

Therefore, we can apply Lemma 13.3.11 with $C = \left((I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\right)^\top$ and obtain

$$\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)(TP)^\top\|_F \leq \sqrt{1+\delta}\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\|_F. \tag{13.64}$$

Replacing Eq. (13.64) in Eq. (13.63) we get

$$\left.\begin{array}{rl} & \|(I - X_\gamma {X_\gamma}^\top){P_k}^\top\|_F \\ \leq & \sqrt{\gamma}\sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\|_F \\ + & \left(\frac{\delta}{2} + \sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\right)\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\|_F \\ = & \left(\sqrt{1+\delta}\frac{2-\delta^2}{1-\delta}\right)\left(\sqrt{\gamma}+1\right) + \frac{\delta}{2}\|(I - X_{\mathsf{opt}}{X_{\mathsf{opt}}}^\top)P^\top\|_F. \end{array}\right\} \tag{13.65}$$

Finally, putting Eq. (13.62) and Eq. (13.65) together we obtain:

$$\left.\begin{aligned} & \|P^\top - X_\gamma {X_\gamma}^\top P^\top\|_F^2 \\ \leq \; & \left(1 + \left(\sqrt{1+\delta}\,\tfrac{2-\delta^2}{1-\delta}\right)\left(\sqrt{\gamma}+1\right) + \tfrac{\delta}{2}\right)^2 \|P^\top - X_{\mathsf{opt}} {X_{\mathsf{opt}}}^\top P^\top\|_F^2, \end{aligned}\right\}$$

which concludes the proof. □

Chapter 14

The kissing number problem

The KNP was introduced in Sect. 2.2.7.10 (also see Appendix B).

> KISSING NUMBER PROBLEM (KNP). Given two integers n, K, determine whether n unit spheres can be arranged around a central unit sphere centered at the origin in $\mathbb{R}^K$, in such a way that (a) all of the surrounding spheres intersect the central sphere in exactly one point, and (b) all pairs of surrounding spheres intersect in at most one point.

The KNP, as given above, is expressed as a decision problem. The optimization version asks to maximize the number of surrounding spheres in $\mathbb{R}^K$. The maximum number n of surrounding spheres is called the *kissing number* in dimension K and denoted $\mathsf{kn}(K)$. Two kissing configurations in 2D and 3D are shown in Fig. 14.1.

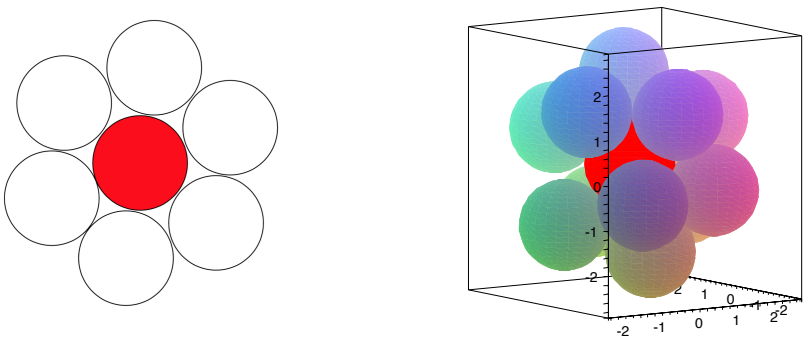


Figure 14.1: Kissing configurations in 2D (left) and 3D (right).

14.1 Basic formulations

Fig. 14.2 (left) shows a geometrical intuition in the case $K = 2$. The corresponding MP formulation is a

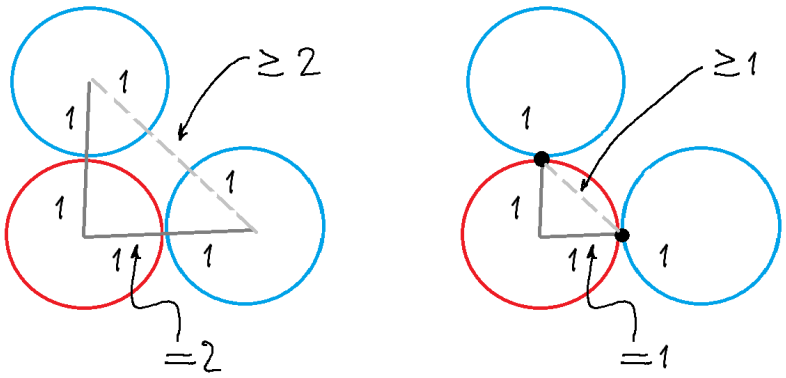


Figure 14.2: Center formulation (left) and point of contact formulation (right).

pure feasibility nonconvex QCP:

$$\begin{array}{rrcl} \forall i \leq n & \|x_i\|_2^2 & = & 4 \\ \forall i < j \leq n & \|x_i - x_j\|_2^2 & \geq & 4, \end{array}$$

where $x_i \in \mathbb{R}^K$ is a vector of decision variables describing the center of the i-th surrounding sphere, for each $i \leq n$.

In fact, it suffices to decide the contact points of each surrounding sphere rather than its center (Fig. 14.2, right), which yields the formulation:

$$\begin{array}{rrcll} \forall i \leq n & \|x_i\|_2^2 & = & 1 & (14.1) \\ \forall i < j \leq n & \|x_i - x_j\|_2^2 & \geq & 1, & (14.2) \end{array}$$

where x_i now describes the point of contact rather than the center of the surrounding sphere.

The equivalence of these two formulations is given by the fact that each point of contact is midway on the segment between the origin and the corresponding sphere center.

14.1.1 In practice

Pure feasibility formulations are usually difficult to solve in practice, since feasibility requires a YES/NO binary answer. Most NLP solvers, on the other hand, have an easier time improving existing feasible solutions than finding some. In particular, all algorithms for finding local optima of NLPs assume the existence of a feasible starting point as input (the implementations of these algorithms usually allow infeasible starting points). See Sect. 2.2.5 for more information.

A more solver-friendly variant of Eq. (14.1)-(14.2) is the following (nonconvex) QCP:

$$\left.\begin{array}{rrcl} \max\limits_{x,\alpha} & \alpha & & \\ \forall i \leq n & ||x_i||^2 & = & 1 \\ \forall i < j \leq n & ||x_i - x_j||^2 & \geq & \alpha \\ \forall i \leq n & x_i & \in & [-1,1]^K \\ & \alpha & \geq & 0, \end{array}\right\} \quad (14.3)$$

where α is a scalar decision variable which relaxes feasibility. Note that α is maximized, so that, if the optimum α^* is at least 1, then the solution x^* of Eq. (14.3) is also feasible in Eq. (14.1)-(14.2). Morover, at global optimality, Eq. (14.3) identifies the positions of the surrounding balls with the pairwise largest minimum separation.

14.2 Spherical codes

The natural generalization of Eq. (14.1)-(14.2) is the concept of a spherical code.

14.2.1 Definition

A K-dimensional spherical z-code is a pair $(z, \mathcal{C})$ where $z \in [-1, 1]$ and $\mathcal{C}$ is a finite subset of the unit sphere $\mathbb{S}^{K-1}$ in $\mathbb{R}^K$, such that, for each $x \neq y \in \mathcal{C}$ we have $\langle x, y \rangle \leq z$.

A spherical code is the continuous equivalent of an error correcting (discrete) code. Each element x of the z-code $\mathcal{C}$ is associated with a subset of the spherical surface which contains all other vectors $u \in \mathbb{S}^{K-1}$ having an angle θ with x such that $\cos\theta > z$. The relation with error correction is that every word of the message is associated to a vector $x \in \mathcal{C}$. After transmission, the recipient receives a vector $u \in \mathbb{S}^{K-1}$ which may not belong to $\mathcal{C}$. On the other hand, it suffices to find the element of $x \in \mathcal{C}$ with smallest angle θ with u: if its cosine exceeds z, then we assume that x is the word which was originally transmitted.

The computational difficulty linked to a K-dimensional z-spherical code $\mathcal{C}$ is its construction. It is not easy to find $n = |\mathcal{C}|$ equally spaced vectors on the surface of $\mathcal{S}^{K-1}$. The MP formulation for constructing $\mathcal{C}$ is as follows.

$$\forall i \leq n \quad \|x_i\|_2^2 = 1 \tag{14.4}$$

$$\forall i < j \leq n \quad \langle x_i, x_j \rangle \leq z. \tag{14.5}$$

14.2.2 Proposition

Eq. (14.1)-(14.2) can be obtained from Eq. (14.4)-(14.5) by setting $z = \frac{1}{2}$.

Proof. Let $x \neq y \in \mathcal{C}$, and θ be the angle between x and y. We want $\cos\theta \leq z$. We note that, since $\|x\|_2^2 = \|y\|_2^2 = 1$, $\cos\theta \leq z$ is equivalent to $\|x - y\|_2^2 \geq (1 - \cos\theta)^2 + \sin^2\theta$ (see Fig. 14.3). Hence,

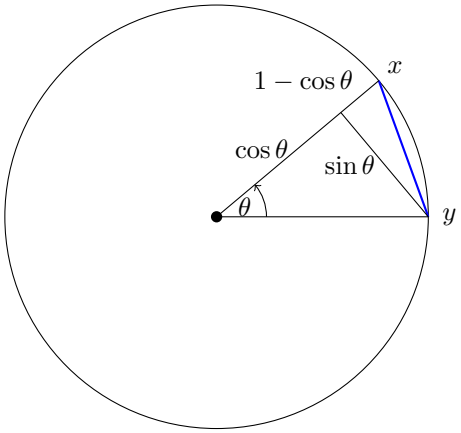


Figure 14.3: The reason why $\|x - y\|_2^2 = (1 - \cos\theta)^2 + \sin^2\theta$.

$$\begin{aligned} \forall i < j \leq n \quad \|x_i - x_j\|_2^2 &= \|x_i\|_2^2 + \|x_j\|_2^2 - 2\,x_i^\top x_j = 1 + 1 - 2\,x_i^\top x_j = 2(1 - x_i^\top x_j) \\ &\geq (1 - \cos\theta)^2 + (\sin\theta)^2 = 1 - 2\cos\theta + \cos^2\theta + \sin^2\theta = 2(1 - \cos\theta), \end{aligned}$$

whence

$$\forall i < j \leq n \quad x_i^\top x_j \leq \cos\theta = z.$$

The result follows since, when $\theta = \pi/3$ (as in Eq. (14.1)-(14.2)), we have $\cos\theta = \frac{1}{2}$. □

As in Sect. 14.1.1, we propose a reformulation of Eq. (14.4)-(14.5) for practical use:

$$\left.\begin{array}{rrcl} \min\limits_{x,z} & z & & \\ \forall i \leq n & ||x_i||^2 & = & 1 \\ \forall i < j \leq n & \langle x_i, x_j \rangle & \leq & z \\ \forall i \leq n & x_i & \in & [-1,1]^K \\ & z & \in & [-1,1]. \end{array}\right\} \tag{14.6}$$

Note that, by having z as a variable instead of a constant, we compute the best spherical z-code for given K, n.

14.3 MINLP formulation

The optimization problem related to the KNP, which aims at maximizing the number of surrounding spheres, is the following Mixed-Integer QCP (MIQCP) [204].

- **Parameters**:
  - K: space dimension;
  - $\bar{n}$: upper bound to $\mathsf{kn}(K)$.
- **Variables**:
  - $x_i \in \mathbb{R}^K$: center of i-th sphere
  - $\alpha_i = 1$ iff i-th sphere is in configuration
- **Formulation**:

$$\left.\begin{array}{rrcll} \max & \sum\limits_{i=1}^{\bar{n}} \alpha_i & & & \\ \forall i \leq \bar{n} & ||x_i||^2 & = & \alpha_i & \\ \forall i < j \leq \bar{n} & ||x_i - x_j||^2 & \geq & \alpha_i \alpha_j & (\star) \\ \forall i \leq \bar{n} & x_i & \in & [-1,1]^K & \\ \forall i \leq \bar{n} & \alpha_i & \in & \{0,1\}. & \end{array}\right\} \tag{14.7}$$

Note that the meaning of the α variable differs from Eq. (14.3). Here, α_i signals the presence of the i-th surrounding ball if $\alpha_i = 1$, or its absence if $\alpha_i = 0$. The globally optimal objective function value of Eq. (14.7) is exactly $\mathsf{kn}(K)$, as long as $\bar{n}$ is a valid upper bound.

We remark that the binary variable product $\alpha_i \alpha_j$ in the distance constraints $(\star)$ of Eq. (14.7) can be linearized exactly using Fortet's inequalities (see Rem. 2.2.8).

14.4 Polar coordinates

Many problems on a sphere benefit from a transformation to polar coordinates, which consists in mapping the point of contact $x_i = (x_{i1}, \ldots, x_{iK})$ of the i-th surrounding sphere to the unit length vector making angles $(\vartheta_{i1}, \ldots, \vartheta_{i,K-1})$ with the coordinate hyperplanes. The (pure feasibility) formulation we obtain is:

$$\forall i < j \leq n \quad \|x_i - x_j\|_2^2 \geq 1 \tag{14.8}$$

$$\forall k \leq K \quad \sin\vartheta_{i,k-1} \prod_{h=k}^{K-1} \cos\vartheta_{ih} = x_{ik}. \tag{14.9}$$

We note that Eq. (14.8) establishes the minimum distance between contact points, and Eq. (14.9) is the change of variables [281, 162]. We also note that we need only decide $\sin\vartheta$ and $\cos\vartheta$ rather than θ. This yields the simpler formulation:

$$\forall i < j \leq n \quad \|x_i - x_j\|_2^2 \geq 1 \tag{14.10}$$

$$\forall k \leq K \quad s_{ik} \prod_{h=k}^{K-1} c_{ih} = x_{ik} \tag{14.11}$$

$$\forall i \leq n, k \leq K \quad s_{ik}^2 + c_{ik}^2 = 1 \tag{14.12}$$

We can replace Eq. (14.11) into Eq. (14.10), which yields a rather formidable PP formulation of degree $2K$. Another possible reformulation consists in taking logarithms on both sides of Eq. (14.11): it removes the products at the cost of introducing a transcendental function.

14.5 Lower bounds

The formulations presented in the previous sections of this chapter are empirically very hard to solve, in particular to global optimality. They suffer from a range of numerical problems, among which formulation symmetry [173, §6].

The following facts were established using a variety of solution algorithms and solvers [180, 162, 173].

- Eq. (14.7) certifies that $\mathsf{kn}(2) = 6$ and $\mathsf{kn}(3) = 12$ whenever $\bar{n} = \mathsf{kn}(K)$, but fails to provide the same certifications if $\bar{n} = \mathsf{kn}(K) + 1$. Moreover, even with $\bar{n} = \mathsf{kn}(K)$, it fails to certify that $\mathsf{kn}(4) = 24$.

- Eq. (14.3) certifies $\mathsf{kn}(2) = 6$, $\mathsf{kn}(3) = 24$, but generally fails to find lower bounds greater than 38 for $\mathsf{kn}(5)$, for which a lower bound $\mathsf{kn}(5) \geq 40$ is known.[1] We remark that $K = 5$ is the smallest open case of the KNP.

- The formulations in Sect. 14.4 are numerically even more challenging.

The only practical usefulness of the formulations presented so far appears to be that they allow the identification of lower bounds to $\mathsf{kn}(K)$ with corresponding sphere configurations. We recall that the KNP is a maximization problem, so lower bounds are obtained by evaluating the objective function value of feasible solutions.

14.6 Upper bounds

Upper bounds on a maximization problem are more difficult to obtain than lower bounds. In this section we examine some of the existing methods.

[1] I was once able to find the known lattice solution $\hat{x}$ with $\mathsf{kn}(5) = 40$ by feeding $\hat{x}$ to a local NLP solver: the solution was not improved, but was deemed feasible. Any slight perturbation of $\hat{x}$ yielded an infeasible point. This suggests that the basin of attraction of $\hat{x}$ is tiny.

14.6.1 The useless SDP

A well-known strategy to obtain bounds on QCPs and MIQCPs in the optimization direction is to formulate and solve SDP relaxations. The SDP relaxation of Eq. (14.3) is as follows [174]:

$$\left.\begin{array}{rrcl} \max\limits_{X\in[-1,1]^{n^2},\alpha\geq 0} & \alpha & & \\ \forall i \leq n & X_{ii} & = & 1 \\ \forall i < j \leq n & X_{ii} + X_{jj} - 2X_{ij} & \geq & \alpha \\ & X & \succeq & 0. \end{array}\right\} \tag{14.13}$$

A "uselessness theorem" was proved in [168] for Eq. (14.13). One aspect of its uselessness in providing upper bounds is that it is independent of K: its bound therefore only depends on n, which means we cannot expect the bound to change as K changes. The uselessness theorem states that, for all $n \geq 2$, the optimal objective function value of Eq. (14.13) is $2n/(n-1)$. In other words, this bound tends to 2 from above as $n \to \infty$. Since $\alpha = 2$ is a (slack) upper bound for any instance (K, n) of the KNP, this SDP relaxation is actually practically useless.

14.6.2 The shadow bound

This bound was the reason why, in 1694, D. Gregory challenged Newton's intuition that $\mathsf{kn}(3) = 12$ (see Appendix B). Consider the central sphere C at the origin with radius 1, another sphere D centered at the origin with radius 3, and a surrounding sphere S adjacent to C. Imagine a spherical light-emitting device at the origin, suppose that the surface of C is transparent, and that the surfaces of D and S are opaque. Then S projects a shadow on the inner surface of D (see Fig. 14.4). The area of this shadow

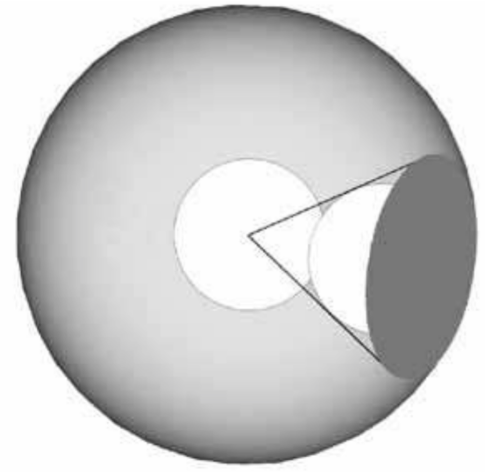


Figure 14.4: The idea behind the shadow bound (from [282]).

is around 7.6, while the total area of the surface of D is 113.1. Therefore there cannot be more than $\frac{113.1}{7.6} = 14.9$ balls surrounding C.

14.6.1 Exercise

Compute the areas of D and of the shadow of S.

14.6.3 The Delsarte bound

Consider the points of contact $x_1, \ldots, x_n$ of the surrounding spheres on the surface of the central ball with unit radius. We let $C = \{x_1, \ldots, x_n\}$ and call this set *spherical code*; remark that all code vectors are unit vectors. We look at the function σ_t mapping the scalar t to the number of pairs of points x_i, x_j having angular separation t, scaled by n; i.e.:

$$\forall t \in [-1,1] \quad \sigma_t = \frac{1}{n}\big|\{(i,j) \mid i,j \leq n \wedge x_i \cdot x_j = t\}\big|.$$

In order to enforce the minimum inter-spherical Euclidean distance required by the KNP, we let $\sigma_t = 0$ for all $t \in (\frac{1}{2}, 1)$, which imposes an angle of at least $\frac{\pi}{3}$ rad $= \arccos\frac{1}{2}$ between each pair of distinct vectors x_i, x_j in the code. Because of their unit length, this is the same as requiring that $x_i \cdot x_j \leq \frac{1}{2}$ for all $i \neq j$. Moreover, because $|C|$ is finite, only finitely many σ_t are nonzero (those corresponding to an angle arising from two vectors in C), which justifies our use of sums instead of integrals. From this discussion, we can state the following facts about the quantities σ_t:

$$\begin{aligned}
\sum_{t\in[-1,1]} \sigma_t = \frac{1}{n}|\text{all index pairs}| &= n \\
\sigma_1 = \frac{1}{n}n &= 1 \\
\forall t \in (1/2, 1) \quad \sigma_t &= 0 \\
\forall t \in [-1,1] \quad \sigma_t &\geq 0 \\
|\{\sigma_t > 0 \mid t \in [-1,1]\}| &< \infty.
\end{aligned}$$

Because we want to maximize n, these facts allow us to write the (semi-infinite) LP

$$\begin{cases}
\max\limits_{\sigma} & \sum\limits_{t\in[-1,1]} \sigma_t & & \\
& \sigma_1 & = & 1 \\
\forall t \in (\frac{1}{2}, 1) & \sigma_t & = & 0 \\
\forall t \in [-1,1] & \sigma_t & \geq & 0,
\end{cases} \tag{14.14}$$

the optimal value of which is an upper bound on $\mathsf{kn}(K)$. We can eliminate the fixed variables to obtain:

$$\begin{cases}
1 + \max\limits_{\sigma} & \sum\limits_{t\in[-1,\frac{1}{2}]} \sigma_t & & \\
\forall t \in [-1,1] & \sigma_t & \geq & 0,
\end{cases} \tag{14.15}$$

14.6.3.1 Valid cuts and Gegenbauer polynomials

Unfortunately, it is obvious that Eq. (14.15) is unbounded. In order to find further valid inequalities for Eq. (14.15), we look for a family $\mathscr{F}$ of functions $\phi : [-1,1] \to \mathbb{R}$ such that, for each $\phi \in \mathscr{F}$, the inequality

$$\sum_{t\in[-1,1]} \phi(t)\sigma_t \geq 0 \tag{14.16}$$

holds. One can then append Eq. (14.16) (for various $\phi \in \mathscr{F}$) to Eq. (14.15) as valid cuts in order to bound its optimal value from above. The best known and most often used family $\mathscr{F}$ is the orthogonal set $\mathscr{G}$ of *Gegenbauer polynomials* (introduced below), for which Eq. (14.16) is known to hold [98]. Since we replaced fixed variables to the corresponding values in Eq. (14.15), we reformulate Eq. (14.16) to:

$$\sum_{t\in[-1,\frac{1}{2}]} \phi(t)\sigma_t + \phi(1) \geq 0 \quad \Rightarrow \quad \sum_{t\in[-1,\frac{1}{2}]} \phi(t)\sigma_t \geq -\phi(1). \tag{14.17}$$

As long as we add a finite number of cuts from Eq. (14.17) to Eq. (14.15), the LP will have a finite number of constraints. On the other hand, Eq. (14.15) is still not easy to solve because it has infinitely many variables (one for every $t \in [-1, 1/2]$). We remark that the inequality sense in Eq. (14.16) was chosen in view of $\phi(t)$ ranging over Gegenbauer polynomials.

Another issue with Eq. (14.15) is that it is independent of K, whereas we want to find bounds for $\mathsf{kn}(K)$. We can encode K in the choice of the functions $\phi \in \mathscr{F}$ to be used in Eq. (14.17): more precisely, we take $\mathscr{F} = \mathscr{G}$, the family of Gegenbauer polynomials. These are univariate polynomials $G_m^K(t)$ depending on two parameters h, K: h encodes the degree and K is the dimension of the host space. It turns out that $G_h^K(t) = P_h^{(K-2)/2,(K-2)/2}(t)$, a *Jacobi polynomial*[2] (see Fig. 14.5). Jacobi polynomials are denoted

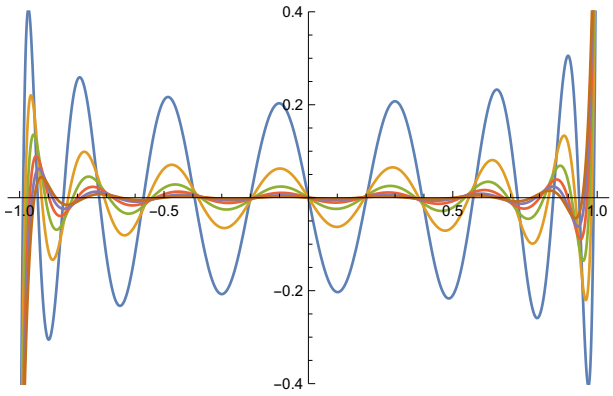


Figure 14.5: Gegenbauer polynomials.

by $P_h^{\alpha,\beta}$, and defined as follows:

$$P_h^{\alpha,\beta} = \frac{1}{2^h} \sum_{i=0}^{h} \binom{h+\alpha}{i} \binom{h+\beta}{h-1} (t+1)^i (t-1)^{h-i}.$$

We let $\mathscr{G}^K$ be the subset Gegenbauer polynomials of dimension K.

14.6.3.2 Primal and dual

We fix some $H = \{1, \ldots, \bar{H}\}$ to index a finite sequence of polynomials in $\mathscr{G}^K$ ordered by increasing degree $h \in H$ (we do not consider the degree zero term $G_0^K(t)$ since it is a positive constant; as such, it would yield a redundant cut). We reformulate Eq. (14.15) by adding valid cuts Eq. (14.17):

$$\left.\begin{array}{rrcl} 1 + \max & \sum_{t \in [-1, \frac{1}{2}]} \sigma_t & & \\ \forall h \in H & \sum_{t \in [-1, \frac{1}{2}]} G_h^K(t) \sigma_t & \geq & -G_h^K(1) \\ \forall t \in [-1, \frac{1}{2}] & \sigma_t & \geq & 0. \end{array}\right\} \tag{14.18}$$

For each $\geq$ constraint of the maximization problem Eq. (14.18) we introduce a dual variable $d_h \leq 0$ (for $h \in H$). For each variable $\sigma_t \geq 0$ (for $t \in [-1, \frac{1}{2}]$) we introduce a dual constraint with $\geq$ sense. The dual turns out to be:

$$\left.\begin{array}{rrcl} 1 + \min & \sum_{h \in H} (-G_h^K(1)) d_h & & \\ \forall t \in [-1, \frac{1}{2}] & \sum_{h \in H} G_h^K(t) d_h & \geq & 1 \\ \forall h \in H & d_h & \leq & 0. \end{array}\right\} \tag{14.19}$$

[2] We remark that some of the literature uses Jacobi parameters $(K-1)/2$ or $(K-3)/2$.

Now we define a new variable vector $c = -d$ and rewrite Eq. (14.19) as:

$$\left.\begin{array}{rrcl} 1 + \min & \sum\limits_{h\in H} G_h^K(1)c_h & & \\ \forall t \in [-1, \frac{1}{2}] & \sum\limits_{h\in H} G_h^K(t)c_h & \leq & -1 \\ \forall h \in H & c_h & \geq & 0. \end{array}\right\} \tag{14.20}$$

14.6.3.3 Delsarte's theorem

The fundamental theorem of Delsarte's LP applied to spherical codes is the following:

14.6.2 Theorem
Let $c_0 > 0$ and $F : [-1, 1] \to \mathbb{R}$ such that:

(i) $\exists J \subseteq \mathbb{N} \cup \{0\}$ *and* $c \in \mathbb{R}_+^{|J|} \geq 0$ *s.t.* $F(t) = \sum\limits_{h\in J} c_h G_h^K(t)$

(ii) $\forall t \in [-1, z]\ F(t) \leq 0$

Then $\mathsf{kn}(K) \leq \frac{F(1)}{c_0}$.

Thm. 14.6.2 was originally proved in [98, Thm. 4.3], but no LP was formulated in that paper. To the best of our knowledge, an LP (with uncountably many variables, but only finitely many of which can be nonzero) was first formulated in [238] (reprinted in [73, Ch. 13]). While the proof given in [98] depends on the properties of Gegenbauer polynomials, [238] simply states that Thm. 14.6.2 can be proved using Eq. (14.20).

We can derive Eq. (14.20) from Thm. 14.6.2: in order to achieve the best upper bound for $\mathsf{kn}(K)$, we look for the smallest possible value of $F(1)/c_0$ by minimizing $F(1)$ and fixing $c_0 = 1$. Setting $H = J \setminus \{0\}$, this yields:

$$\left.\begin{array}{rrcll} \min & F(1) = 1 + \sum\limits_{h\in H} c_h G_h^K(1) & & & \\ \forall t \in [-1, \frac{1}{2}] & 1 + \sum\limits_{h\in H} G_h^K(t)c_h & \leq & 0 & (14.21.1) \\ \forall h \in H & c_h & \geq & 0, & \end{array}\right\} \tag{14.21}$$

which is exactly Eq. (14.20).

14.6.3.4 Pfender's theorem

A variant of Thm. 14.6.2 was proved by Pfender in [246]:

14.6.3 Theorem
Let $c_0 > 0$ and $f : [-1, 1] \to \mathbb{R}$ such that:

(i) $\sum\limits_{i,j\leq n} f(x_i \cdot x_j) \geq 0$

(ii) $\forall t \in [-1, z]\ f(t) + c_0 \leq 0$

(iii) $f(1) + c_0 \leq 1$.

Then $\mathsf{kn}(K) \leq \frac{1}{c_0}$.

Pfender's motivation for proposing this variant is the brevity of the proof, which justifies our repeating it here. *Proof.* ([246]). Let $g(t) = f(t) + c_0$, then:

$$n^2 c_0 \leq n^2 c_0 + \sum_{i,j \leq n} f(x_i \cdot x_j) = \sum_{i,j \leq n} g(x_i \cdot x_j) \leq \sum_{i \leq n} g(x_i \cdot x_i) = n g(1) \leq n,$$

whence $n \leq 1/c_0$. □

While Pfender's theorem (Thm. 14.6.3) and Delsarte's theorem (Thm. 14.6.2) certainly look related, there are two prominent syntactical differences: Pfender adds a constraint on the function at 1, namely $f(1)+c_0 \leq 1$, and Pfender's bound is $1/c_0$. A possible interpretation to reconcile the two theorems is that $F(t) = f(t) + c_0$ for each $t \in [-1, 1]$, where F is the function Delsarte's theorem. Under this assumption, constraints (ii) in both theorems become the same.

We recall that the bound in Delsarte's theorem is $F(1)/c_0$. Using the additional constraint we see that:

$$\mathsf{kn}(K) \leq \frac{F(1)}{c_0} = \frac{f(1) + c_0}{c_0} \leq \frac{1}{c_0}$$

by (iii) in Thm. 14.6.3, whence Delsarte's bound is at least as tight as Pfender's, and possibly tighter. On the other hand, constraint (i) in Delsarte's theorem, which states that $F(t) = \sum_h c_h G_h^K(t)$ for all $t \in [-1, 1]$, is equivalent to $\int_{[-1,1]} F(t)dt \geq 0$ [238, 246, 267]. In our interpretation, this yields $\int_{[-1,1]} (f(t) + c_0)dt \geq 0$. But constraint (i) in Pfender's theorem, equivalent to $\int_{[-1,1]} f(t)dt \geq 0$, is at least as strict, and possibly stricter. In summary, Pfender's theorem makes stronger requirements than Delsarte's and delivers a weaker result. The computational bound values provided by the derived LP is the same as the LP from Delsarte's theorem.[3]

As in the case of Delsarte's theorem (see Eq. (14.21)), we can also derive an LP from Pfender's theorem: we maximize c_0 subject to (i)-(iii). We "model" (i), as in Delsarte's case, by taking $f(t)$ to be a nonnegatively weighted sum of Gegenbauer polynomials (starting with degree 1). This yields:

$$\begin{cases} \max\limits_{c \geq 0} & c_0 & & & \\ \forall t \in [-1, \frac{1}{2}] & c_0 + \sum\limits_{h \in H} c_h G_h^K(t) & \leq & 0 & (14.22.1) \\ & c_0 + \sum\limits_{h \in H} c_h G_h^K(1) & \leq & 1. & (14.22.2) \end{cases} \tag{14.22}$$

14.6.4 Implementable LPs

The standard approach [238, 21, 26] to deal with the semi-infinite LPs in Eq. (14.20) and (14.22) is to only keep finitely many constraints out of the infinite families listed in the formulations, for example by considering a discretization of $[-1, \frac{1}{2}]$. We write both (14.21.1) and (14.22.1) as follows:

$$\forall t \in T \quad c_0 + \sum_{h \in H} c_h G_h^K(t) \leq 0, \tag{14.23}$$

where c_0 is fixed to 1 in (14.21.1), and $T \subset [-1, \frac{1}{2}]$ with $|T|$ finite. T usually consists of finitely many equally spaced values between -1 and $\frac{1}{2}$ [238, 21].

An appropriate choice of T can be made with the help of [73, Ch. 13, §2], a result which is also formally summarized in [21, Lemma 1]. We give it in algorithmic form below.

1. Compute an optimal solution c^* to the finite LP (either Delsarte's or Pfender's) obtained by replacing $[-1, \frac{1}{2}]$ by the finite set T.

[3] This sentence refers to yet unpublished computational experiments by the author.

2. Find the global maximum t^* of the polynomial

$$F^*(t) = c_0^* + f^*(t) = c_0^* + \sum_{h \in H} c_h^* G_h^K(t)$$

with respect to $t \in [-1, \frac{1}{2}]$, as follows:

   (a) compute the set S of all solutions of the polynomial equation $\frac{dF(t)}{dt} = 0$,

   (b) let $t^* = \arg\max\{F(t) \mid t \in S\}$ and $\psi = F(t^*)$.

3. If $\psi \leq 0$ then c^* is feasible w.r.t. all of the infinitely many constraints in the LP: the bound is given by

$$\mathsf{kn}(K) \leq \begin{cases} \lfloor f^*(1) + 1 \rfloor & \text{with Delsarte's LP} \\ \lfloor \frac{1}{c_0^*} \rfloor & \text{with Pfender's LP.} \end{cases}$$

4. Ff $0 < \psi < c_0^*$ then a bound can be obtained as:

$$\mathsf{kn}(K) \leq \begin{cases} \lfloor \frac{f^*(1)+1-\psi}{1-\psi} \rfloor & \text{with Delsarte's LP} \\ \lfloor \frac{1-\psi}{c_0^*-\psi} \rfloor & \text{with Pfender's LP.} \end{cases}$$

5. If $\psi \geq c^*$ then no bound can be inferred: the cardinality of T should be increased, and the LP solved again.

14.6.4 Lemma
The bound formula in Step 4 of the above procedure holds.

Proof. Since ψ is the maximum value of $F^*(t)$ for $t \in [-1, \frac{1}{2}]$, and $\psi > 0$, then $F^*(t) - \psi = c_0^* - \psi - f^*(t) \leq 0$ for all $t \in [-1, \frac{1}{2}]$. Since $F^*(t)$ obeys the conditions of Delsarte's theorem, $F^*(t) - \psi$ does too: in particular, condition (i) follows by setting $c_0 = c_0^* - \psi$ and condition (ii) holds because ψ is a positive constant. Hence the corresponding bound is $\frac{F^*(1)-\psi}{c_0^*-\psi}$. We now recall that $c_0^* = 1$ in Delsarte's LP, and derive $\frac{f^*(1)+1-\psi}{1-\psi}$. For Pfender's LP, we recall that $F^*(1) = 1$, and derive $\frac{1-\psi}{c_0^*-\psi}$, as claimed. □

14.6.4.1 The choice of H

We recall that H is the index set of the finite subfamily $\mathscr{H}$ of functions from $\mathscr{F}$ used to quantify over the sums occurring in Delsarte's LP.

Adding functions to $\mathscr{F}$ has been the preferred method for improving the Delsarte bound so far. In [236], some functions are allowed to violate constraint (14.22.2) in Eq. (14.22) over finitely many values of t, which correspond to cases tackled separately and combinatorially. In [246], a new class of functions satisfying Eq. (14.16) and (14.22.2) has been introduced. Other bound improvements were obtained by considering an extension of the function σ_t to vector triplets instead of pairs, which yields a SDP [26]. Solving these SDPs poses some technical issues with precision, which were addressed in [226]. Extending this idea further (to vector quadruplets) was carried out in [117] but only for binary codes rather than spherical codes.

To the best of our knowledge, however, no-one discusses the problem of appropriately choosing H in view of computational efficiency, precision, bound quality and bound validity. In [73, Ch. 13], the authors simply state that they choose Gegenbauer polynomials up to degree 30 (but also point out that they only needed polynomials of degree up to 14; for $K = 4$ they limited the degree to 9). A somewhat more detailed treatment (but still insufficient to derive an algorithm to decide H) is given in [238].

If $\mathscr{F} = \mathscr{G}^K$ is the family of Gegenbauer polynomials $G_h^K(t)$, H is the set where the parameter h ranges. Large values of m yield polynomials of high degree, which may in turn yield large floating point errors when evaluated. This is therefore a source of imprecision which might, in the worst case, invalidate the bound.

Chapter 15

ACOPF

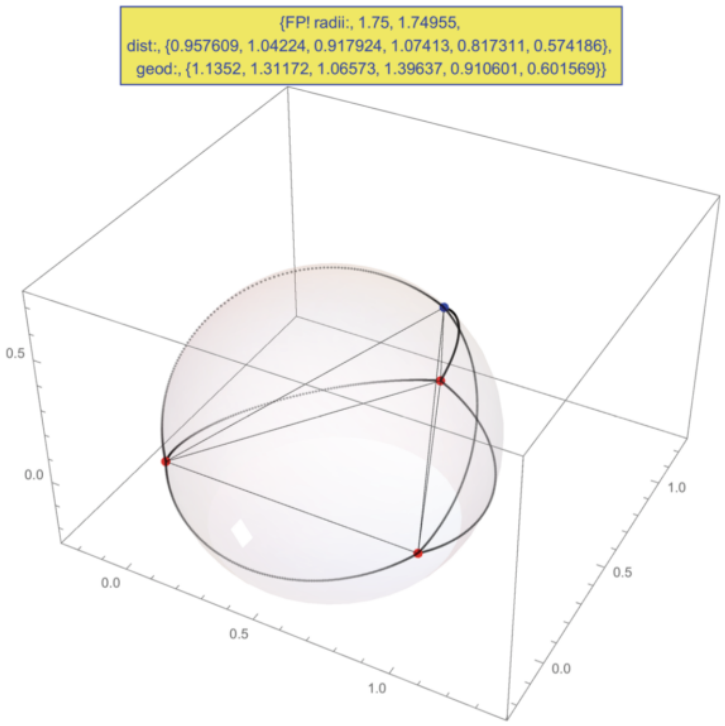
Chapter 16

PMU placement

Appendix A

Fighting over Gödel

In assembling the historic results in DG that seemed most remarkable to me, I stumbled upon a very short paper of Kurt Gödel about the fact that if you can find four points in 3D that match the six edge lengths of a tetrahedron, then you can also find four points on a sphere. The result is nontrivial because in 3D the edges are straight segments, whereas on the sphere the edges are curved.



I found it surprising that Kurt Gödel worked in DG, since he is known as a logician, more precisely the logician who shattered Hilbert's dream of a formalist mathematics, by means of his incompleteness theorems.

After more investigations, it turned out that Gödel wrote exactly two papers on DG. These were not actually papers, but rather short texts found in the *comptes rendus* of Menger's Mathematical Colloquium. Karl Menger was part of the Vienna Circle. When things got rough, and its founder Moritz Schlick was shot dead by his former student Johann Nelböck, Karl distanced himself from the Circle. He

started a regular mathematical seminar at the University of Vienna, featuring some of the most famous mathematicians and logicians of his time (e.g. John von Neumann, Alfred Tarski, Norbert Wiener, Karol Borsuk, Karl Popper). The local mathematicians and students, including Menger and Gödel, featured as a semi-permanent audience, and also frequently gave presentations themselves. This is where Gödel presented his two results in DG; his short papers appear in the proceedings, published by Springer-Verlag in 1998 [221].

While leafing through the proceedings book, I stumbled upon Franz Alt's Afterword (Franz Alt was one of the last survivors of Menger's seminar when Springer published the proceedings). He relates:

> What appears to have been Kurt Gödel's first oral presentation of a proof of his incompleteness theorem, was given at the 24th Colloquium in less than an hour, reported in proceedings [221] so concisely that it takes up little more than a page. [...] There was the unforgettable quiet after Gödel's presentation, ended by what must be the understatement of the century: "That is very interesting. You should publish that." Then a question: "You use Peano's system of axioms. Will it work for other systems?" Gödel, after a few seconds of thought: "Yes, any system broad enough to define the field of the integers."

Among those present at Gödel's presentation of the incompleteness theorem, there was a certain Olga Taussky (later Taussky-Todd): a mathematician who, according to Wikipedia, fixed all (but one) of the (many) errors in Hilbert's papers. After Gödel's reply to the question, she interjected,

> Olga Taussky (half-smiling): "The integers do not constitute a field!"

This comment, taken at face value, is completely inappropriate. While it is true that the integers do not constitute a field, this is knowledge that every undergraduate student in mathematics possesses, and such a mistake is clearly a slip of the tongue on G̈del's part. I find it hard to believe that anyone would make such a remark after any mathematical seminar, let alone G̈del's announcement of his incompleteness theorem (Gödel was by then already famous because of his earlier completeness theorem). So why did Olga, a gifted mathematician herself, made it? Why did she challenge Gödel, and why was she "half-smiling" as she did so?

The final part of the exchange is related by Franz Alt in a way which is consistent with the theory that Olga must have had some further reason for making the remark, other than the detection of an error.

> G̈del, who knew this as well as anyone, and had only spoken carelessly: "Well, the... the... the domain of integrity of the integers." And final relaxing laughter.

I was dumbfounded for a while, until I bought a second-hand copy of a book by Pierre Cassou-Noguès, *Les démons de Gödel*, published by 2007 by Éditions du Seuil. In §I.5, he writes (my translation):

> "Gödel loved women, and made no secret of it", said Olga Taussky, who studied at the University of Vienna at the same time as Gödel. She recounts some episodes, almost pranks, from those years. Gödel has some admirers: young women who complain about his prima donna attitudes, his late rising habits, his spoiled child behaviour. Gödel also has his tricks, e.g. giving appointments to one of his fans in a university room where he knows he'll find another girl he is trying to seduce.

Now the "half-smile" and the public challenge about the "field of integers" are consistent with a broader picture. The main question is to establish whether Olga is a fan being challenged, or one of Gödel's object of desire, or simply an impartial observer stating a fact.

But first, Olga's comment above sounds preposterous if we take into account what is commonly known about Gödel's life. This is a man who became increasingly dependent on his wife Adele, until he let himself die of hunger when she had to leave home for a long stay in hospital. We know of no infidelity during their marriage, and we also know of no interest of Gödel's in everyday life matters, his attention being fully devoted to logic and philosophy.

The truth may not be quite as simple. Gödel himself, in a note from his *nachlass* (the archive of his handwritten papers and notes), states that he sees his life until 1943 as partitioned in three broad periods: 1920-1927, 1928-1936, and 1937-1943. He writes that he did not give enough attention to women in the first and third period, but gave too much of it in the second period. The seminar related by Franz Alt is dated Jan. 22, 1931: right in the middle of Gödel's "too much attention to women" period. So perhaps we simply have the wrong idea of Gödel's relationship with women.

Now, back to the main question. From her half-smiling challenge and her remark about Gödel's tricks with women, I would say Olga was intrigued by Gödel. If this is true, then she cannot have been an impartial observer. Furthermore, the two girls in Olga's tale play a distinctly different role: one is scorned (because she is an admirer and Gödel uses her to attempt to make some other girl jealous), while the other is desired. In my experience, I have rarely seen people recount the "scorned roles" they were forced to play before those they liked or loved. If I am to attribute to Olga one of the two roles in her tale, I would definitely choose the object of desire.

This leaves two questions open: who was the "scorned woman", and was Olga really an object of Gödel's desire? After all, if her tale involves herself, she is heavily biased. She might have wished to be an object of Gödel's desire with all her heart, but this does not mean she actually was. Olga might have thought that Gödel brought "an admirer" to a room where she was working, whereas Gödel simply walked in with a woman he was talking to, for whatever reason. This might have even happened several times, but how many empty rooms can the University of Vienna have, to make such an event so unlikely as to consider it purposeful? There may have been a few rooms dedicated to doctoral students or postdoctoral assistants, as is customary even now. Maybe Olga only remarked when Gödel entered these rooms with a girl, possibly because there were few girls in the mathematics department, or possibly because she was taken with Gödel.

Can this girl, the fan, the admirer, the "scorned role", be Adele, who then became his wife?

Gödel met Adele in 1927, but married her in 1938. They had a difficult start, since she was a waitress and a dancer, and he knew his parents would be opposed to their union. In Jan. 1931, Adele might well be a girl classified by Olga as "one of Gödel's fans" — the girl to whom he gave appointments in rooms where the true object of his desire was present. I do not remember reading that Gödel gave Adele appointments at the university. I remember reading that he often went to see her at the restaurant she worked for.

I do not even know whether, in those times, it was accepted practice for anyone to simply walk into a university department unchallenged. As far as I know, universities were pretty open places as regards the teaching halls and the courtyards, but departments were often behind closed doors. Even if not locked, I would find it unlikely that a complete outsider, such as Adele, would dare walk into a department unaccompanied. Of course it is possible that Gödel brought her in. But then why meet her at the University? If we take Gödel's statement about his excessive interest in women at face value, it might have been more conducive for him to seek a more romantic place than the Ph.D. office in order to spend some time with his girlfriend.

Moreover, Olga says "admirers", not "a single admirer". Adele may be part of this set (the complaints about Gödel's spoiled child habits can certainly be attributed to her), but can she really have made Olga jealous? A waitress and a dancer? So different from herself (and Gödel)? Wouldn't it be more likely that Olga should be jealous of an intellectual peer? Or at least someone within the department? I do not think Adele is the most likely candidate for the "scorned role" in Olga's tale.

Imagine a more normal setting, where the most likely cause for entering a room in one's department is work. Gödel was not well known for working with other people, but Menger's Colloquium proceedings [221] show that Gödel often participated to discussions after seminars. Let us suppose that a few times he sat down with a colleague and discussed a problem. Who, amongst his colleagues, qualifies for this role? Since we are looking for a girl, the choice is very limited. As far as I can see, only two women gave talks at Menger's Colloquium: Olga Taussky, and Laura Klanfer. If we exclude the former, we must consider the latter.

And again we can find some interesting evidence in [221]. Something which might also explain the keen interest of Gödel in DG. Laura Klanfer intervened in the Colloquium twice: "On d-cyclic quadruplets" (Dec. 2, 1931) and "Metric characterization of the sphere" (June 28, 1932). Rather interestingly, Gödel also gave talks on the same dates. In the first seminar, Laura posed an open question: can any tetrahedron in space be embedded isometrically on the sphere? In the seminar on Feb. 18, 1932, Gödel provided an answer to Laura's question [221, p. 198]. His answer is an elegant and nontrivial fixed point construction, which I subsequently extended to an arbitrary number of dimensions [192].

So the fact that Gödel sat down in the Ph.D. room with Laura to discuss her open question, appears as perfectly normal, and possibly absolutely innocent. Notwithstanding, it is possible that Olga witnessed the exchange and became jealous. After all, Olga worked in topological algebra, and there is no trace in the Colloquium proceedings that Gödel ever took an interest in her work.

Laura clearly played a minor part in the Colloquium, whereas Olga was much more present. I do not

know what became of Laura, whereas the contributions of Olga Taussky to mathematics are considerable. While there is no certainty in my inferences, it stands to reason that Olga should have been rather pissed off at Gödel's interest (even if purely intellectual) into someone whom Olga might have considered beneath her level.

Appendix B

Apocryphal history of the kissing number problem

"Kissing" is billiard jargon. British players would say two adjacent billiard balls on the table "kiss".

The term found its way into mathematics thanks to Isaac Newton (whom everyone knows) and David Gregory (a professor of Mathematics at Edinburgh — without having ever obtained a degree — and then Savilian Professor of Astronomy at Oxford thanks to Newton's influence). In the 1690s, scared of the social unrest in Scotland, Gregory left and visited Newton in Cambridge.

According to unsubstantiated rumours and well-established British protocol, the brit and the scot went down the pub for a few pints of ale and a game of pool. There, among kissing balls and fumes of alcohol, they got into a brawl about the number of balls that could kiss a central ball on the billiard table. Still sober enough, they counted them, and came to agree on the number six.

As the number of pints increased, the two started blabbering about gravity-defying floating balls passionately kissing in three dimensions, and disagreed: Newton, embracing the voice of Kepler, said no more than twelve balls could be arranged around a central one. Gregory, who had to gain his master's approval by attempting to be brilliant and surprising, said that perhaps thirteen could fit?

George Szpiro [282] recounts a different story in plus.maths.org/content/newton-and-kissing-problem (some nonsense about astronomy and planets), but since neither he nor I were present at the quabble, his word on the matter is just as good as mine.

Bibliography

[1] S. Aaronson. *Quantum computing since Democritus*. Cambridge Univesity Press, Cambridge, UK, 2013.

[2] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66:671–687, 2003.

[3] N. Adhya, M. Tawarmalani, and N.V. Sahinidis. A Lagrangian approach to the pooling problem. *Industrial and Engineering Chemistry Research*, 38:1956–1972, 1999.

[4] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs: II. Implementation and computational results. *Computers & Chemical Engineering*, 22(9):1159–1179, 1998.

[5] C.S. Adjiman. *Global Optimization Techniques for Process Systems Engineering*. PhD thesis, Princeton University, June 1998.

[6] C.S. Adjiman, I.P. Androulakis, C.D. Maranas, and C.A. Floudas. A global optimization method, αBB, for process design. *Computers & Chemical Engineering*, 20:S419–S424, 1996.

[7] A. Ahmadi, R. Jungers, P. Parrilo, and M. Roozbehani. Joint spectral radius and path-complete graph Lyapunov functions. *SIAM Journal on Control and Optimization*, 52(1):687–717, 2014.

[8] A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: More tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebra and Geometry*, 3(2):193–230, 2019.

[9] A. Ahmadi, A. Olshevsky, P. Parrilo, and J. Tsitsiklis. NP-hardness of deciding convexity of quartic polynomials and related problems. *Mathematical Programming*, 137:453–476, 2013.

[10] M. Aigner. Turán's graph theorem. *American Mathematical Monthly*, 102(9):808–816, 1995.

[11] N. Ailon and B. Chazelle. Approximate nearest neighbors and fast Johnson-Lindenstrauss lemma. In *Proceedings of the Symposium on the Theory Of Computing*, volume '06 of *STOC*, Seattle, 2006. ACM.

[12] F.A. Al-Khayyal and H.D. Sherali. On finitely terminating branch-and-bound algorithms for some global optimization problems. *SIAM Journal of Optimization*, 10(4):1049–1057, 2000.

[13] A. Alexandrov. *Convex Polyhedra*. Springer, Berlin, 2005 (translated from Russian ed. 1950).

[14] A. Alfakih, A. Khandani, and H. Wolkowicz. Solving Euclidean distance matrix completion problems via semidefinite programming. *Computational Optimization and Applications*, 12:13–30, 1999.

[15] G. Allen. Sparse higher-order principal components analysis. In N. Lawrence and M. Girolami, editors, *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 27–36, La Palma, 2012. PMLR.

[16] Z. Allen-Zhu, R. Gelashvili, S. Micali, and N. Shavit. Sparse sign-consistent Johnson-Lindenstrauss matrices: Compression with neuroscience-based constraints. *Proceedings of the National Academy of Sciences*, 111(47):16872–16876, 2014.

[17] D. Aloise, P. Hansen, and L. Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming A*, 131:195–220, 2012.

[18] N. Alon and J. Spencer. *The probabilistic method*. Wiley, Hoboken, NJ, 2008.

[19] D. Amelunxen, M. Lotz, M. McCoy, and J. Tropp. Living on the edge: phase transitions in convex programs with random data. *Information and Inference: A Journal of the IMA*, 3:224–294, 2014.

[20] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, December 1995.

[21] K. Anstreicher. The thirteen spheres: a new proof. *Discrete and Computational Geometry*, 31:613–625, 2004.

[22] T. Apostol. *Mathematical Analysis*. Addison-Wesley, Reading, MA, 1961.

[23] R. Arriaga and S. Vempala. An algorithmic theory of learning: Robust concepts and random projection. *Machine Learning*, 63:161–182, 2006.

[24] L. Asimow and B. Roth. The rigidity of graphs. *Transactions of the AMS*, 245:279–289, 1978.

[25] C. Audet, J. Brimberg, P. Hansen, S. Le Digabel, and N. Mladenović. Pooling problem: Alternate formulations and solution methods. *Management Science*, 50(6):761–776, 2004.

[26] C. Bachoc and F. Vallentin. New upper bounds for kissing numbers from semidefinite programming. *Journal of the AMS*, 21:909–924, 2008.

[27] J. Bachrach and C. Taylor. Localization in sensor networks. In I. Stojmenović, editor, *Handbook of Sensor Networks*, pages 3627–3643. Wiley, 2005.

[28] M. Bardet, J.-Ch. Faugère, and B. Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of International Conference on Polynomial System Solving*, 2004.

[29] G. Barker and D. Carlson. Cones of diagonally dominant matrices. *Pacific Journal of Mathematics*, 57(1):15–32, 1975.

[30] A. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete and Computational Geometry*, 13:189–202, 1995.

[31] A. Barvinok. Measure concentration in optimization. *Mathematical Programming*, 79:33–53, 1997.

[32] A. Barvinok. *A Course in Convexity*. Number 54 in Graduate Studies in Mathematics. AMS, Providence, RI, 2002.

[33] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*. Springer, New York, 2006.

[34] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, Chichester, second edition, 1993.

[35] N. Beeker, S. Gaubert, C. Glusa, and L. Liberti. Is the distance geometry problem in NP? In Mucherino et al. [233], pages 85–94.

[36] E. Bell. The iterated exponential integers. *Annals of Mathematics*, 39:539–557, 1938.

[37] P. Belotti, S. Cafieri, J. Lee, L. Liberti, and A. Miller. On the composition of convex envelopes for quadrilinear terms. In A. Chinculuun and *et al.*, editors, *Optimization, Simulation and Control*, volume 76 of *SOIA*, pages 1–16. Springer, Berlin, 2013.

[38] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4):597–634, 2009.

[39] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, Princeton, NJ, 2009.

[40] Y. Bengio. Deep learning for AI, 2017. Presentation at the MIP 2017 workshop.

[41] K. Bennett and O. Mangasarian. Bilinear separation of two sets in n-space. *Computational Optimization and Applications*, 2(3):207–227, 1993.

[42] E. Berlekamp, J. Conway, and R. Guy. *Winning ways for your mathematical plays, vol. 2*. Academic Press, 1982.

[43] D. Bertsekas. *Convex optimization theory*. Athena Scientific, Nashua, 2009.

[44] D. Bienstock and A. Michalka. Polynomial solvability of variants of the trust-region subproblem. In *Proceedings of the 25th annual ACM Symposium on Discrete Algorithms*, volume 25 of *SODA*, pages 380–390, Philadelphia, 2014. ACM.

[45] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 2011.

[46] P. Biswas, T. Lian, T. Wang, and Y. Ye. Semidefinite programming based algorithms for sensor network localization. *ACM Transactions in Sensor Networks*, 2:188–220, 2006.

[47] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions, and universal machines. *Bulletin of the AMS*, 21(1):1–46, 1989.

[48] L. Blumenthal. *Theory and Applications of Distance Geometry*. Oxford University Press, Oxford, 1953.

[49] I. Bomze. Evolution towards the maximum clique. *Journal of Global Optimization*, 10:143–164, 1997.

[50] I. Bomze. Copositive optimization — Recent developments and applications. *European Journal of Operational Research*, 216:509–520, 2012.

[51] I. Bomze, M. Dür, E. De Klerk, C. Roos, A. Quist, and T. Terlaky. On copositive programming and standard quadratic optimization problems. *Journal of Global Optimization*, 18:301–320, 2000.

[52] P. Bonami and J. Lee. `BONMIN` User's Manual. Technical report, IBM Corporation, June 2007.

[53] I. Borg and P. Groenen. *Modern Multidimensional Scaling*. Springer, New York, second edition, 2010.

[54] C. Boutsidis, A. Zouzias, and P. Drineas. Random projections for k-means clustering. In *Advances in Neural Information Processing Systems*, NIPS, pages 298–306, La Jolla, 2010. NIPS Foundation.

[55] S. Boyd and L. Vandenberghe. *Convex Optimization*. CUP, Cambridge, 2004.

[56] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008.

[57] J. Brimberg, P. Hansen, and N. Mladenoviç. Convergence of variable neighbourhood search. Technical report, GERAD, 2008.

[58] A. Brook, D. Kendrick, and A. Meeraus. GAMS, a user's guide. *ACM SIGNUM Newsletter*, 23(3-4):10–11, 1988.

[59] B. Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero-dimensional polynomial ideal. *Journal of Symbolic Computation*, 41:475–511, 2006.

[60] S. Cafieri, J. Lee, and L. Liberti. On convex relaxations of quadrilinear terms. *Journal of Global Optimization*, 47:661–685, 2010.

[61] E. Candès. The mathematics of sparsity. In S.Y. Jang, Y.R. Kim, D.-W. Lee, and I. Yie, editors, *Proceedings of the International Congress of Mathematicians*, volume I. Kyung Moon SA, Seoul, 2014.

[62] E. Candès and T. Tao. Reflections on compressed sensing. *IEEE Information Theory Society Newsletter*, 58(4):14–17, 2008.

[63] A. Cassioli, B. Bordeaux, G. Bouvier, A. Mucherino, R. Alves, L. Liberti, M. Nilges, C. Lavor, and T. Malliavin. An algorithm to enumerate all possible protein conformations verifying a set of distance constraints. *BMC Bioinformatics*, 16:23–38, 2015.

[64] A.-L. Cauchy. Sur les polygones et les polyèdres. *Journal de l'École Polytechnique*, 16(9):87–99, 1813.

[65] A. Cayley. A theorem in the geometry of position. *Cambridge Mathematical Journal*, II:267–271, 1841.

[66] Y.-J. Chang and B. Wah. Polynomial Programming using Gröbner bases. Technical report, University of Illinois at Urbana-Champaign, 1994.

[67] V. Chvátal. *Linear Programming*. Freeman & C., New York, 1983.

[68] D. Cifuentes and P. Parrilo. Exploiting chordal structure in polynomial ideas: a Gröbner basis approach. *SIAM Journal of Discrete Mathematics*, 30(3):1534–1570, 2016.

[69] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1965.

[70] COIN-OR. *Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT*, 2006.

[71] G. Collins. Quantifier elimination for real closed fields. *ACM SIGSAM Bulletin*, 8(3):80–90, 1974.

[72] R. Connelly. A counterexample to the rigidity conjecture for polyhedra. *Publications Mathématiques de l'IHES*, 47:333–338, 1978.

[73] J. Conway and N. Sloane, editors. *Sphere Packings, Lattices and Groups*. Springer, Berlin, 1993.

[74] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Symposium on the Theory of Computing*, STOC, pages 151–158, New York, 1971. ACM.

[75] G. Cornuéjols and M. Dawande. A class of hard small 0-1 programs. In R. Bixby, E. Boyd, and R. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *LNCS*, pages 284–293, Berlin, 1998. Springer.

[76] G. Cornuéjols, L. Liberti, and G. Nannicini. Improved strategies for branching on general disjunctions. *Mathematical Programming A*, 130:225–247, 2011.

[77] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman & Hall, Boca Raton, 2001.

[78] H. Cramer. *Mathematical Methods of Statistics*. Princeton University Press, Princeton, NJ, 1946.

[79] A.E. Csallner, T. Csendes, and M.C. Markót. Multisection in interval branch-and-bound methods for global optimization i. theoretical results. *Journal of Global Optimization*, 16:371–392, 2000.

[80] A.E. Csallner, T. Csendes, and M.C. Markót. Multisection in interval branch-and-bound methods for global optimization II. Numerical tests. *Journal of Global Optimization*, 16:219–228, 2000.

[81] M. Cucuringu, Y. Lipman, and A. Singer. Sensor network localization by eigenvector synchronization over the Euclidean group. *ACM Transactions on Sensor Networks*, 8:1–42, 2012.

[82] M. Cucuringu, A. Singer, and D. Cowburn. Eigenvector synchronization, graph ridigity and the molecule problem. *Information and Inference: a journal of the IMA*, 1:21–67, 2012.

[83] J. Currie and D. Wilson. OPTI: Lowering the Barrier Between Open Source Optimizers and the Industrial MATLAB User. In N. Sahinidis and J. Pinto, editors, *Foundations of Computer-Aided Process Operations*, Savannah, Georgia, USA, 8–11 January 2012.

[84] C. D'Ambrosio, L. Liberti, P.-L. Poirion, and K. Vu. Random projections for quadratic programming. Technical Report 2019-7-7322, Optimization Online, 2019.

[85] C. D'Ambrosio, L. Liberti, P.-L. Poirion, and K. Vu. Random projections for quadratic programs. *Mathematical Programming B*, 183:619–647, 2020.

[86] S. Damelin and W. Miller. *The mathematics of signal processing*. CUP, Cambridge, 2012.

[87] G. Dantzig. Reminiscences about the origins of linear programming. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: the state of the art*. Springer, Berlin, 1983.

[88] G. Dantzig. The Diet Problem. *Interfaces*, 20(4):43–47, 1990.

[89] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[90] S. Dasgupta and A. Gupta. An elementary proof of a theorem by Johnson and Lindenstrauss. *Random Structures and Algorithms*, 22:60–65, 2002.

[91] A. D'Aspremont, F. Bach, and L. El Ghaoui. Approximation bounds for sparse principal component analysis. *Mathematical Programming B*, 148:89–110, 2014.

[92] M. Davenport, M. Duarte, Y. Eldar, and G. Kutyniok. Introduction to compressed sensing. In Y. Eldar and G. Kutyniok, editors, *Compressed Sensing: Theory and Applications*, page 1–64. CUP, Cambridge, 2012.

[93] T. Davidović, L. Liberti, N. Maculan, and N. Mladenović. Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In *MISTA Proceedings*, 2007.

[94] M. Davis. Arithmetical problems and recursively enumerable predicates. *Journal of Symbolic Logic*, 18(1), 1953.

[95] M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, 74(3):425–436, 1961.

[96] R. Davis, C. Ernst, and D. Wu. Protein structure determination via an efficient geometric build-up algorithm. *BMC Structural Biology*, 10(Suppl 1):S7, 2010.

[97] N. de Bruijn. *Asymptotic methods in analysis*. Dover, New York, 1981.

[98] P. Delsarte, J.M. Goethals, and J.J. Seidel. Spherical codes and designs. *Geometriæ Dedicata*, 6:363–388, 1977.

[99] P. Demartines and J. Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, 1997.

[100] S. Dey, R. Mazumder, M. Molinaro, and G. Wang. Sparse principal component analysis and its ℓ_1-relaxation. Technical Report 1712.00800v1, arXiv, 2017.

[101] G. Dias and L. Liberti. Diagonally dominant programming in distance geometry. In R. Cerulli, S. Fujishige, and R. Mahjoub, editors, *International Symposium in Combinatorial Optimization*, volume 9849 of *LNCS*, pages 225–236, New York, 2016. Springer.

[102] Y. Ding, N. Krislock, J. Qian, and H. Wolkowicz. Sensor network localization, Euclidean distance matrix completions, and graph realization. *Optimization and Engineering*, 11:45–66, 2010.

[103] M. Dür. Copositive programming — A survey. In M. Diehl and *et al.*, editors, *Recent advances in Optimization and its Applications in Engineering*. Springer, Heidelberg, 2010.

[104] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[105] T.G.W. Epperly. *Global Optimization of Nonconvex Nonlinear Programs using Parallel Branch and Bound*. PhD thesis, University of Winsconsin – Madison, 1995.

[106] T.G.W. Epperly and E.N. Pistikopoulos. A reduced space branch and bound algorithm for global optimization. *Journal of Global Optimization*, 11:287:311, 1997.

[107] T. Eren, D. Goldenberg, W. Whiteley, Y. Yang, A. Morse, B. Anderson, and P. Belhumeur. Rigidity, computation, and randomization in network localization. *IEEE*, pages 2673–2684, 2004.

[108] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiæ Scientiarum Imperialis Petropolitanæ*, 8:128–140, 1736.

[109] L. Euler. Continuatio fragmentorum ex adversariis mathematicis depromptorum: II Geometria, 97. In P. Fuss and N. Fuss, editors, *Opera postuma mathematica et physica anno 1844 detecta*, volume I, pages 494–496. Eggers & C., Petropolis, 1862.

[110] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. Wiley, New York, 1968.

[111] M. Fischetti. *Lezioni di Ricerca Operativa (in Italian)*. Edizioni Libreria Progetto, Padova, 1999.

[112] R. Fletcher. *Practical Methods of Optimization*. Wiley, Chichester, second edition, 1991.

[113] R. Fortet. Applications de l'algèbre de Boole en recherche opérationelle. *Revue Française de Recherche Opérationelle*, 4:17–26, 1960.

[114] L.R. Foulds, D. Haughland, and K. Jornsten. A bilinear approach to the pooling problem. *Optimization*, 24:165–180, 1992.

[115] R. Fourer and D. Gay. *The AMPL Book*. Duxbury Press, Pacific Grove, 2002.

[116] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[117] D.C. Gijswijt, H.D. Mittelmann, and A. Schrijver. Semidefinite code bounds based on quadruple distances. *IEEE Transactions on Information Theory*, 58(5):2697–2705, 2012.

[118] P. Gill, W. Murray, A. Saunders, J. Tomlin, and M. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209, 1986.

[119] P.E. Gill. *User's Guide for SNOPT 5.3*. Systems Optimization Laboratory, Department of EESOR, Stanford University, California, February 1999.

[120] P.E. Gill. *User's guide for SNOPT version 7.2*. Systems Optimization Laboratory, Stanford University, California, 2006.

[121] H. Gluck. Almost all simply connected closed surfaces are rigid. In A. Dold and B. Eckmann, editors, *Geometric Topology*, volume 438 of *Lecture Notes in Mathematics*, pages 225–239, Berlin, 1975. Springer.

[122] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1930.

[123] A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[124] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[125] D. Gonçalves, A. Mucherino, C. Lavor, and L. Liberti. Recent advances on the interval distance geometry problem. *Journal of Global Optimization*, 69:525–545, 2017.

[126] J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity*. AMS, 1993.

[127] M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[128] C. Guéret, C. Prins, and M. Sevaux. *Applications of optimization with Xpress-MP*. Dash Optimization, Bilsworth, 2000.

[129] K. Hägglöf, P.O. Lindberg, and L. Svensson. Computing global minima to polynomial optimization problems using Gröbner bases. *Journal of Global Optimization*, 7(2):115:125, 1995.

[130] M. Hall. *Combinatorial Theory*. Wiley, New York, 2nd edition, 1986.

[131] P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79:191–215, 1997.

[132] W. Hart, C. Laird, J.-P. Watson, and D. Woodruff. *Pyomo — Optimization modelling in Python*. Springer, New York, 2012.

[133] T. Havel and K. Wüthrich. An evaluation of the combined use of nuclear magnetic resonance and distance geometry for the determination of protein conformations in solution. *Journal of Molecular Biology*, 182(2):281–294, 1985.

[134] C.A. Haverly. Studies of the behaviour of recursion for the pooling problem. *ACM SIGMAP Bulletin*, 25:19–28, 1978.

[135] R. Helgason, J. Kennington, and H. Lall. A polynomially bounded algorithm for a singly constrained quadratic program. *Mathematical Programming*, 18:338–343, 1980.

[136] L. Henkin, P. Suppes, and A. Tarski, editors. *The axiomatic method with special reference to geometry and physics*. North-Holland, Amsterdam, 1959.

[137] Heron. *Metrica*, volume I. Alexandria, ∼50AD.

[138] H. Hijazi and L. Liberti. Constraint qualification failure in action. *Operations Research Letters*, 44:503–506, 2016.

[139] D. Hilbert. *Grundlagen der Geometrie*. Teubner, Leipzig, 1903.

[140] Roland Hildebrand. The extreme rays of the 5×5 copositive cone. *Linear Algebra and its Applications*, 437:1538–1547, 2012.

[141] D. Hochbaum. Complexity and algorithms for nonlinear optimization problems. *4OR*, 3(3):171–216, 2005.

[142] K. Holmström and M. Edvall. The tomlab optimization environment. In J. Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages 369–376. Springer, Boston, 2004.

[143] R. Horst and P.M. Pardalos, editors. *Handbook of Global Optimization*, volume 1. Kluwer Academic Publishers, Dordrecht, 1995.

[144] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933.

[145] IBM. *ILOG CPLEX 12.6 User's Manual*. IBM, 2014.

[146] IBM. *ILOG CPLEX 12.8 User's Manual*. IBM, 2017.

[147] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Foundations of Computer Science*, volume 42 of *FOCS*, pages 10–33, Washington, DC, 2001. IEEE.

[148] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Symposium on the Theory Of Computing*, volume 30 of *STOC*, pages 604–613, New York, 1998. ACM.

[149] P. Indyk and A. Naor. Nearest neighbor preserving embeddings. *ACM Transactions on Algorithms*, 3(3):Art. 31, 2007.

[150] B. Jansen, C. Roos, T. Terlaky, and J.-Ph. Vial. Interior-point methodology for linear programming: duality, sensitivity analysis and computational aspects. Technical Report 28, TU Delft, 1993.

[151] R. Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Operations Research*, 21(1):221–224, 1973.

[152] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In G. Hedlund, editor, *Conference in Modern Analysis and Probability*, volume 26 of *Contemporary Mathematics*, pages 189–206, Providence, RI, 1984. AMS.

[153] I. Jolliffe. *Principal Component Analysis*. Springer, Berlin, 2nd edition, 2010.

[154] J. Jones. Universal diophantine equation. *Journal of Symbolic Logic*, 47(3):549–571, 1982.

[155] D. Kane and J. Nelson. Sparser Johnson-Lindenstrauss transforms. *Journal of the ACM*, 61(1):4, 2014.

[156] I. Kantor, J. Matoušek, and R. Šámal. *Mathematics++: Selected topics beyond the basic courses*. Number 75 in Student Mathematical Library. AMS, Providence, RI, 2015.

[157] R. Karp. Reducibility among combinatorial problems. In R. Miller and W. Thatcher, editors, *Complexity of Computer Computations*, volume 5 of *IBM Research Symposia*, pages 85–104, New York, 1972. Plenum.

[158] P. Kesavan and P.I. Barton. Generalized branch-and-cut framework for mixed-integer nonlinear optimization problems. *Computers & Chemical Engineering*, 24:1361–1366, 2000.

[159] R. Koenker. *Quantile regression*. CUP, Cambridge, 2005.

[160] B. Korte and J. Vygen. *Combinatorial Optimization, Theory and Algorithms*. Springer, Berlin, 2000.

[161] V. Kovačević-Vujčić, M. Čangalović, M. Ašić, L. Ivanović, and M. Dražić. Tabu search methodology in global optimization. *Computers and Mathematics with Applications*, 37:125–133, 1999.

[162] S. Kucherenko, P. Belotti, L. Liberti, and N. Maculan. New formulations for the kissing number problem. *Discrete Applied Mathematics*, 155(14):1837–1841, 2007.

[163] J.-B. Lasserre. *An introduction to polynomial and semi-algebraic optimization*. CUP, Cambridge, 2015.

[164] M. Laurent. Matrix completion problems. In C. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, pages 1967–1975. Springer, New York, second edition, 2009.

[165] C. Lavor, L. Liberti, and N. Maculan. Computational experience with the molecular distance geometry problem. In J. Pintér, editor, *Global Optimization: Scientific and Engineering Case Studies*, pages 213–225. Springer, Berlin, 2006.

[166] C. Lavor, L. Liberti, and A. Mucherino. The *interval* Branch-and-Prune algorithm for the discretizable molecular distance geometry problem with inexact distances. *Journal of Global Optimization*, 56:855–871, 2013.

[167] C. Lavor, A. Mucherino, L. Liberti, and N. Maculan. On the computation of protein backbones by using artificial backbones of hydrogens. *Journal of Global Optimization*, 50:329–344, 2011.

[168] J. Lee and L. Liberti. On an SDP relaxation for kissing number. *Optimization Letters*, 14:417–422, 2020.

[169] J. Leech. The problem of the thirteen spheres. *Mathematical Gazette*, 40:22–23, 1956.

[170] L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College London, UK, March 2004.

[171] L. Liberti. Reformulations in mathematical programming: Definitions and systematics. *RAIRO-RO*, 43(1):55–86, 2009.

[172] L. Liberti. Reformulations in mathematical programming: Automatic symmetry detection and exploitation. *Mathematical Programming A*, 131:273–304, 2012.

[173] L. Liberti. Symmetry in mathematical programming. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *IMA*, pages 263–286. Springer, New York, 2012.

[174] L. Liberti. Mathematical programming bounds for kissing numbers. In A. Sforza and C. Sterle, editors, *Optimization and Decision Science: Methodologies and Applications (AIRO-ODS17)*, volume 217 of *Proceedings in Mathematics and Statistics*, pages 213–222, New York, 2017. Springer.

[175] L. Liberti. Undecidability and hardness in mixed-integer nonlinear programming. *RAIRO-Operations Research*, 53:81–109, 2019.

[176] L. Liberti. A new distance geometry method for constructing word and sentence vectors. In *Companion Proceedings of the Web Conference (DL4G Workshop)*, volume 20 of *WWW*, New York, 2020. ACM.

[177] L. Liberti. Distance geometry and data science. *TOP*, 28:271–339, 220.

[178] L. Liberti, S. Cafieri, and F. Tarissan. Reformulations in mathematical programming: A computational approach. In A. Abraham, A.-E. Hassanien, P. Siarry, and A. Engelbrecht, editors, *Foundations of Computational Intelligence Vol. 3*, number 203 in Studies in Computational Intelligence, pages 153–234. Springer, Berlin, 2009.

[179] L. Liberti and M. Dražic. Variable neighbourhood search for the global optimization of constrained NLPs. In *Proceedings of GO Workshop, Almeria, Spain*, 2005.

[180] L. Liberti and S. Kucherenko. Comparison of deterministic and stochastic approaches to global optimization. *International Transactions in Operational Research*, 12:263–285, 2005.

[181] L. Liberti and C. Lavor. On a relationship between graph realizability and distance matrix completion. In A. Migdalas, A. Sifaleras, C. Georgiadis, J. Papathanaiou, and E. Stiakakis, editors, *Optimization theory, decision making, and operational research applications*, volume 31 of *Proceedings in Mathematics & Statistics*, pages 39–48, Berlin, 2013. Springer.

[182] L. Liberti and C. Lavor. Six mathematical gems in the history of distance geometry. *International Transactions in Operational Research*, 23:897–920, 2016.

[183] L. Liberti and C. Lavor. *Euclidean Distance Geometry: An Introduction*. Springer, New York, 2017.

[184] L. Liberti, C. Lavor, J. Alencar, and G. Abud. Counting the number of solutions of ^{k}DMDGP instances. In F. Nielsen and F. Barbaresco, editors, *Geometric Science of Information*, volume 8085 of *LNCS*, pages 224–230, New York, 2013. Springer.

[185] L. Liberti, C. Lavor, N. Maculan, and A. Mucherino. Euclidean distance geometry and applications. *SIAM Review*, 56(1):3–69, 2014.

[186] L. Liberti, C. Lavor, A. Mucherino, and N. Maculan. Molecular distance geometry methods: from continuous to discrete. *International Transactions in Operational Research*, 18:33–51, 2010.

[187] L. Liberti, N. Maculan, and S. Kucherenko. The kissing number problem: a new result from global optimization. In L. Liberti and F. Maffioli, editors, *CTW04 Workshop on Graphs and Combinatorial Optimization*, volume 17 of *Electronic Notes in Discrete Mathematics*, pages 203–207, Amsterdam, 2004. Elsevier.

[188] L. Liberti and F. Marinelli. Mathematical programming: Turing completeness and applications to software analysis. *Journal of Combinatorial Optimization*, 28(1):82–104, 2014.

[189] L. Liberti, N. Mladenović, and G. Nannicini. A recipe for finding good solutions to MINLPs. *Mathematical Programming Computation*, 3:349–390, 2011.

[190] L. Liberti and C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25:157–168, 2003.

[191] L. Liberti and C. Pantelides. An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization*, 36:161–189, 2006.

[192] L. Liberti, G. Swirszcz, and C. Lavor. Distance geometry on the sphere. In J. Akiyama and *et al.*, editors, *JCDCG*^{2}, volume 9943 of *LNCS*, pages 204–215, New York, 2016. Springer.

[193] L. Liberti and K. Vu. Barvinok's naive algorithm in distance geometry. *Operations Research Letters*, 46:476–481, 2018.

[194] C. Ling, J. Nie, L. Qi, and Y. Ye. Biquadratic optimization over unit spheres and semidefinite programming relaxations. *SIAM Journal on Optimization*, 20(3):1286–1310, 2009.

[195] M. Locatelli. Simulated annealing algorithms for global optimization. In Pardalos and Romeijn [243], pages 179–229.

[196] M. Locatelli and U. Raber. On convergence of the simplicial branch-and-bound algorithm based on ω-subdivisions. *Journal of Optimization Theory and Applications*, 107(1):69–79, October 2000.

[197] M. Locatelli and F. Schoen. Random linkage: a family of acceptance/rejection algorithms for global optimization. *Mathematical Programming*, 85(2):379–396, 1999.

[198] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the International Symposium of Computer-Aided Control Systems Design*, volume 1 of *CACSD*, Piscataway, 2004. IEEE.

[199] R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[200] L. Lovasz. *Combinatorial problems and exercises*. North-Holland, Amsterdam, 1993.

[201] S. Lucidi and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *Journal of Optimization Theory and Applications*, 62(2):255–277, 1989.

[202] R. Lyndon. *Notes on logic.* Number 6 in Mathematical Studies. Van Nostrand, New York, 1966.

[203] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. University of California Press, 1967.

[204] N. Maculan, P. Michelon, and J. MacGregor Smith. Bounds on the kissing numbers in $\mathbb{R}^n$: Mathematical programming formulations. Technical report, University of Massachusetts, Amherst, USA, 1996.

[205] A. Majumdar, A. Ahmadi, and R. Tedrake. Control and verification of high-dimensional systems with dsos and sdsos programming. In *Conference on Decision and Control*, volume 53, pages 394–401, Piscataway, 2014. IEEE.

[206] A. Makhorin. *GNU Linear Programming Kit.* Free Software Foundation, `http://www.gnu.org/software/glpk/`, 2003.

[207] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, 1999.

[208] The MathWorks, Inc., Natick, MA. *MATLAB R2014a*, 2014.

[209] The MathWorks, Inc., Natick, MA. *MATLAB R2017a*, 2017.

[210] Y. Matiyasevich. Enumerable sets are diophantine. *Soviet Mathematics: Doklady*, 11:354–357, 1970.

[211] J. Matoušek. On variants of the Johnson-Lindenstrauss lemma. *Random Structures and Algorithms*, 33:142–156, 2008.

[212] J. Matoušek. Lecture notes on metric embeddings. Technical report, ETH Zürich, 2013.

[213] T. Matsui. NP-hardness of linear multiplicative programming and related problems. *Journal of Global Optimization*, 9:113–119, 1996.

[214] J. Maxwell. On reciprocal figures and diagrams of forces. *Philosophical Magazine*, 27(182):250–261, 1864.

[215] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:146–175, 1976.

[216] N. Megiddo. On the complexity of polyhedral separability. *Discrete and Computational Geometry*, 3:325–337, 1988.

[217] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures.* Springer, Berlin, 2008.

[218] L. Mencarelli, Y. Sahraoui, and L. Liberti. A multiplicative weights update algorithm for MINLP. *EURO Journal on Computational Optimization*, 5:31–86, 2017.

[219] K. Menger. Untersuchungen über allgemeine Metrik. *Mathematische Annalen*, 100:75–163, 1928.

[220] K. Menger. New foundation of Euclidean geometry. *American Journal of Mathematics*, 53(4):721–745, 1931.

[221] K. Menger, editor. *Ergebnisse eines Mathematischen Kolloquiums.* Springer, Wien, 1998.

[222] J. Milnor. On the Betti numbers of real varieties. *Proceedings of the AMS*, 15:275–280, 1964.

[223] J. Milnor. *Topology from the differentiable viewpoint.* University Press of Virginia, Charlottesville, 1969.

[224] M. Minsky. Size and structure of universal turing machines using tag systems. In *Recursive Function Theory*, volume 5 of *Symposia in Pure Mathematics*, pages 229–238. AMS, Providence, RI, 1962.

[225] R. Misener and C. Floudas. Global optimization of large-scale generalized pooling problems: quadratically constrained MINLP models. *Industrial Engineering and Chemical Research*, 49:5424–5438, 2010.

[226] H. Mittelmann and F. Vallentin. High-accuracy semidefinite programming bounds for kissing numbers. *Experimental Mathematics*, 19(2):175–179, 2010.

[227] A. Moitra. *Algorithmic aspects of Machine Learning.* CUP, Cambridge, 2018.

[228] R. Montague. Universal grammar. *Theoria*, 36(3):373–398, 1970.

[229] R. Montague. *Formal Philosophy.* Yale University Press, London, 1974.

[230] R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis.* SIAM, Philadelphia, 2009.

[231] J. Moré and Z. Wu. Distance geometry optimization for protein structures. *Journal of Global Optimization*, 15:219–234, 1999.

[232] T. Motzkin and E. Straus. Maxima for graphs and a new proof of a theorem of Turán. *Canadian Journal of Mathematics*, 17:533–540, 1965.

[233] A. Mucherino, C. Lavor, L. Liberti, and N. Maculan, editors. *Distance Geometry: Theory, Methods, and Applications.* Springer, New York, 2013.

[234] M. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.

[235] K. Murty and S. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.

[236] O. Musin. The kissing number in four dimensions. *Annals of Mathematics*, 168:1–32, 2008.

[237] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, New York, 1988.

[238] A. Odlyzko and N. Sloane. New bounds on the number of unit spheres that can touch a unit sphere in n dimensions. *Journal of Combinatorial Theory A*, 26:210–214, 1979.

[239] M. Padberg. Classical cuts for mixed-integer programming and branch-and-cut. *Annals of Operations Research*, 139:321–352, 2005.

[240] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover, New York, 1998.

[241] P. Pardalos and S. Vavasis. Quadratic Programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1:15–22, 1991.

[242] P. Pardalos and S. Vavasis. Open questions in complexity theory for numerical optimization. *Mathematical Programming*, 57:337–339, 1992.

[243] P.M. Pardalos and H.E. Romeijn, editors. *Handbook of Global Optimization*, volume 2. Kluwer Academic Publishers, Dordrecht, 2002.

[244] P.M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is NP-hard. *Operations Research Letters*, 7(1):33–35, February 1988.

[245] A. Pfeffer. *Practical Probabilistic Programming*. Manning Publications, Shelter Island, NY, 2016.

[246] F. Pfender. Improved delsarte bounds for spherical codes in small dimensions. *Journal of Combinatorial Theory A*, 114(6):1133–1147, 2007.

[247] N.-T. Pham. Quantile regression in large energy datasets. Master's thesis, LIX, Ecole Poltyechnique, 2018.

[248] F. Potra and S. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124:281–302, 2000.

[249] C. Pugh. *Real Mathematical Analysis*. Springer, Berlin, 2002.

[250] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[251] D. Ratz and T. Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7:183–207, 1995.

[252] R. Reams, G. Chatham, W. Glunt, D. McDonald, and T. Hayden. Determining protein structure using the distance geometry program APA. *Computers and Chemistry*, 23:153–163, 1999.

[253] J. Renegar and M. Shub. Unified complexity analysis for Newton LP methods. *Mathematical Programming*, 53:1–16, 1992.

[254] Y. Roghozin. Small universal Turing machines. *Theoretical Computer Science*, 168:215–240, 1996.

[255] N. Rojas and F. Thomas. Application of distance geometry to tracing coupler curves of pin-jointed linkages. *Journal of Mechanisms and Robotics*, 5(2):021001, 2013.

[256] H.S. Ryoo and N.V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–566, 1995.

[257] M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal components analysis of a graph, and its relationships to spectral clustering. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proceedings of the European Conference in Machine Learning (ECML)*, volume 3201 of *LNAI*, pages 371–383, Berlin, 2004. Springer.

[258] G. Sagnol. *PICOS: A Python Interface for Conic Optimization Solvers*. Zuse Institut Berlin, 2016.

[259] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974.

[260] E. Salgado, A. Scozzari, F. Tardella, and L. Liberti. Alternating current optimal power flow with generator selection. In J. Lee, G. Rinaldi, and R. Mahjoub, editors, *Combinatorial Optimization (Proceedings of ISCO 2018)*, volume 10856 of *LNCS*, pages 364–375, 2018.

[261] A. Báez Sánchez and C. Lavor. On the estimation of unknown distances for a class of Euclidean distance matrix completion problems with interval data. *Linear Algebra and its Applications*, 592:287–305, 2020.

[262] T. Sarlós. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, volume 47 of *FOCS*, pages 143–152, Washington, 2006. IEEE.

[263] J. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. *Proceedings of 17th Allerton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[264] T. Schlick. *Molecular modelling and simulation: an interdisciplinary guide*. Springer, New York, 2002.

[265] F. Schoen. Two-phase methods for global optimization. In Pardalos and Romeijn [243], pages 151–177.

[266] I. Schoenberg. Remarks to Maurice Fréchet's article "Sur la définition axiomatique d'une classe d'espaces distanciés vectoriellement applicable sur l'espace de Hilbert". *Annals of Mathematics*, 36(3):724–732, 1935.

[267] I. Schoenberg. Positive definite functions on spheres. *Duke Mathematical Journal*, 9(1):96–108, 1942.

[268] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.

[269] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

[270] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning*. CUP, New York, 2014.

[271] C. Shannon. A universal Turing machine with two internal states. In C. Shannon and J. McCarthy, editors, *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 157–165, Princeton, NJ, 1956. Princeton University Press.

[272] J.P. Shectman and N.V. Sahinidis. A finite algorithm for global minimization of separable concave programs. *Journal of Global Optimization*, 12:1–36, 1998.

[273] H. Sherali and W. Adams. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems. *Discrete Applied Mathematics*, 52:83–106, 1994.

[274] H. Sherali and P. Driscoll. Evolution and state-of-the-art in integer programming. *Journal of Computational and Applied Mathematics*, 124:319–340, 2000.

[275] A. Singer. Angular synchronization by eigenvectors and semidefinite programming. *Applied and Computational Harmonic Analysis*, 30:20–36, 2011.

[276] E. Smith. *On the Optimal Design of Continuous Processes*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.

[277] E. Smith and C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 23:457–478, 1999.

[278] J.-L. Starck, M. Elad, and D. Donoho. Image decomposition via the combination of sparse representations and a variational approach. *IEEE Transactions on Image Processing*, 14(10):1570–1582, 2005.

[279] H. Steinhaus. Sur la division des corps matériels en parties. *Bulletin de l'Académie Polonaise des Sciences Cl. III*, 4(12):801–804, 1956.

[280] D. Steinley. K-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59:1–34, 2006.

[281] P. Szabłowski. Uniform distributions on spheres in finite dimensional l_α and their generalizations. *Journal of Multivariate Analysis*, 64:103–117, 1998.

[282] G. Szpiro. Newton and the kissing problem. *Plus magazine (online)*, 23, January 2003.

[283] H.A. Taha. *Operations Research: An Introduction*. MacMillan, New York, 1992.

[284] A. Tarski. A decision method for elementary algebra and geometry. Technical Report R-109, Rand Corporation, 1951.

[285] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

[286] M. Tawarmalani and N.V. Sahinidis. Convexification and global optimization of the pooling problem. *Mathematical Programming*, (submitted).

[287] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2322, 2000.

[288] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1937.

[289] H. Tuy. *Convex Analysis and Global Optimization*. Kluwer Academic Publishers, Dordrecht, 1998.

[290] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming A*, 130:249–294, 2011.

[291] P. Varignon. *Nouvelle Mécanique*. Claude Jombert, Paris, 1725.

[292] S. Vavasis. Quadratic programming is in NP. *Information Processing Letters*, 36:73–77, 1990.

[293] S. Vavasis. Complexity issues in global optimization: A survey. In Horst and Pardalos [143], pages 27–41.

[294] S. Vavasis and R. Zippel. Proving polynomial-time for sphere-constrained quadratic programming. Technical Report 90-1182, Dept. of Comp. Sci., Cornell University, 1990.

[295] S.A. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, Oxford, 1991.

[296] S. Vempala. *The Random Projection Method*. Number 65 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS, Providence, RI, 2004.

[297] S. Venkatasubramanian and Q. Wang. The Johnson-Lindenstrauss transform: An empirical study. In *Algorithm Engineering and Experiments*, volume 13 of *ALENEX*, pages 164–173, Providence, RI, 2011. SIAM.

[298] R. Vershynin. *High-dimensional probability*. CUP, Cambridge, 2018.

[299] R. Vidal, Y. Ma, and S. Sastry. *Generalized Principal Component Analysis*. Springer, New York, 2016.

[300] V. Visweswaran and C. A. Floudas. New formulations and branching strategies for the GOP algorithm. In I.E. Grossmann, editor, *Global Optimization in Engineering Design*. Kluwer Academic Publishers, Dordrecht, 1996.

[301] K. Vu, P.-L. Poirion, C. D'Ambrosio, and L. Liberti. Random projections for quadratic programs over a Euclidean ball. In A. Lodi and *et al.*, editors, *Integer Programming and Combinatorial Optimization (IPCO)*, volume 11480 of *LNCS*, pages 442–452, New York, 2019. Springer.

[302] K. Vu, P.-L. Poirion, and L. Liberti. Random projections for linear programming. *Mathematics of Operations Research*, 43(4):1051–1071, 2018.

[303] K. Vu, P.-L. Poirion, and L. Liberti. Gaussian random projections for Euclidean membership problems. *Discrete Applied Mathematics*, 253:93–102, 2019.

[304] Wikipedia. Principal component analysis, 2019. [Online; accessed 190726].

[305] Wikipedia. Random variable, 2020. [Online; accessed 201023].

[306] H.P. Williams. *Model Building in Mathematical Programming*. Wiley, Chichester, 4th edition, 1999.

[307] C. Witzgall. An all-integer programming algorithm with parabolic constraints. *Journal of the Society of Industrial and Applied Mathematics*, 11:855–870, 1963.

[308] L.A. Wolsey. *Integer Programming*. Wiley, New York, 1998.

[309] D. Woodruff. Sketching as a tool for linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.

[310] D. Wu, Z. Wu, and Y. Yuan. Rigid versus unique determination of protein structures with geometric buildup. *Optimization Letters*, 2(3):319–331, 2008.

[311] K. Wüthrich. Protein structure determination in solution by nuclear magnetic resonance spectroscopy. *Science*, 243:45–50, 1989.

[312] Y. Yemini. The positioning problem — a draft of an intermediate summary. In *Proceedings of the Conference on Distributed Sensor Networks*, pages 137–145, Pittsburgh, 1978. Carnegie-Mellon University.

[313] K. Yu, Z. Lu, and J. Stander. Quantile regression: applications and current research areas. *The Statistician*, 52(3):331–350, 2003.

[314] L. Zhang, M. Mahdavi, R. Jin, T. Yang, and S. Zhu. Recovering the optimal solution by dual random projection. In S. Shalev-Shwartz and I. Steinwart, editors, *Conference on Learning Theory (COLT)*, volume 30 of *Proceedings of Machine Learning Research*, pages 135–157. ⟨jmlr.org⟩, 2013.

[315] W. Zhu. Unsolvability of some optimization problems. *Applied Mathematics and Computation*, 174:921–926, 2006.

Index