École Polytechnique

Problems and exercises in Operations Research

Leo Liberti[1]

Last update: November 20, 2010

[1]Some exercises have been proposed by other authors, as detailed in the text. All the solutions, however, are by the author, who takes full responsibility for their accuracy (or lack thereof).

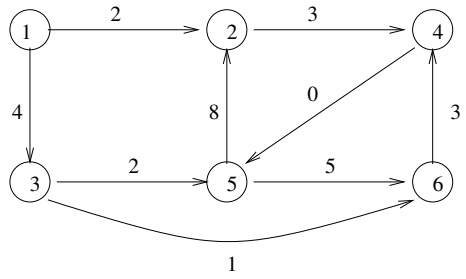
Contents

Chapter 1

Optimization on graphs

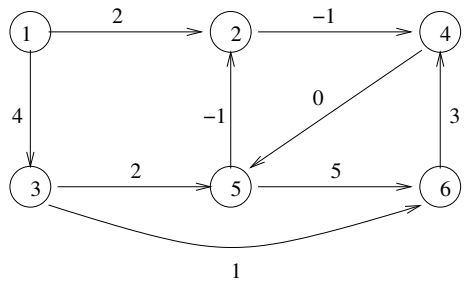
1.1 Dijkstra's algorithm

Use Dijkstra's algorithm to find the shortest path tree in the graph below using vertex 1 as source.



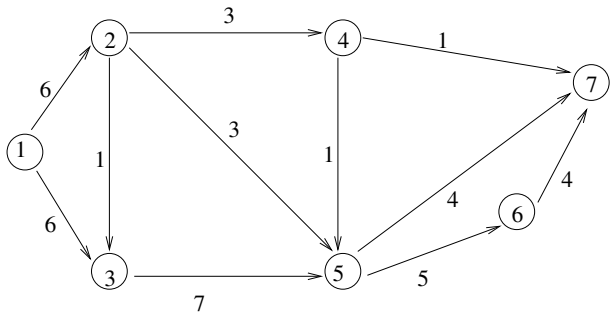
1.2 Bellman-Ford's algorithm

Check whether the graph below has negative cycles using Bellman-Ford's algorithm and 1 as a source vertex.



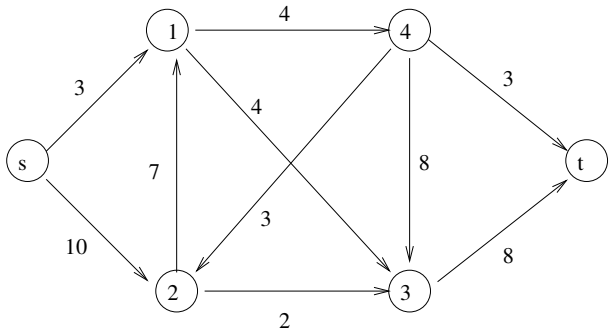
1.3 Maximum flow

Determine a maximum flow from node 1 to node 7 in the network $G = (V, A)$ below (the values on the arcs (i, j) are the arc capacities k_{ij}). Also find a cut having minimum capacity.



1.4 Minimum cut

Find the mimum cut in the graph below (arc capacities are marked on the arcs). What algorithm did you use?



1.5 Renewal plan

A small firm buys a new production machinery costing 12000 euros. In order to decrease maintenance costs, it is possible to sell the machinery second-hand and buy a new one. The maintenance costs and possible gains derived from selling the machinery second-hand are given below (for the next 5 years):

age (years)	costs (keuro)	gain (keuro)
0	2	-
1	4	7
2	5	6
3	9	2
4	12	1

Determine a renewal plan for the machinery which minimizes the total operation cost over a 5-year period. [*E. Amaldi, Politecnico di Milano*]

1.6 Connected subgraphs

Consider the complete undirected graph $K_n = (V, E)$ where $V = \{0, \ldots, n-1\}$ and $E = \{\{u, v\} \mid u, v \in V\}$. Let $U = \{i \mod n \mid i \geq 0\}$ and $F = \{\{i \mod n, (i+2) \mod n\} \mid i \geq 0\}$. Show that (a) $H = (U, F)$ is a subgraph of G and that (b) H is connected if and only if n is odd.

1.7 Strong connection

Consider the complete undirected graph $K_n = (V, E)$ and orient the edges arbitrarily into an arc set A so that for each vertex $v \in V$, $|\delta^+(v)| \geq 1$ and $|\delta^-(v)| \geq 1$. Show that the resulting directed graph $G = (V, A)$ is strongly connected.

Chapter 2

Linear programming

2.1 Graphical solution

Consider the problem

$$\begin{array}{lll} \min\limits_{x} & cx & \\ & Ax & \geq b \\ & x & \geq 0 \end{array}$$

where $x = (x_1, x_2)^T$, $c = (16, 25)$, $b = (4, 5, 9)^T$, and

$$A = \begin{pmatrix} 1 & 7 \\ 1 & 5 \\ 2 & 3 \end{pmatrix}.$$

1. Solve the problem graphically.
2. Write the problem in standard form. Identify B and N for the optimal vertex of the feasible polyhedron.

[*E. Amaldi, Politecnico di Milano*]

2.2 Geometry of LP

Consider the following LP problem.

$$\begin{array}{lll} \max z^* = 3x_1 + 2x_2 & & (*) \\ 2x_1 + x_2 & \leq 4 & (2.1) \\ -2x_1 + x_2 & \leq 2 & (2.2) \\ x_1 - x_2 & \leq 1 & (2.3) \\ x_1, x_2 & \geq 0. & \end{array}$$

1. Solve the problem graphically, specifying the variable values and z^* at the optimum.
2. Determine the bases associated to all the vertices of the feasible polyhedron.

3. Specify the sequence of the bases visited by the simplex algorithm to reach the solution (choose x_1 as the first variable entering the basis).
4. Determine the value of the reduced costs relative to the basic solutions associated to the following vertices, expressed as intersections of lines in $\mathbb{R}^2$: (a) (Eq. 2.1) $\cap$ (Eq. 2.2); (b) ((Eq. 2.1) $\cap$ (Eq. 2.3), where (Eq. i) is the equation obtained by inequality (i) replacing $\leq$ with $=$.
5. Verify geometrically that the objective function gradient can be expressed as a non-negative linear combination of the active constraint gradients only in the optimal vertex (keep in mind that the constraints must all be cast in the $\leq$ form, since the optimization direction is maximization — e.g. $x_1 \geq 0$ should be written as $-x_1 \leq 0$).
6. Say for which values of the RHS coefficient b_1 in constraint (2.1) the optimal basis does not change.
7. Say for which values of the objective function coefficients the optimal vertex is $((x_1 = 0) \cap (\text{Eq. } 2.2))$, where $x_1 = 0$ is the equation of the ordinate axis in $\mathbb{R}^2$.
8. For which values of the RHS coefficient associated to (2.2) the feasible region is (a) empty (b) contains only one solution?
9. For which values of the objective function coefficient c_1 there is more than one optimal solution?

[*M. Trubian, Università Statale di Milano*]

2.3 Simplex method

Solve the following LP problem using the simplex method:

$$\begin{aligned} \min z = \quad & x_1 - 2x_2 & \\ & 2x_1 + 3x_3 & = 1 \\ & 3x_1 + 2x_2 - x_3 & = 5 \\ & x_1, x_2, x_3 & \geq 0. \end{aligned}$$

Use the two-phase simplex method (the first phase identifies an initial basis) and Bland's rule (for a choice of the entering and exiting basis which ensures algorithmic convergence). [*E. Amaldi, Politecnico di Milano*]

2.4 Duality

What is the dual of the following LP problems?

1.
$$\left.\begin{array}{rrcl} \min_x & 3x_1 + 5x_2 - x_3 & & \\ & x_1 - x_2 + x_3 & \leq & 3 \\ & 2x_1 - 3x_2 & \leq & 4 \\ & x & \geq & 0 \end{array}\right\} \tag{2.4}$$

2.
$$\left.\begin{array}{rrcl} \min_x & x_1 - x_2 - x_3 & & \\ & -3x_1 - x_2 + x_3 & \leq & 3 \\ & 2x_1 - 3x_2 - 2x_3 & \geq & 4 \\ & x_1 - x_3 & = & 2 \\ & x_1 & \geq & 0 \\ & x_2 & \geq & 0 \end{array}\right\} \tag{2.5}$$

3.

$$\left.\begin{array}{rrcl} \max_x & x_1 - x_2 - 2x_3 + 3 & & \\ & -3x_1 - x_2 + x_3 & \leq & 3 \\ & 2x_1 - 3x_2 & \geq & 4x_3 \\ & x_1 - x_3 & = & x_2 \\ & x_1 & \geq & 0 \\ & x_2 & \leq & 0 \end{array}\right\} \tag{2.6}$$

2.5 Geometrical interpretation of the simplex algorithm

Solve the following problem

$$\left.\begin{array}{rrcl} \max_x & x_1 + x_2 & & \\ & -x_1 + x_2 & \leq & 1 \\ & 2x_1 + x_2 & \leq & 4 \\ & x_1 & \geq & 0 \\ & x_2 & \leq & 0 \end{array}\right\}$$

using the simplex algorithm. Start from the initial point $\bar{x} = (1, 0)$.

2.6 Complementary slackness

Consider the problem

$$\begin{array}{rrrrl} \max & 2x_1 & + & x_2 & \\ & x_1 & + & 2x_2 & \leq 14 \\ & 2x_1 & - & x_2 & \leq 10 \\ & x_1 & - & x_2 & \leq 3 \\ & x_1 & , & x_2 & \geq 0. \end{array}$$

1. Write the dual problem.
2. Verify that $\bar{x} = (\frac{20}{3}, \frac{11}{3})$ is a feasible solution.
3. Show that $\bar{x}$ is optimal using the complementary slackness theorem, and determine the optimal solution of the dual problem. [*Pietro Belotti, Carnegie Mellon University*]

2.7 Sensitivity analysis

Consider the problem:

$$\left.\begin{array}{rrcl} \min & x_1 - 5x_2 & & \\ & -x_1 + x_2 & \leq & 5 \\ & x_1 + 4x_2 & \leq & 40 \\ & 2x_1 + x_2 & \leq & 20 \\ & x_1, x_2 & \geq & 0. \end{array}\right\}$$

1. Check that the feasible solution $x^* = (4, 9)$ is also optimal.
2. Which among the constraints' right hand sides should be changed to decrease the optimal objective function value, supposing this change does not change the optimal basis? Should the change be a decrease or an increase?

2.8 Dual simplex method

Solve the following LP problem using the dual simplex method.

$$\begin{array}{lrl} \min & 3x_1 + 4x_2 + 5x_3 & \\ & 2x_1 + 2x_2 + x_3 & \geq 6 \\ & x_1 + 2x_2 + 3x_3 & \geq 5 \\ & x_1, x_2, x_3 & \geq 0. \end{array}$$

What are the advantages with respect to the primal simplex method?

Chapter 3

Integer programming

3.1 Piecewise linear objective

Reformulate the problem $\min\{f(x) \mid x \in \mathbb{R}_{\geq 0}\}$, where:

$$f(x) = \begin{cases} -x+1 & 0 \leq x < 1 \\ x-1 & 1 \leq x < 2 \\ \frac{1}{2}x & 2 \leq x \leq 3 \end{cases}$$

as a Mixed-Integer Linear Programming problem.

3.2 Gomory cuts

Solve the following problem using Gomory's cutting plane algorithm.

$$\left.\begin{array}{lrcl} \min & x_1 - 2x_2 & & \\ & -4x_1 + 6x_2 & \leq & 9 \\ & x_1 + x_2 & \leq & 4 \\ & x \geq 0 & , & x \in \mathbb{Z}^2 \end{array}\right\}$$

[Bertsimas & Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, Belmont, 1997.]

3.3 Branch and Bound I

Solve the following problem using the Branch and Bound algorithm.

$$\left.\begin{array}{lrcl} \max & 2x_1 + 3x_2 & & \\ & x_1 + 2x_2 & \leq & 3 \\ & 6x_1 + 8x_2 & \leq & 15 \\ & x_1, x_2 & \in & \mathbb{Z}_+ \end{array}\right\}$$

Each LP subproblem may be solved graphically.

3.4 Branch and Bound II

Solve the following problem using the Branch and Bound algorithm.

$$\begin{aligned} \max z^* \quad &= 3x_1 + 4x_2 \\ 2x_1 + x_2 \quad &\leq 6 \\ 2x_1 + 3x_2 \quad &\leq 9 \\ x_1, x_2 \quad &\geq 0, \text{ intere} \end{aligned}$$

Each LP subproblem may be solved graphically.

3.5 Knapsack Branch and Bound

An investment bank has a total budget of 14 million euros, and can make 4 types of investments (numbered 1,2,3,4). The following tables specifies the amount to be invested and the net revenue for each investment. Each investment must be made in full if made at all.

Investment	1	2	3	4
Amount	5	7	4	3
Net revenue	16	22	12	8

Formulate an integer linear program to maximize the total net revenue. Suggest a way, beside the simplex algorithm, to solve the continuous relaxation of the problem, and use it within a Branch-and-Bound algorithm to solve the problem. [*E. Amaldi, Politecnico di Milano*]

Chapter 4

Easy modelling problems

4.1 Compact storage of similar sequences

One practical problem encountered during the DNA mapping process is that of compactly storing extremely long DNA sequences of the same length which do not differ greatly. We consider here a simplified version of the problem with sequences of 2 symbols only (0 and 1). The *Hamming distance* between two sequences $a, b \in \mathbb{F}_2^n$ is defined as $\sum_{i=1}^{n} |a_i - b_i|$, i.e. the number of bits which should be flipped to transform a into b. For example, on the following set of 6 sequences below, the distance matrix is as follows:

1. `011100011101`
2. `101101011001`
3. `110100111001`
4. `101001111101`
5. `100100111101`
6. `010101011100`

	1	2	3	4	5	6
1	0	4	4	5	4	3
2	-	0	4	3	4	5
3	-	-	0	5	2	5
4	-	-	-	0	3	6
5	-	-	-	-	0	5
6	-	-	-	-	-	0

As long as the Hamming distances are not too large, a compact storage scheme can be envisaged where we only store one complete sequence and all the differences which allow the reconstruction of the other sequences. Explain how this problem can be formulated to find a spanning tree of minimum cost in a graph. Solve the problem for the instance given above. [*E. Amaldi, Politecnico di Milano*]

4.2 Communication of secret messages

Given a communication network the probability that a secret message is intercepted along a link connecting node i to j is p_{ij}. Explain how you can model the problem of broadcasting the secret message to every node minimizing the interception probability as a minimum spanning tree problem on a graph. [*E. Amaldi, Politecnico di Milano*]

4.3 Mixed production

A firm is planning the production of 3 products A_1,A_2,A_3. In a month production can be active for 22 days. In the following tables are given: maximum demands (units=100kg), price (\$/100Kg), production costs (per 100Kg of product), and production quotas (maximum amount of 100kg units of product that would be produced in a day if all production lines were dedicated to the product).

Product	A_1	A_2	A_3
Maximum demand	5300	4500	5400
Selling price	\$124	\$109	\$115
Production cost	\$73.30	\$52.90	\$65.40
Production quota	500	450	550

1. Formulate an AMPL model to determine the production plan to maximize the total income.
2. Change the mathematical program and the AMPL model to cater for a fixed activation cost on the production line, as follows:

Product	A_1	A_2	A_3
Activation cost	\$170000	\$150000	\$100000

3. Change the mathematical program and the AMPL model to cater for both the fixed activation cost and for a minimum production batch:

Product	A_1	A_2	A_3
Minimum batch	20	20	16

[*E. Amaldi, Politecnico di Milano*]

4.4 Production planning

A firm is planning the production of 3 products A_1, A_2, A_3 over a time horizon of 4 months (january to april). Demand for the products over the months is as follows:

Demand	January	February	March	April
A_1	5300	1200	7400	5300
A_2	4500	5400	6500	7200
A_3	4400	6700	12500	13200

Prices, production costs, production quotas, activation costs and minimum batches (see Ex. 4.3 for definitions of these quantities) are:

Product	A_1	A_2	A_3
Unit prices	\$124	\$109	\$115
Activation costs	\$150000	\$150000	\$100000
Production costs	\$73.30	\$52.90	\$65.40
Production quotas	500	450	550
Minimum batches	20	20	16

There are 23 productive days in january, 20 in february, 23 in march and 22 in april. The activation status of a production line can be changed every month. Minimum batches are monthly.

Moreover, storage space can be rented at monthly rates of \$3.50 for A_1, \$4.00 for A_2 and \$3.00 for A_3. Each product takes the same amount of storage space. The total available volume is 800 units.

Write a mathematical program to maximize the income, and solve it with AMPL. [*E. Amaldi, Politecnico di Milano*]

4.5 Transportation

An Italian transportation firm should carry some empty containers from its 6 stores (in Verona, Perugia, Rome, Pescara, Taranto and Lamezia) to the main national ports (Genoa, Venice, Ancona, Naples, Bari). The container stocks at the stores are the following:

	Empty containers
Verona	10
Perugia	12
Rome	20
Pescara	24
Taranto	18
Lamezia	40

The demands at the ports are as follows:

	Container demand
Genoa	20
Venice	15
Ancona	25
Naples	33
Bari	21

Transportation is carried out by a fleet of lorries. The transportation cost for each container is proportional to the distance travelled by the lorry, and amounts to 30 euro / km. Every lorry can carry at most 2 containers. Distances are as follows:

	Genoa	Venice	Ancona	Naples	Bari
Verona	290 km	115 km	355 km	715 km	810 km
Perugia	380 km	340 km	165 km	380 km	610 km
Rome	505 km	530 km	285 km	220 km	450 km
Pescara	655 km	450 km	155 km	240 km	315 km
Taranto	1010 km	840 km	550 km	305 km	95 km
Lamezia	1072 km	1097 km	747 km	372 km	333 km

Write a mathematical program to find the minimal cost transportation policy and solve it with AMPL. [*E. Amaldi, Politecnico di Milano*]

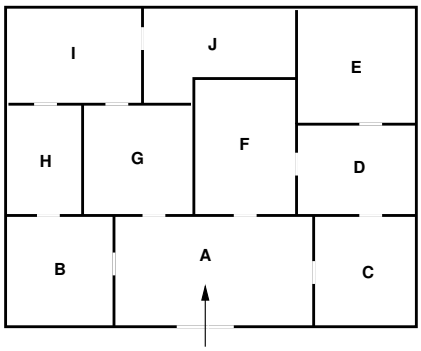
4.6 Project planning with precedences

A project consists of the following 7 activities, whose length in days is given in brackets: A (4), B (3), C (5), D (2), E (10), F (10), G (1). The following precedences are also given: $A \to G, D$; $E, G \to F$;

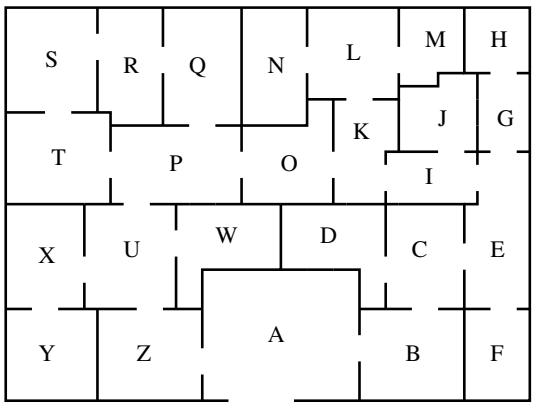
$D, F \to C$; $F \to B$. Each day of work costs 1000 euros; furthermore a special machinery must be rented from the beginning of activity A to the end of activity B at a daily cost of 5000 euros. Formulate this as an LP problem and suggest an algorithm for solving it. [*F. Malucelli, Politecnico di Milano*]

4.7 Museum guards

A museum director must decide how many guards should be employed to control a new wing. Budget cuts have forced him to station guards at each door, guarding two rooms at once. Formulate a mathematical program to minimize the number of guards. Solve the problem on the map below using AMPL.



Also solve the problem on the following map.



[*P. Belotti, Carnegie Mellon University*]

4.8 Inheritance

A rich aristocrat passes away, leaving the following legacy:

- A Caillebotte picture: 25000$
- A bust of Diocletian: 5000$
- A Yuan dinasty chinese vase: 20000$

- A 911 Porsche: 40000$
- Three diamonds: 12000$ each
- A Louis XV sofa: 3000$
- Two very precious Jack Russell race dogs: 3000$ each (the will asserts that they may not be separated)
- A sculpture dated 200 A.D.: 10000$
- A sailing boat: 15000$
- A Harley Davidson motorbike: 10000$
- A piece of furniture that once belonged to Cavour: 13.000$,

which must be shared between the two sons. What is the partition that minimizes the difference between the values of the two parts? Formulate a mathematical program and solve it with AMPL. [*P. Belotti, Carnegie Mellon*]

4.9 Carelland

The independent state of Carelland mainly exports four goods: steel, engines, electronic components and plastics. The Chancellor of the Exchequer (a.k.a. the minister of economy) of Carelland wants to maximize exports and minimize imports. The unit prices on the world markets for steel, engines, electronics and plastics, expressed in the local currency (the Klunz) are, respectively: 500, 1500, 300, 1200. Producing 1 steel unit requires 0.02 engine units, 0.01 plastics units, 250 Klunz in other imported goods and 6 man-months of work. Producing 1 engine unit requires 0.8 steel units, 0.15 electronics units, 0.11 plastics units, 300 Klunz in imported goods and 1 man-year. One electronics unit requires: 0.01 steel units, 0.01 engine units, 0.05 plastics units, 50 Klunz in imported goods and 6 man-months. One plastics unit requires: 0.03 engine units, 0.2 steel units, 0.05 electronics units, 300 Klunz in imported goods and 2 man-years. Engine production is limited to 650000 units, plastics production to 60000 units. The total available workforce is 830000 each year. Steel, engines, electronics and plastics cannot be imported. Write a mathematical program that maximizes the gross internal product and solve the problem with AMPL. [*G. Carello, Politecnico di Milano*]

4.10 CPU Scheduling

10 tasks must be run on 3 CPUs at 1.33, 2 and 2.66 GHz (each processor can run only one task at a time). The number of elementary operations of the tasks (expressed in billions of instructions (BI)) is as follows:

process	1	2	3	4	5	6	7
BI	1.1	2.1	3	1	0.7	5	3

Schedule tasks to processors so that the completion time of the last task is minimized. Solve the problem with AMPL.

4.11 Dyeing plant

A fabric dyeing plant has 3 dyeing baths. Each batch of fabric must be dyed in each bath in the order: first, second, third bath. The plant must colour five batches of fabric of different sizes. Dyeing batch i in bath j takes a time s_{ij} expressed in hours in the matrix below:

$$\begin{pmatrix} 3 & 1 & 1 \\ 2 & 1.5 & 1 \\ 3 & 1.2 & 1.3 \\ 2 & 2 & 2 \\ 2.1 & 2 & 3 \end{pmatrix}.$$

Schedule the dyeing operations in the baths so that the ending time of the last batch is minimized.

4.12 Parking

On Dantzig Street cars can be parked on both sides of the street. Mr. Edmonds, who lives at number 1, is organizing a party for around 30 people, who will arrive in 15 cars. The length of the i-th car is λ_i, expressed in meters as follows:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
λ_i	4	4.5	5	4.1	2.4	5.2	3.7	3.5	3.2	4.5	2.3	3.3	3.8	4.6	3

In order to avoid bothering the neighbours, Mr. Edmonds would like to arrange the parking on both sides of the street so that the length of the street occupied by his friends' cars should be minimum. Give a mathematical programming formulation and solve the problem with AMPL.

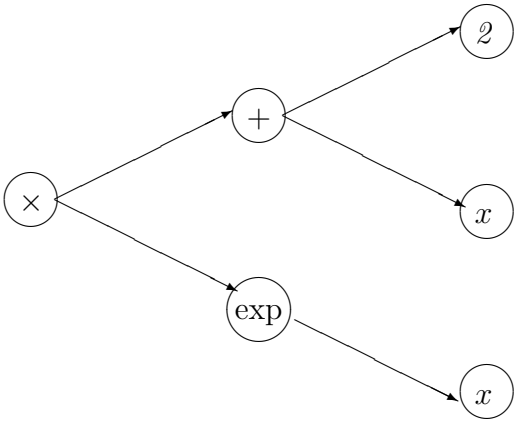
How does the program change if on exactly one of the street sides the cars should not occupy more than 15m?

Chapter 5

Difficult modelling problems

5.1 Checksum

An *expression parser* is a program that reads mathematical expressions (input by the user as strings) and evaluates their values on a set of variable values. This is done by representing the mathematical expression as a directed binary tree. The leaf nodes represent variables or constants; the other nodes represent binary (or unary) operators such as arithmetic (+, -, *, /, power) or transcendental (sin, cos, tan, log, exp) operators. The unary operators are represented by a node with only one arc in its outgoing star, whereas the binary operators have two arcs. The figure below is the binary expression tree for $(x + 2)e^x$.



The expression parser consists of several subroutines.

- `main()`: the program entry point;
- `parse()`: reads the string containing the mathematical expression and transforms it into a binary expression tree;
- `gettoken()`: returns and deletes the next semantic token (variable, constant, operator, brackets) from the mathematical expression string buffer;
- `ungettoken()`: pushes the current semantic token back in the mathematical expression string buffer;
- `readexpr()`: reads the operators with precedence 4 (lowest: +,-);

- `readterm()`: reads the operators with precedence 3 (*, /);
- `readpower()`: reads the operators with precedence 2 (power);
- `readprimitive()`: reads the operators of precedence 1 (functions, expressions in brackets);
- `sum(term a, term b)`: make a tree $+ \begin{array}{cc} \nearrow & a \\ \searrow & b \end{array}$;
- `difference(term a, term b)`: make a tree $- \begin{array}{cc} \nearrow & a \\ \searrow & b \end{array}$;
- `product(term a, term b)`: make a tree $* \begin{array}{cc} \nearrow & a \\ \searrow & b \end{array}$;
- `fraction(term a, term b)`: make a tree $/ \begin{array}{cc} \nearrow & a \\ \searrow & b \end{array}$;
- `power(term a, term b)`: make a tree $\wedge \begin{array}{cc} \nearrow & a \\ \searrow & b \end{array}$;
- `minus(term a)`: make a tree $- \rightarrow a$;
- `logarithm(term a)`: make a tree make a tree $\log \rightarrow a$;
- `exponential(term a)`: make a tree make a tree $\exp \rightarrow a$;
- `sine(term a)`: make a tree make a tree $\sin \rightarrow a$;
- `cosine(term a)`: make a tree make a tree $\cos \rightarrow a$;
- `tangent(term a)`: make a tree make a tree $\tan \rightarrow a$;
- `variable(var x)`: make a leaf node x;
- `number(double d)`: make a leaf node d;
- `readdata()`: reads a table of variable values from a file;
- `evaluate()`: computes the value of the binary tree when substituting each variable with the corresponding value;
- `printresult()`: print the results.

For each function we give the list of called functions and the quantity of data to be passed during the call.

- main: readdata (64KB), parse (2KB), evaluate (66KB), printresult(64KB)
- evaluate: evaluate (3KB)
- parse: gettoken (0.1KB), readexpr (1KB)
- readprimitive: gettoken (0.1KB), variable (0.5KB), number (0.2KB), logarithm (1KB), exponential (1KB), sine (1KB), cosine (1KB), tangent (1KB), minus (1KB), readexpr (2KB)
- readpower: power (2KB), readprimitive (1KB)
- readterm: readpower (2KB), product (2KB), fraction (2KB)
- readexpr: readterm (2KB), sum (2KB), difference (2KB)

- gettoken: ungettoken (0.1KB)

Each function call requires a bidirectional data exchange between the calling and the called function. In order to guarantee data integrity during the function call, we require that a checksum operation be performed on the data exchanged between the pair (calling function, called function). Such pairs are called *checksum pairs*. Since the checksum operation is costly in terms of CPU time, we limit these operations so that no function may be involved in more than one checksum pair. Naturally though, we would like to maximize the total quantity of data undergoing a checksum.

1. Formulate a mathematical program to solve the problem, and solve the given instance with AMPL.

2. Modify the model to ensure that `readprimitive()` and `readexpr()` are a checksum pair. How does the solution change?

5.2 Eight queens

Formulate an integer linear program to solve the problem of positioning eight queens on the chessboard so that no queen is under threat by any other queen. Solve this program with AMPL. [*P. Belotti, Carnegie Mellon University*]

5.3 Production management

A firm which produces only one type of product has 40 workers. Each one of them produces 20 units per month. The demand varies during the semester according to the following table:

Month	1	2	3	4	5	6
Demand (units)	700	600	500	800	900	800

In order to increase/decrease production depending on the demand, the firm can offer some (paid) extra working time (each worker can produce at most 6 additional units per month at unit cost of 5 euros), use a storage space (10 euros/month per unit of product), employ or dismiss personnel (the number of employed workers can vary by at most ± 5 per month at an additional price of 500 euros per employment and 700 euros per dismissal).

At the outset, the storage space is empty, and we require that it should be empty at the end of the semester. Formulate a mathematical program that maximizes the revenues, and solve it with AMPL. How does the objective function change when all variables are relaxed to be continuous? [*E. Amaldi, Politecnico di Milano*]

5.4 The travelling salesman problem

A travelling salesman must visit 7 customers in 7 different locations whose (symmetric) distance matrix is:

	1	2	3	4	5	6	7
1	-	86	49	57	31	69	50
2		-	68	79	93	24	5
3			-	16	7	72	67
4				-	90	69	1
5					-	86	59
6						-	81

Formulate a mathematical program to determine a visit sequence starting at ending at location 1, which minimizes the travelled distance, and solve it with AMPL. Knowing that the distances obey a triangular inequality and are symmetric, propose a suitable heuristic method.

5.5 Optimal rocket control 1

A rocket of mass m is launched at sea level and has to reach an altitude H within time T. Let $y(t)$ be the altitude of the rocket at time t and $u(t)$ the force acting on the rocket at time t in the vertical direction. Assume $u(t)$ may not exceed a given value b, that the rocket has constant mass m throughout, and that the gravity acceleration g is constant in the interval $[0, H]$. Discretizing time $t \in [0, T]$ in n intervals, propose a linear program to determine, for each $k \leq n$, the force $u(t_k)$ acting on the rocket so that the total consumed energy is minimum. Solve the problem with AMPL with the following data: $m = 2140$kg, $H = 23$km, $T = 1$min, $b = 60000$N, $n = 20$.

5.6 Double monopoly

In 2021, all the world assets are held by the AA (Antidemocratic Authorities) bank. In 2022, a heroical judge manages to apply the ancient (but still existing) anti-trust regulations to the AA bank, right before being brutally decapitated by its vicious corporate killers. A double monopoly situation is thus established with the birth of the BB (Bastard Business) bank. In a whirlpool of flagrantly illegal acts that are approved thanks to the general public being hypnotized by the highly popular television programme "The International Big Brother 26", the AA bank manages to insert a codicil in the anti-trust law that ensures that BB may not draw any customer without the prior approval of AA. However, in an era where the conflict of interest is not only accepted but even applauded, AA's lawyers are the same as BB's, so the same codicil is inserted in the law in favour of BB. When the law is finally enforced, as usually happens when two big competitors share the market, AA and BB get together and develop some plans to make as much money as they can without damaging each other too much. It decided that each bank must, independently of the codicil, make at least one investment. When operations begin, the following situation occurs: there are 6 big customers and a myriad of small private individuals with their piddly savings accounts which are progressively deprived of everything they have and which can, to the end of this exercise (but let's face it — to every other end, too), be entirely disregarded. The effect of the codicil and of the bank agreement brings about a situation where the quantity of money earned by AA in a given investment is exactly the same as the quantity of money lost by BB for not having made the same investment (and vice versa). The following 6×6 matrix $A = (a_{ij})$ represents the revenues and losses of the two banks:

$$A = \begin{pmatrix} 1 & -2 & 1 & 3 & -5 & 2 \\ 4 & 1 & 1 & 3 & 2 & -1 \\ 1 & 10 & -6 & 3 & 1 & 4 \\ 2 & 2 & 1 & 3 & 3 & -8 \\ -12 & 1 & 3 & -4 & 2 & 1 \\ 3 & 3 & -1 & 4 & 2 & 2 \end{pmatrix}.$$

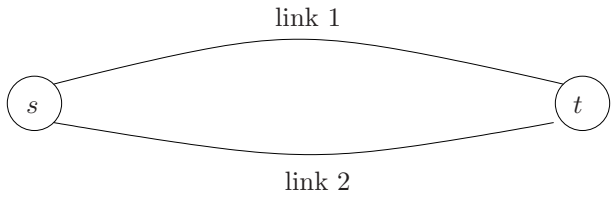
a_{ij} represents the revenue of AA and loss of BB if AA invests *all* of its budget in client i and BB *all* of its budget in client j. The banks may naturally decide to get more than one client, but without exceeding their budgets (which amount to exactly the same amount: 1 fantastillion dollars). Even to the bank managers it appears evident that AA's optimal strategy is to maximize the expected revenues and BB's to minimize the expected loss. Write two linear programs: one to model the expected revenues of AA and the other to model the losses of BB. Comment on the relations between the models.

Chapter 6

Telecommunication networks

6.1 Packet routing

There are n data flows that must be routed from a source node s to a destination node t following one of two possible links, with capacity $u_1 = 1$ and $u_2 = 2$ respectively.



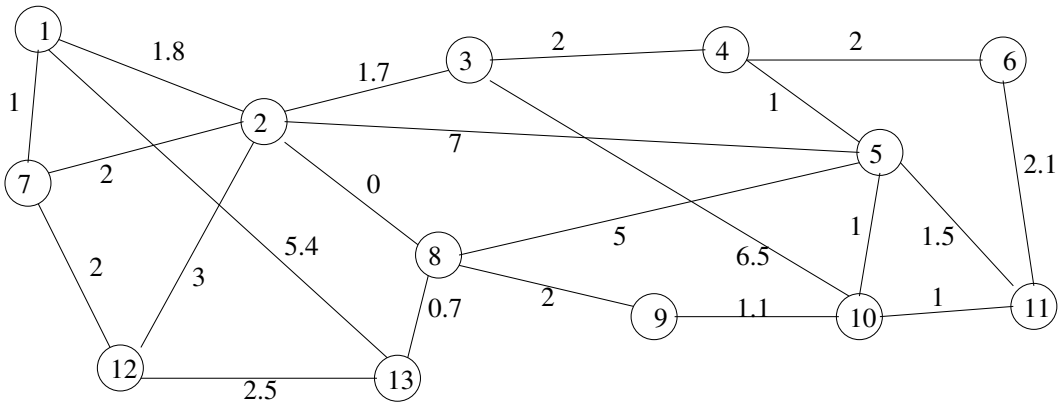
The company handling link 2 is 30% more expensive than the company handling link 1. The table below specifies the demands to be routed and the cost on link 1.

Demand	Required capacity (Mbps)	Cost on link 1
1	0.3	200
2	0.2	200
3	0.4	250
4	0.1	150
5	0.2	200
6	0.2	200
7	0.5	700
8	0.1	150
9	0.1	150
10	0.6	900

Formulate a mathematical program to minimize the routing cost of all the demands. How would you change the model to generalize to a situation with m possible parallel links between s and t?

6.2 Network Design

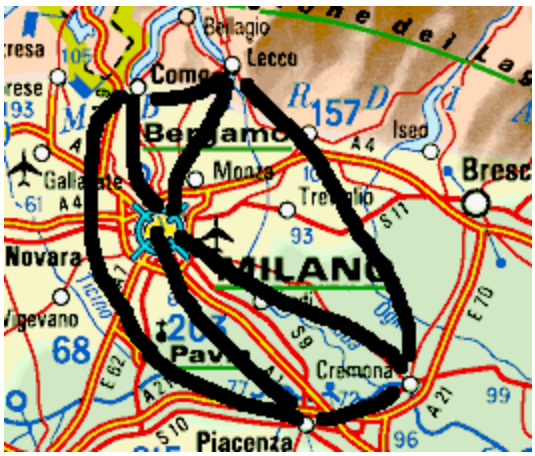
Orange is the unique owner and handler of the telecom network in the figure below.



The costs on the links are proportional to the distances $d(i, j)$ between the nodes, expressed in units of 10km. Because of anti-trust regulations, Orange must delegate to SFR and Bouygtel two subnetworks each having at least two nodes (with Orange handling the third part). Orange therefore needs to design a backbone network to connect the three subnetworks. Transforming an existing link into a backbone link costs $c = 25$ euros/km. Formulate a mathematical program to minimize the cost of implementing a backbone connecting the three subnetworks, and solve it with AMPL. How does the solution change if Orange decides to partition its network in 4 subnetworks instead of 3?

6.3 Network Routing

The main telephone network backbone connecting the different campuses of the Politecnico di Milano (at Milano, Como, Lecco, Piacenza, Cremona) has grown over the years to its present state without a clear organized plan. Politecnico asked its main network provider to optimize the routing of all the traffic demands to see whether the installed capacity is excessive. The network topology is as depicted below.



For each link there is a pair (u, c) where u is the link capacity (Mb/s) and c the link length (km).

1. Como, Lecco: (200, 30)
2. Como, Milano: (260, 50)
3. Como, Piacenza: (200, 110)
4. Lecco, Milano: (260, 55)

5. Lecco, Cremona: (200, 150)
6. Milano, Piacenza: (260, 72)
7. Milano, Cremona: (260, 90)
8. Piacenza, Cremona: (200, 100)

The traffic demands to be routed (in Mb/s) are as follows.

1. Como, Lecco: 20
2. Como, Piacenza: 30
3. Milano, Como: 50
4. Milano, Lecco: 40
5. Milano, Piacenza: 60
6. Milano, Cremona: 25
7. Cremona, Lecco: 35
8. Cremona, Piacenza: 30

The current routing is as given below.

1. Como → Lecco
2. Como → Milano → Piacenza
3. Milano → Lecco → Como
4. Milano → Como → Lecco
5. Milano → Piacenza
6. Milano → Piacenza → Cremona
7. Cremona → Milano → Piacenza → Como → Lecco
8. Cremona → Milano → Como → Lecco → Milano → Piacenza

Formulate a mathematical program to find an optimal network routing. [with *P. Belotti, Carnegie Mellon University*]

Chapter 7

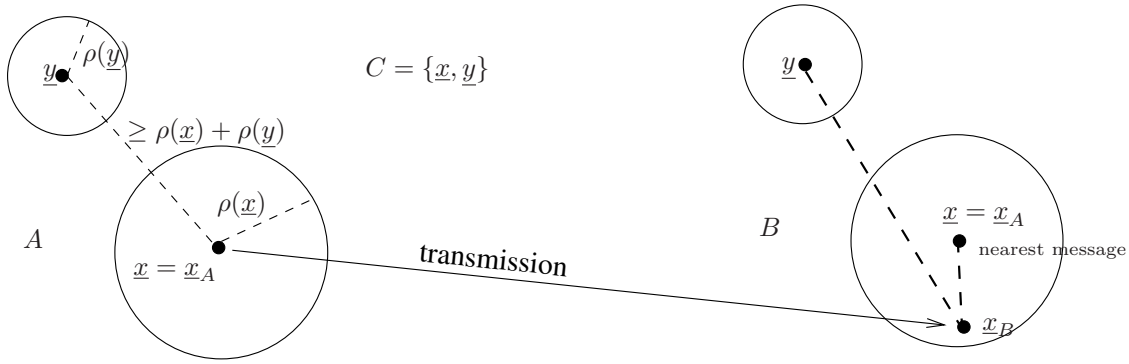
Nonlinear programming

7.1 Error correcting codes

A message sent by A to B is represented by a vector $\underline{z} = (\underline{z}_1, \ldots, \underline{z}_m) \in \mathbb{R}^m$. An *Error Correcting Code* (ECC) is a finite set C (with $|C| = n$) of messages with an associated function $\rho : C \to \mathbb{R}$, such that for each pair of distinct messages $\underline{x}, \underline{y} \in C$ the inequality $||\underline{x} - \underline{y}|| \geq \rho(\underline{x}) + \rho(\underline{y})$ holds. The *correction radius* of code C is given by

$$R_C = \min_{\underline{x} \in C} \rho(\underline{x}),$$

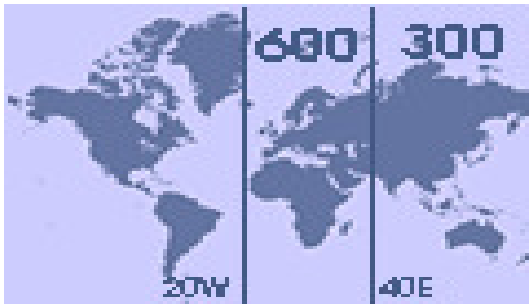
and represents the maximum error that can be corrected by the code. Assume both A and B know the code C and that their communication line is faulty. A message $\underline{x}_A \in C$ sent by A gets to B as $\underline{x}_B \notin C$ because of the faults. Supposing the error in $\underline{x}_B$ is strictly less than R_C, B is able to reconstruct the original message $\underline{x}_A$ looking for the message $\underline{x} \in C$ closest to $\underline{x}_B$ as in the figure below.



Formulate a (nonlinear) mathematical program to build an ECC C of 10 messages in $\mathbb{R}^{12}$ (where all message components are in $[0, 1]$) so that the correction radius is maximized.

7.2 Airplane maintenance

Boeing needs to build 5 maintenance centers for the Euro-Asian area. The construction cost for each center is 300 million euros in the European area (between 20°W and 40°E) and 150 million euros in the Asian area (between 40°E and 160°E), as shown below.



Each center can service up to 60 airplanes each year. The centers should service the airports with the highest number of Boeing customers, as detailed in the table below (airport name, geographical coordinates, expected number of airplanes/year needing maintenance).

Airport	**Coordinates**		**N. of planes**
London Heathrow	51°N	0°W	30
Frankfurt	51°N	8°E	35
Lisboa	38°N	9°W	12
Zürich	47°N	8°E	18
Roma Fiumicino	41°N	12°E	13
Abu Dhabi	24°N	54°E	8
Moskva Sheremetyevo	55°N	37°E	15
Vladivostok	43°N	132°E	7
Sydney	33°S	151°E	32
Tokyo	35°N	139°E	40
Johannesburg	26°S	28°E	11
New Dehli	28°N	77°E	20

The total cost is given by the construction cost plus the expected servicing cost, which depends linearly on the distance an airplane needs to travel to reach the maintenance center (weighted by 50 euro/km). We assume earth is a perfect sphere, so that the shortest distance between two points with geographical coordinates (δ_1, φ_1) and (δ_2, φ_2) is given by:

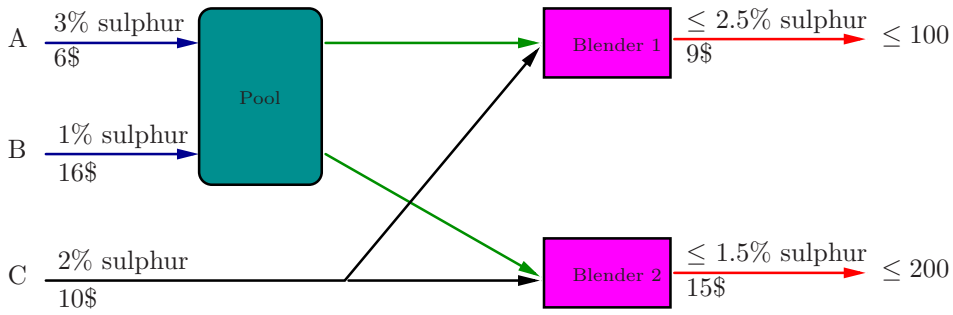
$$d(\delta_1, \varphi_1, \delta_2, \varphi_2) = 2r \operatorname{asin}\sqrt{\sin^2\left(\frac{\delta_1 - \delta_2}{2}\right) + \cos\delta_1 \cos\delta_2 \sin^2\left(\frac{\varphi_1 - \varphi_2}{2}\right)},$$

where r, the earth radius, is 6371km.

Formulate a (nonlinear) mathematical program to minimize the operation costs.

7.3 Pooling problem

The blending plant in a refinery is composed by a pool and two mixers as shown in the picture below.



Two types of crude A, B whose unitary cost and sulphur percentage are 6\$, 16\$ and 3%, 1% respectively enter the pool through two input valves. The output of the pool is then carried to the mixers, together with some crude of type C (unit cost 10%, sulphur percentage 2%) which enters the plant through a third input valve. Mixer 1 must produce petrol containing at most 2.5% sulphur, which will be sold at a unitary price of 9\$. Mixer 2 must produce a more refined petrol containing at most 1.5% sulphur, sold at a unitary price of 15\$. The maximum market demand for the refined petrol 1 is of 100 units, and of 200 units for refined petrol 2. Formulate a (nonlinear) mathematical program to maximize revenues. Does your program describe a convex programming problem? Propose and implement (with AMPL and CPLEX) a heuristic algorithm to find a feasible, and hopefully good, solution to the problem.

[Haverly, *Studies of the behaviour of recursion for the pooling problem*, ACM SIGMAP Bulletin **25**:19-28, 1978]

7.4 Optimal rocket control 2

A rocket of mass m is launched at sea level and has to reach an altitude H within time T. Let $y(t)$ be the altitude of the rocket at time t and $u(t)$ the force acting on the rocket at time t in the vertical direction. Assume: (a) $u(t)$ may not exceed a given value b; (b) the rocket has initial mass $m = m_0 + c$ (where c is the mass of the fuel) and loses $\alpha u(t)$ kg of mass (burnt fuel) each second; (c) the gravity acceleration g is constant in the interval $[0, H]$. Discretizing time $t \in [0, T]$ in n intervals, propose a (nonlinear) mathematical program to determine, for each $k \leq n$, the force $u(t_k)$ acting on the rocket so that the total consumed energy is minimum.

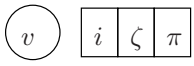
Solutions

Chapter 8

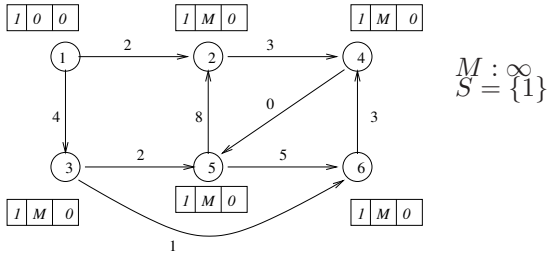
Optimization on graphs: Solutions

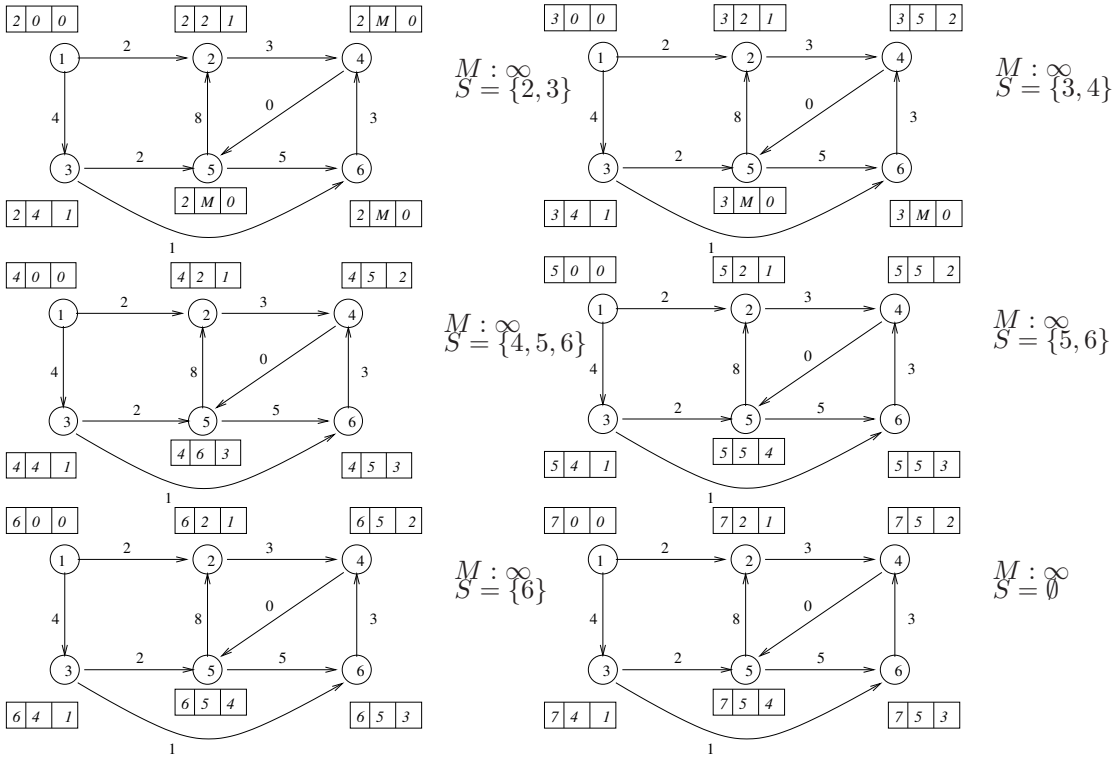
8.1 Dijkstra's algorithm: Solution

Dijkstra's algorithm is used for computing shortest paths from one root node to every other node of a directed or undirected graph without cycles of negative cost. To each node $v \in V$ in the graph we associated a label $\zeta(v)$ (initially set at ∞). For each node v we store the node $\pi(v)$ which precedes it in the shortest path. Graphically, we indicate labels and predecessors at each iteration directly on the graph as follows:



In the above picture, v is the vertex, i the iteration of the algorithm, ζ the label and π the cost. We initialize the set of reached vertices $S = \{r\}$ (a vertex v is *reached* if it is a candidate for being chosen as a settled vertex at the next iteration; a vertex v is *settled* if a shortest path to from r to v has been found). At each iteration, we settle the node h in S with minimum label $\zeta(h)$. For each node $w \in \delta^+(h)$, if $\zeta(h) + c_{hw} < \zeta(w)$ we update $\zeta(w) \to \zeta(h) + c_{hw}$ and $\pi(w) \to h$, where c_{hw} is the cost of the arc (h, w). We remove h from S and add w to S if it is not already in S.





8.2 Bellman-Ford algorithm: Solution

The Bellman-Ford algorithm also works in presence of negative cost cycles (it works in the sense that if there is a negative cycle it is detected and the algorithm terminates). We solve the problem with the same notation as Ex. 8.1, save that we indicate the reached vertices by Q as a reminder that Q is not simply a set but a FIFO (first in, first out) queue. At each iteration we choose the oldest node $h \in Q$ and we explore its outgoing star as in Dijkstra's algorithm. Since each node may not be visited more than $n-1$ times if there are no cycles of negative cost, it suffices to keep track of the number of times each node is visited. Should a node be visited n times, then there is a cycle of negative cost in the graph. More precisely, Q turns out to be:

$$1, 2, 3, 4, 5, 6, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2.$$

Since vertex 2 was visited 6 times and $n = 6$, the cycle $(2, 4, 5, 2)$ has negative cycle and the algorithm terminates.

8.3 Maximum flow: Solution

A *network* is a weighted directed graph with a source node $s \in V$ and a destination node $t \in V$; we associate a capacity $k_{ij} \geq 0$ to each arc $(i, j) \in A$. A *flow from s to t* is a function $x : A \to \mathbb{R}$ such that $0 \leq x(i, j) \leq k_{ij}$ for each $(i, j) \in A$ (we also denote $x(i, j)$ as x_{ij}). A flow is *feasible* if for each vertex $i \in V \setminus \{s, t\}$ we have

$$\sum_{j \in \delta^-(i)} x_{ji} = \sum_{j \in \delta^+(i)} x_{ij}. \tag{8.1}$$

The *value* φ of the flow x is the sum of the flows on the arcs coming out of s, i.e. $\varphi = \sum_{i \in \delta^+(s)} x_{si}$. Each non-empty node subset $S \subsetneq V$ induces a partition of V in $S, V \setminus S$. Given a non-empty proper

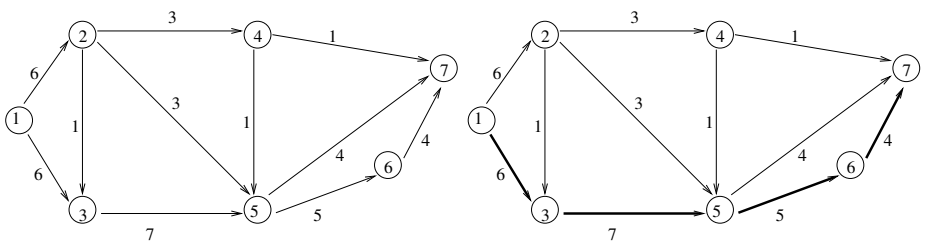
node subset $S \subsetneq V$ containing s and not t, the *cut* induced by S (denoted by $\delta(S)$) is the subset of all arcs in A having an adjacent vertex in S and the other in $V \setminus S$ (we say that the cut $\delta(S)$ separates s from t). The *directed cut from* S (denoted by $\delta^+(S)$) is defined as $\{(i,j) \in \delta(S) \mid i \in S \wedge j \in V \setminus S\}$. The *directed cut into* S (denoted by $\delta^-(S)$) is defined as $\{(i,j) \in \delta(S) \mid j \in S \wedge i \in V \setminus S\}$. The *capacity* of the cut $\delta(S)$ is $k(S) = \sum_{(i,j)\in\delta(S)} k_{ij}$. Similar definitions are applied to directed cuts, with notations $k^+(S), k^-(S)$. A *minimum cut* in a network is a directed cut $\delta^+(S^*)$ such that for all other non-empty node subsets S such that $s \in S$ and $t \notin S$ we have $k^+(S^*) \leq k^+(S)$. The flow through the cut $\delta(S)$ is $\varphi(S) = \sum_{(i,j)\in\delta^+(S)} x_{ij} - \sum_{(i,j)\in\delta^-(S)} x_{ij}$.

The Maximum Flow / Minimum Cut theorem asserts that the maximum value a feasible flow can take is the same as the capacity of a minimum cut in the network. Let φ^* be the maximum value over all feasible flows and $\delta^+(S^*)$ be a minimum cut in the network. The proof is sketched below.

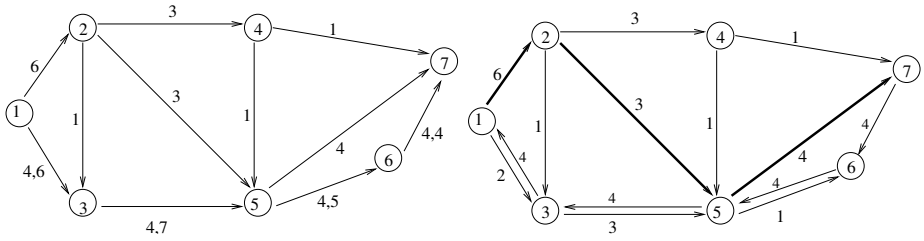
- By Eq. (8.1), the flow through any cut $\delta(S)$ must have a value equal to the flow on the arcs of the cut $\delta^+(\{s\})$. This also holds for the minimum cut, hence $\varphi^* = \varphi(S^*)$.
- By definition, $\varphi(S^*) = \sum_{(i,j)\in\delta^+(S^*)} x_{ij} - \sum_{(i,j)\in\delta^-(S^*)} x_{ij}$. We remark that this quantity is maximum when the first sum attains maximum value and the second has minimum value.
- Since the flow φ^* is maximum and$0 \leq x_{ij} \leq k_{ij}$ for each $(i,j) \in A$, $\varphi(S^*)$ is maximum when $x_{ij} = k_{ij}$ for each $(i,j) \in \delta^+(S^*)$ and $x_{ij} = 0$ for each $(i,j) \in \delta^-(S^*)$.
- We can infer that $\varphi(S^*) = \sum_{(i,j)\in\delta^+(S^*)} k_{ij}$, which by definition is the capacity of the minimum cut. Hence $\varphi^* = k^+(S^*)$ as required.

We can now solve the exercise with the Ford-Fulkerson algorithm. We start from the feasible flow having value 0. The *incremental network* $\bar{G}_0$ associated to the initial feasible flow x^0 represents all potential flow variations with respect to the current feasible flow. More precisely, the incremental network $\bar{G} = (V, \bar{A})$ of G with respect to the flow x is a network such that: (a) for each arc $(i,j) \in A$ such that $x_{ij} = 0$ there is an arc $(i,j) \in \bar{A}$ weighted by $w_{ij} = k_{ij}$; (b) for each arc $(i,j) \in A$ such that $x_{ij} = k_{ij}$ there is an arc $(j,i) \in \bar{A}$ weighted by $w_{ji} = k_{ij}$; (c) for each arc $(i,j) \in A$ such that $0 < x_{ij} < k_{ij}$ there is an arc $(i,j) \in \bar{A}$ weighted by $w_{ij} = k_{ij} - x_{ij}$ and an arc $(j,i) \in \bar{A}$ weighted by $w_{ji} = x_{ij}$. An *augmenting* $s-t$*-path* in an incremental network is a path $p \subseteq \bar{A}$ in $\bar{G}$ from s to t such that $\beta(p) = \min_{(i,j)\in p} w_{ij} > 0$

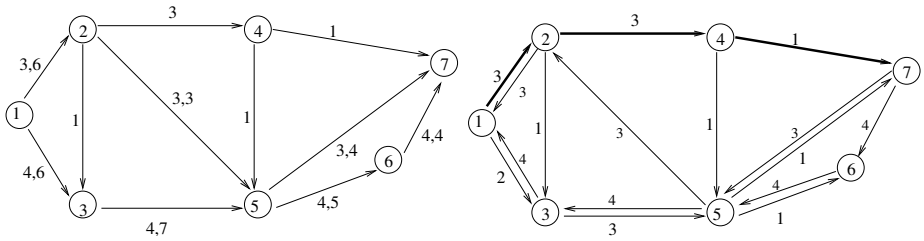
Since $x_{ij} = 0$ for all $(i,j) \in A$, $\bar{G}_0$ is the same as G. At iteration h, we need to find an augmenting $s-t$-path in the incremental network $\bar{G}_h$ with respect to the current feasible flow x^h. If such a path exists we define x^{h+1} as follows: (a) for all $(i,j) \in p$ such that $(i,j) \in A$ let $x^{h+1}_{ij} = x^h_{ij} + w_{ij}$; (b) for all $(i,j) \in p$ such that $(j,i) \in A$ let $x^{h+1}_{ji} \leftarrow x^h_{ji} - w_{ij}$. The value φ of the flow is updated to $\varphi + \beta(p)$. If no such path exists, the algorithm terminates: x^h is the maximum flow. The pictures below show the network G on the left and the incremental network $\bar{G}$ on the right.



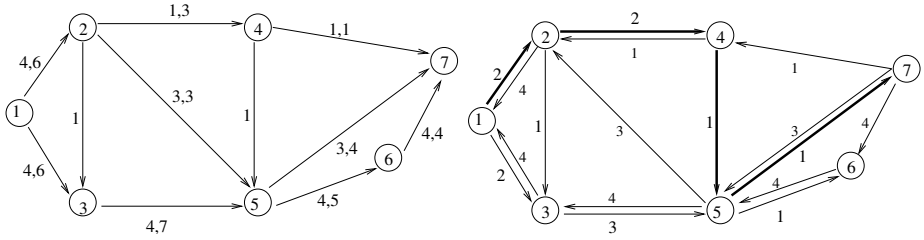
The augmenting path p in $\bar{G}_0$ is $\{(1,3),(3,5),(5,6),(6,7)\}$ with $\beta(p) = 4$. The new feasible flow has value $\varphi = 0 + 4 = 4$. The incremental network $\bar{G}_1$ is shown on the right, below.



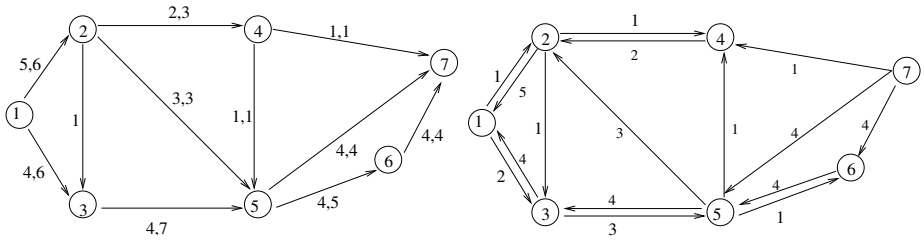
The next augmenting path is $\{(1,2),(2,5),(5,7)\}$ with $\beta = 3$. The next flow has value $\varphi = 4 + 3 = 7$. The incremental network $\bar{G}_2$ is shown on the right below.



Next augmenting path: $\{(1,2),(2,4),(4,7)\}$ with $\beta = 1$; $\varphi = 8$.



Next and final augmenting path: $\{(1,2),(2,4),(4,5),(5,7)\}$ with $\beta = 1$; $\varphi = 9$.



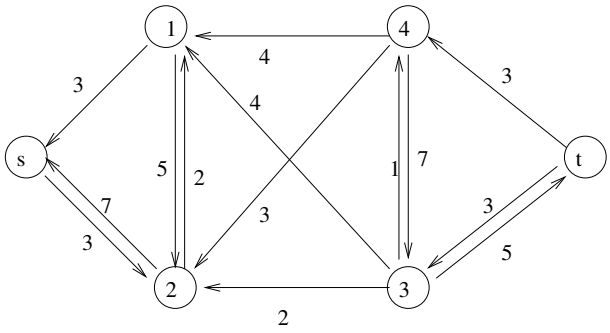
There are no more augmenting paths from 1 to 7 in the incremental network above, so the maximum flow has value 9.

A minimum cut in the network can be found as follows: on the incremental network $\bar{G}_4$ of the last iteration, from node 1 we can only reach nodes 2,3,4,5,6. The subset of nodes $S = \{1,2,3,4,5,6\}$ determines a minimum cut $\{(4,7),(5,7),(6,7)\}$ with capacity $4 + 4 + 1 = 9$.

8.4 Minimum cut: Solution

We employ Ford-Fulkerson's algorithm to find the maximum flow. We then apply the graph exploration algorithm to the residual network found at the last iteration of the Ford-Fulkerson algorithm, starting

from the source node s. The subset S of the nodes visited by the graph exploration algorithm generates the minimum cut $\delta^+(S)$. More precisely, the last iteration of the Ford-Fulkerson algorithm identifies a flow having value 10 through the paths $s-1-4-t$, $s-2-3-t$, $s-2-1-3-t$, $s-2-1-4-3-t$. The residual network at the last iteration is:



The graph exploration algorithm starting from s applied to the above graph finds $S = \{s, 2, 1\}$, so the cut with minimum capacity is given by arcs (i, j) with $i \in S$ and $j \in V \backslash S$:

$$(1, 4), (1, 3), (2, 3)$$

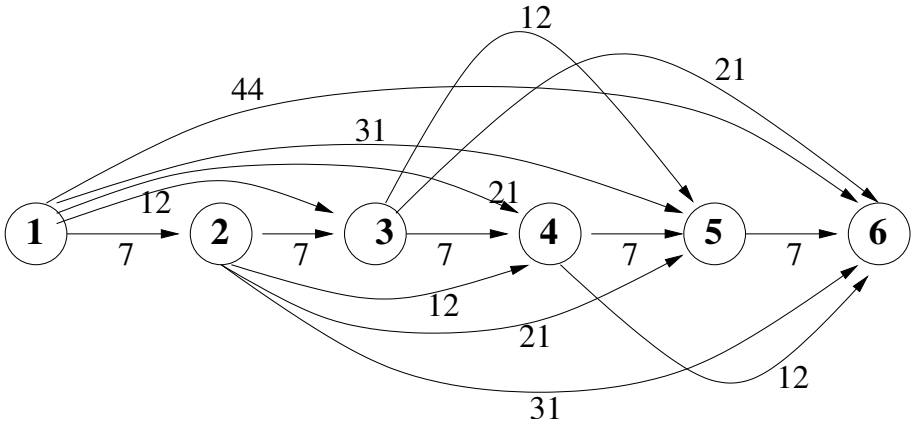
with capacity 4+4+2=10.

8.5 Renewal plan: Solution

Consider a directed graph with 6 nodes. Nodes 1 to 5 are associated to the start of each year. The sixth node represents the end of the planning period. For each $i < 6$ and $j > i$, arc (i, j) represents the occurrence that a production machinery bought at the beginning of the i-th year has been sold at the beginning of the j-th year. The cost c_{ij} associated to the arc (i, j) is given by:

$$c_{ij} = a_i + \sum_{k=i}^{j-1} m_k - r_j,$$

where a_i is the price of a new machinery (equal to 12000 euros), m_k is the maintenance cost in the k-th year and r_j is the gain from the sale of the old machinery. We obtain the following graph:



Any path from 1 to 6 represents a renewal plan; the cost of the path is the cost of the plan. We have to find a shortest path from node 1 to node 6. To this end we may either apply Dijkstra's algorithm

stopping as soon as node 6 has been settled, or we may notice that the graph is acyclic. This allows us to solve the problem using a dynamic programming technique. We obtain the following values:

1. $\zeta(1) = 0$;
2. $\zeta(2) = 7$, $\pi(2) = 1$;
3. $\zeta(3) = 12$, $\pi(3) = 1$;
4. $\zeta(4) = 19$, $\pi(4) = 3$;
5. $\zeta(5) = 24$, $\pi(5) = 3$;
6. $\zeta(6) = 31$, $\pi(6) = 5$.

The shortest path (having cost 31) is $1 \to 3 \to 5 \to 6$. In other words, the firm should buy new machinery every two years. Note that this solution is not unique.

8.6 Connected subgraphs: Solution

(a) For each integer $i \geq 0$, $i \mod n$ and $(i+2) \mod n$ are strictly smaller than n, hence in V. Since E includes all possible pairs $\{i, j\} \in V$, $F \subseteq E$. (b) ($\Rightarrow$) Assume first that H is connected; then there must be a path between vertices 0 and 1, i.e. there must be an integer k and a sequence of pairs $\{0, 2\}, \{2, 4\}, \ldots, \{2k-2, 2k\}$ such that $2k \mod n = 1$: this means $2k = qn + 1$ for some integer q, which implies $n = \frac{2k-1}{q}$, which is odd for all values of k, q. ($\Leftarrow$) Suppose now n is odd: then for all integers j we can always find an integer q such that $qn + j$ is divisible by 2 (choose q odd if j is odd, and q even if j is even), therefore if we let $k = \frac{qn+j}{2}$ then k is an integer, hence $\forall j \; \exists k \; (2k \mod n) = j$, which implies that 0 is connected with every other vertex $j \in V$, which proves that H is connected.

8.7 Strong connection: Solution

Suppose, to get a contradiction, that there exist vertices $u, v \in V$ such that no directed path can be found in G from u to v. Since the undirected graph K_n is complete, the set P_{uv} of all (undirected) paths from u to v in K_n is non-empty. Let $\bar{P}_{uv}$ the set of arc sequences in G corresponding to each (undirected) path in P_{uv}. Since u, v are disconnected in G, this means in each sequence $p \in P_{uv}$ there is either (a) at least an arc (t, z) such that z has no outgoing arc in G, which implies $|\delta^+(z)| = 0$ against the hypothesis, or (b) v is such that $|\delta^-(v)| = 0$, again against the hypothesis.

Chapter 9

Linear programming: Solutions

9.1 Graphical solution: Solution

1. Equations associated to system $Ax = b$:

$$x_1 + 7x_2 = 4 \tag{9.1}$$
$$x_1 + 5x_2 = 5 \tag{9.2}$$
$$2x_1 + 3x_2 = 9 \tag{9.3}$$

Draw the corresponding lines on the Cartesian plane (see Fig. 9.1). The objective function is

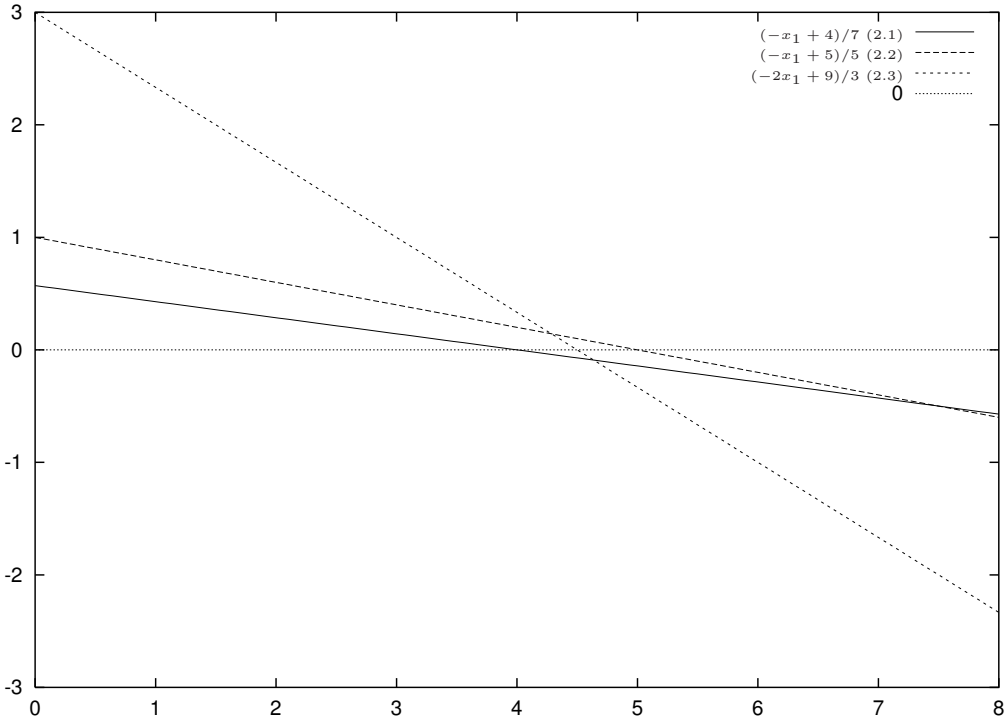


Figure 9.1: The feasible polyhedron is unbounded.

$16x_1 + 25x_2$. If we set $16x_1 + 25x_2 = q$ we obtain the parametric line equation $x_2 = -\frac{16}{25}x_1 + \frac{q}{25}$. It

is evident from Fig. 9.2 that the optimal solution is at vertex R of the feasible polyhedron. Vertex

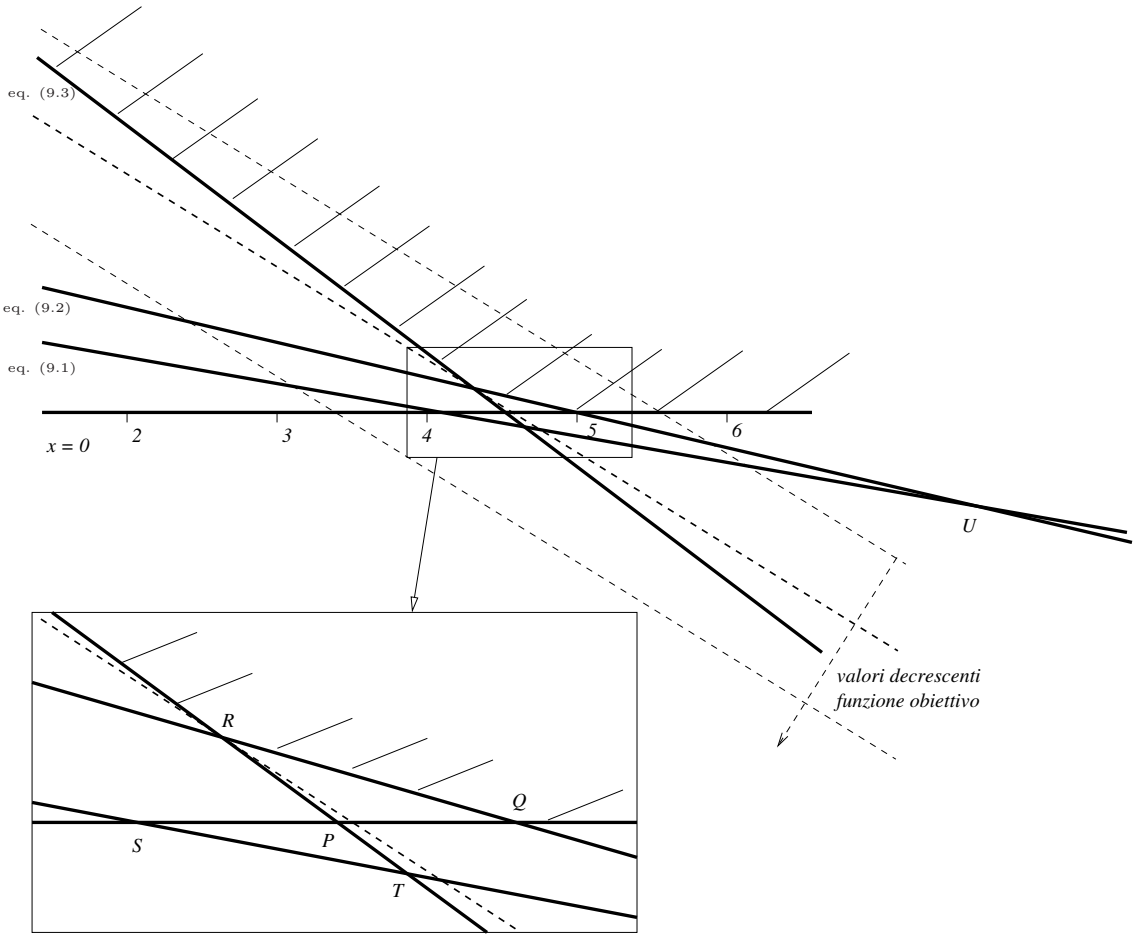


Figure 9.2: Graphical solution of the problem.

R is the intersection of (9.2) and (9.3). We can obtain its coordinates by solving the system

$$\begin{aligned} x_1 + 5x_2 &= 5 \\ 2x_1 + 3x_2 &= 9 \end{aligned}$$

which implies $x_1 = 5(1 - x_2)$ and hence $x_2 = \frac{1}{7}$ and $x_1 = \frac{30}{7}$.

2. By writing the problem in standard form we must introduce three slack variables s_1, s_2, s_3 associated to each of the constraints. Let then $x' = (x_1, x_2, s_1, s_2, s_3)^\top$, $c' = (16, 25, 0, 0, 0)$ and

$$A' = \left(\begin{array}{cc|ccc} 1 & 7 & -1 & 0 & 0 \\ 1 & 5 & 0 & -1 & 0 \\ 2 & 3 & 0 & 0 & -1 \end{array}\right) = (A| - I).$$

The problem in standard form is given by

$$\begin{aligned} \min_{x'} \quad & c'x' \\ & A'x' = b \\ & x' \geq 0 \end{aligned}$$

Since vertex R is the intersection of lines (9.1) and (9.2), the corresponding constraints have the relative slack variables s_2, s_3 equal to zero in R. If we set the basic variables to $x_B = (x_1, x_2, s_1)$ and the nonbasics to $x_N = (s_2, s_3)$ we obtain a partition $x' = (x_B|x_N)$ of the variables to which there corresponds a partition of the matrix columns

$$A' = \left(\begin{array}{ccc|cc} 1 & 7 & -1 & 0 & 0 \\ 1 & 5 & 0 & -1 & 0 \\ 2 & 3 & 0 & 0 & -1 \end{array}\right) = (B|N).$$

The value of the basic variables x_B in R is given by $B^{-1}b$. We obtain

$$B^{-1} = \frac{1}{7}\begin{pmatrix} 0 & -3 & 5 \\ 0 & 2 & -1 \\ -7 & 11 & -2 \end{pmatrix}$$

and hence $B^{-1}b = (\frac{30}{7}, \frac{1}{7}, \frac{9}{7})$. Note that the values for x_1 and x_2 correspond to those computed above.

9.2 Geometry of LP: Solution

1. The equations associated to the constraints (2.1), (2.2), (2.3) are:

$$\begin{aligned} 2x_1 + x_2 &= 4 \quad \text{(Eq. 2.1)} \\ -2x_1 + x_2 &= 2 \quad \text{(Eq. 2.2)} \\ x_1 - x_2 &= 1, \quad \text{(Eq. 2.3)} \end{aligned}$$

that is,

$$\begin{aligned} x_2 &= -2x_1 + 4 \\ x_2 &= 2x_1 + 2 \\ x_2 &= x_1 - 1, \end{aligned}$$

which can easily be drawn as lines in the Cartesian plane x_1, x_2. The objective function $(*)$ may be represented by the parametric line family $x_2 = -\frac{3}{2} + q$. The feasible polyhedron is $PQROS$, represented in Fig. 9.3. The optimal solution is in vertex $P = (\frac{1}{2}, 3)$, and the value of the objective in that point is $z^* = \frac{15}{2}$.

2. We write the problem in standard form introducing 3 slack variables s_1, s_2, s_3 associated with each of the constraints. Let $x' = (x_1, x_2, s_1, s_2, s_3)^T$, $c' = (-3, -2, 0, 0, 0)$, $b = (b_1, b_2, b_3)^T = (4, 2, 1)^T$ and

$$A' = \left(\begin{array}{cc|ccc} 2 & 1 & 1 & 0 & 0 \\ -2 & 1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{array}\right) = (A|I).$$

The problem in standard form is:

$$\begin{aligned} \min_{x'} \quad & c'x' \\ & A'x' = b \\ & x' \geq 0. \end{aligned}$$

(a) Vertex P: $s_1 = 0, s_2 = 0$, hence $x_B = (x_1, x_2, s_3)$, $x_N = (s_1, s_2)$,

$$B = \begin{pmatrix} 2 & 1 & 0 \\ -2 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}, \quad N = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

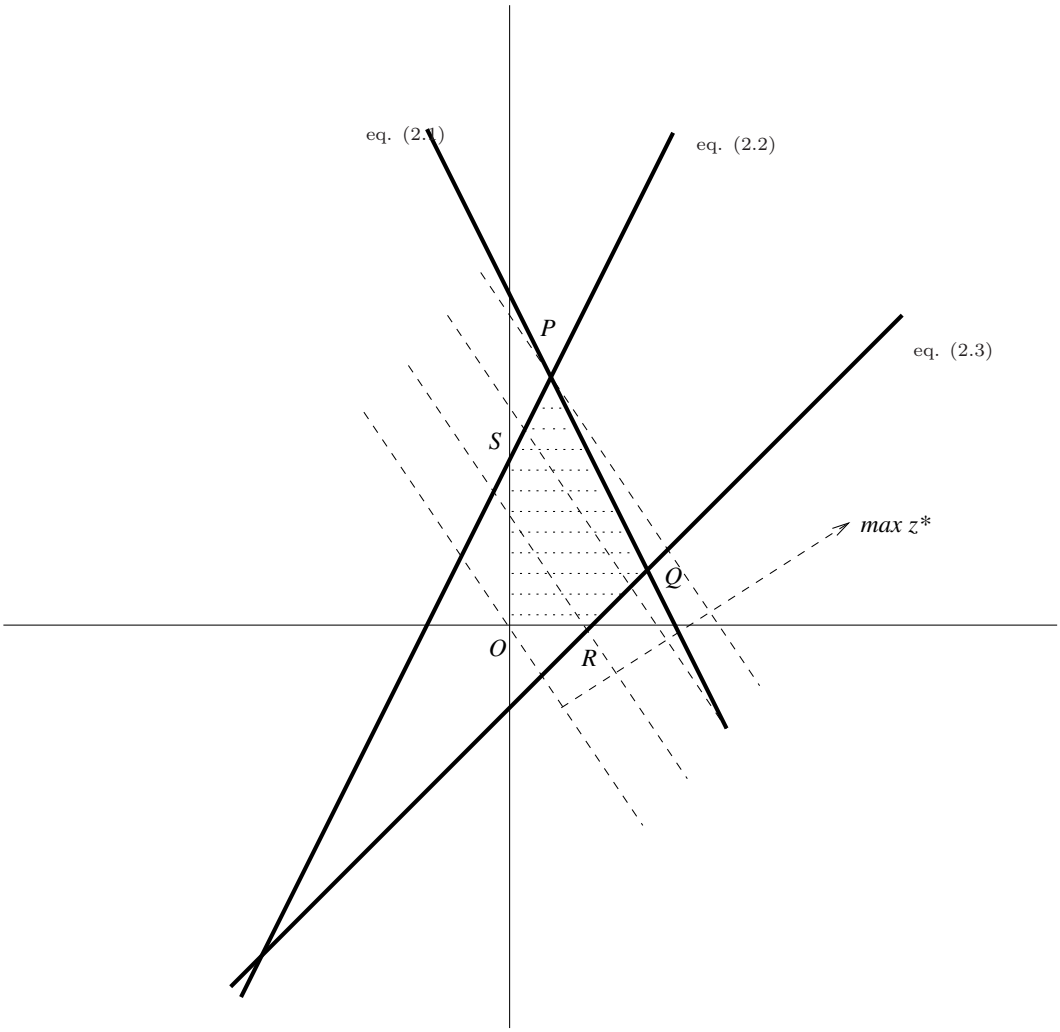


Figure 9.3: The feasible polyhedron.

(b) Vertex Q: $s_1 = 0, s_3 = 0$, hence $x_B = (x_1, x_2, s_2)$, $x_N = (s_1, s_3)$,

$$B = \begin{pmatrix} 2 & 1 & 0 \\ -2 & 1 & 1 \\ 1 & -1 & 0 \end{pmatrix}, \quad N = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

(c) Vertex R: $x_2 = 0, s_3 = 0$, hence $x_B = (x_1, s_1, s_2)$, $x_N = (x_2, s_3)$,

$$B = \begin{pmatrix} 2 & 1 & 0 \\ -2 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad N = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ -1 & 1 \end{pmatrix}.$$

(d) Vertex O: $x_1 = 0, x_2 = 0$, hence $x_B = (s_1, s_2, s_3)$, $x_N = (x_1, x_2)$,

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad N = \begin{pmatrix} 2 & 1 \\ -2 & 1 \\ 1 & -1 \end{pmatrix}.$$

(e) Vertex S: $x_1 = 0, s_2 = 0$, hence $x_B = (x_2, s_1, s_3)$, $x_N = (x_1, s_2)$,

$$B = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}, \quad N = \begin{pmatrix} 2 & 0 \\ -2 & 1 \\ 1 & 0 \end{pmatrix}.$$

3. We can take the vector of slack variables as the initial feasible basic variables (variables x_1, x_2 are nonbasics and take the value 0, which is consistent with the fact that the vertex corresponding to the initial feasible basis is vertex O). Let x_h be the variables which enters the basis. In order to find the exiting variable, we compute $\theta = \min\{\frac{\bar{b}_i}{\bar{a}_{ih}} \mid i \leq 3 \wedge \bar{a}_{ih} > 0\}$, where $\bar{a}_{ih}$ is the i-th element of the h-th column in the matrix $B^{-1}N$, and $\bar{b}_i$ is the i-th element of $B^{-1}b$. The text of the problem tells us to use $h = 1$. Since $\theta = \min\{\frac{4}{2}, \frac{1}{1}\} = 1$ (because $\frac{2}{-2} < 0$ the element $\frac{\bar{b}_2}{\bar{a}_{21}}$ is not taken into account) and $1 = \theta = \frac{\bar{b}_3}{\bar{a}_{31}}$, the index of the exiting basis is 3, i.e. the third variable of the current basis, which is s_3. The first visited vertex is R, corresponding to the basis (x_1, s_1, s_2). The subsequent vertices visited by the simplex algorithm are Q and then P.

4. The vertex in (Eq. (2.1) $\cap$ Eq. (2.2)) is P and the vertex in (Eq. (2.1) $\cap$ Eq. (2.3)) is Q. The reduced costs are given by the equation $\bar{c} = c^\top - c_B^\top B^{-1} A'$, where the reduced costs for the basic variables are equal to 0 and those for the nonbasic variables may be nonzero: we want to determine $\bar{c}_N = c_N^\top - c_B^\top B^{-1} N$. In P, B and N are as in point (2a) above, hence

$$B^{-1}N = \frac{1}{4}\begin{pmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 3 \end{pmatrix}.$$

Since $c'_B = (-3, -2, 0)$ and $c'_N = (0, 0)$, we have $\bar{c}_N = (\frac{7}{4}, \frac{1}{4})$. Since both values are greater than 0, the basis in P is optimal. In Q, B, N are as in point (2b) above, hence

$$B^{-1}N = \frac{1}{3}\begin{pmatrix} 1 & 1 \\ 1 & -2 \\ 1 & 4 \end{pmatrix}$$

Since $c'_B = (-3, -2, 0)$, we have $\bar{c}_N = (\frac{5}{3}, -\frac{1}{3})$. This tells us that Q is not an optimal solution.

5. The objective function gradient is a conic combination of the active constraint gradients only in an optimal point. In other words, this condition asserts that if the only improving directions are infeasible, then the vertex is optimum. In this instance, the optimal vertex is $P = (\frac{1}{2}, 3)$. The objective gradient is $\nabla f = (3, 2)$ (constant for each x_1, x_2). The constraints which are active in P are (2.1) and (2.2), with gradients $(2, 1)$ and $(-2, 1)$. We solve the system

$$\lambda_1 \begin{pmatrix} 2 \\ 1 \end{pmatrix} + \lambda_2 \begin{pmatrix} -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

and verify that $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$. The solution of the system is $\lambda_1 = \frac{7}{4}$ and $\lambda_2 = \frac{1}{4}$. Since both are strictly positive, the condition is verified for the optimal vertex P (see Fig. 9.4). We now check that the objective function gradient is *not* a conic combination of the active constraint gradients in the non-optimal vertices Q, R, O, S.

- Vertex Q. Active constraints (2.1), (2.3) with gradients $(2, 1)$ and $(1, -1)$. We get $\lambda_1 = \frac{5}{3}$, $\lambda_2 = -\frac{1}{3} < 0$.
- Vertice R. Active constraints (2.3), $-x_2 \leq 0$ with gradients $(1, -1)$ and $(0, -1)$. We get $\lambda_1 = 3, \lambda_2 = -5 < 0$.
- Vertex O. Active constraints $-x_1 \leq 0$, $-x_2 \leq 0$ with gradients $(-1, 0)$ and $(0, -1)$. We get $\lambda_1 = -3 < 0, \lambda_2 = -2 < 0$.
- Vertex S. Active constraints $-x_1 \leq 0$, (2.2) with gradients $(-1, 0)$ and $(-2, 1)$. We get $\lambda_1 = -7 < 0, \lambda_2 = 2$.

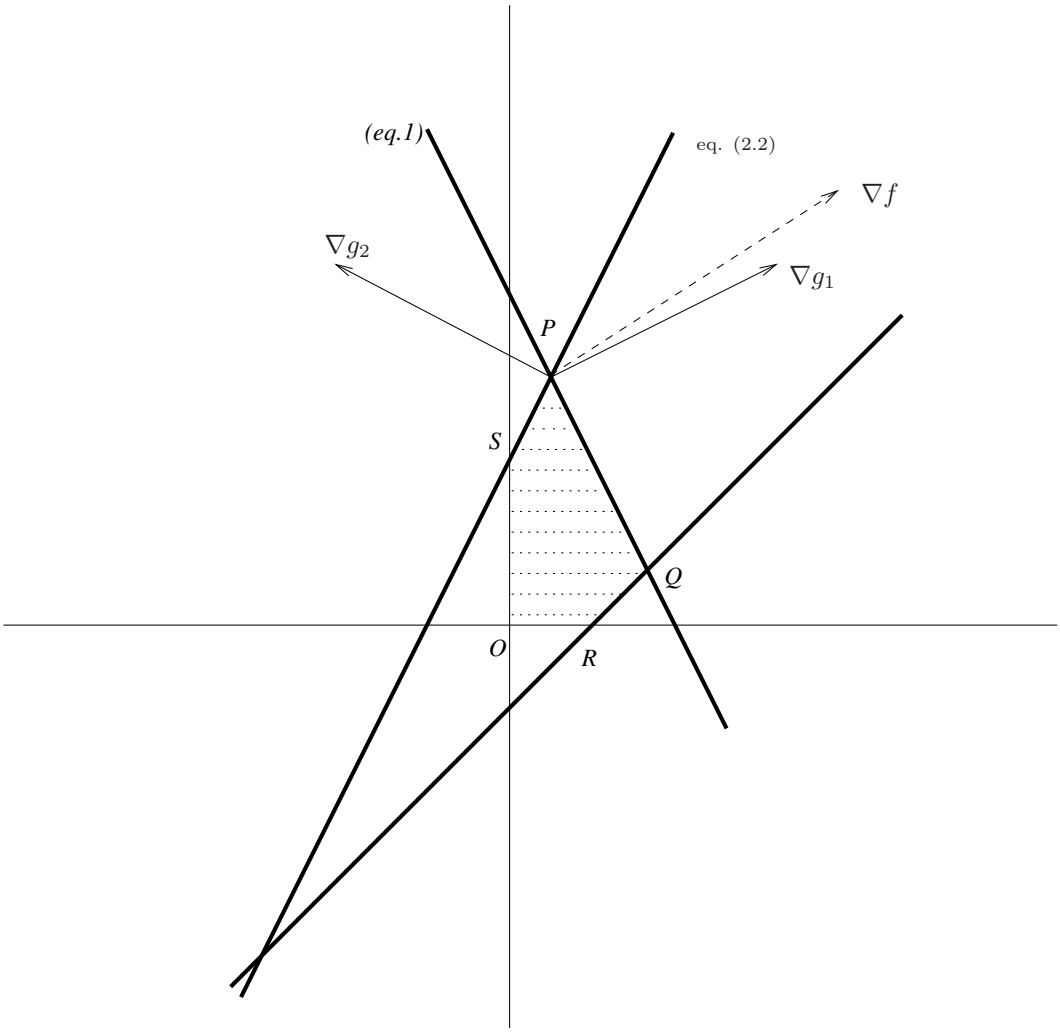


Figure 9.4: Optimality at P: the vector ∇f is in the cone generated by ∇g_1 and ∇g_2, where $g_1(x) \leq 4$ is (2.1) and $g_2(x) \leq 2$ is (2.2).

6. By inspection, for $b_1 \to \infty$ and $b_1 \to S_y = 2$ the optimal basis does not change. The case $b_1 = S_y$ is degenerate. For $0 < b_1 < S_y$ we get $x_1 = 0$, which therefore exits the basis (s_2 enters it, since (2.2) ceases to be active). For $b_1 = 0$ there is only one feasible point $(0, 0)$ and for $b_1 < 0$ the feasible region is empty.

7. Consider the family of lines $x_2 = mx_1 + q$ where $m > 2$, shown in Fig. 9.5. By inspection, every objective function of the form $\max -mx_1 + x_2$ where $m > 2$ has optimum S on the polyhedron $PQROS$.

8. Let $Q = (Q_x, Q_y)$, and $x_2(x_1) = 2x_1 + b_2$ the family of lines parallel to that of Eq. (2.2). For $x_2(Q_x) < Q_y$ the feasible region is empty. Since Q is the intersection of Eq. (2.1) and Eq. (2.3), we get $Q = (\frac{5}{3}, \frac{2}{3})$. We therefore require $x_2(\frac{5}{3}) < \frac{2}{3}$, i.e. $\frac{10}{3} + b_2 < \frac{2}{3}$, that is $b_2 < -\frac{8}{3}$. If the three lines defined by the constraints meet in Q then the feasible region only has one single point. This happens if $b_2 = -\frac{8}{3}$.

9. If the family of lines given by the objective, namely $c_1x_1 + 2x_2 = q$, is parallel to one of the sides of the polyhedron, and its optimization direction is towards the outside of the polyhedron (relative to the side to which it is parallel) then there are multiple optimal solutions. For $c_1 = 4$ we get

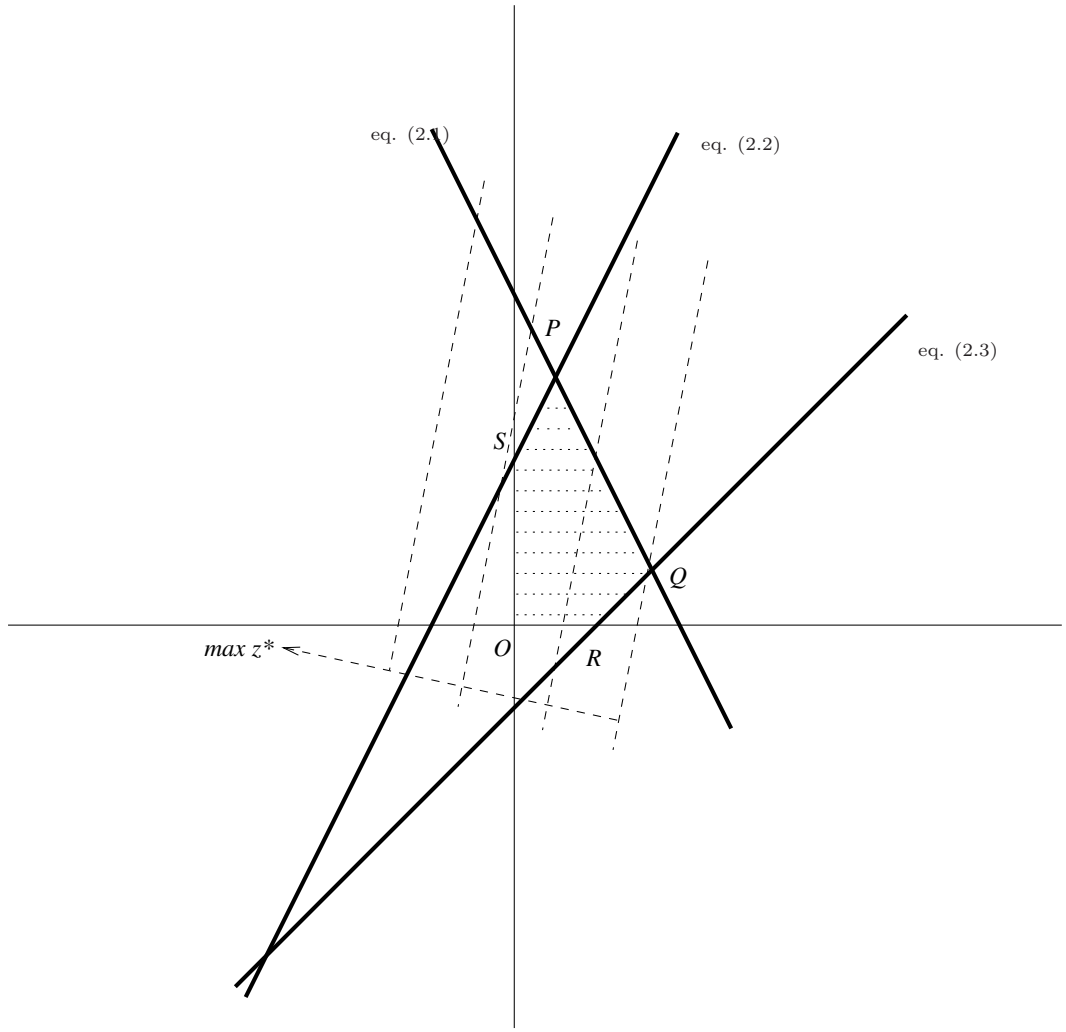


Figure 9.5: Objective function such that S is optimum.

$x_2 = -2x_1 + \frac{q}{2}$, which is parallel to Eq. (2.1). For $c_1 = -4$ we get $x_2 = 2x_1 + \frac{q}{2}$, which is parallel to (2.2). For $c_1 = 0$ we get $x_2 = \frac{q}{2}$, which is parallel to lato $x_1 = 0$: this choice is not acceptable, however, because for increasing q, x_2 decreases, so the maximization direction is towards the semi-space $x_1 \geq 0$ which contains the polyhedron. For $c_1 = -2$ we have $x_2 = x_1 + \frac{q}{2}$, which is parallel to the $\overline{QR}$ side (but again has maximization direction towards the polyhedron).

9.3 Simplex method: Solution

The problem is already in standard form as there are no inequality constraints and all variables are constrained to be non-negative. Constraints (2.1)-(2.3) can be written as the system $Ax = b$ where $x = (x_1, x_2, x_3)$, $b = (1, 5)$ and

$$A = \begin{pmatrix} 2 & 0 & 3 \\ 3 & 2 & -1 \end{pmatrix}.$$

No initial feasible basic solution is immediately evident. We therefore need a two-phase simplex method (the first phase is used to locate an initial feasible basis).

First phase. We use the simplex method to solve an auxiliary problem designed to find an initial feasible basis for the original problem. We add an auxiliary variable $y_i \geq 0$ for each equation constraint in the standard form problem. The problem constraints are reformulated to $A'\bar{x} = b$ where $\bar{x} = (x_1, x_2, x_3, y_1, y_2)$ and

$$A' = \begin{pmatrix} 2 & 0 & 3 & 1 & 0 \\ 3 & 2 & -1 & 0 & 1 \end{pmatrix}$$

Intuitively, y_i are a measure of the distance of the current basis from the feasible region of the original problem. If the auxiliary problem has a solution such that $y_i = 0$ for all i, that solution is also feasible in the original problem. Since y_i are constrained to be non-negative, it suffices to ask that $\sum_i y_i = 0$ to enforce $y_i = 0$ for all i. We can therefore choose $v = \sum_i y_i$ as the objective function of the auxiliary problem:

$$\begin{aligned} \min v = \quad & y_1 + y_2 & \\ & 2x_1 + 3x_3 + y_1 & = 1 \\ & 3x_1 + 2x_2 - x_3 + y_2 & = 5 \\ & x_1, x_2, x_3, y_1, y_2 & \geq 0. \end{aligned}$$

If the feasible region of the original problem is non-empty, we will necessarily find $v = 0$ and $y_i = 0$ for all i. The auxiliary problem for the present instance is $\min\{y_1 + y_2 \mid \bar{A}\bar{x} = b, x \geq 0, y \geq 0\}$. We shall solve it using the simplex algorithm.

The initial feasible basis for the auxiliary problem is $\bar{x}_B = (y_1, y_2)$. We express the basic variables in function of the nonbasics:

$$\begin{aligned} y_1 &= 1 - 2x_1 - 3x_3 \\ y_2 &= 5 - 3x_1 - 2x_2 + x_3. \end{aligned}$$

The objective is therefore $v = y_1 + y_2 = 6 - 5x_1 - 2x_2 - 2x_3$. We write these information in tableau form as:

-6	-5	-2	-2	0	0
1	2	0	3	1	0
5	3	2	-1	0	1

The reduced costs of the nonbasic variables $\bar{x}_N = (x_1, x_2, x_3)$ are all negative; since we are minimizing, each nonbasic might enter the basis. By Bland's anti-cycling rule, we choose the variable with least index, that is x_1. There are two limits to the growth of x_1, given by $y_1 = 1 - 2x_1$ and $y_2 = 5 - 3x_1$; the growth should be bounded by $\min\{\frac{1}{2}, \frac{5}{3}\} = \frac{1}{2}$, which corresponds to the basic variable y_1: the variable y_1 exits the basis. We pivot on the coefficient 2 in position (1,1) in the tableau:

1. divide row 1 by 2;
2. add 5 times row 1 to row 0;
3. subtract 3 times row 1 from row 2.

We obtain the following tableau:

$-\frac{7}{2}$	0	-2	$\frac{11}{2}$	$\frac{5}{2}$	0
$\frac{1}{2}$	1	0	$\frac{3}{2}$	$\frac{1}{2}$	0
$\frac{7}{2}$	0	2	$-\frac{11}{2}$	$-\frac{3}{2}$	1

The only negative reduced cost is that of x_2, so x_2 enters the basis. The only bound to the growth of x_2 is given by $y_2 = \frac{7}{2} - 2x_2$, hence $x_2 \leq \frac{7}{4}$, and y_2 exits the basis. We pivot on coefficient 2 in position (2,2):

1. add row 2 to row 0;
2. divide row 2 by 2.

We get the tableau:

0	0	0	0	1	1
$\frac{1}{2}$	1	0	$\frac{3}{2}$	$\frac{1}{2}$	0
$\frac{7}{4}$	0	1	$-\frac{11}{4}$	$-\frac{3}{4}$	$\frac{1}{2}$

All the reduced costs of the nonbasics (x_3, y_1, y_2) are non-negative, hence the first phase of the algorithm terminates. We showed that the feasible region of the original problem is non-empty; the initial feasible basis of the original problem is $x_B = (x_1, x_2)$.

SECOND PHASE. Objective function: $x_1 - 2x_2$. Initial feasible basis: (x_1, x_2). Nonbasic variable: x_3. We express x_1, x_2 in function of x_3:

$$\begin{aligned} x_1 &= \frac{1}{2} - \frac{3}{2}x_3 \\ x_2 &= \frac{7}{4} + \frac{11}{4}x_3 \end{aligned}$$

The objective, expressed in function of x_3, is $-3-7x_3$. We eliminate from our tableau all columns relative to the auxiliary variables:

3	0	0	-7
$\frac{1}{2}$	1	0	$\frac{3}{2}$
$\frac{7}{4}$	0	1	$-\frac{11}{4}$

Since we are minimizing the objective, x_3 enters the basis as it has a negative reduced cost. Since the only bound to the growth of x_3 is given by $x_1 = \frac{1}{2} - \frac{3}{2}x_3$, x_1 exits the basis. We pivot on the coefficient $\frac{3}{2}$ in position (3,1):

1. divide row 1 by $\frac{3}{2}$;
2. add 7 times row 1 to row 0;
3. add $-\frac{11}{4}$ times row 1 to row 3.

The new tableau is:

$\frac{16}{3}$	$\frac{14}{3}$	0	0
$\frac{1}{3}$	$\frac{2}{3}$	0	1
$\frac{8}{3}$	$\frac{11}{6}$	1	0

All the reduced costs being non-negative, the algorithm terminates with optimal solution $(0, \frac{8}{3}, \frac{1}{3})$ and optimal objective function value $-\frac{16}{3}$.

9.4 Duality: Solution

The dual problem can be obtained mechanically from the primal problem as follows. We associate a dual variable to each primal constraint and reformulate the primal problem following the rules below:

Primal	Dual
min	max
variables x	constraints
constraints	variables y
objective coefficients c	constraint right hand sides c
constraint right hand sides b	objective coefficients b
$A_i x \geq b_i$	$y_i \geq 0$
$A_i x \leq b_i$	$y_i \leq 0$
$A_i x = b_i$	y_i unconstrained
$x_j \geq 0$	$yA^j \leq c_j$
$x_j \leq 0$	$yA^j \geq c_j$
x_j unconstrained	$yA^j = c_j$

In the above table, A_i is the i-th row of A and A^j is the j-th column.

1.
$$\left.\begin{array}{rrcl} \max_y & 3y_1 + 4y_2 & & \\ & y_1 + 2y_2 & \leq & 3 \\ & -y_1 - 3y_2 & \leq & 5 \\ & y_1 & \leq & -1 \\ & y_1, y_2 & \leq & 0 \end{array}\right\} \tag{9.4}$$

2.
$$\left.\begin{array}{rrcl} \max_y & 3y_1 + 4y_2 + 2y_3 & & \\ & -3y_1 + 2y_2 + y_3 & \leq & 1 \\ & -y_1 - 3y_2 & \leq & -1 \\ & y_1 - 2y_2 - y_3 & = & -1 \\ & y_1 & \leq & 0 \\ & y_2 & \geq & 0 \\ & y_3 & & \text{unconstrained} \end{array}\right\} \tag{9.5}$$

3.
$$\left.\begin{array}{rrcl} \min_y & -3y_1 + 3 & & \\ & -3y_1 + 2y_2 + y_3 & \leq & -1 \\ & -y_1 - 3y_2 - y_3 & \geq & 1 \\ & y_1 - 4y_2 - y_3 & = & 2 \\ & y_1 & \leq & 0 \\ & y_2 & \geq & 0 \\ & y_3 & & \text{unconstrained} \end{array}\right\} \tag{9.6}$$

9.5 Geometrical interpretation of the simplex algorithm: Solution

We give here a lengthy solution where every step is inferred from first principles and geometrical considerations. We remark that it is not required of the student to proceed as below. The usual solution in tableau form will suffice. We believe, however, that reading this solution will help the student to identify the geometrical reasons for the "mechanical" steps in the simplex algorithm in tableau form.

Let $M = \{1, 2, 3, 4\}$ be the set of indices of the problem constraints: $-x_1 + x_2 \leq 1$, $2x_1 + x_2 \leq 4$, $-x_1 \leq 0$, $-x_2 \leq 0$. The simplex algorithm, schematized geometrically, is as follows:

1. From the feasible solution $\bar{x}$ move to another feasible solution $x^{(1)}$ corresponding to a vertex of the feasible polyhedron. Let $k = 1$.

2. Is there a feasible direction from $x^{(k)}$ which increases the objective function value (recall we are maximizing the objective)? If not, the algorithm terminates with optimal solution $x^{(k)}$.
3. From $x^{(k)}$ move to an adjacent polyhedron vertex $x^{(k+1)}$ with lower objective function value. If no such vertex can be found, the algorithm terminates and the problem is unbounded. Otherwise, repeat from 2.

The operation of finding the next vertex $x^{(k+1)}$ (adjacent to the current vertex $x^{(k)}$ can be described as follows:

1. find a feasible increasing direction vector ξ
2. find a feasible step value λ
3. compute $x^{(k+1)} = x^{(k)} + \lambda\xi$.

The given problem is $\max\{cx \mid Ax \leq b\}$, where

$$A = \begin{pmatrix} -1 & 1 \\ 2 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix},$$

$b = (1, 4, 0, 0)^\top$ and $c = (1, 1)$.

Notation. Let $I(x)$ be the set of indices of the constraints which are active in x (i.e. the problem constraints which are satisfied at equality by x). Let $\bar{I}(x)$ be $M \backslash I(x)$. Let A_I be the submatrix of A consisting of the rows indexed by the set I. Likewise, let b_I be the subvector of b indexed by the set I. Let A_i be the i-th row of A.

9.5.1 Iteration 1: Finding the initial vertex

Since the initial solution $\bar{x} = (1, 0)$ is feasible in the problem constraints but it does not identify a vertex of the feasible polyhedron (why?), it is necessary to find an initial vertex.

Feasible direction. In $\bar{x} = (1, 0)$ the set of active constraint index is $I(\bar{x}) = \{4\}$. We can therefore "move" the solution along the constraint $x_2 = 0$ until we reach the first basic feasible solution (corresponding to a feasible vertex). In order to to that, we solve the system $A_I\xi = 0, c\xi = 1$ with $A_I = (0, -1)$ to find $-\xi_2 = 0$ and $\xi_1 = 1$, and hence $\xi = (1, 0)^\top$. This procedure works because the rank of the system $A_I\xi = 0$ (that is, 1) is strictly smaller than the number of components of ξ (that is, 2). Should $\bar{x}$ already be a vertex this condition would not be verified and this procedure would not work. See the discussion for iteration 2 (Seection 9.5.2).

Feasible step. In order to "move" to a basic solution, we have to compute the step λ so that $\bar{x} + \lambda\xi$ is feasible. The only constraints are those that are inactive at $\bar{x}$ (the active ones being already satified by definition):

$$A_{\bar{I}}(\bar{x} + \lambda\xi) \leq b_{\bar{I}}. \tag{9.7}$$

If $A_i\xi \leq 0$, then Eq. (9.7) is verified: we only need consider the indices i such that $A_i\xi > 0$. Choose λ as follows:

$$\lambda = \min\left\{ \frac{b_i - A_i\bar{x}}{A_i\xi} \mid i \in \bar{I}(x), A_i\xi > 0 \right\}. \tag{9.8}$$

This way, for the inequalities in (9.7) with $A_i\xi > 0$ we have:

$$A_i\bar{x} + \lambda A_i\xi \leq A_i\bar{x} + \frac{b_i - A_i\bar{x}}{A_i\xi} A_i\xi = b_i.$$

Notice that (9.7) is satisfied. In the particular instance given in this problem, we have $\bar{I}(x) = \{1,2,3\}$, $\xi = (1,0)$ as above and hence $A_1\xi = -1 < 0$, $A_2\xi = 2 > 0$, $A_3\xi = -1 < 0$ and $\lambda = \frac{b_2 - A_2\bar{x}}{A_2\xi} = \frac{4-2}{2} = 1$. Finally we obtain $x^{(1)} = \bar{x} + \lambda\xi = (2,0)^\top$.

9.5.2 Iteration 2: Finding a better vertex

Feasible direction. The parameters of this iteration are the following: $I(x^{(1)}) = \{2,4\}$, $A_I = \begin{pmatrix} 2 & 1 \\ 0 & -1 \end{pmatrix}$, $A_I^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & -1 \end{pmatrix}$. The system $A_I\xi = 0$ has rank equal to the number of components of ξ (that is, 2) so its solution would yield $\xi = 0$, which is not an increasing direction.

The optimality conditions in a point x are that there should be no increasing feasible directions in x. Therefore the system

$$c\xi > 0, \qquad A_I\xi \leq 0 \tag{9.9}$$

should have no solutions (recall we want to actually find an increasing feasible direction, so we are looking for a solution of (9.9)). We can reformulate (9.9) to the restricted problem[1] $\max\{c\xi \mid A_I\xi \leq 0\}$. The dual of the restricted problem is $\min\{\eta 0 \mid ; \eta A_I = c, \eta \geq 0\}$, which is the equivalent to determining whether the system $\eta A_I = c, \eta \geq 0$ is feasible, since the objective function is 0. If we find a feasible η then the restricted problem should be such that $\max c\xi = \min \eta 0 = 0$, which shows that system (9.9) is not feasible, and hence that x is optimal. On the other hand, if we find η such that $\eta A_I = c$ but $\eta \not\geq 0$ we can derive a feasible direction. We shall therefore suppose that there is an index h such that $\eta_h < 0$. Let u_h be the vector with 1 in the h-st component and 0 on everywhere else. We have $\eta_h = cA_I^{-1}u_h < 0$. The feasible increasing direction is $\xi = -A_I^{-1}u_h$: we obtain $c\xi = -cA_I^{-1}u_h > 0$ and $A_I\xi = -A_IA_I^{-1}u_h = -u_h < 0$, which means that ξ is a feasible solution for the restricted problem equivalent to (9.9).

Applied to the current instance, we obtain: $\bar{\eta}A_I = c$, whence $\bar{\eta} = cA_I^{-1} = (\frac{1}{2}, -\frac{1}{2})$. Since $-1/2 < 0$, we define $h = 2$, $u_h = (0,1)$ and hence $\xi = -A_I^{-1}u_h = (-\frac{1}{2}, 1)^\top$.

Feasible step. As in iteration 1: $\bar{I}(x^{(1)}) = \{1,3\}$, $A_{\bar{I}} = \begin{pmatrix} -1 & 1 \\ -1 & 0 \end{pmatrix}$, and hence $A_1\xi = \frac{3}{2} > 0$ e $A_2\xi = \frac{1}{2} > 0$. Furthermore $A_1x^{(1)} = -2$ and $A_2x^{(2)} = -2$, so that $\lambda = \min\left\{\frac{1-(-2)}{3/2}, \frac{0-(-2)}{1/2}\right\} = \min\{2,4\} = 2$. Therefore $x^{(2)} = x^{(1)} + \lambda\xi = (2,0)^\top + 2(-1/2,1)^\top = (1,2)^\top$.

9.5.3 Iteration 3: Algorithm termination

Feasible direction. Current parameters: $I(x^{(1)}) = \{1,2\}$, $A_I = \begin{pmatrix} -1 & 1 \\ 2 & 1 \end{pmatrix}$, $A_I^{-1} = \begin{pmatrix} -\frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & \frac{1}{3} \end{pmatrix}$. We follow the same procedure as in iteration 2. To find an increasing direction we have $\bar{\eta}A_I = c$, whence $\bar{\eta} = cA_I^{-1} = (\frac{1}{3}, \frac{2}{3})$. Since $\bar{\eta} > 0$, the algorithm terminates with optimal solution $x^{(2)} = (1,2)^\top$.

9.6 Complementary slackness: Solution

1. The dual of the given problem is:

$$\begin{array}{rrcrcrl} \min & 14y_1 & + & 10y_2 & + & 3y_3 & \\ & y_1 & + & 2y_2 & + & y_3 & \geq 2 \\ & 2y_1 & - & y_2 & - & y_3 & \geq 1 \\ & y_1 & , & y_2 & , & y_3 & \geq 0 \end{array}$$

[1] If we obtain ξ such that $c\xi > 0$ the problem is unbounded, for if ξ is feasible, $\alpha\xi$ is feasible for each $\alpha > 0$, whence the objective function $c\xi$ can increase without bounds as α increases.

2. $\bar{x} = (\frac{20}{3}, \frac{11}{3})$ satisfies the constraints of the primal problem, so it is a feasible solution.

3. By complementary slackness, if $\bar{x} = (x_1, x_2)$ is feasible in the primal, $\bar{y} = (y_1, y_2, y_3)$ is feasible in the dual, and both satisfy the equations:

$$\begin{aligned} y_i(a_i^T x - b_i) &= 0 \quad \forall i \\ (c_j - y^T A_j)x_j &= 0 \quad \forall j \end{aligned}$$

then $\bar{x}$ is optimal in the primal and $\bar{y}$ in the dual. We get:

$$\begin{aligned} y_1(x_1 + 2x_2 - 14) &= 0 \\ y_2(2x_1 - x_2 - 10) &= 0 \\ y_3(x_1 - x_2 - 3) &= 0 \\ x_1(2 - y_1 - 2y_2 - y_3) &= 0 \\ x_2(1 - 2y_1 + y_2 + y_3) &= 0. \end{aligned}$$

Since $\bar{x} = (\frac{20}{3}, \frac{11}{3})$ satisfies the 1st and 3rd constraints in the primal but not the second, $y_2 = 0$. Since $x_1 \neq 0$ and $x_2 \neq 0$, we also have:

$$\begin{aligned} y_1 + 2y_2 + y_3 &= 2 \\ 2y_1 - y_2 - y_3 &= 1. \end{aligned}$$

Therefore, since $y_2 = 0$, we obtain $y_1 = 1$ and $y_3 = 1$. Since $\bar{y} = (1, 0, 1)$ is a dual feasible solution (satisfies the dual constraints), and the primal/dual solution pair $(\bar{x}, \bar{y})$ satisfies the complementary slackness conditions, $\bar{x}$ is the optimal solution to the primal problem. Again by complementary slackness, we also have that $\bar{y} = (1, 0, 1)$ is the optimal solution of the dual problem.

9.7 Sensitivity analysis: Solution

1. The dual of the given problem is:

$$\left.\begin{array}{lrcl} \min & 5y_1 + 40y_2 + 20y_3 & & \\ & -y_1 + y_2 + 2y_3 & \leq & 1 \\ & y_1 + 4y_2 + y_3 & \leq & -5 \\ & y_1, y_2, y_3 & \leq & 0. \end{array}\right\}$$

The solution $x^* = (4, 9)$ satisfies the 1st and 2nd constraints as equalities, and the 3rd constraint as a proper inequality; it is therefore a feasible solution. Since the problem has 3 constraints, we introduce 3 dual variables y_1, y_2, y_3. The complementary slackness conditions are:

$$\begin{aligned} y_1(-x_1 + x_2 - 5) &= 0 \\ y_2(x_1 + 4x_2 - 40) &= 0 \\ y_3(2x_1 + x_2 - 20) &= 0 \\ x_1(1 + y_1 - y_2 - 2y_3) &= 0 \\ x_2(-5 - y_1 - 4y_2 - y_3) &= 0. \end{aligned}$$

Since the 3rd constraint is not satisfied at equality by x^*, we have $y_3 = 0$. The last two equations become:

$$\begin{aligned} y_1 - y_2 &= -1 \\ -y_1 - 4y_2 &= 5, \end{aligned}$$

yielding a solution $y_1 = -\frac{9}{5}$ and $y_2 = -\frac{4}{5}$. The primal solution $x^* = (4, 9)$ and the dual solution $y^* = (-\frac{9}{5}, -\frac{4}{5}, 0)$ are both feasible in the respective proiblems and satisfy the complementary slackness conditions, and are therefore optimal.

2. The objective function of the dual is yb. If we perturb the b coefficients to take the values $b+\varepsilon$ for some "small" ε vector, we obtain $y(b+\varepsilon) = yb + y\varepsilon$. Since we have $c^\top x^* = y^* b$ at the optimum, the perturbed optimum is $c^\top x^* + y^*\varepsilon$. In other words the dual optimal solution y^* represents the variation of the objective function with respect to the unit variation in the constraints right hand side coefficients. In the present case a unit variation to b_1 yields a difference of $\frac{9}{5}$ in objective function cost, whilst for constraint 2 we get $\frac{4}{5}$ and for 3 we get zero. It is therefore convenient to increase b_1. Notice also that y^* is a bound to the amount of money that may be invested to increase b_1 by one unit.

9.8 Dual simplex method: Solution

The primal simplex method works by visiting a sequence of feasible bases converging to the optimal basis; in other words, it maintains feasibility while aiming to optimality. The dual simplex method, on the other hand, maintains optimality whilst working towards feasibility: it visits a sequence of dual feasible bases (corresponding to primal infeasible bases whose objective function value is "super-optimal") whilst working towards primal feasibility. Typically, we start with an infeasible initial basis where all the reduced costs of the nonbasic variables are non-negative (for minimization). I.e. the initial basis is already optimal (with respect to its nonbasic reduced costs) but it is infeasible.

		x_1	x_2	x_3	x_4	x_5
$-z$	0	3	4	5	0	0
x_4	-6	$\boxed{-2}$	-2	-1	1	0
x_5	-5	-1	-2	-3	0	1

Observe that $\bar{b}_4 = -6 < 0$ and hence the value of x_4 is not primal feasible; therefore x_r (with $r = 4$) exits the basis and is set to 0. We now wish to find an index $s \leq n$ such that x_s can enter the basis taking the place of x_r. The pivot element $\bar{a}_{rs}$ is determined by finding the minimum value $\frac{\bar{c}_s}{|\bar{a}_{rs}|}$ in

$$\{\frac{\bar{c}_j}{|\bar{a}_{rj}|} \mid \bar{a}_{rj} < 0, 1 \leq j \leq n\}.$$

We briefly explain why. Should $\bar{a}_{rj} \geq 0$ for every $j \leq n$, a pivot in $\bar{a}_{rj}$ would not change the sign of $\bar{b}_r$: this would mean that the primal is infeasible (no change of basis would yield $x_r \geq 0$). Should there be a j such that $\bar{a}_{rj} < 0$, however, a pivot operation in $\bar{a}_{rj}$ would make x_r primal feasible by setting it to 0 and making it exit the basis. This is why we only consider negative coefficients. Let us now see how the choice of s changes the coefficients of the objective function. The pivoting operations on the objective row will update it so that $\bar{c}_j \leftarrow \bar{c}_j - \frac{\bar{c}_s}{\bar{a}_{rs}}\bar{a}_{rj}$ for each $j \leq n$. To maintain dual feasibility it is necessary that $\bar{c}_j \geq 0$ for each $j \leq n$, and hence that $\bar{c}_j - \frac{\bar{c}_s}{\bar{a}_{rs}}\bar{a}_{rj} \geq 0$: so we need

$$\forall\, j \leq n \text{ such that } \bar{a}_{rj} < 0, \quad \frac{\bar{c}_s}{|\bar{a}_{rs}|} \leq \frac{\bar{c}_j}{|\bar{a}_{rj}|}.$$

In this instance we have

$$\frac{\bar{c}_1}{|\bar{a}_{11}|} = \frac{3}{2}; \frac{\bar{c}_2}{|\bar{a}_{12}|} = 2; \frac{\bar{c}_2}{|\bar{a}_{13}|} = 5.$$

The pivot element is $\bar{a}_{11}$ (as shown in the tableau) and the entering variable x_1. The pivot operation yields the new tableau:

		x_1	x_2	x_3	x_4	x_5
$-z$	-9	0	1	7/2	3/2	0
x_1	3	1	1	1/2	-1/2	0
x_5	-2	0	$\boxed{-1}$	-5/2	-1/2	1

The variable x_5 exits the basis (row 2) and x_2 enters (column 2). We get a tableau which is primal feasible (and hence optimal):

		x_1	x_2	x_3	x_4	x_5
$-z$	-11	0	0	1	1	1
x_1	1	1	0	-2	-1	1
x_2	2	0	1	5/2	1/2	-1

The optimal objective function value went from 0 to 9 to 11.

Chapter 10

Integer programming: Solutions

10.1 Piecewise linear objective: Solution

Let $a_0 = 0$, $a_1 = 1$, $a_2 = 2$, $a_3 = 3$. We get $f(a_0) = 1$, $f(a_1) = 0$, $f(a_2) = 1$, $f(a_3) = 3/2$. Since f is piecewise linear, for $x \in [a_i, a_{i+1})$ and $i \in \{0, 1, 2\}$, f can be written as $f(x) = \lambda_i f(a_i) + \lambda_{i+1} f(a_{i+1})$ with $\lambda_i + \lambda_{i+1} = 1$ eand $\lambda_i, \lambda_{i+1} \geq 0$, where λ are real variables expressing the affince dependence of f on the interval $[f(a_i), f(a_{i+1})]$. We can therefore write

$$f(x) = \sum_{i=0}^{3} \lambda_i f(a_i)$$

and impose that at most 2 variables λ (having consecutive indices) should be strictly positive. In order to formalize this condition we employ binary variables signalling which interval is active. Let $z_1 = 1$ if $0 \leq x < 1$ and 0 otherwise, $z_2 = 1$ if $1 \leq x < 2$ and 0 otherwise, and $z_3 = 1$ if $x \geq 0$ and 0 otherwise. In order to express that exactly one interval is active, we use the constraint:

$$\sum_{i=1}^{3} z_i = 1.$$

In order to express the condition on the positivity of at most 2 λ variables (having consecutive indices) we use the following constraints:

$$\begin{array}{rcl} \lambda_0 & \leq & z_1 \\ \lambda_1 & \leq & z_1 + z_2 \\ \lambda_2 & \leq & z_2 + z_3 \\ \lambda_3 & \leq & z_3. \end{array}$$

We obtain:

$$\left.\begin{array}{rrcl} \min & \lambda_0 + \lambda_2 + \frac{3}{2}\lambda_3 & & \\ & \lambda_0 & \leq & z_1 \\ & \lambda_1 & \leq & z_1 + z_2 \\ & \lambda_2 & \leq & z_2 + z_3 \\ & \lambda_3 & \leq & z_3 \\ & z_1 + z_2 + z_3 & = & 1 \\ & \lambda_0, \lambda_1, \lambda_2, \lambda_3 & \geq & 0 \\ & z_1, z_2, z_3 & \in & \{0, 1\}. \end{array}\right\}$$

10.2 Gomory Cuts: Solution

First of all, we reformulate the problem from canonical to standard form:

$$\left.\begin{array}{lrcl} \min & x_1 - 2x_2 & & \\ t.c. & -4x_1 + 6x_2 + x_3 & = & 9 \\ & x_1 + x_2 + x_4 & = & 4 \\ & x \geq 0 & , & x_1, x_2 \in \mathbb{Z} \end{array}\right\}$$

where x_3, x_4 are slack variables.

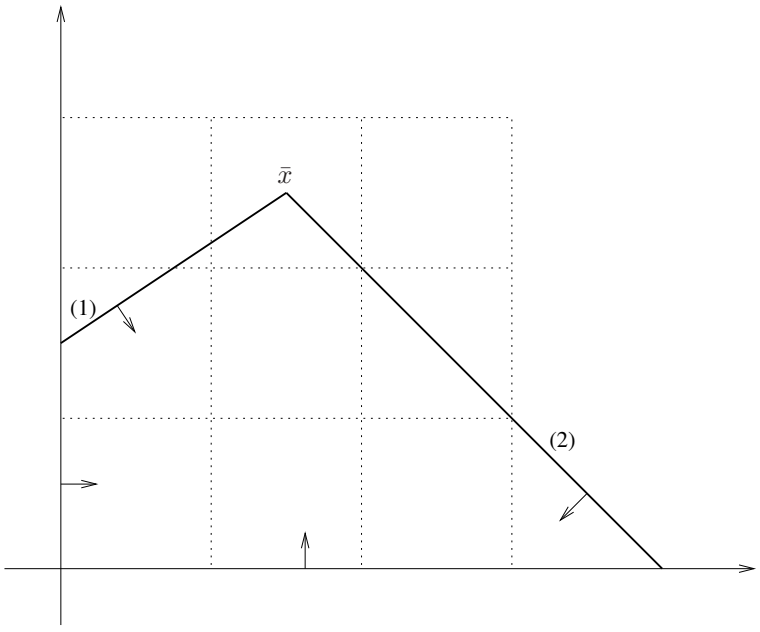
Using the simplex method applied to the feasible basis $x_B = (x_3, x_4)$, we obtain the following tableaux sequence (the pivot element is emphasized as $\boxed{\text{p}}$):

	x_1	x_2	x_3	x_4
0	1	-2	0	0
9	-4	$\boxed{6}$	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
$\frac{5}{2}$	$\boxed{\frac{5}{3}}$	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	$-\frac{1}{10}$	$\frac{3}{5}$

Therefore the optimal solution of the relaxation is $\bar{x} = (\frac{3}{2}, \frac{5}{2})$, with slack variables $x_3 = x_4 = 0$ (also see the figure below).



From the optimal tableau, we derive a Gomory cut from the first row $x_2 + \frac{1}{10}x_3 + \frac{2}{5}x_4 = \frac{5}{2}$. The Gomory cut is expressed as

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor, \tag{10.1}$$

where N is the set of indices of the nonbasic variables and i is the index of the chosen row. In this case, we obtain $x_2 \leq 2$.

We now have to add the Gomory cut $x_2 \leq 2$ in the current (optimal) simplex tableau. By definition, a valid cut "cuts off" the current optimal relaxed solution from the feasible region, the currently optimal

basis ceases to be feasible after the Gomory cut is inserted. The primal simplex algorithm relies on having a feasible basis at all times, so it cannot be used. The dual simplex algorithm, on the other hand, relies on having a basis which is always optimal (or super-optimal) and works towards reaching feasibility. The currently optimal basis becomes "super-optimal" after the insertion of the Gomory cut in the sense that the new optimal solution will surely have a higher objective function value (recall we are minimizing the objective) since it is constrained by one more inequality.

In order to insert the constraint $x_2 \leq 2$ in the tableau it is necessary to express it in function of the nonbasic variables x_3, x_4. If from (10.1) we subtract the i-th row of the optimal tableau

$$x_i + \sum_{j \in N} \bar{a}_{ij} x_j \leq \bar{b}_i$$

we obtain the Gomory cut in fractional form:

$$\sum_{j \in N} (\lfloor \bar{a}_{ij} \rfloor - \bar{a}_{ij}) x_j \leq (\lfloor \bar{b}_i \rfloor - \bar{b}_i).$$

Applied to our instance this is:

$$-\frac{1}{10} x_3 - \frac{2}{5} x_4 \leq -\frac{1}{2}.$$

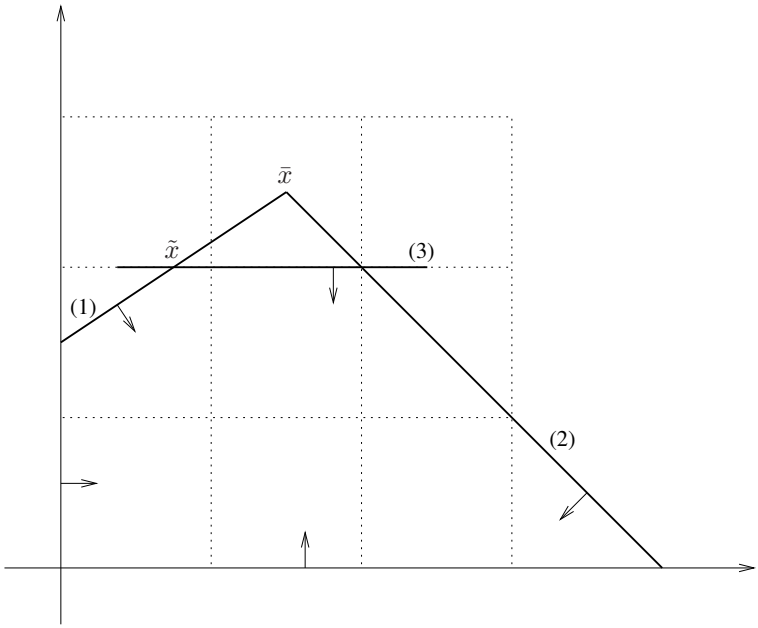
Since the simplex algorithm in tableau form requires all inequalities to be in equation form, we need to add a slack variable $x_5 \geq 0$ to the problem:

$$-\frac{1}{10} x_3 - \frac{2}{5} x_4 + x_5 = -\frac{1}{2}.$$

In this form, the equation can be added to the currently optimal tableau, which gains a new row and column (corresponding respectively to the new cut and the new slack variable, which is inserted 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 $x_2 \leq 2$, depicted in the figure below as constraint (3).



We carry out an iteration of the dual simplex algorithm using the new tableau. The reduced costs are all non-negative, but $\bar{b}_3 = -\frac{1}{2} < 0$ implies that $x_5 = \bar{b}_3$ has nagative value and so is not primal feasible (recall $x_5 \geq 0$ is a constraint in the problem). We pick x_5 to exit the basis. The entering variable is given by the index j such that

$$j = \text{argmin}\{\frac{\bar{c}_j}{|\bar{a}_{ij}|} \mid j \leq n \wedge \bar{a}_{ij} < 0\},$$

which in this case is the minimum index in $\{3, \frac{1}{2}\}$, i.e. $j = 4$. Therefore x_4 enters the basis instead of x_5, and the pivot element is that indicated in the tableau above. We get the new tableau

	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	$-\frac{5}{2}$

with optimal solution $\tilde{x} = (\frac{3}{4}, 2)$. The solution is not yet integer, so we choose the second row:

$$x_1 - \frac{1}{4}x_3 + \frac{3}{2}x_5 = \frac{3}{4},$$

whence we obtain the Gomory cut

$$x_1 - x_3 + x_5 \leq 0,$$

which expressed in the original problem variable is

$$-3x_1 + 5x_2 \leq 7.$$

The fractional form of this Gomory cut is

$$-\frac{3}{4}x_3 - \frac{1}{2}x_5 \leq -\frac{3}{4}.$$

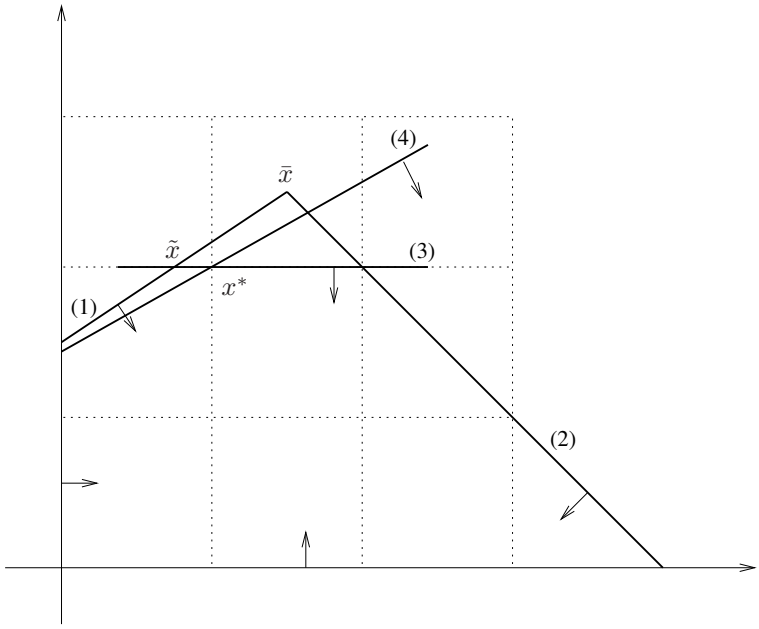
We insert it in the tableau as before and we obtain:

	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

where the pivot element is emphasized (row 4 was selected because $\bar{b}_4 < 0$, column 5 because $\frac{\bar{c}_3}{|\bar{a}_{43}|} = \frac{1}{3} < 1 = \frac{\bar{c}_5}{|\bar{a}_{45}|}$). Pivoting yields

	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	$\frac{2}{3}$	$-\frac{4}{3}$

which corresponds to the optimal integer solution $x^* = (1, 2)$ depicted below.



10.3 Branch and Bound I: Solution

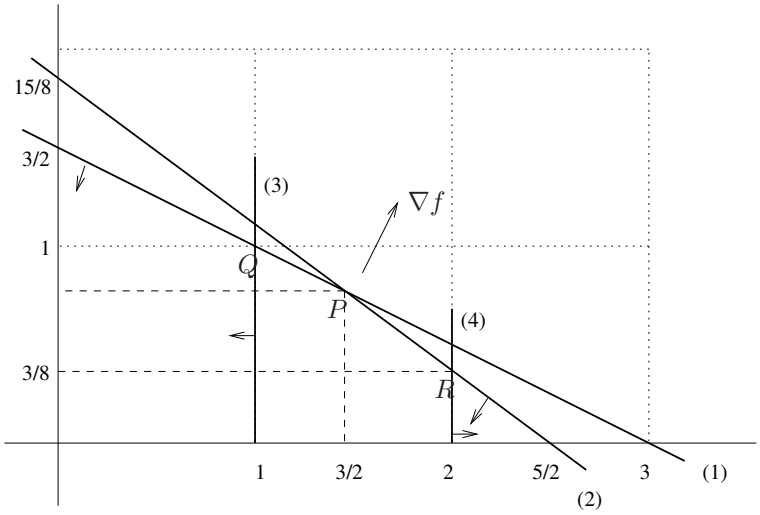
Branch-and-Bound is a tree-like search which relies on recursively partitioning the solution space and examining the subproblem (also called the *Branch-and-Bound node*) consisting of the original problem restricted to each partition set. A lower bound (upper bound if considering a maximization problem, as in this instance) is computed for each subproblem by solving its linear relaxation using linear programming techniques. A node is *fathomed* (i.e. no more recursive partitioning is carried out on that node) if one of the three following conditions occurs: (a) the relaxation solution is integer; (b) the problem is infeasible; (c) the relaxed objective function value is worse than the objective function value of the best integral solution found so far. Should any subproblem be unbounded, the original problem is also unbounded.

Partitioning is carried out in a number of ways. One of the simplest way is to choose the variable x_i whose non-integral solution value $\bar{x}_i$ is closest to midpoint of the interval $[\lfloor \bar{x}_i \rfloor, \lceil \bar{x}_i \rceil]$ and to enforce the constraints $x_i \leq \lfloor \bar{x}_i \rfloor$ and respectively $x_i \geq \lceil \bar{x}_i \rceil$ in the two new created subproblems. This is a partition of the solution space because it excludes no integral value. At any particular iteration, the subproblem to be chosen for examination is that whose parent in the tree has the best relaxed objective value. The algorithm terminates when there are no more subproblems to be solved.

Let $x^* = (\infty, \infty)$ be the best solution so far (the *incumbent*) and $f^* = -\infty$ its objective function value. At the first node N_1 we solve the following subproblem:

$$\left.\begin{array}{rrcl} \max & 2x_1 + 3x_2 & & \\ & x_1 + 2x_2 & \leq & 3 \\ & 6x_1 + 8x_2 & \leq & 15 \\ & x_1, x_2 & \in & \mathbb{R}_+. \end{array}\right\}$$

The drawing below shows that the solution is the intersection P of the line (1) defined by $x_1 + 2x_2 = 3$ and the line (2) defined by $6x_1 + 8x_2 = 15$.

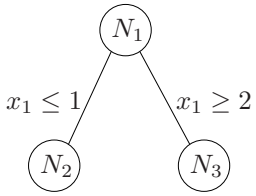


We obtain an upper bound $f = \frac{21}{4}$ with fractional solution $x = P = (\frac{3}{2}, \frac{3}{4})$. We choose x_1 for branching because its fractional part is closest to $\frac{1}{2}$. We form two subnodes N_2, N_3; in N_2 we add the constraint $x_1 \leq 1$ (line (3) $x_1 = 1$) and in N_3 the constraint $x_1 \geq 2$ (line (4) $x_1 = 2$).

In N_2 the solution is at the intersection Q of the lines (1) and (3): we therefore obtain $x = Q = (1, 1)$ and $f = 5$; since the solution is integer, the node is fathomed and no further branching occurs. Since $x \in \mathbb{Z}+^2$ e $f > f^*$, we update $x^* = x$ and $f^* = f$.

In N_3 the solution is at the intersection R of lines (2) and (4): we get $x = R = (2, \frac{3}{8})$ and $f = \frac{41}{8}$. Since the upper bound is 41/8 and $\lfloor \frac{41}{8} \rfloor = 5$, every integer solution in this node or its recursive partitions would have objective function value necessary worst than f^*. Hence the node is fathomed and no further branching occurs in this node.

Since there are no more nodes to be examined, the algorithm terminates with solution $x^* = (1, 1)$. The picture below represents the Branch-and-Bound tree.



10.4 Branch and Bound II: Solution

At each node we solve the relaxed LP problem graphically: by inspection we identify which among the vertices of the feasible polyhedron is optimal, and the two lines at whose intersection it lies. To find the coordinates of the optimal vertex, we solve a small sytem of equations. The solution is given in Fig. 10.4. The tree nodes are visited in the following order: P1, P2, P3, P4, P5, P6, P7. After having solved P7, node P6 is fathomed because its integer solution is worse than P7's; node P2 is equally fathomed because its upper bound (we are maximizing) is worse than the best integer solution (in P7): $x^* = (0, 3)$, $z^* = 12$. Notice that whenever the optimal relaxed objective value is fractional, the node upper bound is given by $\lfloor \bar{z} \rfloor$. For example, in P1 the upper bound is $\lfloor \frac{51}{4} \rfloor = 12$.

10.5 Knapsack Branch and Bound: Solution

The formulation of the problem is as follows:

$$\begin{array}{rrl} \max & 16x_1 + 22x_2 + 12x_3 + 8x_4 & \\ & 5x_1 + 7x_2 + 4x_3 + 3x_4 & \leq 14 \\ & x_1, x_2, x_3, x_4 & \in \{0, 1\}. \end{array}$$

In order to solve the continuous relaxation of this integer programming problem, we order the ratios between revenue and cost of each investment:

$$(16/5, 22/7, 12/4, 8/3) = (3.2, 3.14, 3, 2.66).$$

It follows that 1,2,3,4 is the preference order given by revenue per unit investment cost. We now take fractions of the various investments in the given preference order (recall we are solving the continuous relaxation now, so fractions are allowed) so that (a) the fractions grow to be at most 1 and (b) the knapsack constraint $5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$ is satisfied. For example, at node 1 we have $x_1 = 1$ (total invested amount: 5 million euros), $x_2 = 1$ (total invested amount: 5+7=12 million euros), $x_3 = \frac{1}{2}$ (total invested amount: 12+2=14 million euros). We can use this solution method within a Branch and Bound algorithm. The latter adds constraints of the type $x_i \leq 0$ or $x_i \geq 1$ at each relaxed subproblem, which must be easily taken into account by forcing either the zero or the total investment of particular type.

The Branch and Bound tree is given in Fig. 10.2.

- Index t specifies the order of subproblem solution.
- *Upper bound computation* (recall we are maximizing). At each node we solve the linear program associated to the subproblem with the method described above. If its solution $\bar{z}$ is fractional, since the coefficients and solution of the problem are integer, we can consider $\lfloor \bar{z} \rfloor$ as the upper bound.
- *Lower bound computation*. The lower bound (LB in Fig. 10.2) is used to fathom those nodes where $\bar{z} \leq$ LB (which may not possibly contain an optimal solution). Whilst the upper bound is computed at each node, the lower bound is initialized at $-\infty$ and updated during the algorithm. Each time

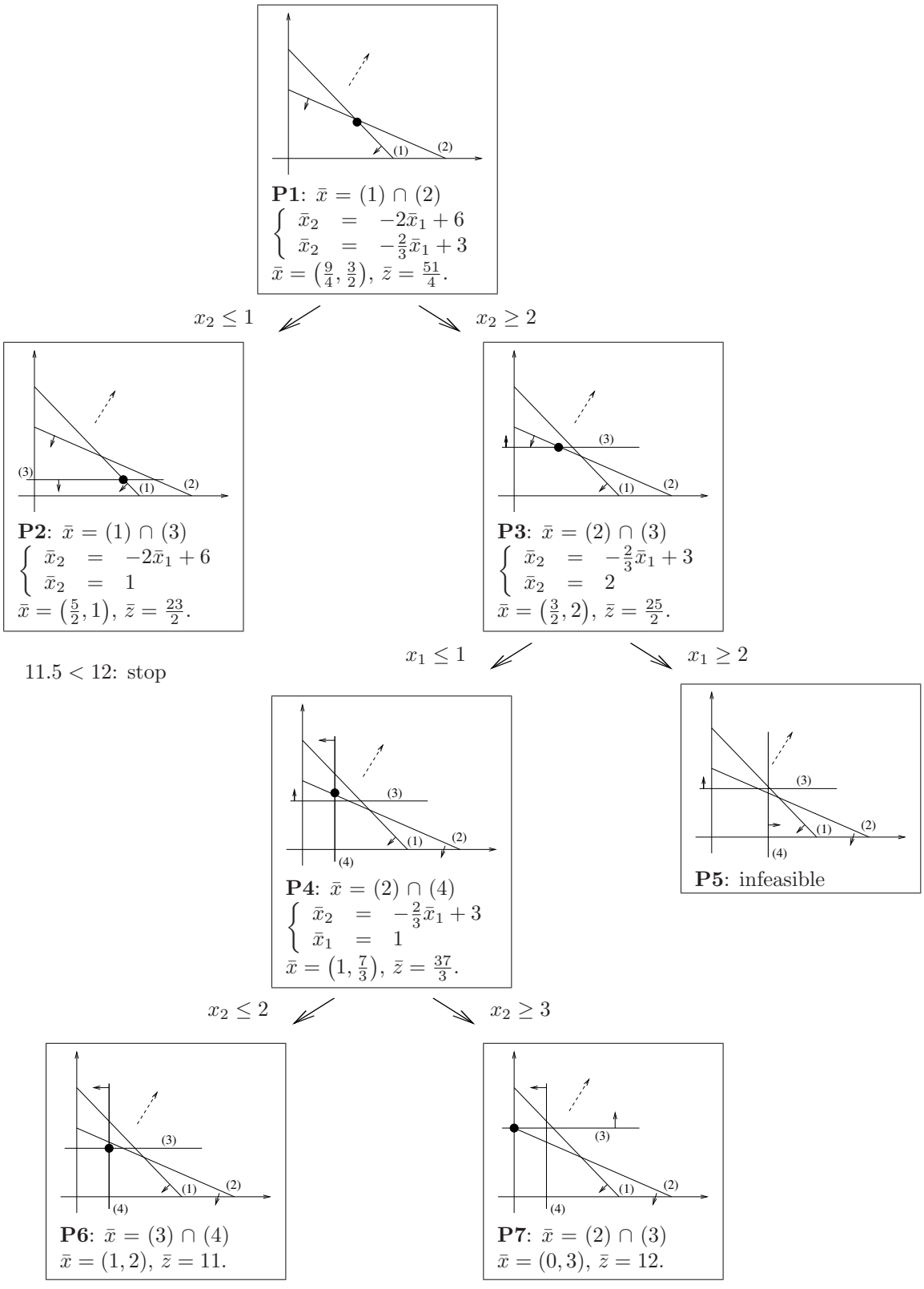


Figure 10.1: Branch-and-Bound tree for Exercise 3.4.

a subproblem generates an integral solution with (integer) value $\bar{z}$, if $\bar{z} >$ LB then LB is updated to $\bar{z}$. For example, in subproblem 4 we find an integer solution with $\bar{z} = 36$. Since $LB = -\infty$ and

$\bar{z} > -\infty$, LB is updated to 36.

- In subproblem 6 we find an integral solution with $\bar{z} = 42$, so LB = 42. Therefore fathoms node 4.
- Subproblem 7 is infeasible because if $\bar{x}_1 = \bar{x}_2 = \bar{x}_3 = 1$, then the required invested amount is 16 (more than the budget).
- Subproblem 8 is fathomed because $\bar{z} = 38$ is smaller than the current LB (which is 42), so node 8 cannot contain an optimal solution.
- At node 9 we have $\bar{z} = 42\frac{6}{7}$, which makes the upper bound $\lfloor \bar{z} \rfloor = 42$. Again, this means that node 9 may not contain a solution better than what we already found at node 6. Therefore node 9 is fathomed. As there are no more open subproblems, the algorithm terminates with optimal solution $x^* = (0, 1, 1, 1)$ with value 42.

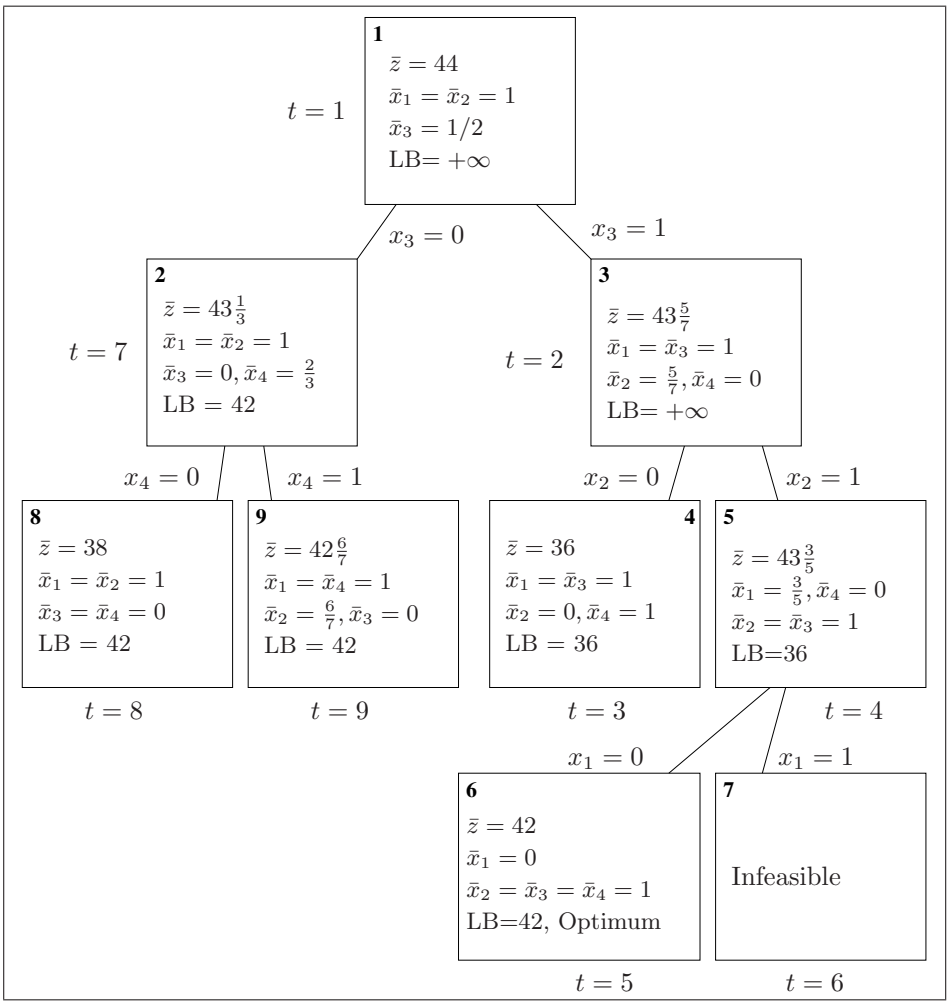


Figure 10.2: Branch and Bound tree for Exercise 3.5.

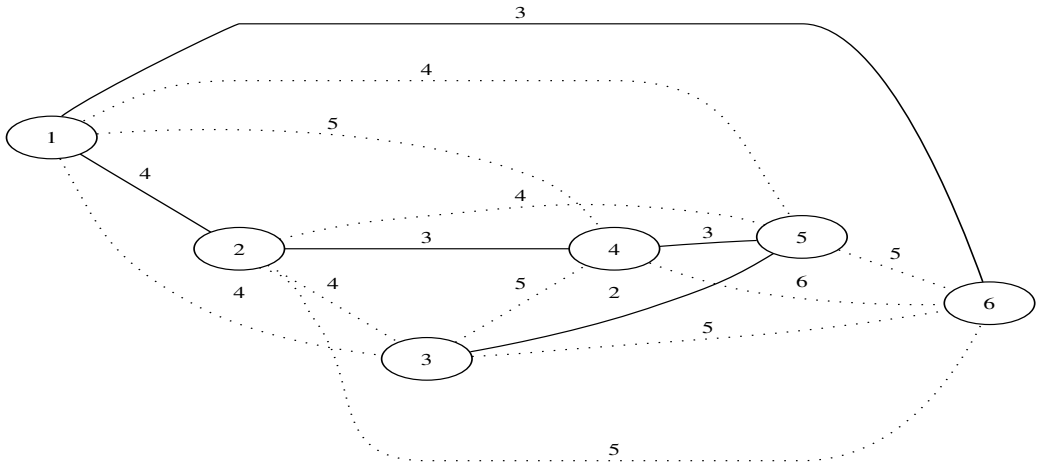
Chapter 11

Easy modelling problems: solutions

11.1 Compact storage of similar sequences: Solution

Consider a complete undirected graph $G = (V, E)$ where each vertex is a sequence and the weight of an edge $\{i, j\} \in E$ is given by the Hamming distance between sequence i and sequence j. To each edge $\{i, j\} \in E$ we also associate the sequence of bit flips necessary to transform sequence i into sequence j. A minimum cost spanning tree in G provides the most economical way to recover all possible sequences starting from only one of these sequences.

The instance in the exercise yields a minimum spanning tree having cost 15.



11.2 Communication of secret messages: Solution

The communication network is represented by a directed graph $G = (V, A)$. Each arc (i, j) is weighted by its probability $1 - p_{ij}$ that the the message is not intercepted along the arc. In order to broadcast the message to all nodes we want to find a subset of arcs which is connected, reaches all nodes, and has no cycle (otherwise the interception probability might increase). In other words, a spanning tree. The spanning tree T should maximize the chances that the message arrives at each node without interception,

i.e.:

$$\max_{\text{all } T}\{\prod_{\{i,j\}\in T}(1-p_{ij}) \mid T \text{ spanning tree}\}. \tag{11.1}$$

Since the Prim (and Kruskal) algorithms for finding optimum spanning trees deal with the case when the cost of the tree is the sum of the costs of the edges, we cannot use those algorithms to solve the problem.

However, we can reformulate the problem by requiring the spanning tree T which maximizes the modified objective function $\log \prod_{\{i,j\}\in T}(1-p_{ij})$. This will change the value of the objective function associated to the solution but not the solution itself, since the log function is monotonic increasing.

$$\begin{aligned}
\log \quad & \max_{\text{all } T}\{\prod_{\{i,j\}\in T}(1-p_{ij}) \mid T \text{ spanning tree}\} = \\
= \quad & \max_{\text{all } T}\{\log \prod_{\{i,j\}\in T}(1-p_{ij}) \mid T \text{ spanning tree}\} = \\
= \quad & \max_{\text{all } T}\{\sum_{\{i,j\}\in T}\log(1-p_{ij}) \mid T \text{ spanning tree}\}.
\end{aligned}$$

The latter is a "proper" minimum spanning tree problem on the graph G where each arc $(i,j) \in A$ is weighted by $\log(1-p_{ij})$, and can be solved using either Prim's algorithm.

11.3 Mixed production: Solution

11.3.1 Formulation

- *Indicex*: Let i be an index on the set $\{1,2,3\}$.
- *Parameters*:
  - P: number of production days in a month;
  - d_i: maximum market demand for product i;
  - v_i: selling price for product i;
  - c_i: production cost for product i;
  - q_i: maximum production quota for product i;
  - a_i: activation cost for the plant producing i;
  - l_i: minimum batch of product i.
- *Variables*:
  - x_i: quantity of product i to produce ($x_i \geq 0$);
  - y_i: activation status of product i (1 if active, 0 otherwise).
- *Objective function*:

$$\max \sum_i ((v_i - c_i)x_i - a_i y_i)$$

- *Constraints*:
  1. (demand): for each i, $x_i \leq d_i$;
  2. (production): $\sum_i \frac{x_i}{q_i} \leq P$;
  3. (activation): for each i, $x_i \leq P q_i y_i$;
  4. (minimum batch): for each i, $x_i \geq l_i y_i$;

11.3.2 AMPL model, data, run

```
# mixedproduction.mod

set PRODUCTS;

param days >= 0;
param demand { PRODUCTS } >= 0;
param price { PRODUCTS } >= 0;
param cost { PRODUCTS } >= 0;
param quota { PRODUCTS } >= 0;
param activ_cost { PRODUCTS } >= 0;    # activation costs
param min_batch { PRODUCTS } >= 0;     # minimum batches

var x { PRODUCTS } >= 0;               # quantity of product
var y { PRODUCTS } >= 0, binary;       # activation of production lines

maximize revenue: sum {i in PRODUCTS}
((price[i] - cost[i]) * x[i] - activ_cost[i] * y[i]);

subject to requirement {i in PRODUCTS}:
x[i] <= demand[i];

subject to production:
sum {i in PRODUCTS} (x[i] / quota[i]) <= days;

subject to activation {i in PRODUCTS}:
x[i] <= days * quota[i] * y[i];

subject to batches {i in PRODUCTS}:
        x[i] >= min_batch[i] * y[i];


# mixedproduction.dat

set PRODUCTS := A1 A2 A3 ;

param days := 22;
param : demand  price   cost  quota  activ_cost min_batch :=
     A1 5300    124     73.30   500      170000        20
     A2 4500    109     52.90   450      150000        20
     A3 5400    115     65.40   550      100000        16 ;


# mixedproduction.run

model mixedproduction.mod;
data mixedproduction.dat;
option solver cplexstudent;
solve;
display x;
display y;
```

11.3.3 CPLEX solution

```
.

CPLEX 7.1.0: optimal integer solution; objective 220690
```

```
5 MIP simplex iterations
0 branch-and-bound nodes
ampl: display x;
x [*] :=
A1     0
A2  4500
A3  5400
;

ampl: display y;
y [*] :=
A1  0
A2  1
A3  1
;
```

11.4 Production planning: Solution

11.4.1 Formulation

- *Indices*:
  - i: an index on the set $\pi = \{A_1, A_2, A_3\}$;
  - j: an index on the set $\mu = \{1, 2, 3, 4\}$.
- *Parameters*:
  - P_j: number of production days in month j;
  - d_{ij}: maximum demand for product i in month j;
  - v_i: selling price for product i;
  - c_i: production cost of product i;
  - q_i: maximum production quota of product i;
  - a_i: activation cost for production i;
  - l_i: minimum batch for production i;
  - m_i: storage cost for product i;
  - C: storage capacity in number of units.
- *Variables*:
  - x_{ij}: quantity of product i produced during month j;
  - w_{ij}: quantity of product i sold during month j;
  - z_{ij}: quantity of product i stocked during month j;
  - y_{ij}: activation status for production i: (1=active, 0=inactive).

  All variables are constrained to be non-negative. y_{ij} are binary variables.
- *Objective function*:

$$\max \sum_{i=1}^{3} \left(v_i \sum_{j=1}^{4} w_{ij} - c_i \sum_{j=1}^{4} x_{ij} - m_i \sum_{j=1}^{4} z_{ij} - a_i \sum_{j=1}^{4} y_{ij} \right).$$

- *Constraints*:
  1. (requirement): for each i, j: $w_{ij} \leq d_{ij}$;
  2. (production): per each j: $\sum_{i=1}^{3} \frac{x_{ij}}{q_i} \leq P_j$;
  3. (balance): for each i, j: $z_{i,j-1} + x_{ij} = z_{ij} + w_{ij}$;
  4. (capacity): for each j: $\sum_{i=1}^{3} z_{ij} \leq C$;
  5. (activation): for each i, j: $x_{ij} \leq P_j q_i y_{ij}$;
  6. (minimum batch): for each i, j: $x_{ij} \geq l_i y_{ij}$;
  7. (december): for each i: $z_{i0} = 0$.

11.4.2 AMPL model, data, run

```
# productionplan.mod

set PRODUCTS;

param Months;

set MONTHS := 1..Months;
set MONTHS0 := MONTHS union {0};

param days{MONTHS} >= 0;
param demand { PRODUCTS, MONTHS } >= 0;
param price { PRODUCTS } >= 0;
param cost { PRODUCTS } >= 0;
param quota { PRODUCTS } >= 0;
param activation { PRODUCTS } >= 0;
param batch { PRODUCTS } >= 0;
param storage { PRODUCTS } >= 0;
param capacity >= 0;

var x { PRODUCTS, MONTHS } >= 0;
var w { PRODUCTS, MONTHS } >= 0;
var z { PRODUCTS, MONTHS0 } >= 0;
var y { PRODUCTS, MONTHS } >= 0, binary;

maximize revenue:
sum {i in PRODUCTS}
(price[i] * sum {j in MONTHS} w[i,j] -
 cost[i]   * sum {j in MONTHS} x[i,j] -
 storage[i]  * sum {j in MONTHS} z[i,j] -
                activation[i] * sum {j in MONTHS} y[i,j]) ;

subject to requirement {i in PRODUCTS, j in MONTHS}:
w[i,j] <= demand[i,j];

subject to production {j in MONTHS}:
sum {i in PRODUCTS} (x[i,j] / quota[i]) <= days[j];

subject to bilance {i in PRODUCTS, j in MONTHS}:
z[i,j-1] + x[i,j] = z[i,j] + w[i,j];

subject to capacitymag {j in MONTHS}:
sum {i in PRODUCTS} z[i,j] <= capacity;
```

```
subject to active {i in PRODUCTS, j in MONTHS}:
        x[i,j] <= days[j]*quota[i]*y[i,j];

subject to minbatch {i in PRODUCTS, j in MONTHS}:
        x[i,j] >= batch[i]*y[i,j];


# productionplan.dat

set PRODUCTS := A1 A2 A3 ;

param Months := 4 ;

param days :=
     1  23
     2  20
     3  23
     4  22 ;

param demand:  1       2          3          4    :=
        A1     5300   1200      7400        5300
        A2     4500   5400      6500        7200
        A3     4400   6700     12500       13200 ;

param : price   cost quota  activation batch storage :=
     A1     124 73.30 500    150000      20    3.5
     A2     109 52.90 450    150000      20    4
     A3     115 65.40 550    100000      16    3   ;

param capacity := 800 ;

let {i in PRODUCTS} z[i,0] := 0;
fix {i in PRODUCTS} z[i,0];


# productionplan.run

model productionplan.mod;
data productionplan.dat;
option solver cplexstudent;
solve;
option display_round 4;
display revenue;
display x;
display y;
quit;
```

11.4.3 CPLEX solution

```
CPLEX 7.1.0: optimal integer solution; objective 1581550
47 MIP simplex iterations
0 branch-and-bound nodes
guadagno = 1581550.0000

x :=
A1 1    6100.0000
A1 2       0.0000
A1 3       0.0000
```

```
A1 4       0.0000
A2 1       0.0000
A2 2    3518.1818
A2 3       0.0000
A2 4       0.0000
A3 1    4400.0000
A3 2    6700.0000
A3 3   12650.0000
A3 4   12100.0000 ;

y :=
A1 1   1.0000
A1 2   0.0000
A1 3   0.0000
A1 4   0.0000
A2 1   0.0000
A2 2   1.0000
A2 3   0.0000
A2 4   0.0000
A3 1   1.0000
A3 2   1.0000
A3 3   1.0000
A3 4   1.0000 ;
```

11.5 Transportation: Solution

11.5.1 Formulation

- *Indices*:
  - i: index on the set $\{1, \ldots, M\}$ (stores);
  - j: index on the set $\{1, \ldots, P\}$ (ports);
- *Parameters*:
  - m_i: availability (in number of containers) at i-th store;
  - r_j: demand at j-th port;
  - d_{ij}: distance between store i and port j;
  - C: unit transportation cost (per km).
- *Variables*:
  - x_{ij}: number of containers sent from store i to port j;
  - y_{ij}: number of lorries travelling from store i to port j;

  All variables are constrained to be non-negative.
- *Objective function*:

$$\min \sum_{i=1}^{M} \sum_{j=1}^{P} C d_{ij} y_{ij}$$

- *Constraints*:
  1. (store availability) for each $i \leq M$: $\sum_{j=1}^{P} x_{ij} \leq m_i$;
  2. (port demand) for each $j \leq P$: $\sum_{i=1}^{M} x_{ij} \geq r_j$;
  3. (lorry capacity) for each $i \leq M$, $j \leq P$, $2y_{ij} \geq x_{ij}$.

11.5.2 AMPL model, data, run

```
# transportation.mod

set STORES;
set PORTS;

param availability { STORES } >= 0;
param demand { PORTS } >= 0;
param distance { STORES, PORTS } >= 0;
param costkm >= 0;

var x { STORES, PORTS } >= 0;
var y { STORES, PORTS } >= 0, integer;

minimize cost:
sum {i in STORES, j in PORTS} costkm * distance[i,j] * y[i,j];


subject to avail {i in STORES}:
sum {j in PORTS} x[i,j] <= availability[i];

subject to request {j in PORTS}:
sum {i in STORES} x[i,j] >= demand[j];

subject to lorrycap {i in STORES, j in PORTS}:
        2*y[i,j] >= x[i,j];


# transportation.dat

set STORES := Verona Perugia Rome Pescara Taranto Lamezia;
set PORTS := Genoa Venice Ancona Naples Bari;

param availability :=
        Verona   10
        Perugia  12
        Rome     20
        Pescara  24
        Taranto  18
        Lamezia  40 ;

param demand :=
        Genoa    20
        Venice   15
        Ancona   25
        Naples   33
        Bari     21 ;

param distance :
                 Genoa  Venice  Ancona   Naples  Bari :=
        Verona    290     115    355      715     810
        Perugia   380     340    165      380     610
        Rome      505     530    285      220     450
        Pescara   655     450    155      240     315
        Taranto  1010     840    550      305      95
        Lamezia  1072    1097    747      372     333 ;

param costkm := 300;
```

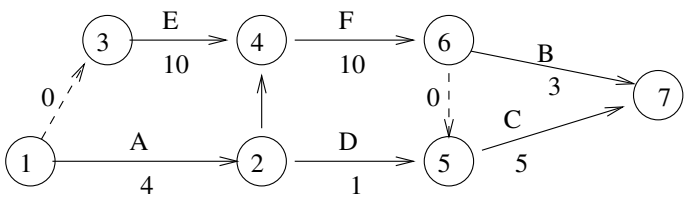
```
# transportation.run

model transportation.mod;
data transportation.dat;
option solver cplexstudent;
solve;
option display_round 4;
display cost;
display x;
display y;
```

11.5.3 CPLEX solution

```
CPLEX 7.1.0: optimal integer solution; objective 4685100
70 MIP simplex iterations
0 branch-and-bound nodes
costo = 4685100.0000

x [*,*]
:         Ancona     Bari     Genova    Napoli    Venezia    :=
Lamezia    0.0000    4.0000    0.0000   26.0000    0.0000
Perugia    1.0000    0.0000    6.0000    0.0000    5.0000
Pescara   24.0000    0.0000    0.0000    0.0000    0.0000
Roma       0.0000    0.0000   14.0000    6.0000    0.0000
Taranto    0.0000   17.0000    0.0000    1.0000    0.0000
Verona     0.0000    0.0000    0.0000    0.0000   10.0000
;

y [*,*]
:         Ancona     Bari     Genova    Napoli    Venezia    :=
Lamezia    0.0000    2.0000    0.0000   13.0000    0.0000
Perugia    1.0000    0.0000    3.0000    0.0000    3.0000
Pescara   12.0000    0.0000    0.0000    0.0000    0.0000
Roma       0.0000    0.0000    7.0000    3.0000    0.0000
Taranto    0.0000    9.0000    0.0000    1.0000    0.0000
Verona     0.0000    0.0000    0.0000    0.0000    5.0000
;
```

11.6 Project planning with precedences: Solution

The precedence graph $G = (V, A)$ (which associates to each arc an activity) is as follows.



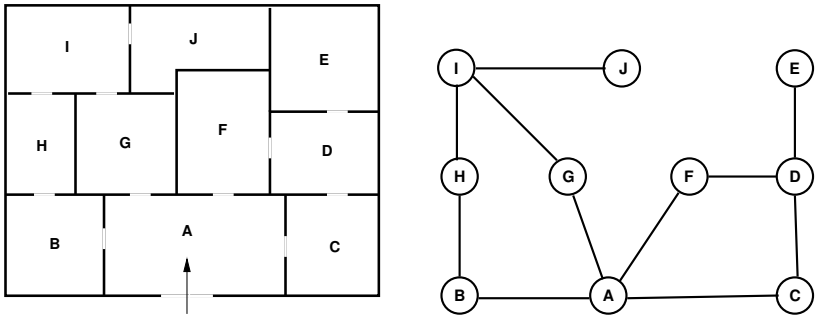
To each vertex $i \in V$ we associate a variable t_i (the starting time of the activities represented by arcs in $\bar{\delta}^+(i)$. The mathematical programming formulation of the problem is:

$$\begin{aligned} \min \quad & t_7 - t_1 + 5000(t_4 - t_2) \\ & t_i + d_{ij} \leq t_j \quad \forall\, (i,j) \in A, \end{aligned}$$

where d_{ij} is the cost of the arc (i, j).

11.7 Museum guards: Solution

The problem can be formalized by representing each museum room by a vertex $v \in V$ of an undirected graph $G = (V, E)$. There is an edge between two vertices if there is a door leading from one room to the other; this way, edges represent the possibility of there being a guard on a door. We want to choose the smallest subset $F \subseteq E$ of edges *covering* all vertices, i.e. such that for all $v \in V$ there is $w \in V$ with $\{v, w\} \in F$.



To each $\{i, j\} \in E$ we associated a binary variable x_{ij} is assigned the value 1 if there is a guard on the door represented by edge $\{i, j\}$ and 0 otherwise.

11.7.1 Formulation

- *Parameters.* $G = (V, A)$: graph description of the museum topology.
- *Variables.* x_{ij}: 1 if edge $\{i, j\} \in E$ is to be included in F, 0 otherwise.
- *Objective function*
$$\min \sum_{\{i,j\} \in E} x_{ij}$$
- *Constraints.* (Vertex cover): $\sum_{j \in V : \{i,j\} \in E} x_{ij} \geq 1 \quad \forall i \in V.$

11.7.2 AMPL model, data, run

```
# museum.mod

param n >= 0, integer;
set V := 1..n;
set E within {V,V};
var x{E} binary;
```

```
minimize cost : sum{(i,j) in E} x[i,j];
subject to vertexcover {i in V} :
  sum{j in V : (i,j) in E} x[i,j] + sum{j in V : (j,i) in E} x[j,i] >= 1;
```

```
# museum.dat

param n := 10;
set E :=
  1 2
  1 3
  1 6
  1 7
  2 8
  3 4
  4 5
  7 9
  8 9
  9 10 ;
```

```
# museum.run

model museum.mod;
data museum.dat;
option solver cplexstudent;
solve;
display cost;
display x;
```

11.7.3 CPLEX solution

```
CPLEX 7.1.0: optimal integer solution; objective 6
2 MIP simplex iterations
0 branch-and-bound nodes
cost = 6

x :=
1 2    0
1 3    1
1 6    1
1 7    1
2 8    1
3 4    0
4 5    1
7 9    0
8 9    0
9 10   1
;
```

11.8 Inheritance: Solution

The problem may be formalized as follows: given a set A of n elements each with an evaluation function $v : A \to \mathbb{R}$, we want to find a partition of A in A_1, A_2 such that

$$|v(A_1) - v(A_2)| = |\sum_{a \in A_1} v(a) - \sum_{a \in A_2} v(a)|$$

is minimum. This is known as the SUBSET-SUM problem.

It can be modelled using mathematical programming by introducing binary variables x_a, y_a for each $a \in A$, such that $x_a = 1$ and $y_a = 0$ if object a is assigned to brother x, and $x_a = 0$ and $y_a = 1$ if a is assigned to y. We naturally need the constraint

$$\forall a \in A \quad (x_a + y_a = 1).$$

The objective function to be minimized is:

$$\min |\sum_{a \in A_1} v_a x_a - \sum_{a \in A_2} v_a y_a|,$$

which ensures that the inheritance is split between the two brothers as fairly as possible. Because of the absolute value, this formulation is nonlinear.

Let $V = \sum_{a \in A} v(a)$ be the total value of the inheritance. The SUBSET-SUM can also be described as follows:

- maximize the inheritance assigned to one of the brothers with the constraint that it should not exceed $V/2$;
- minimize the inheritance assigned to one of the brothers with the constraint that it should not be less than $V/2$.

This interpretation gives us two integer linear programming formulations:

$$\left.\begin{array}{rll} \max & \sum_{a \in A} v_a x_a & \\ \text{s.t.} & \sum_{a \in A} v_a x_a & \leq \frac{V}{2} \sum_{a \in A} v_a \\ \forall a \in A & x_a \in \{0,1\} & \end{array}\right\}$$

$$\left.\begin{array}{rll} \min & \sum_{a \in A} v_a x_a & \\ \text{s.t.} & \sum_{a \in A} v_a x_a & \geq \frac{V}{2} \sum_{a \in A} v_a \\ \forall a \in A & x_a \in \{0,1\} & \end{array}\right\}$$

11.8.1 AMPL model, data, run

```
# subsetsum.mod

param n;
param v {1..n};
param V := sum {i in 1..n} v [i];
var x {1..n} binary;
minimize cost: sum {i in 1..n} v[i] * x[i];
subject to limit: sum {i in 1..n} v [i]* x[i] >= 0.5 * V;
```

```
# subsetsum.dat

param n := 13;
param: v :=
  1 25000
  2 5000
  3 20000
  4 40000
  5 12000
  6 12000
  7 12000
  8 3000
  9 6000
  10 10000
  11 15000
  12 10000
  13 13000;
```

```
# subsetsum.run

model subsetsum.mod;
data subsetsum.dat;
option solver cplexstudent;
solve;
display cost;
display x;
```

11.8.2 CPLEX solution

```
CPLEX 8.1.0: optimal integer solution; objective 92000
7 MIP simplex iterations
0 branch-and-bound nodes
cost = 92000

x [*] :=
 1  1
 2  0
 3  1
 4  0
 5  0
 6  0
 7  0
 8  1
 9  1
10  0
11  1
12  1
13  1
;
```

11.9 Carelland: Solution

Miximize the profits (exported quantities - produced quantities) subject to the constraints on production, amount of work and balance between produced and exported products.

11.9.1 Formulation

Parameters:

- P: set of products;
- H: total available amount of work (man-years);
- M_i maximum possible production for product $i \in P$;
- p_i market price for product $i \in P$;
- m_i amount of raw materials necessary to manufacture a unit of product $i \in P$;
- h_i amount of work required to manufacture a unit of product $i \in P$;

Variabili:

- x_a, x_m, x_p, x_e: produced units of steel, engines, plastics and electronics
- y_a, y_m, y_p, y_e: exported units of steel, engines, plastics and electronics.

Model:

$$\begin{aligned}
\max \sum_{i \in P} p_i y_i - \sum_{i \in P} m_i x_i & \\
\sum_{i \in P} h_i x_i \leq H & \\
x_i \leq M_i & \quad \forall i \in P \\
y_a + 0.8x_m + 0.01x_e + 0.2x_p = x_a & \\
y_m + 0.02x_a + 0.01x_e + 0.03x_p = x_m & \\
y_e + 0.15x_m + 0.05x_p = x_e & \\
y_p + 0.01x_a + 0.11x_m + 0.05x_e = x_p & \\
x_i, y_i \geq 0 & \quad \forall i \in P
\end{aligned}$$

11.9.2 AMPL model, data, run

```
# carelland.mod

set PRODUCTS;

param p {PRODUCTS} >= 0;
param HMan >=0;
param Max {PRODUCTS} >=0;
param m {PRODUCTS} >= 0;
param h {PRODUCTS} >= 0;
param a {PRODUCTS, PRODUCTS} >=0;


var x { PRODUCTS } >= 0;
var y { PRODUCTS } >= 0;
```

```
maximize klunz:
sum {i in PRODUCTS} (p[i]*y[i] - m[i]*x[i]);

subject to limit{i in PRODUCTS}:
x[i] <= Max[i];

subject to work:
sum {i in PRODUCTS} h[i]*x[i]<=HMan;


subject to balance{i in PRODUCTS} :
y[i] + sum{j in PRODUCTS}(a[j,i]*x[j]) = x[i];
```

```
# carelland.dat

set PRODUCTS := steel plastics electronics engines;

param HMan:= 830000;
param :             p          m     h        Max     :=
     steel        500        250     0.5     2000000
     plastics    1200        300     2         60000
     electronics  300         50     0.5      650000
     engines     1500        300     1       2000000  ;

param a:       steel           plastics         electronics        engines :=
steel           0                 0.01             0                0.02
plastics        0.2               0                0.05             0.03
electronics     0.01              0.05             0                0.01
engines         0.8               0.11             0.15             0 ;
```

```
# carelland.run

model carelland.mod;
data carelland.dat;
option solver cplexstudent;
solve;
display profit;
display x;
display y;
```

11.9.3 CPLEX solution

```
CPLEX 8.1.0: optimal solution; objective 435431250
9 dual simplex iterations (6 in phase I)
klunz = 435431000

x [*] :=
electronics   74375
    engines  475833
   plastics   60000
      steel  393958
;
```

```
y [*] :=
electronics       0
    engines  465410
   plastics       0
      steel     547.917
;
```

11.10 CPU Scheduling: Solution

- *Indices*:
  - i, j: indices on a set P of tasks;
  - k: index on a set C of CPUs.
- *Parameters*:
  - b_i: number of BI (billion instructions) in task i;
  - s_k: speed of CPU k in GHz;
  - $W_{\max}$: upper bound for completion time of all tasks.
- *Variables*:
  - $x_i \geq 0$: starting time of task i;
  - $y_i \in \mathbb{Z}_+$: CPU ID to which task i is assigned;
  - $z_{ik} = 1$ if task i is assigned to CPU k, 0 otherwise;
  - $\sigma_{ij} = 1$ if task i ends before task j starts, 0 otherwise;
  - $\varepsilon_{ij} = 1$ if task i is executed on a CPU having lower ID than task j;
  - $L_i \geq 0$: length of task i;
  - $W \geq 0$: completion time of all tasks.
- *Objective function*:
$$\min W$$
- *Constraints*:
  - (times) $\forall i \in P \quad (W \geq x_i + L_i)$
  - (lengths) $\forall i \in P \quad (L_i = \sum_{k \in C} \frac{b_i}{s_k} z_{ik})$;
  - (assignment) $\forall i \in P \quad (\sum_{k \in C} z_{ik} = 1)$;
  - (cpudef) $\forall i \in P \quad (y_i = \sum_{k \in C} k z_{ik})$
  - (horizontal non-overlapping) $\forall i \neq j \in P \quad (x_j - x_i \geq L_i - (1 - \sigma_{ij}) W_{\max})$
  - (vertical non-overlapping) $\forall i \neq j \in P \quad (y_j - y_i \geq 1 - (1 - \varepsilon_{ij})|P|)$
  - (at least one position) $\forall i \neq j \in P \quad (\sigma_{ij} + \sigma_{ji} + \varepsilon_{ij} + \varepsilon_{ji} \geq 1)$
  - (horizontal: at most one) $\forall i \neq j \in P \quad (\sigma_{ij} + \sigma_{ji} \leq 1)$
  - (vertical: at most one) $\forall i \neq j \in P \quad (\varepsilon_{ij} + \varepsilon_{ji} \leq 1)$

11.10.1 AMPL model, data, run

```
# cpuscheduling.mod

param p > 0, integer;
param c > 0, integer;
set P := 1..p;
set C := 1..c;

param b{P} >= 0;
param s{C} >= 0;
param Wmax default sum{i in P} b[i] / (min{k in C} s[k]);

var x{P} >= 0;              # starting time of task
var y{P} >= 0;              # CPU of task
var z{P,C} binary;          # assignment task/CPU
var sigma{P,P} binary;      # is task i scheduled before task j?
var epsilon{P,P} binary;    # is task i scheduled on a lower CPU than task j?
var L{P} >= 0;              # length of task
var W >= 0;                 # makespan

minimize makespan: W;

subject to times{i in P} : W >= x[i] + L[i];

subject to lengths{i in P} : L[i] = sum{k in C} (b[i]/s[k]) * z[i,k];

subject to assignment{i in P} : sum{k in C} z[i,k] = 1;

subject to cpudef{i in P} : y[i] = sum{k in C} k * z[i,k];

subject to hnonoverlapping{i in P, j in P : i != j} :
  x[j] - x[i] >= L[i] - (1-sigma[i,j]) * Wmax;

subject to vnonoverlapping{i in P, j in P : i != j} :
  y[j] - y[i] >= 1 - (1-epsilon[i,j]) * card(P);

subject to atleastone{i in P, j in P : i != j} :
  sigma[i,j] + sigma[j,i] + epsilon[i,j] + epsilon[j,i] >= 1;

subject to hatmostone{i in P, j in P : i != j} :
  sigma[i,j] + sigma[j,i] <= 1;

subject to vatmostone{i in P, j in P : i != j} :
  epsilon[i,j] + epsilon[j,i] <= 1;


# cpuscheduling.dat

param p := 7;
param c := 3;

param : b :=
1 1.1
2 2.1
3 3.0
4 1.0
5 0.7
6 5.0
```

```
7 3.0 ;

param : s :=
1 1.33
2 2.00
3 2.66 ;


# cpuscheduling.run

model cpuscheduling.mod;
data cpuscheduling.dat;
option solver cplex;
option cplex_options "mipdisplay=2";
solve;
display makespan;
for{k in C} {
  printf "CPU %d : ", k;
  for{i in P : z[i,k] = 1} {
    printf "[%d:%f] ", i, x[i];
  }
  printf "\n";
}
```

11.10.2 CPLEX solution

```
CPLEX 8.1.0: optimal integer solution; objective 2.781954887
40175 MIP simplex iterations
8463 branch-and-bound nodes
makespan = 2.78195

CPU 1 : [5:0.000000] [7:0.526316]
CPU 2 : [1:1.731955] [3:0.000000] [4:2.281955]
CPU 3 : [2:0.000000] [6:0.789474]
```

11.11 Dyeing plant: Solution

- *Indices*:
  - i, j: index on the set L of fabric batches;
  - k: index on the set $V = \{1, \ldots, v\}$ of dyeing baths;
- *Parameters*:
  - s_{ik}: time necessary to dye batch i in bath k;
  - M: upper bound to completion time of last bath.
- *Variables*:
  - $t_{ik} \geq 0$: starting time for dyeing batch i in bath k;
  - $T \geq 0$: completion time for last batch;
  - $y_{ijk} = 1$ if batch i is to be dyed before batch j in bath k, 0 otherwise.
- *Objective function*:

$$\min T$$

- *Constraints*:
  - (sequential) $\forall i \in L, k \in V \setminus \{v\} \quad (t_{ik} + s_{ik} \leq t_{i(k+1)})$;
  - (last bath) $\forall i \in L \quad (t_{iv} + s_{iv} \leq T)$;
  - (non overlapping) $\forall i, j \in L, k \in V, i \neq j \quad (t_{ik} + s_{ik} \leq t_{jk} + M(1 - y_{ijk}))$;
  - (disjunction) $\forall i, j \in L, k \in V, i \neq j \quad (y_{ijk} + y_{jik} = 1)$.

11.11.1 AMPL model, data, run

```
# dyeing.mod

param l >= 1;
param v >= 1;

set L := 1..l;
set V := 1..v;

param s{L,V} >= 0;
param M default sum{i in L, k in V} s[i,k] ;

var t{L,V} >= 0;
var T >= 0;
var y{L,L,V} binary;

minimize makespan : T;

subject to sequential{i in L, k in V diff {v}} : t[i,k] + s[i,k] <= t[i,k+1];

subject to lastbath{i in L} : t[i,v] + s[i,v] <= T;

subject to nonoverlap{i in L, j in L, k in V : i != j} :
  t[i,k] + s[i,k] <= t[j,k] + M * (1 - y[i,j,k]);

subject to disjunction{i in L, j in L, k in V : i != j} :
  y[i,j,k] + y[j,i,k] = 1;


# dyeing.dat

param l := 5;
param v := 3;

param s :  1   2   3 :=
1         3.0 1.0 1.0
2         2.0 1.5 1.0
3         3.0 1.2 1.3
4         2.0 2.0 2.0
5         2.1 2.0 3.0 ;


# dyeing.run

model dyeing.mod;
data dyeing.dat;
option solver cplex;
option cplex_options "mipdisplay=2";
solve;
```

```
display makespan;
for {i in L} {
  printf "batch %d : ", i;
  for {k in V} {
    printf "[%f] ", t[i,k];
  }
  printf "\n";
}
```

11.11.2 CPLEX solution

```
CPLEX 8.1.0: optimal integer solution; objective 14.1
1618 MIP simplex iterations
362 branch-and-bound nodes
makespan = 14.1

batch 1 : [9.100000] [12.100000] [13.100000]
batch 2 : [7.100000] [9.100000] [12.100000]
batch 3 : [4.100000] [7.100000] [8.300000]
batch 4 : [2.100000] [4.100000] [9.600000]
batch 5 : [0.000000] [2.100000] [4.100000]
```

11.12 Parking: Solution

- *Indices*:
  - i: index on the set $N = \{1, \ldots, n\}$ of cars;
  - j: index on the set $M = \{1, 2\}$ of car lines (one per street side).
- *Parameters*:
  - λ_i: length of car i;
  - L: upper bound on the car line length;
  - μ: upper bound for the sum of car lengths.
- *Variables*:
  - $x_{ij} = 1$ if i is parked on line j and 0 otherwise;
  - $t_j \geq 0$: length of car line j;
  - $y_j = 1$ if $t_j \leq L$ and 0 otherwise.
- *Objective function*:
$$\min \max_{j \in M} t_j.$$
- *Constraints*:
  - (car line length definition) $\forall j \in M \quad (t_j = \sum_{i \in N} \lambda_i x_{ij})$;
  - (assignment of cars to lines) $\forall i \in N \ (\sum_{j \in M} x_{ij} = 1)$;
  - (Q2: constraint disjunction) $\forall j \in M \quad (t_j \leq L + \mu(1 - y_j))$;
  - (Q2: constraint on one line only) $(\sum_{j \in M} y_j = 1)$.

11.12.1 AMPL model, data, run

```
# parking.mod

param n > 0;
param m > 0;

set N := 1..n;
set M := 1..m;

param lambda{N} >= 0;
param mu := sum{i in N} lambda[i];
param L >= 0;

var x{N,M} binary;
var t{N} >= 0;
var y{M} binary;
var T >= 0;

minimize minmaxobj: T;

subject to minmax {j in M} : T >= t[j];

subject to carlinedef {j in M} :
  t[j] = sum{i in N} lambda[i] * x[i,j];

subject to assignment {i in N} : sum{j in M} x[i,j] = 1;

## second question

#subject to disjunction {j in M} : t[j] <= L + mu * (1 - y[j]);

#subject to onelineonly : sum{j in M} y[j] = 1;



# parking.dat

param n := 15;
param m := 2;
param L := 15;

param : lambda :=
1  4.0
2  4.5
3  5.0
4  4.1
5  2.4
6  5.2
7  3.7
8  3.5
9  3.2
10 4.5
11 2.3
12 3.3
13 3.8
14 4.6
15 3.0 ;
```

```
# parking.run

model parking.mod;
data parking.dat;
option solver cplex;
option cplex_options "mipdisplay=2";
solve;
display minmaxobj;
for {j in M} {
  printf "line %d (length = %f) : ", j, sum{i in N : x[i,j] = 1} lambda[i];
  for {i in N : x[i,j] = 1} {
    printf "%d ", i;
  }
  printf "\n";
}
```

11.12.2 CPLEX solution

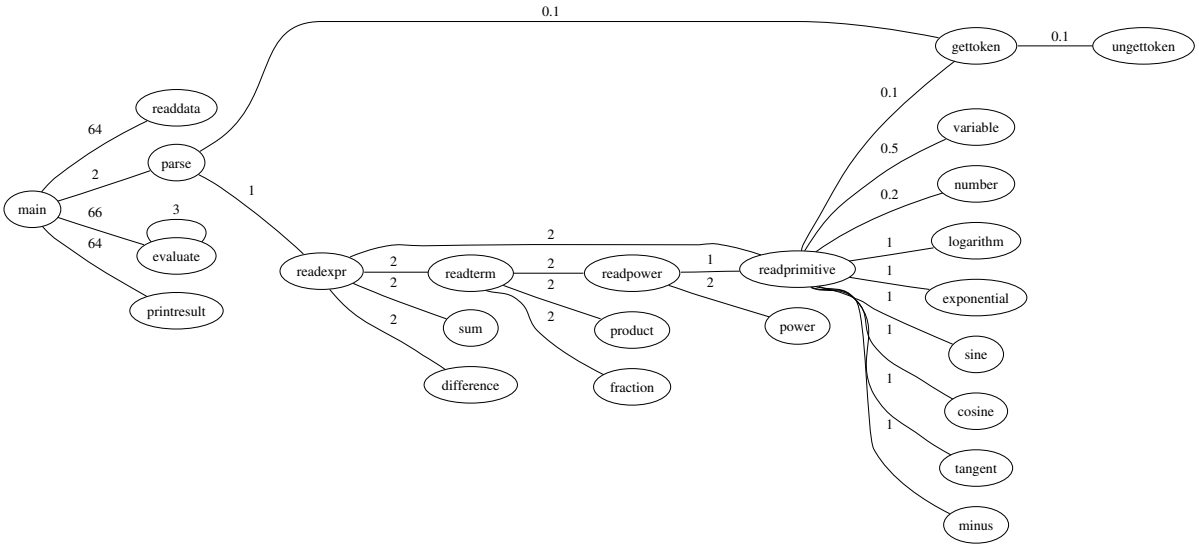
```
CPLEX 8.1.0: optimal integer solution; objective 42.1
56 MIP simplex iterations
50 branch-and-bound nodes
minmaxobj = 42.1

line 1 (length = 42.100000) : 2 3 4 5 6 7 8 11 13 14 15
line 2 (length = 15.000000) : 1 9 10 12
```

Chapter 12

Difficult modelling problems: Solutions

12.1 Checksum: Solution

We represent each subroutine with a vertex in an undirected graph $G = (V, E)$. For each $u, v \in V$, $\{u, v\} \in E$ if subroutine u calls subroutine v (or vice versa). Each edge $\{i, j\} \in E$ is weighted by the quantity p_{ij} of data exchanged between the subroutines. We want to choose a subset $L \subseteq E$ such that for each $u \in V$ there is $v \in V$ with $\{u, v\} \in L$ (i.e. L covers V), such that each vertex $v \in V$ is adjacent to exactly 1 edge in L and such that the total weight $p(L) = \sum_{\{i,j\} \in L} p_{ij}$ is maximum. G is shown below.



12.1.1 Formulation

- *Parameters*: for each $\{i, j\} \in E$, p_{ij} is the weight on the edge.
- *Variables*: for each $\{i, j\} \in E$, $x_{ij} = 1$ if $\{i, j\} \in L$ and 0 otherwise.

- *Objective function*:

$$\max \sum_{e=\{i,j\}\in E} p_{ij}x_{ij}$$

- *Constraints*:

$$\forall i \in V \qquad \sum_{j\in V|\{i,j\}\in E} x_{ij} = 1; \tag{12.1}$$

$$\forall \{i,j\} \in E \qquad x_{ij} \in \{0,1\}. \tag{12.2}$$

12.1.2 AMPL model, data, run

```
# checksum.mod

set V;
set E within {V,V};
param p{E};
var x{E} binary;
maximize data : sum{(i,j) in E} p[i,j] * x[i,j];
subject to assignment {i in V} :
  sum{j in V : i != j and (i,j) in E} x[i,j] +
  sum{j in V : i != j and (j,i) in E} x[j,i] +
    if ((i,i) in E) then x[i,i] else 0 <= 1;
```

```
# checksum.dat

set V := main readdata parse evaluate printresult gettoken readexpr
         readprimitive variable number logarithm exponential sine cosine
         tangent minus power readpower readterm product fraction sum ;
set E :=
 main   readdata
 main   parse
 main   evaluate
 main   printresult
 evaluate   evaluate
 parse   gettoken
 parse   readexpr
 readprimitive   gettoken
 readprimitive   variable
 readprimitive   number
 readprimitive   logarithm
 readprimitive   exponential
 readprimitive   sine
 readprimitive   cosine
 readprimitive   tangent
 readprimitive   minus
 readprimitive   readexpr
 readpower   power
 readpower   readprimitive
 readterm   readpower
 readterm   product
 readterm   fraction
 readexpr   readterm
 readexpr   sum   ;

param p :=
```

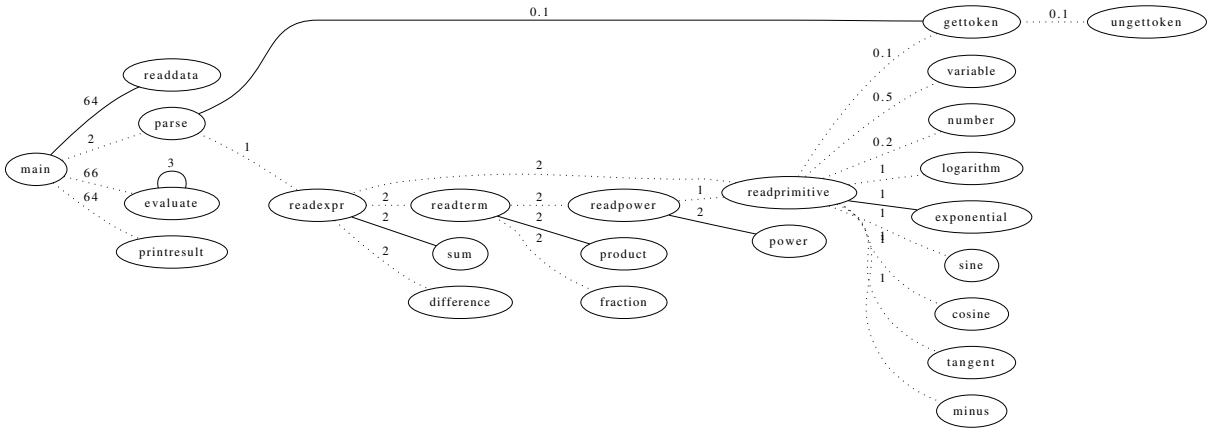
```
 main   readdata    64
 main   parse    2
 main   evaluate    66
 main   printresult    64
 evaluate   evaluate    3
 parse   gettoken    0.1
 parse   readexpr    1
 readprimitive   gettoken    0.1
 readprimitive   variable    0.5
 readprimitive   number    0.2
 readprimitive   logarithm    1
 readprimitive   exponential    1
 readprimitive   sine    1
 readprimitive   cosine    1
 readprimitive   tangent    1
 readprimitive   minus    1
 readprimitive   readexpr    2
 readpower   power    2
 readpower   readprimitive    1
 readterm   readpower    2
 readterm   product    2
 readterm   fraction    2
 readexpr   readterm    2
 readexpr   sum    2 ;
```

```
# checksum.run

model checksum.mod;
data checksum.dat;
option solver cplexstudent;
solve;
display data;
printf "L = {\n";
for {(i,j) in E : x[i,j] = 1} {
  printf "      (%s,%s)\n", i, j;
}
printf "    }\n";
```

12.1.3 CPLEX solution

```
CPLEX 11.0.1: optimal integer solution; objective 74.1
0 MIP simplex iterations
0 branch-and-bound nodes
data = 74.1

L = {
      (main,readdata)
      (evaluate,evaluate)
      (parse,gettoken)
      (readprimitive,exponential)
      (readpower,power)
      (readterm,product)
      (readexpr,sum)
    }
```

The picture below is the solution represented on the graph.



12.2 Eight queens: Solution

Each queen commands a row, a column and two diagonals on the chessboard. The problem is to position the queens so that no row, column or diagonal should be assigned to more than one queen. We associate a binary variable x_{ij} to each cell (i, j) on the chessboard, such that $x_{ij} = 1$ if there is a queen at (i, j) and 0 otherwise. The $(1, 1)$ cell is at the north-west corner. We maximize the number of queens on the chessboard subject to the assignment and cardinality constraints.

12.2.1 Formulation

- *Parameters.* n: number of cells on one side of the (square) chessboard.
- *Variables.* $x_{ij} = 1$ if there is a queen in (i, j), and 0 otherwise.
- *Objective function.*

$$\sum_{i,j \leq n} x_{ij}$$

- *Constraints.*
  1. (at most one queen per row):

$$\forall i \leq n \; (\sum_{j \leq n} x_{ij} \leq 1);$$

  2. (at most one queen per column):

$$\forall j \leq n \; (\sum_{i \leq n} x_{ij} \leq 1);$$

  3. (at most one queen per NW-SE diagonal):

$$\forall i, j \leq n \; (\sum_{h \leq n : h < i, h < j} x_{(i-h)(j-h)} + \sum_{h \leq n : h+i \leq n, h+j \leq n} x_{(i+h)(j+h)} \leq 1);$$

  4. (at most one queen per SW-NE diagonal):

$$\forall i, j \leq n \; (\sum_{h \leq n : h < i, h+j \leq n} x_{(i-h)(j+h)} + \sum_{h \leq n : h+i \leq n, h < j} x_{(i+h)(j-h)} \leq 1);$$

5. (at most 8 queens on the chessboard):

$$\sum_{i,j \leq n} x_{ij} \leq 8;$$

Notice that for $n \leq 8$ the cardinality constraint (≤ 8 queens) is redundant, as it is a consequence of the assignment constraints.

12.2.2 AMPL model, run

```
# eightqueens.mod

param n >= 0, default 8;
set N := 1..n;
var x{N,N} binary;
maximize queens : sum{i in N, j in N} x[i,j];
subject to rows {i in N} : sum{j in N} x[i,j] <= 1;
subject to cols {j in N} : sum{i in N} x[i,j] <= 1;
subject to diagNW {i in N, j in N} :
  sum{h in N : h < i and h < j} x[i-h,j-h] +
    sum{h in N : h+i<=n and h+j<=n} x[i+h,j+h] <= 1;
subject to diagSW {i in N, j in N} :
  sum{h in N : h < i and h+j<=n} x[i-h,j+h] +
    sum{h in N : h+i<=n and h < j} x[i+h,j-h] <= 1;
```

```
# eightqueens.run

model eightqueens.mod;
option solver cplexstudent;
solve;
display queens;
display x;
```

12.2.3 CPLEX solution

```
CPLEX 7.1.0: optimal integer solution; objective 8
122 MIP simplex iterations
0 branch-and-bound nodes
queens = 8

x [*,*]
:   1   2   3   4   5   6   7   8    :=
1   0   0   0   1   0   0   0   0
2   0   0   0   0   0   1   0   0
3   1   0   0   0   0   0   0   0
4   0   0   0   0   1   0   0   0
5   0   1   0   0   0   0   0   0
6   0   0   0   0   0   0   0   1
7   0   0   1   0   0   0   0   0
8   0   0   0   0   0   0   1   0
;
```

12.3 Production management

12.3.1 Formulation

- *Index*: i: index on set $\{1, \ldots, N\}$ of the months of the planning horizon.
- *Parameters*:
  - F: initial number of employees ($F = 40$);
  - P: product units produced by each employee with normal work ($P = 20$);
  - r_i: demand at i-th month;
  - U_s: maximum number of additional units that can be produced per employee ($U_s = 6$);
  - C_s: cost of each additional unit ($C_s = 5$);
  - C_m: unit stocking cost ($C_m = 10$);
  - V: maximum workforce variation because of hiring and firing ($V = 5$);
  - C_a: fixed costs when hiring an employee ($C_a = 500$);
  - C_l: fixed costs when firing an employee ($C_l = 700$);
- *Variables*:
  - x_i: number of employees at i-th month;
  - y_i: additional units produced in the i-th month;
  - z_i: stocked units at the i-th month;
  - w_i: money spent on hiring / firing people at the i-th month.
- *Objective function*:
$$\min \sum_{i=1}^{N} (C_s y_i + C_m z_i + w_i)$$
- *Constraints*:
  1. (production) for each $i \leq N$: $y_i \leq U_s x_i$;
  2. (hiring) for each $i \leq N$: $x_i - x_{i-1} \leq V$;
  3. (firing) for each $i \leq N$: $x_{i-1} - x_i \leq V$;
  4. (stock balance) for each $i \leq N$: $P x_i + z_{i-1} + y_i = r_i + z_i$;
  5. (hiring policy) for each $i \leq N$: $w_i \geq C_a(x_i - x_{i-1})$;
  6. (firing policy) for each $i \leq N$: $w_i \geq C_l(x_{i-1} - x_i)$.

12.3.2 AMPL model, data, run

```
# productionmgt.mod

set MONTHS;
param demand {MONTHS};
param employee_start;
param employee_production;
param max_extra;
param cost_extra;
param storagecost;
param max_hired;
```

```
param max_fired;
param hiringcost;
param firingcost;

var workers {{0} union MONTHS} >= 0, integer;
var additionalprod {MONTHS} >= 0 ;
var stocking {{0} union MONTHS} >= 0;
var policy {MONTHS} >= 0;

minimize cost : sum {i in MONTHS} (cost_extra * additionalprod[i] +
                                   storagecost * stocking[i] + policy[i]);

subject to production {i in MONTHS} :
         additionalprod[i] <= max_extra * workers[i];

subject to hiring {i in MONTHS} :
         workers[i] - workers[i - 1] <= max_hired;

subject to firing {i in MONTHS} :
         workers[i - 1] - workers[i] <= max_fired;

subject to storagebalance {i in MONTHS} :
         employee_production * workers[i] + stocking[i - 1] +
         additionalprod[i] = demand[i] + stocking[i];

subject to hiringpolicy {i in MONTHS} :
         policy[i] >= hiringcost * (workers[i] - workers[i - 1]);

subject to firingpolicy {i in MONTHS} :
         policy[i] >= firingcost * (workers[i - 1] - workers[i]);



# gestioneimpresa.dat - dati AMPL per il problema di gestione impresa

set MONTHS := 1 2 3 4 5 6 ;
param demand :=
        1 700
        2 600
        3 500
        4 800
        5 900
        6 800 ;
param employee_start := 40;
param employee_production := 20;
param max_extra := 6;
param cost_extra := 5;
param storagecost := 10;
param max_hired := 5;
param max_fired := 5;
param hiringcost := 500;
param firingcost := 700;

let stocking[0] := 0;
let workers[0] := employee_start;
fix stocking[0];
fix workers[0];

# productionmgt.dat
```

```
model productionmgt.mod;
data productionmgt.dat;
option solver cplex;
solve;
display cost;
display workers;
display additionalprod;
display stocking;
display policy;
quit;
```

CPLEX Solution

```
CPLEX 8.1.0: optimal integer solution; objective 10000
9 MIP simplex iterations
0 branch-and-bound nodes
costo = 10000

operai [*] :=
0  40
1  35
2  33
3  33
4  33
5  33
6  33
;

prod_straord [*] :=
1    0
2    0
3    0
4    0
5  160
6  140
;

magazzino [*] :=
0    0
1    0
2   60
3  220
4   80
5    0
6    0
;

politica [*] :=
1  3500
2  1400
3     0
4     0
5     0
6     0
;
```

12.4 The travelling salesman problem: Solution

The Travelling Salesman Problem (TSP) can be formalized as the search for a Hamiltonian tour (i.e. a tour that passes through every vertex at most once) of minimum cost in a complete directed graph $G = (V, A)$ where V is the set of customers and A the set of distances between customer sites.

12.4.1 Formulation

- *Parameters:*
  1. i, j: indices on set V of customers;
  2. d_{ij}: distance between customers i and j.
- *Variables:* for each $i \neq j \in V$, $x_{ij} = 1$ if the salesman travels directly from i to j and 0 otherwise.
- *Objective function:*
$$\min \sum_{i \neq j \in V} d_{ij} x_{ij}.$$
- *Constraints:*
  1. (one successor) $\forall i \in V \quad \sum_{j \in V, j \neq i} x_{ij} = 1$;
  2. (one predecessor) $\forall j \in V \quad \sum_{i \in V, i \neq j} x_{ij} = 1$;
  3. (subtour elimination) $\forall S \subsetneq V \quad \sum_{i \in S, j \in V \setminus S} x_{ij} \geq 1$.

The last constraints ensures that for each (nontrivial) partition of $V = S \dot{\cup} (V \setminus S)$ with $1 \leq |S| < |V|$, the salesman must travel out of S. This prevents the solution from consisting of a set of disconnected tours (which would be feasible with the first two constraints). Since the number of proper subsets S of V is exponential in $|V|$, the instance size is also exponential in $|V|$, which is not an acceptable feature. We must therefore devise a solution method that circumvents the problem.

Initially, we solve the problem with the subtour elimination constraints relaxed (thus, the instance size becomes polynomial in $|V|$), which is called the *master problem*. We then test the obtained solution to see if the tour γ passing through 1 passes through every vertex in V. If so, the solution satisfies the subtour elimination constraints, is optimal and the algorithm terminates. Otherwise we set S to be the set of vertices incident in γ, generate one subtour elimination constraint for S, add it to the master problem, and solve it again iteratively until no more subtours are present in the solution.

12.4.2 AMPL model, data

```
# tsp.mod

param n > 0, integer;
set V := 1..n;
param d{V,V} >= 0;
param subtours >= 0, integer, default 0;
set S{1..subtours};

var x{V,V} binary;
```

```
minimize distance : sum{i in V, j in V : i != j} d[i,j] * x[i,j];
subject to successor {i in V} : sum{j in V : i != j} x[i,j] = 1;
subject to predecessor {j in V} : sum{i in V : i != j} x[i,j] = 1;
subject to subtour_elim {k in 1..subtours} :
  sum{i in S[k], j in V diff S[k]} x[i,j] >= 1;
```

```
# tsp.dat

param n := 7;
param d : 1  2  3  4  5  6  7 :=
1         0 86 49 57 31 69 50
2         0  0 68 79 93 24  5
3         0  0  0 16  7 72 67
4         0  0  0  0 90 69  1
5         0  0  0  0  0 86 59
6         0  0  0  0  0  0 81
7         0  0  0  0  0  0  0 ;
```

12.4.3 Algorithm

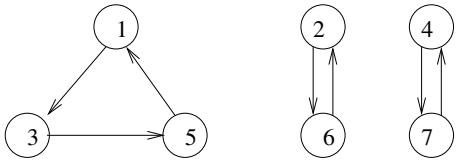
We first solve the model with `subtours` (the number of subtour elimination inequalities) at its default value 0 — this is the initial master problem.

```
model tsp.mod;
data tsp.dat;
for {i in V, j in V : i > j} {
  let d[i,j] := d[j,i];
}
option solver cplexstudent;
solve;
display distance;
display x;
```

We obtain the following CPLEX solution, corresponding to three distinct subtours : $(1, 3, 5, 1)$, $(2, 6, 2)$ e $(4, 7, 4)$.

```
distance = 137

x [*,*]
:   1   2   3   4   5   6   7    :=
1   0   0   1   0   0   0   0
2   0   0   0   0   0   1   0
3   0   0   0   0   1   0   0
4   0   0   0   0   0   0   1
5   1   0   0   0   0   0   0
6   0   1   0   0   0   0   0
7   0   0   0   1   0   0   0 ;
```

This solution is as in the picture below.



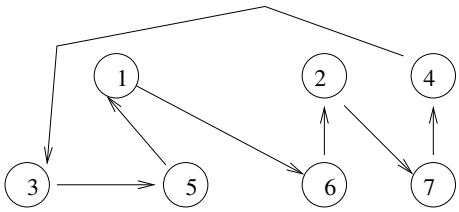
We add a subtour elimination constraint with $S = \{1, 3, 5\}$. In practice, it suffices to add the following instructions before the `solve;` command:

```
let subtours := 1;
let S[1] := { 1, 3, 5 };
```

We get the following solution:

```
distance = 153

x [*,*]
:   1   2   3   4   5   6   7    :=
1   0   0   0   0   0   1   0
2   0   0   0   0   0   0   1
3   0   0   0   0   1   0   0
4   0   0   1   0   0   0   0
5   1   0   0   0   0   0   0
6   0   1   0   0   0   0   0
7   0   0   0   1   0   0   0 ;
```

Since this corresponds to a Hamiltonian tour, it is an optimal solution (pictured below).



We now implement this in an AMPL run file.

```
# tsp.run
option solver cplex;
# don't print CPLEX messages
option solver_msg 0;

model tsp.mod;
data tsp.dat;

# distance matrix must be symmetric
for {i in V, j in V : i > j} {
  let d[i,j] := d[j,i];
}

# data structures required for the algorithm
let subtours := 0;
param successorvertex{V} >= 0, integer;
param currentvertex >= 0, integer;
param termination binary;
let termination := 0;

# iterative part
repeat while (termination = 0) {

  # solve the problem
```

```
  solve > /dev/null;

  let subtours := subtours + 1;
  if (subtours > 1) then {
    printf "  breaking the subtour...\n";
  }

  # find the successors of each vertex
  for {i in V} {
    let successorvertex[i] := sum{j in V : j != i} j * x[i,j];
  }

  # find a subtour
  let currentvertex := 1;
  let S[subtours] := {};
  repeat {
    let S[subtours] := S[subtours] union {currentvertex};
    let currentvertex := successorvertex[currentvertex];
  } until (currentvertex = 1);

  # print the subtour we wish to eliminate
  printf "(sub)tour: (";
  for {i in S[subtours]} {
    printf "%d -> ", i ;
  }
  printf "1)\n";

  # termination condition
  if (card(S[subtours]) >= n) then {
    # Hamiltonian tour, terminate
    let termination := 1;
  }
}

printf "tour cost: %g\n", distance;
```

12.4.4 CPLEX solution

```
(sub)tour: (1 -> 3 -> 5 -> 1)
  breaking the subtour...
(sub)tour: (1 -> 6 -> 2 -> 7 -> 4 -> 3 -> 5 -> 1)
tour cost: 153
```

12.4.5 Heuristic solution

A solution algorithm for a minimization problem is a *k-approximation algorithm* if the yielded solution has objective function value $\bar{f}$ such that $\bar{f} \leq kf^*$, where f^* is the value of an optimal solution. There exists a well-known $\frac{3}{2}$-approximation heuristic for the metric symmetric TSP (i.e. where distances are symmetric and obey a triangular inequality) described by Christofides. It works as follows:

1. Find a minimum cost spanning tree (MST) $T = (V, E(T))$ in the (undirected, as distances are

symmetric) graph $G = (V, E)$. The z-solution of the following formulation:

$$\left.\begin{array}{rll} \min\limits_{y\geq 0, z\in\{0,1\}} & \sum\limits_{\{i,j\}\in E} d_{ij} z_{ij} & \\ \forall u \neq v \in V, i \in V \smallsetminus \{u,v\} & \sum\limits_{\substack{j\in V \\ \{i,j\}\in E}} (y^{uv}_{ij} - y^{uv}_{ji}) & = \quad 0 \\ \forall u \neq v \in V & \sum\limits_{\substack{i\in V \\ i\neq u}} (y^{uv}_{iu} - y^{uv}_{ui}) & = \quad 1 \\ \forall u \neq v \in V, \{i,j\} \in E & y^{uv}_{ij} & \leq \quad z_{ij} \\ \forall u \neq v \in V, \{i,j\} \in E & y^{uv}_{ji} & \leq \quad z_{ji} \\ \forall \{i,j\} \in V & z_{ij} & = \quad z_{ji} \\ & \sum\limits_{\{i,j\}\in E} z_{ij} & = \quad n-1 \end{array}\right\} \tag{12.3}$$

defines the set of edges in an MST.

2. Let $M = (V(M), E(M))$ be a minimum cost complete matching between the vertices $V(M) \subseteq V$ such that their star degree in T is odd (i.e. such that $|\bar{\delta}(v) \cap E(T)| \mod 2 = 1$ for $v \in V(M)$, where $\bar{\delta}(v)$ is the set of edges adjacent to v). The solution of the following formulation:

$$\left.\begin{array}{rll} \min\limits_{w\in\{0,1\}} & \sum\limits_{\{i,j\}\in E} d_{ij} w_{ij} & \\ \forall u \in V : |\delta_T(u)| \bmod 2 = 1 & \sum\limits_{\substack{v\in V: v\neq u \\ |\delta_T(v)| \bmod 2 = 1}} w_{uv} & = \quad 1 \end{array}\right\} \tag{12.4}$$

defines the set of edges of a complete matching in T.

3. Consider the subgraph $L = T \cup M = (V, E(T) \cup E(M))$: it is a Eulerian cycle because by construction every vertex has even star degree (we added a matchine edge incident to all those vertices that had an odd star in T).

4. For each $v \in V$ with more than two incident edges (i.e. such that $|\delta(v) \cap (T \cup M)| > 2$) we contract all pairs of edges adjacent to v but one. The contraction operation on v consists in replacing a pair of adjacent edges $\{u, v\}, \{v, w\} \in T \cup M$ by the edge $\{u, w\}$. This operation is always possible because the graph is complete. At the end of this operation each node in V has one predecessor and one successor only, and L is a Hamiltonian cycle.

Thm. *Let $\bar{f}$ be the cost of L and f^* be the cost of an optimal Hamiltonian cycle L^*. Then $\bar{f} \leq \frac{3}{2} f^*$.*

Proof. For a set of edges $S \subseteq E$, let $f(S) = \sum_{\{i,j\}\in S} d_{ij}$. Every Hamiltonian cycle (including L^*) can be seen as a spanning tree union an edge. Since T is of minimum cost, $f(E(T)) \leq f^*$. Let $(v_1, \ldots, v_{2m})$ be the vertices of $V(M)$ ordered as in L^*. Since L^* is a Hamiltonian cycle, we can find two "alternating" matchings of $V(M)$ given by $M_1 = \{\{v_1, v_2\}, \{v_3, v_4\}, \ldots\}$ and $M_2 = \{\{v_2, v_3\}, \{v_4, v_5\}, \ldots\}$. Since $M_1 \cup M_2 \subseteq L^*$, we have $f(M_1 \cup M_2) \leq f^*$. Furthermore, since M is an optimal matching in $V(M)$, both M_1 and M_2 have greater than or equal cost to M, so $f(M_1 \cup M_2) \geq 2f(E(M))$. Putting the last two statements together, we have $f(M) \leq \frac{1}{2} f^*$. Furthermore, because distances are Euclidean and they obey the strict triangular inequality, $\bar{f} \leq f(T \cup M)$ (in other words, visiting more nodes cannot decrease the total cost). Therefore $\bar{f} \leq f(T) + f(M) \leq f^* + \frac{1}{2} f^* = \frac{3}{2} f^*$. □

Applying Christofides' algorithm to the instance of the exercise, we find a Hamiltonian cycle with total travelled distance 153 (i.e. we find an optimal solution), as shown in Fig. 12.1, 12.2.

The heuristic is implemented as an AMPL run file as follows.

```
# Christofides' heuristic

# don't print solver messages
option solver_msg 0;
```

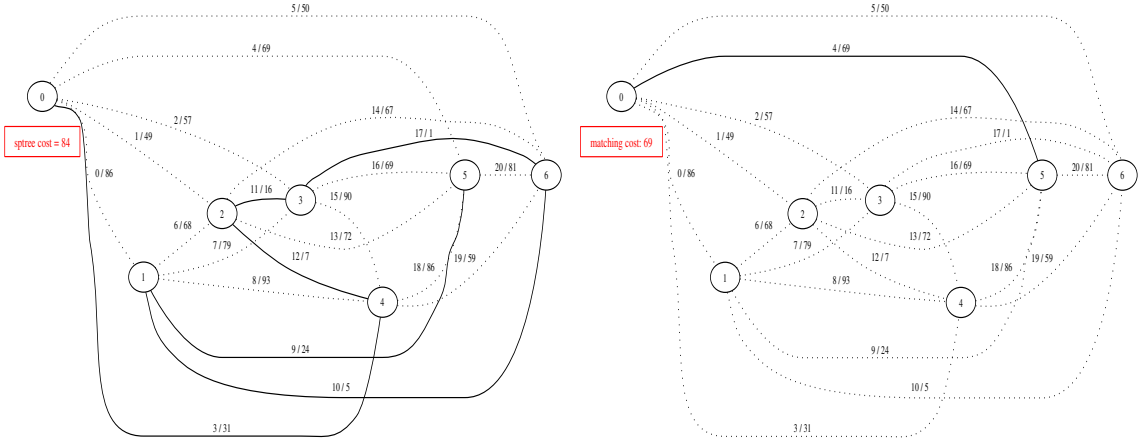


Figure 12.1: The minimum cost spanning tree T and the minimum matching M.

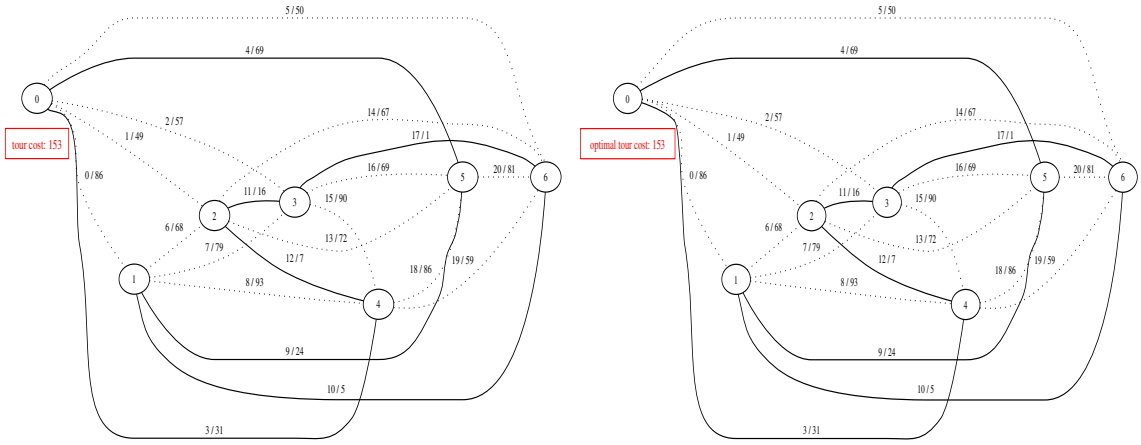


Figure 12.2: The heuristic and optimal solutions.

```
# vertices
param n integer, > 0;
set V := 1..n;

# edge weights
param d{V,V} >= 0;

# choose solver
option solver cplex;

# spanning tree subproblem
model christofides-sptree.mod;
problem T : x, z, mst,
            flow0, flow1, flow_on_tree1, flow_on_tree2,
            spanning_tree, symm_sptree;
data tsp.dat;
# symmetric weights
for {i in V, j in V : i > j} {
  let d[i,j] := d[j,i];
```

```
}
solve > /dev/null;

# matching subproblem
param odd{v in V};
# compute vertices with odd degree in the spanning tree
let {v in V} odd[v] := (sum{u in V : u != v and z[v,u] = 1} 1) mod 2;
model christofides-matching.mod;
problem M : w, odd_matching, matching;
solve > /dev/null;

# contract edges around vertices with degree > 2 in sptree union matching
param tour{u in V, v in V : u != v} binary;
let {u in V, v in V : u != v} tour[u,v] := max(0, z[u,v] + w[u,v]);
param contract{v in V};
let {v in V} contract[v] := card({u in V : u != v and tour[u,v] = 1});
param contract_pairs integer, default 0;
for {u in V, v in V, i in V : u != v and v != i and u != i and
     tour[u,v] = 1 and tour[v,i] = 1 and contract[v] > 2} {
  if (contract_pairs >= 2) then {
    let tour[u,v] := 0;
    let tour[v,i] := 0;
    let tour[u,i] := 1;
  }
  let contract_pairs := contract_pairs + 1;
}

# print tour
param v1 integer, default 1;
param v2 integer, default 1;
printf "tour: %d", v1;
set T_set default {v1};
param h_cost default 0;
repeat while(card(T_set) < n) {
  let v2 := min {v in V : v1 != v and (v not in T_set) and tour[v1,v] = 1} v;
  let h_cost := h_cost + d[v1,v2];
  printf ",%d", v2;
  let v1 := v2;
  let T_set := T_set union {v1};
}
let h_cost := h_cost + d[1,v2];
printf "; cost = %g\n", h_cost;
```

It rests on two model files for finding a spanning tree and a matching. The AMPL mod file for spanning tree is the following.

```
# Christofides' heuristic: spanning tree

# flow variables
var x{V,V,V,V} >= 0;
# tree variables
var z{V,V} binary;

# MST objective
minimize mst: sum{u in V, v in V : u != v} d[u,v]*z[u,v];

subject to flow0 {u in V, v in V, i in V : u != v and i != u and i != v} :
  sum {j in V : i != j} (x[i,j,u,v] - x[j,i,u,v]) = 0;
```

```
subject to flow1 {u in V, v in V : u != v} :
  sum {i in V : i != u} (x[u,i,u,v] - x[i,u,u,v]) = 1;

subject to flow_on_tree1 {u in V, v in V, i in V, j in V : u != v and i != j} :
  x[i,j,u,v] <= z[i,j];

subject to flow_on_tree2 {u in V, v in V, i in V, j in V : u != v and i != j} :
  x[j,i,u,v] <= z[j,i];

subject to spanning_tree : sum {u in V, v in V : u < v} z[u,v] = n - 1;

subject to symm_sptree {u in V, v in V : u < v} : z[v,u] = z[u,v];
```

The AMPL mod file for matchings is the following.

```
# Christofides' heuristic: matching

# matching variables
var w{V,V} binary;

# matching objective
minimize odd_matching : sum{u in V, v in V : u != v} d[u,v]*w[u,v];

subject to matching {u in V : odd[u]} :
  sum{v in V : u != v and odd[v]} w[u,v] = 1;
```

The output on `tsp.dat` is:

```
tour: 1,5,3,4,7,2,6; cost = 153
```

i.e. the heuristic solution is optimal in this case.

12.5 Optimal rocket control 1: Solution

The equation of motion of the rocket is:

$$\forall t \in [0,T] \quad m\frac{\partial^2 y(t)}{\partial t^2} + mg = u(t).$$

At time 0 (resp. T), the rocket must be at height 0 (resp. H); velocity at time 0 is 0, so $y(0) = v(0) = 0$,$y(T) = H$. The force acting on the rocket must not exceed b, so $|u(t)| \leq b$ for each $t \in [0,T]$. We must determine $u(t)$ so that the energy is minimum. Our objective function is thus $E = \int_0^T |u(t)|dt$. We obtain a nonlinear problem with time dependency:

$$\begin{aligned}
\min \quad & \int_0^T |u(t)|dt \\
\forall t \in [0,T] \quad & |u(t)| \leq b \\
\forall t \in [0,T] \quad & m\frac{\partial^2 y(t)}{\partial t^2} + mg = u(t) \\
& y(0) = 0 \\
& y(T) = H \\
& v(0) = 0.
\end{aligned}$$

First we remove the time dependency. We consider a discretization of the interval $[0, T]$ in n subintervals, with $t_1 = 0$, $\Delta t = \frac{T}{n}$, $t_{n+1} = t_n + \Delta t = T$ and $t_k = t_1 + k\Delta t$ for each $k \leq n$. Let $y_k = y(t_k)$ and $v_k = \frac{\partial y(t)}{\partial y}\big|_{t_k}$ for each $k \leq n+1$. For $k \leq n$, the time derivative of y at t_k is approximated by $\frac{y_{k+1}-y_k}{\Delta t}$, so we set $v_k = \frac{y_{k+1}-y_k}{\Delta t}$ for each $k \leq n$. The second time derivative of y at t_k is similarly approximated by $\frac{v_{k+1}-v_k}{\Delta t}$, hence $\frac{v_{k+1}-v_k}{\Delta t} = \frac{u_k}{m} - g$, where $u_k = u(t_k)$ for $k \leq n$. We obtain the following nonlinear program:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{n} |u_k| \\
\forall k \leq n \quad & y_{k+1} - y_k = v_k \Delta t \\
\forall k \leq n \quad & v_{k+1} - v_k = (\frac{u_k}{m} - g)\Delta t \\
\forall k \leq n+1 \quad & |u_k| \leq b \\
& y_1 = 0 \\
& y_{n+1} = H \\
& v_1 = 0 \\
\forall k \leq n+1 \quad & 0 \leq y_k \leq H \\
\forall k \leq n+1 \quad & v_k \geq 0 \\
\forall k \leq n+1 \quad & u_k \in \mathbb{R}.
\end{aligned}$$

We reformulate it to a linear program by introducing variables w_k for $k \leq n+1$, which replace each nonlinear term $|u_k|$. We introduce the constraints $u_k \leq w_k$, $u_k \geq -w_k$, $w_k \geq 0$ and obtain the following LP:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{n} w_k \\
\forall k \leq n \quad & y_{k+1} - y_k = v_k \Delta t \\
\forall k \leq n \quad & v_{k+1} - v_k = (\frac{u_k}{m} - g)\Delta t \\
\forall k \leq n \quad & w_k \leq b \\
& y_1 = 0 \\
& y_{n+1} = H \\
& v_1 = 0 \\
\forall k \leq n+1 \quad & -w_k \leq u_k \leq w_k \\
\forall k \leq n+1 \quad & 0 \leq y_k \leq H \\
\forall k \leq n+1 \quad & v_k, w_k \geq 0 \\
\forall k \leq n+1 \quad & u_k \in \mathbb{R}.
\end{aligned}$$

12.5.1 AMPL: model, run

```
## rocket.mod

# time horizon
param T >= 0, default 60;
# height to reach
param H >= 0, default 23000;
# mass of rocket
param m >= 0, default 2140;
# limit on force
param b >= 0, default 60000;
```

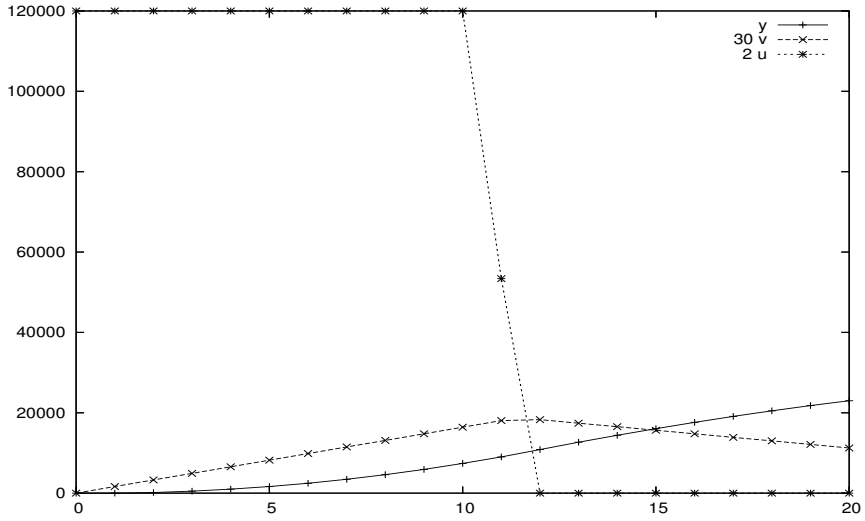
```
# number of time intervals
param n >= 0, default 20;
# gravity acceleration
param g default 9.8;
# Delta t
param Dt := T / n;

set N := 1..n+1;
set N1 := 1..n;

# height
var y{N} >= 0, <= H;
# velocity
var v{N} >= 0;
# force
var u{N};
# linearization
var w{N} >= 0;

minimize energy : sum{k in N1} w[k];

subject to velocity {k in N1} : y[k+1] - y[k] = Dt*v[k];
subject to force {k in N1} : v[k+1] - v[k] = Dt*(u[k]/m - g);
subject to forcelimit {k in N1} : w[k] <= b;
subject to sealevel: y[1] = 0;
subject to height : y[n+1] = H;
subject to still: v[1] = 0;
subject to linearization1 {k in N}: u[k] + w[k] >= 0;
subject to linearization2 {k in N}: u[k] - w[k] <= 0;


# rocket.run

model rocket.mod;
option solver cplex;
solve;
display energy;

for {i in N} {
  printf "%d %f %f %f\n", i, y[i], v[i], u[i];
}
```

12.5.2 CPLEX solution

```
CPLEX 11.0.1: optimal solution; objective 686696.1111
39 dual simplex iterations (0 in phase I)
energy = 686696

1 0.000000 0.000000 60000.000000
2 0.000000 54.712150 60000.000000
3 164.136449 109.424299 60000.000000
4 492.409346 164.136449 60000.000000
5 984.818692 218.848598 60000.000000
6 1641.364486 273.560748 60000.000000
7 2462.046729 328.272897 60000.000000
8 3446.865421 382.985047 60000.000000
9 4595.820561 437.697196 60000.000000
10 5908.912150 492.409346 60000.000000
```

```
11 7386.140187 547.121495 60000.000000
12 9027.504673 601.833645 26696.111111
13 10833.005607 609.858100 0.000000
14 12662.579907 580.458100 0.000000
15 14403.954206 551.058100 0.000000
16 16057.128505 521.658100 0.000000
17 17622.102804 492.258100 0.000000
18 19098.877103 462.858100 0.000000
19 20487.451402 433.458100 0.000000
20 21787.825701 404.058100 0.000000
21 23000.000000 374.658100 0.000000
```

The picture below shows height, velocity and force in function of the time interval index. Velocity and force were scaled to be visible on the same graph as height.



12.6 Double monopoly: Solution

The model to maximize the minimum expected return for AA is as follows: for $i \leq n$, let x_i be the fraction of the budget invested in customer i. Suppose we know that BB will choose investment $j \leq n$: then the expected return is $g_j(x) = \sum_{i=1}^{n} a_{ij} x_i$. Minimizing this value on j we get $f_A(x) = \min\{g_j(x) \mid j \leq n\}$. Since x_i are non-negative budget fractions, we have $\sum_{i=1}^{n} x_i = 1$ e $x_i \geq 0$ for each $i \leq n$.

$$\left.\begin{array}{rrcl} \max_x & f_A(x) & & \\ & \sum_{i=1}^{n} x_i & = & 1 \\ & x & \geq & 0. \end{array}\right\}$$

This is equivalent to:

$$\left.\begin{array}{rrcll} \max_x & v & & & \\ & \sum_{i=1}^{n} a_{ij} x_i & \geq & v & \forall\, j \leq n \\ & \sum_{i=1}^{n} x_i & = & 1 & \\ & x & \geq & 0. & \end{array}\right\}$$

We now apply the transformation $X_i = x_i/v$ for each $i \leq n$, obtaining $\sum_{i=1}^{n} x_i/v \sum_{i=1}^{n} X_i = 1/v$. Since $\max v = \min 1/v$ for each $v > 0$, we can re-write the model as:

$$\left.\begin{array}{rrcll} \min_X & \sum_{i=1}^{n} X_i & & & \\ & \sum_{i=1}^{n} a_{ij} X_i & \geq & 1 & \forall\, j \leq n \\ & X & \geq & 0. & \end{array}\right\} \tag{12.5}$$

The model to minimize the maximum expected loss for BB is as follows. For $j \leq n$, let y_j be the non-negative budget fraction of BB invested in customer j. As above, $f_B(x) = \max\{\sum_{j=1}^n y_j a_{ij} \mid i \leq n\}$, $\sum_{j=1}^n y_j = 1$ and $y_j \geq 0$ for each $j \leq n$.

$$\left.\begin{array}{rrcl} \min_y & f_B(x) & & \\ & \sum_{j=1}^n y_j & = & 1 \\ & y & \geq & 0, \end{array}\right\}$$

which is equivalent to:

$$\left.\begin{array}{rrcll} \min_y & z & & & \\ & \sum_{j=1}^n y_j a_{ij} & \leq & z & \forall\, j \leq n \\ & \sum_{j=1}^n y_j & = & 1 & \\ & y & \geq & 0. & \end{array}\right\}$$

We transform $Y_j = y_j/z$ for each $j \leq n$:

$$\left.\begin{array}{rrcll} \max_Y & \sum_{j=1}^n Y_j & & & \\ & \sum_{j=1}^n Y_j a_{ij} & \leq & 1 & \forall\, j \leq n \\ & Y & \geq & 0. & \end{array}\right\} \tag{12.6}$$

It is straightforward to verify that (12.6) is the dual of (12.5).

Chapter 13

Telecommunication networks: Solutions

13.1 Packet routing: Solution

13.1.1 Formulation for 2 links

- *Indices*:
  i: index on the set of demands $F = \{1, \ldots, n\}$.

- *Parameters*:

  - a_i: capacity used by demand i ($\forall i \in F$);
  - c_i: routing cost for demand i on link 1 ($\forall i \in F$);
  - p: cost percentage difference between routing on link 2 and link 1;
  - u_1: capacity installed on link 1;
  - u_2: capacity installed on link 2.

- *Variables*:

  - $x_{i1} = 1$ if packet i is routed on link 1, 0 otherwise ($\forall i \in F$)
  - $x_{i2} = 1$ if packet i is routed on link 2, 0 otherwise ($\forall i \in F$)

- *Objective function*:
  $\min \sum_{i=1}^{n} (c_i x_{i1} + (p+1) c_i x_{i2})$

- *Constraints*:

  - $\forall i \in F \ (x_{i1} + x_{i2} = 1)$;
  - $\sum_{i=1}^{n} a_i x_{i1} \leq u_1$;
  - $\sum_{i=1}^{n} a_i x_{i2} \leq u_2$.

13.1.2 Formulation for m links

- *Indices*:
  1. i: index on the set of demands $F = \{1, \ldots, n\}$;
  2. j: index on the set of links $L = \{1, \ldots, m\}$.
- *Parameters*:
  - a_i: capacity used by demand i ($\forall i \in F$);
  - c_i: routing cost for demand i on link 1 ($\forall i \in F$);
  - p_j: cost percentage difference between routing on link j and link 1 ($\forall j \in L$);
  - u_j: capacity installed on link j ($\forall j \in L$).
- *Variables*:
  $x_{ij} = 1$ if packet i is routed on link j, 0 otherwise ($\forall i \in F, j \in L$).
- *Objective function*:
  $\min \sum_{j=1}^{m} \sum_{i=1}^{n} (p_j + 1) c_i x_{ij}$.
- *Constraints*:
  - $\forall i \in F \ (\sum_{j=1}^{m} x_{ij} = 1)$;
  - $\forall j \in L \ (\sum_{i=1}^{n} a_i x_{ij} \leq u_j)$.

13.1.3 AMPL model, data, run

```
# packetrouting.mod

param n > 0;
set F := 1..n;
param a{F} >= 0;
param c{F} >= 0;
param p default 0.3;
param u1 >= 0;
param u2 >= 0;

var x1{F} binary;
var x2{F} binary;

minimize objfun :
  sum{i in F} (c[i]*x1[i] + (p+1)*c[i]*x2[i]);

subject to knapsack1 : sum{i in F} a[i]*x1[i] <= u1;
subject to knapsack2 : sum{i in F} a[i]*x2[i] <= u2;
subject to assignment {i in F} : x1[i] + x2[i] = 1;



# packetrouting.dat

param n := 10;
param : a c :=
1  0.3 200
2  0.2 200
```

```
3  0.4 250
4  0.1 150
5  0.2 200
6  0.2 200
7  0.5 700
8  0.1 150
9  0.1 150
10 0.6 900 ;
param u1 := 1;
param u2 := 2;


# packetrouting.run

model packetrouting.mod;
data packetrouting.dat;
option solver cplex;
solve;
display objfun;
for {i in F} {
  printf "packet %d on link %d\n", i, if (x1[i] = 1) then 1 else 2;
}
```

13.1.4 CPLEX solution

.

```
CPLEX 8.1.0: optimal integer solution; objective 3610
2 MIP simplex iterations
0 branch-and-bound nodes
objfun = 3610

packet 1 on link 2
packet 2 on link 1
packet 3 on link 2
packet 4 on link 1
packet 5 on link 2
packet 6 on link 2
packet 7 on link 2
packet 8 on link 2
packet 9 on link 1
packet 10 on link 1
```

13.2 Network Design: Solution

Let $G = (V, E)$ be the graph of the network. The problem can be formalized as looking for the partition of V in three disjoint subsets V_1, V_2, V_3 such that the sum of the backbone update cost are minimum on the edges having one adjacent vertex in a set of the partition, and the other adjacent vertex in another set of the partition. This problem is often called GRAPH PARTITIONING or MIN-k-CUT problem.

13.2.1 Formulation and linearization

- *Indices*: $i, j \in V$ and $h, k \in K = \{1, 2, 3\}$.

- *Parameters*:
  - for each $\{i,j\} \in E$, d_{ij} is the edge weight (distance between i and j);
  - c: backbone updating cost;
  - m: minimum cardinality of the subnetworks.
- *Variables*: for each $i \in V$, $h \in K$, let $x_{ih} = 1$ if vertex i is in V_h, and 0 otherwise.
- *Objective function*:

$$\min \frac{1}{2} \sum_{h \neq k \in K} \sum_{\{i,j\} \in E} c d_{ij} x_{ih} x_{jk}$$

- *Constraints*:

$$\forall i \in V \quad \sum_{k \in K} x_{ik} = 1; \text{ (assignment)} \tag{13.1}$$

$$\forall h \in K \quad \sum_{i \in V} x_{ik} \geq m; \text{ (subnetwork cardinality).} \tag{13.2}$$

This formulation involves products between binary variables, and can therefore be classified as a Binary Quadratic Program (BQP). Its feasible region is nonconvex (due to the integrality constraints and the quadratic terms), and the continuous relaxation of its feasible region is also nonconvex (due to the quadratic terms). This poses additional problems to the calculation of the lower bound within Branch-and-Bound (BB) type solution algorithms. However, the formulation can be linearized exactly, which means that there exists a Mixed-Integer Linear Programming (MILP) formulation of the problem whose projection in the x-space of the BQP yields exactly the same feasible region. The above program can be reformulated as follows:

1. replace each quadratic product $x_{ih}x_{jk}$ by a continuous *linearization variable* w_{ij}^{hk} constrained by $0 \leq w_{ij}^{hk} \leq 1$;
2. add the following constraints to the formulation:

$$\forall \{i,j\} \in E, h \neq k \in K \quad w_{ij}^{hk} \geq x_{ih} + x_{jk} - 1 \quad (\text{if } x_{ih} = x_{jk} = 1,\ w_{ij}^{hk} = 1) \tag{13.3}$$

$$\forall \{i,j\} \in E, h \neq k \in K \quad w_{ij}^{hk} \leq x_{ih} \quad (\text{if } x_{ih} = 0,\ w_{ij}^{hk} = 0) \tag{13.4}$$

$$\forall \{i,j\} \in E, h \neq k \in K \quad w_{ij}^{hk} \leq x_{jk} \quad (\text{if } x_{jk} = 0,\ w_{ij}^{hk} = 0). \tag{13.5}$$

Constraints (13.3)-(13.5) are a way to express the equation $w_{ij}^{hk} = x_{ih}x_{jk}$ (i.e. the condition vertex i assigned to subnetwork h and vertex j assigned to subnetwork k) without introducing quadratic products in the formulation. The resulting formulation is a MILP whose continuous relaxation is a Linear Programming problem (hence it is convex, which implies that each local optimum is also global — so it can be safely used to compute lower bounds in BB algorithms such as that implemented in CPLEX).

13.2.2 AMPL model, data, run

```
# network design
param n >= 0, integer;
param k >= 1, integer;
set V := 1..n;
set K := 1..k;
param c >= 0;
param m >= 0, integer;
param d{V,V} >= 0 default 0;
```

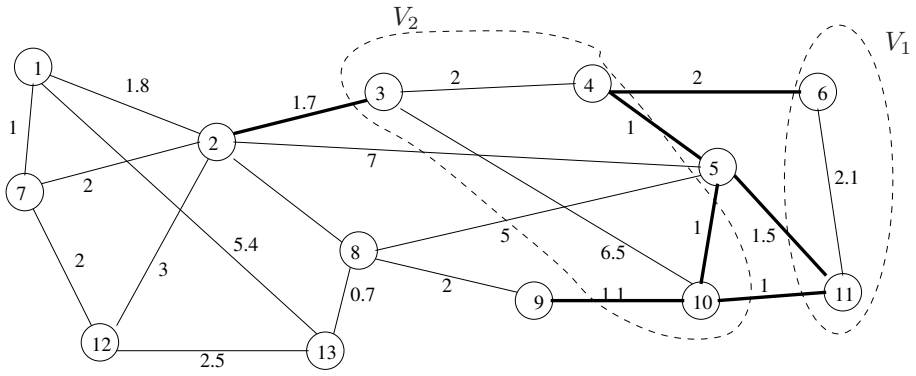
```
var x{V,K} binary;
var w{V,V,K,K} >= 0, <= 1;

minimize cost :  sum{h in K, l in K, i in V, j in V :
                      h != l and i < j and d[i,j] > 0} c*d[i,j]*w[i,j,h,l];
subject to assignment {i in V} : sum{h in K} x[i,h] = 1;
subject to cardinality {h in K} : sum{i in V} x[i,h] >= m;
subject to linearization {h in K, l in K, i in V, j in V :
                            h != l and i < j and d[i,j] > 0} :
                            w[i,j,h,l] >= x[i,h] + x[j,l] - 1;

# netdes.dat
param n := 13;
param k := 3;
param c := 25;
param m := 2;
param d :=
  1  2  1.8
  1  7  1
  2  3  1.7
  2  5  7
  2  7  2
  2  12 3
  3  4  2
  3  10 6.5
  4  5  1
  4  6  2
  5  8  5
  5  10 1
  5  11 1.5
  6  11 2.1
  7  12 2
  8  9  2
  8  13 0.7
  9  10 1.1
  10 11 1
  12 13 2.5 ;

# netdes.run

model netdes.mod;
data netdes.dat;
for {i in V, j in V : i < j} {
  let d[j,i] := d[i,j];
}
option solver cplexstudent;
solve;
display cost;
for {h in K} {
  printf "subnetwork %d:", h;
  for {i in V} {
    if (x[i,h] == 1) then {
      printf " %d", i;
    }
  }
  printf "\n";
}
```

13.2.3 CPLEX solution

For $k = 3$:

```
CPLEX 8.1.0: optimal integer solution; objective 232.5
1779 MIP simplex iterations
267 branch-and-bound nodes
cost = 232.5

subnetwork 1: 6 11
subnetwork 2: 3 4 10
subnetwork 3: 1 2 5 7 8 9 12 13
```

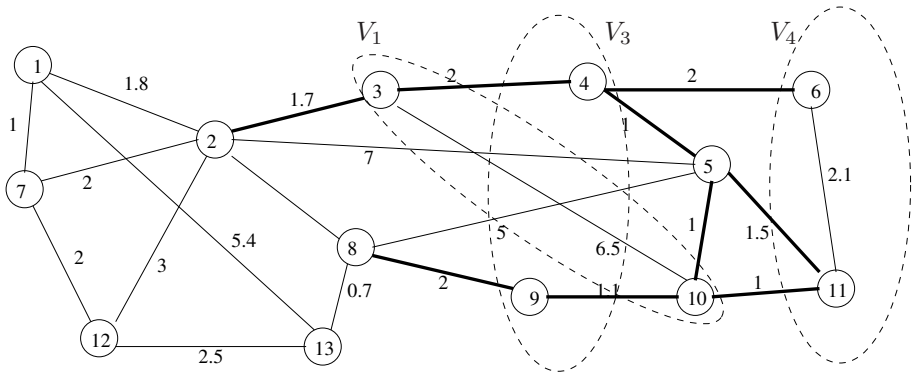
The solution is in the picture below.



For $k = 4$:

```
CPLEX 8.1.0: optimal integer solution; objective 332.5
18620 MIP simplex iterations
1403 branch-and-bound nodes
cost = 332.5

subnetwork 1: 1 2 5 7 8 12 13
subnetwork 2: 4 9
subnetwork 3: 3 10
subnetwork 4: 6 11
```

The solution is in the picture below.



13.3 Network Routing: Solution

We use a path formulation where each variable represents a possible path. Of course the number of paths in a graph is exponential in the size of the graph; so, in order to deal with polynomially-sized formulations only, we need to employ an iterative solution algorithm where a master problem and an auxiliary problem (both smaller in size than the complete exponential-sized formulation) are solved iteratively until convergence. This approach is similar to the one used to solve the Travelling Salesman Problem (TSP) exercise (Eg. 5.4, solution in Section 12.4); whereas in the TSP we generated one new (violated) constraint at each iteration from an exponentially-sized set of subtour elimination constraints, in this case we are going to generate one new path variable to be inserted in the model at each iteration, making sure that said variable has a negative reduced cost (cf. the simplex algorithm), so that it has a chance to decrease the objective function value. Such an approach is called a *column generation algorithm*, whereas the approach used to solve the TSP is a *constraint generation algorithm*.

The network topology is described by an undirected graph $G = (V, E)$ where V is the set of campuses and E is the set of inter-campus links. Let P be the set of possible paths in G, $P_{st} \subseteq P$ the subset of possible paths between campus s and campus t, and $P^{ij} \subseteq P$ the subset of possible paths travelling on link $\{i, j\} \in E$.

- *Indices*:
  1. i, j, s, t: indices in V;
  2. p: index on P, the set of paths of G.
- *Parameters*:
  1. d_{st}: traffic demands for $s \neq t$;
  2. u_{ij}: capacity installed on link $\{i, j\}$.
  3. c_{ij}: length of link $\{i, j\}$; the path length c^p is given by the sum $c^p = \sum_{\{i,j\} \in p} c_{ij}$ of the lengths of the links in p.
- *Variables*. x_p: quantity of traffic on path p.
- *Objective function*:
$$\min_x \sum_{p \in P} c^p x_p$$
- *Constraints*:
  1. (demand satisfaction) for each $s, t \in V$ such that $d_{st} > 0$
$$\sum_{p \in P_{st}} x_p = d_{st};$$
  2. (link capacities) for each link $\{i, j\} \in E$:
$$\sum_{p \in P^{ij}} x_p \leq u_{ij}.$$

At the outset, the master problem is defined starting from an initial set $P_0 \subseteq P$ of paths yielding a feasible routing. Of course the routing is optimal with respect to all the paths in P_0 but not to all possible paths in P. We now have to find a new path that is likely to reduce the objective function cost, or prove that no such path exists. Consider that in a dual LP formulation all variables are reformulated to constraints and vice versa. Thus, optimality within the master problem can be interpreted as follows: all the dual constraints in the dual of the master problem are satisfied. Non optimality with respect to

P, in terms of the primal master problem, means: there may be a path variable in $P \setminus P_0$ whose reduced cost is negative. In terms of the dual of the master problem, this means: there may be a dual constraint which is not satisfied by the current solution. We thus have to find an unsatisfied dual constraint from the set of dual constraints corresponding to the paths in $P \setminus P_0$. The dual of the master problem is:

$$\max \quad \sum_{s,t:s\neq t} d_{st} y_{st} + \sum_{\{i,j\}\in E} u_{ij} z_{ij} \tag{13.6}$$

$$\text{s.t.} \quad y_{st} + \sum_{\{i,j\}\in p} z_{ij} \leq c^p \qquad \forall \{s,t\} : s \neq t \forall p \in P_{st} \tag{13.7}$$

$$z_{ij} \leq 0, \tag{13.8}$$

where y_{st} are the dual variables corresponding to the primal constraints (1) and z_{ij} are those corresponding to the primal constraints (2). We remark that the value of these dual variables is known after the solution of the master problem. The reduced cost π_p corresponding to a primal variable x_p is the slack of the corresponding dual constraint, which is

$$\pi_p = c^p - y_{st} + \sum_{\{i,j\}\in p} z_{ij}.$$

Since we are interested in primal variables x_p with negative reduced cost $\pi_p < 0$ (i.e. those with violated dual constraint), we must find a path p such that $c^p + \sum_{\{i,j\}\in p} z_{ij} = \sum_{\{i,j\}\in p}(c_{ij} + z_{ij}) < y_{st}$. The auxiliary problem (also called the *pricing problem*) is a point-to-point shortest path problem from s to t on G where the edges are weighted by $c_{ij} + z_{ij}$.

Once the auxiliary problem is solved with solution p having cost $\pi_p < y_{st}$, we insert a new variable (column) x_p in the master problem. The process is iterated until for each pair of campuses s, t no more shortest paths having cost $< y_{st}$ are found. The master solution is optimal.

Thus, we need to solve two separate LP problems: the master problem (given above) and the auxiliary problem (which we solve using a network flow mathematical programming formulation):

- *Parameters*:
  1. $G = (V, A)$ where $(i, j) \in A$ se $\{i, j\} \in E$;
  2. source node s, destination node t;
  3. for each $(i, j) \in A$, c_{ij} is the arc weight;
  4. for each $(i, j) \in A$, z_{ij} is the value of the dual variable corresponding to the primal demand constraint having indices i, j in the primal problem.
- *Variables*: for each $(i, j) \in A$, f_{ij} is the flow on the arc.
- *Objective function*:
$$\min \sum_{\{i,j\}\in E} (z_{ij} + c_{ij}) f_{ij}$$
- *Constraints*:
  1. (flow conservation at source):
$$\sum_{j\in\delta^+(s)} f_{sj} - \sum_{j\in\delta^-(s)} f_{js} = 1;$$
  2. (flow conservation at other nodes):
$$\forall i \in V\backslash\{s,t\} \quad \sum_{j\in\delta^+(i)} f_{ij} = \sum_{j\in\delta^-(i)} f_{ji}.$$

The overall iterative algorithm is as follows. The master problem is solved within a main loop. After each master solution, and for each s, t with $d_{st} > 0$, we solve the auxiliary problem to find a shortest $s-t$-path having cost $< y_{st}$. If no such paths are found, the main loop terminates. Otherwise, the set of variables x_p corresponding to all paths p found by the auxiliary problems is added to the master problem.

In practice, we record P_{st} using two vectors `origin[p]` and `destination[p]` indexed on the paths p; P^{ij} is represented by a path-edge incidence matrix `incid[p,i,j]` whose value is 1 if $\{i, j\} \in p$ and 0 otherwise.

13.3.1 AMPL model, data, run

```
# colgen.mod

# nodes
set V;
# edge lengths
param c{V,V} >= 0, default 0;
# link capacities
param u{V,V} >= 0;
# traffic demands
param d{V,V} >= 0, default 0;
# current number of paths in the master problem
param paths >= 0, integer;
# set of paths
set P := 1..paths;
# path origins
param origin{P} symbolic;
# path destinations
param destination{P} symbolic;
# path-edge incidence matrix
param incid{P,V,V} binary, default 0;
# source node used in the auxiliary problem
param sour symbolic;
# destination node used in the auxiliary problem
param dest symbolic;

# quantity of traffic along a path (master problem)
var x{P} >= 0;
# flow along an edge (auxiliary problem)
var f{i in V, j in V : c[i,j] > 0 or c[j,i] > 0} >= 0;

# master problem formulation
minimize cost : sum{p in P, i in V, j in V :
  c[i,j] > 0 and incid[p,i,j] == 1} c[i,j] * x[p];
subject to demand {s in V, t in V : d[s,t] > 0} :
  sum{p in P : origin[p] == s and destination[p] == t} x[p] >= d[s,t];
subject to capacity {i in V, j in V : c[i,j] > 0} :
  sum{p in P : incid[p,i,j] == 1} x[p] <= u[i,j];

# auxiliary problem formulation
param cbar{V,V};
minimize flow :
  sum{i in V, j in V : c[i,j] > 0} cbar[i,j] * f[i,j];
subject to source :
  sum{j in V : c[sour,j] > 0} f[sour,j] -
  sum{j in V : c[j,sour] > 0} f[j,sour] = 1;
```

```
subject to conservation {i in V : i != sour and i != dest} :
  sum{j in V : c[i,j] > 0} (f[i,j] - f[j,i]) = 0;


# colgen.dat

set V := como cremona lecco milan piacenza ;

param :          c   u  :=
como lecco       30 200
como milan       50 260
como piacenza    110 200
lecco milan      55 260
lecco cremona    150 200
milan piacenza   72 260
milan cremona    90 260
piacenza cremona 100 200 ;

param            d :=
como lecco       20
como piacenza    30
milan como       50
milan lecco      40
milan piacenza   60
milan cremona    25
cremona lecco    35
cremona piacenza 30 ;

param incid :=
1 como lecco 1
2 como milan 1
2 milan piacenza 1
3 milan lecco 1
3 lecco como 1
4 milan como 1
4 como lecco 1
5 milan piacenza 1
6 milan piacenza 1
6 piacenza cremona 1
7 cremona milan 1
7 milan piacenza 1
7 piacenza como 1
7 como lecco 1
8 cremona milan 1
8 milan como 1
8 como lecco 1
8 lecco milan 1
8 milan piacenza 1 ;

param origin :=
1 como
2 como
3 milan
4 milan
5 milan
6 milan
7 cremona
8 cremona ;
```

```
param destination :=
1 lecco
2 piacenza
3 como
4 lecco
5 piacenza
6 cremona
7 lecco
8 piacenza ;

param paths := 8;


# colgen.run

model colgen.mod;
data colgen.dat;
option solver_msg 0;
option solver cplex;
option cplex_options "lpdisplay=0";

# c, u are symmetric matrices: complete data
for {i in V, j in V : c[i,j] > 0} {
  let c[j,i] := c[i,j];
  let u[j,i] := u[i,j];
}

param termination binary, default 0;

problem master : x, cost, demand, capacity;
problem auxiliary : f, flow, source, conservation;

repeat while (termination == 0) {

  problem master; solve master > /dev/null;

  let termination := 1;

  for {s in V, t in V : d[s,t] > 0} {
    let sour := s;
    let dest := t;
    for {i in V, j in V : c[i,j] > 0} {
      let cbar[i,j] := (capacity[i,j] + c[i,j]);
    }

    problem auxiliary; solve auxiliary > /dev/null;

    if (flow < demand[s,t]) then {
      let paths := paths + 1;

      for {i in V, j in V : c[i,j] > 0 and f[i,j] > 0.5} {
        let incid[paths, i, j] := 1;
      }

      let origin[paths] := s;
      let destination[paths] := t;

      let termination := 0;
    }
  }
```

```
};

printf "Routing cost %.1f\n", cost;
for {p in P : x[p] > 0} {
  printf "  path %d, traffic %.1f:", p, x[p];
  for {i in V, j in V : c[i,j] > 0 and incid[p,i,j] == 1} {
    printf " (%s,%s)", i, j;
  }
  printf "\n";
}
```

13.3.2 CPLEX Solution

```
Routing cost 23245.0
  path 1, traffic 20.0: (como,lecco)
  path 5, traffic 60.0: (milan,piacenza)
  path 9, traffic 30.0: (como,piacenza)
  path 10, traffic 35.0: (cremona,milan) (milan,lecco)
  path 11, traffic 30.0: (cremona,piacenza)
  path 12, traffic 50.0: (milan,como)
  path 13, traffic 25.0: (milan,cremona)
  path 14, traffic 40.0: (milan,lecco)
```

Chapter 14

Nonlinear programming: Solutions

14.1 Error correcting codes: Solution

1. *Indices*: $j \leq m$, $i \leq n$.
2. *Variables*:
   - $\underline{x}^i \in \mathbb{R}^m$: position of i-th message;
   - $\rho_i \in \mathbb{R}_+$: value of ρ on $\underline{x}^i$
3. *Objective function*:
$$\max \min_{i \leq n} \rho_i$$
4. *Constraints*:
   - (coordinates limits)
$$0 \leq \underline{x}^i_j \leq 1 \qquad \forall\, i \leq n, j \leq m$$
   - (distances)
$$||\underline{x}^i - \underline{x}^k|| \geq \rho_i + \rho_k \qquad \forall\, i, k \leq n$$

14.2 Airplane maintenance: Solution

1. *Indices*:
   - $i \leq n$ = number of maintenance centers;
   - $j \leq m$ = number of airports.
2. Parameters:
   - δ_j: latitude of airport j;
   - φ_j: longitude of airport j;
   - A_j: expected number of airplanes/year leaving from airport j;
   - r: earth radius;
   - P: capacity of centers (number of airplanes);
   - C_1: construction cost between 20°W e 40°E

- C_2: construction cost between 40°E e 160°E
- α: proportionality between distance and cost;

3. *Variables*:
   - x_i: latitude of center i ($90°\text{S} \leq x_i \leq 90°\text{N}$)
   - y_i: longitude of center i ($20°\text{W} \leq y_i \leq 160°\text{E}$)
   - d_{ij}: geodesic distance between center i and airport j ($d_{ij} \geq 0$)
   - w_{ij}: number of airplanes going to center i and coming from airport j ($0 \leq w_{ij} \leq A_j$)
   - $z_i = 1$ if center i is built between 20°W e 40°E, 0 otherwise
4. *Objective function*:
$$\min \sum_{i \leq n} \left(C_1(1 - z_i) + C_2 z_i + \alpha \sum_{j \leq m} w_{ij} d_{ij} \right)$$
5. *Constraints*:
   - *Distances*:
$$d_{ij} = 2r \operatorname{asin} \sqrt{\sin^2\left(\frac{x_i - \delta_j}{2}\right) + \cos x_i \cos \delta_j \sin^2\left(\frac{y_i - \varphi_j}{2}\right)} \qquad \forall i \leq n, j \leq m;$$
   - (maintenance)
$$\sum_{i \leq n} w_{ij} = A_j \qquad \forall j \leq m;$$
   - (capacity)
$$\sum_{j \leq m} w_{ij} \leq P \qquad \forall i \leq n;$$
   - (z definition)
$$\begin{aligned} y_i - 40° &\leq 360°(1 - z_i) \qquad \forall\, i \leq n \\ y_i - 40° &\geq -360° z_i \qquad \forall\, i \leq n \end{aligned}$$
   - (variables domains)
$$\begin{aligned} x_i, y_i, d_{ij} &\in \mathbb{R}_+ \qquad \forall\, i \leq n, j \leq m \\ w_{ij} &\in \mathbb{Z}_+ \qquad \forall\, i \leq n, j \leq m \\ z_i &\in \{0, 1\} \qquad \forall\, i \leq n. \end{aligned}$$

14.3 Pooling problem: Solution

Variables:

- x_A, x_B, x_C: crude in input valves A,B,C;
- y_{11}: petrol between pool and mixer 1;
- y_{12}: petrol between pool and mixer 2;
- y_{21}: petrol between input valve C and mixer 1;

- y_{22}: petrol between input valve C and mixer 2;
- p: sulphur percentage in petrol out of pool.

Formulation:

$$\begin{array}{rll} \max\limits_{x,y,p} & 9(y_{11}+y_{21})+15(y_{12}+y_{22}) & \text{revenue} \\ & -(6x_A+16x_B+10x_C) & \text{cost} \\ \text{t.c.} & x_A+x_B-y_{11}-y_{12}=0 & \text{mass balance in pool} \\ & x_C-y_{21}-y_{22}=0 & \text{mass balance in C} \\ & y_{11}+y_{21}\leq 100 & \text{market demand 1} \\ & y_{12}+y_{22}\leq 200 & \text{market demand 2} \\ & 3x_A+x_B=p(y_{11}+y_{12}) & \text{sulphur balance in pool} \\ & py_{11}+2y_{21}\leq 2.5(y_{11}+y_{21}) & \text{quality petrol 1} \\ & py_{12}+2y_{22}\leq 1.5(y_{12}+y_{22}) & \text{quality petrol 2} \end{array}$$

This problem is nonconvex due to the bilinear terms in p, y in the constraints, for an equation constraint is convex only if it is linear.

Notice this problem has a bilinear structure (i.e. we can define two sets of variables P and Y such that all product terms in the problem have the form py where $p \in P$ and $y \in Y$ — in other words there are no squares). This suggests that by fixing all variables in either set, the resulting subproblem is simply a Linear Programming (LP) problem that can be solved by CPLEX. Let $P(p, y)$ denote the full problem. For fixed values p' and y', let $P(y) = P(p, y|p = p')$ and $P(p) = P(p, y|y = y')$ (thus, $P(y)$ and $P(p)$ are LPs). The algorithm is as follows.

1. Let $k = 1$, $f_0 = -\infty$, $\varepsilon > 0$ and choose p' randomly.
2. Let y' be the optimal solution of $P(y)$.
3. Let p' be the optimal solution of $P(p)$ and f_k be its objective function value.
4. If $|f_k - f_{k-1}| > \varepsilon$ go to Step 2, otherwise terminate with solution (p', y') and objective function value f_k.

This algorithm is known as the Haverly Recursion Algorithm (HRA) and is a particular example of the Successive Linear Programming (SLP) technique for solving Nonlinear Programming (NLP) problems.

14.3.1 AMPL: model, data, run

```
# haverly.mod - Haverly's pooling problem

set X default 1..3;
set D default 1..2;

param d{D};
param q{D};
param r{D};
param c{X};
param pL := 0;
param pU := 5;

var x{X} >= 0, <= 300;
var y{D,D} >= 0, <= 200;
var p >= pL, <= pU;

maximize profit : sum{j in D} r[j] * (sum{i in D} y[i,j]) -
```

```
                  sum{k in X} c[k] * x[k];

subject to mass_balance_pool : x[1] + x[2] = sum{j in D} y[1,j];
subject to mass_balance_C    : x[3] = sum{j in D} y[2,j];
subject to demand{j in D}    : sum{i in D} y[i,j] <= d[j];
subject to sulphur_balance   : 3*x[1] + x[2] = p * (sum{j in D} y[1,j]);
subject to quality{j in D}   :
  p * y[1,j] + 2 * y[2,j] <= q[j] * sum{i in D} y[i,j];
```

```
# haverly.dat

param d :=
1 100
2 200;

param q :=
1 2.5
2 1.5;

param r :=
1 9
2 15;

param c :=
1 6
2 16
3 10;
```

```
# HRA (SLP) algorithm for Haverly's pooling problem
model haverly.mod;
data haverly.dat;

option solver cplex;
option solver_msg 0;

let p := Uniform(pL, pU);

param infinity := 100000;
param epsilon := 0.1;
param f0 default -infinity;
param f default infinity;

repeat while (abs(f - f0) > epsilon) {
  # create and solve P(y)
  fix p;
  unfix {j in D} y[1,j];
  solve;
  printf "current profit from P(y) is %f\n", profit;

  # create and solve P(p)
  unfix p;
  fix {j in D} y[1,j];
  solve;

  let f0 := f;
  let f := profit;
  printf "current profit from P(p) is %f\n", profit;
```

```
}
printf "maximal profit = %f, optimal percentage of sulphur = %f\n", f, p;
```

14.3.2 CPLEX solution

```
current profit from P(y) is 0.000000
current profit from P(p) is 0.000000
current profit from P(y) is 400.000000
current profit from P(p) is 400.000000
current profit from P(y) is 400.000000
current profit from P(p) is 400.000000
maximal profit = 400.000000, optimal percentage of sulphur = 1.000000
```

We remark that the HRA is very sensitive to the performance of the underlying LP solver (CPLEX in this case). So much so that with different versions of CPLEX we obtain different solutions — this is due to the fact that floating point errors in the LP solver may influence the next HRA iteration significantly. The above results were obtained with CPLEX v.9 in the "student" variant (downloaded as an AMPL solver). With CPLEX v.10.1 the performance is remarkably different: in order to obtain convergence to a profit > 398 it was necessary to insert another external loop where in case of failure p was fixed to a random value before restarting. Termination was obtained via a threshold value set at 380. The code is shown below.

```
# HRA (SLP) algorithm for Haverly's pooling problem

model haverly.mod;
data haverly.dat;

option presolve 0;
option solver cplex;
option solver_msg 0;

let p := Uniform(pL, pU);

param threshold := 380;
param infinity := 100000;
param epsilon := 0.001;
param f0 default -infinity;
param f default infinity;
param itn default 1;
param termination binary, default 0;

repeat while (!termination) {
  let p := Uniform(pL, pU);
  let f := infinity;
  let f0 := -infinity;

  repeat while (abs(f - f0) > epsilon) {

    # create and solve P(y)
    fix p;
    unfix {j in D} y[1,j];
    solve;
```

```
    # create and solve P(p)
    unfix p;
    fix {j in D} y[1,j];
    solve;

    # end iteration
    let f0 := f;
    let f := profit;
    printf "itn %d: profit(P(y),P(p))=(%g,%g), p=%g\n", itn, f0, f, p;
    let itn := itn + 1;
  }
  if (f >= threshold) then {
    let termination := 1;
  }
}

printf "maximal profit = %f, optimal percentage of sulphur = %f\n", f, p;
```

The solution (with CPLEX 10.1) is as follows.

```
itn 1: profit(P(y),P(p))=(100000,0), p=0.949365
itn 2: profit(P(y),P(p))=(0,0), p=0.949365
itn 3: profit(P(y),P(p))=(100000,0), p=4.60946
itn 4: profit(P(y),P(p))=(0,0), p=4.60946
itn 5: profit(P(y),P(p))=(100000,0), p=4.78578
itn 6: profit(P(y),P(p))=(0,0), p=4.78578
itn 7: profit(P(y),P(p))=(100000,0), p=0.528629
itn 8: profit(P(y),P(p))=(0,0), p=0.528629
itn 9: profit(P(y),P(p))=(100000,0), p=3.57053
itn 10: profit(P(y),P(p))=(0,0), p=3.57053
itn 11: profit(P(y),P(p))=(100000,84.0074), p=2.75766
itn 12: profit(P(y),P(p))=(84.0074,84.0074), p=2.75766
itn 13: profit(P(y),P(p))=(100000,353.871), p=1.31568
itn 14: profit(P(y),P(p))=(353.871,353.871), p=1.31568
itn 15: profit(P(y),P(p))=(100000,0), p=1.74802
itn 16: profit(P(y),P(p))=(0,0), p=1.74802
itn 17: profit(P(y),P(p))=(100000,0), p=2.03623
itn 18: profit(P(y),P(p))=(0,0), p=2.03623
itn 19: profit(P(y),P(p))=(100000,0), p=3.32606
itn 20: profit(P(y),P(p))=(0,0), p=3.32606
itn 21: profit(P(y),P(p))=(100000,93.1195), p=2.87904
itn 22: profit(P(y),P(p))=(93.1195,93.1195), p=2.87904
itn 23: profit(P(y),P(p))=(100000,0), p=4.71011
itn 24: profit(P(y),P(p))=(0,0), p=4.71011
itn 25: profit(P(y),P(p))=(100000,0), p=1.81762
itn 26: profit(P(y),P(p))=(0,0), p=1.81762
itn 27: profit(P(y),P(p))=(100000,0), p=0.0154438
itn 28: profit(P(y),P(p))=(0,0), p=0.0154438
itn 29: profit(P(y),P(p))=(100000,0), p=3.77799
itn 30: profit(P(y),P(p))=(0,0), p=3.77799
itn 31: profit(P(y),P(p))=(100000,0), p=2.25051
itn 32: profit(P(y),P(p))=(0,0), p=2.25051
itn 33: profit(P(y),P(p))=(100000,0), p=0.850609
itn 34: profit(P(y),P(p))=(0,0), p=0.850609
itn 35: profit(P(y),P(p))=(100000,0), p=3.93874
itn 36: profit(P(y),P(p))=(0,0), p=3.93874
itn 37: profit(P(y),P(p))=(100000,0), p=4.18904
```

```
itn 38: profit(P(y),P(p))=(0,0), p=4.18904
itn 39: profit(P(y),P(p))=(100000,97.4144), p=2.95083
itn 40: profit(P(y),P(p))=(97.4144,97.4144), p=2.95083
itn 41: profit(P(y),P(p))=(100000,398.429), p=1.01546
itn 42: profit(P(y),P(p))=(398.429,398.429), p=1.01546
maximal profit = 398.429299, optimal percentage of sulphur = 1.015464
```

14.4 Optimal rocket control 2: Solution

If the rocket leaves with a mass c of fuel burning as $\alpha u(t)$ kg s^{-1}, its equation of motion is:

$$\forall t \in [0,T] \quad m(t)\left(\frac{\partial^2 y(t)}{\partial t^2} + g\right) = u(t),$$

where $m(t) = m_0 + c - \int_0^t \alpha u(t')dt'$ and $m_0 \leq m(t) \leq m_0 + c$ for $t \in [0,T]$.

At time 0 (resp. T), the rocket must be at height 0 (resp. H); velocity at time 0 is 0, so $y(0) = v(0) = 0$,$y(T) = H$. The force acting on the rocket must not exceed b, so $|u(t)| \leq b$ for each $t \in [0,T]$. We must determine $u(t)$ so that the energy is minimum. Our objective function is thus $E = \int_0^T |u(t)|dt$. We obtain a nonlinear problem with time dependency:

$$\begin{array}{rl} \min & \displaystyle\int_0^T |u(t)|dt \\ \forall t \in [0,T] & |u(t)| \leq b \\ \forall t \in [0,T] & m(t)(\frac{\partial^2 y(t)}{\partial t^2} + g) = u(t) \\ & y(0) = 0 \\ & y(T) = H \\ & v(0) = 0. \end{array}$$

First we remove the time dependency. We consider a discretization of the interval $[0,T]$ in n subintervals, with $t_1 = 0$, $\Delta t = \frac{T}{n}$, $t_{n+1} = t_n + \Delta t = T$ e $t_k = t_1 + k\Delta t$ for each $k \leq n$. Let $y_k = y(t_k)$ and $v_k = \frac{\partial y(t)}{\partial y}\big|_{t_k}$ for each $k \leq n+1$. We introduce variables m_k (rocket mass at time t_k) and γ_k (approximation of $\int_0^{t_k} u(t)dt$) for $k \leq n+1$; we require $\frac{\gamma_{k+1} - \gamma_k}{\Delta t} = u_k$ for $k \leq n$. We obtain the following

model:

$$\begin{array}{rl}
\min & \sum_{k=1}^{n} |u_k| \\
\forall k \leq n & y_{k+1} - y_k = v_k \Delta t \\
\forall k \leq n & m_k v_{k+1} - m_k v_k = (u_k - m_k g)\Delta t \\
\forall k \leq n & m_k = m_0 + c - \alpha \gamma_k \\
\forall k \leq n & \gamma_{k+1} - \gamma_k = u_k \Delta t \\
\forall k \leq n+1 & |u_k| \leq b \\
& y_1 = 0 \\
& y_{n+1} = H \\
& v_1 = 0 \\
& m_1 = m_0 + c \\
\forall k \leq n+1 & 0 \leq y_k \leq H \\
\forall k \leq n+1 & v_k \geq 0 \\
\forall k \leq n+1 & m_k \geq m_0 \\
\forall k \leq n+1 & u_k \in \mathbb{R}.
\end{array}$$

The bilinear terms in m_k and v_k make the problem nonconvex.