in A. Nixon and M. Sitharam (eds.), Rigidity in Action, Fields Communications Series, FIelds Institute, Toronto (to appear)

Distance geometry with and without the graph

LEO LIBERTI[1], CARLILE LAVOR[2]

1. *LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, F-91128 Palaiseau, France*
   Email:liberti@lix.polytechnique.fr
2. *IMECC, University of Campinas, 13081-970, Campinas, Brazil*
   Email:clavor@unicamp.br

May 21, 2025

Abstract

We survey theoretical, algorithmic, and computational results at the intersection of distance geometry problems and mathematical programming, both with and without adjacencies as part of the input. While mathematical programming methods can solve large-scale distance geometry problems with adjacencies, they are severely challenged in the absence thereof.

Preface

This is one of many surveys we have co-authored about Distance Geometry (DG) [83, 64, 81, 18, 76, 44, 78, 70] over the years: the reader might then wonder what we could possibly survey that we have not already surveyed eight times in the last fifteen years.

In [83] we reviewed many continuous solution methods for the DG problem (which we define below) and a few discrete ones, among which ABBIE [50] and Branch-and-Prune [79], the most prominent application of DG being that of proteins. In [64] we surveyed several variants of the DG problem related to the Branch-and-Prune algorithm, the only DG application being (again) proteins. In [81] we surveyed what we thought was most of the field of DG: many problem variants, applications, theoretical results, and methods, among which the Branch-and-Prune and proteins. When [34] was published, however, we discovered we had neglected a large part of DG field: the "unassigned" side of things, where the weighted graph turned into a list of scalar weights. The survey [18], which was co-authored by only one of us, discusses methods for assigned and unassigned DG problems, focusing on rigidity, coordinates computation, nanostructures, and proteins. The survey [76] is at the opposite end of the "recent advances" survey style, as it focuses on six deep theorems about DG in the history of mathematics, from Heron of Alexandria to Kurt Gödel. The paper [44] reviews a very specific aspect of the Branch-and-Prune algorithm, namely the discretization of the search space, but it also proposes new ideas, and so it is only partly a survey. In [78], we review those branches of DG that lead to major open problems. Finally, [70], which was written by only one of us, surveys the interfaces between DG and other fields of applied mathematics and theoretical computer science: mathematical programming, linear algebra, computational complexity, natural language processing, artificial intelligence, machine learning, data science, and statistics.

Like [18], this survey focuses on the interplay between assigned and unassigned DG problems, and considers proteins as applications. Unlike [18], however, this survey explores mathematical programming formulations for both problems, discusses the relationship between "unassigned" and "assigned" versions of the problem, and presents a comparative computational benchmark for most of the proposed formulations. We hope that this unconventional computational angle will make mathematical programming formulations come to life, instead of remaining purely theoretical constructs.

The reader might also wonder why we are so keen on writing surveys. Aside from the odd one out [44], which is not quite just a survey, one possible explanation is the following. As well as many of our co-authors, we have had the good fortune to be invited to give talks in our careers. After delivering the talk, we were often asked (by whomever had invited us) to write a survey. This happened with [83, 64, 76]. We were also approached by some

Editors-in-Chief of journals (or book editors) that routinely commission surveys: this happened with [81, 18, 78, 70]. The current survey was invited by Meera Sitharam and Tony Nixon, who co-organized several thematic programs at the Fields Institute. We could therefore answer "it's not *us*, it's *them*!". But the truth is that writing surveys teaches us a lot: by re-organizing the material in a way that makes sense for us (and, we hope, for the reader too) we see the whole mountain, where previously we had only glimpsed at some peaks. So it is not quite them, it is really just us.

Contents

1 Introduction

DG problems come in two broad forms: assigned and unassigned. Their input is always a set of distances, but the word "distance" is ambiguous. On the one hand, "one meter", or "two-point-six Ångström" are distances. On the other hand, "there are 542 km between Toronto and Montreal" is also a distance. These two concepts are related, but their formalization is very different. For the first, a positive scalar suffices. For the second, one also needs two distinct points: more precisely, we assign the scalar (542) to an edge $\{\mathsf{Toronto}, \mathsf{Montreal}\}$. Since a set of edges defines a graph, DG problems can be defined on a graph (distance scalars are assigned to edges) or without a graph (distance scalars are "unassigned").

The output of DG problems, however, is always the same: we ask for a set of points in a Euclidean space such that each given distance appears exactly once as the Euclidean distance between two of the points. The points correspond to vertices of the graph in the assigned form; in the unassigned form, the number of points n is given as part of the input. This set of points is called *realization*.

Take the unassigned form. Once a realization is given that is compatible with the distance scalars, it is possible to construct a graph, since each scalar is now the length of a segment between two points. This tells us that the assigned form is easier than the unassigned one, since the graph is already given as part of the input. In the unassigned form we have to compute an assignment of distance scalars to edges as well as constructing a realization compatible with this assignment. This plays out in problem formulations as well: from each formulation of the assigned form, we can derive a more complicated formulation of the unassigned form by adding some assignment variables.

1.1 Problem definitions

We now give the formal decision version of the two problem forms, with and without the graph. For an integer p we let $[p] = \{1, \dots, p\}$. We recall that an assignment $\alpha : S \to T$ is an injective but not necessarily surjective function.

- DISTANCE GEOMETRY PROBLEM (DGP). Given an integer $K > 0$ and a simple undirected edge-weighted graph $G = (V, E, d)$, determine whether there exists a realization $x : V \to \mathbb{R}^K$ such that

$$\forall \{u, v\} \in E \quad \|x_u - x_v\|_2^2 = d_{uv}^2, \tag{1}$$

  where d_{uv} is the weight of the edge $\{u, v\}$ and $x_v \in \mathbb{R}^K$ is the position vector of vertex v.

- UNASSIGNED DISTANCE GEOMETRY PROBLEM (UDGP). Given two positive integers K, n and a list $L = (\delta_\ell \mid \ell \leq m)$ of positive scalars, determine whether there exists an assignment $\alpha : [m] \to [n] \times [n]$ and a realization $x : [n] \to \mathbb{R}^K$ such that:

$$\forall \ell \leq m \quad \|x_{\alpha_{\ell_1}} - x_{\alpha_{\ell_2}}\|_2^2 = \delta_\ell^2, \tag{2}$$

  where $\alpha(\ell) = (\alpha_{\ell_1}, \alpha_{\ell_2})$ for all $\ell \leq m$.

We remark that the UDGP is invariant with respect to the order of L (which is in fact a multi-set represented by a list). We write Eq. (1)-(2) in squared form because they avoid the square root relative to the ℓ_2-norm, which is computationally problematic. We also note that, if an assignment α is given, any UDGP instance becomes a DGP one, since the assignment allows the construction of a simple undirected graph edge-weighted by L.

As stated, the DGP and UDGP are decision problems having a single-bit (YES/NO) output, with a certificate for YES instances. This certificate is a realization x for the DGP, and a couple (realization x, assignment α) for the UDGP. DGP instances may be NO for two reasons: the given input is incompatible with any possible realization in K dimensions, or the given input is incompatible with realizations in *any* dimensions (or with the metric axioms). UDGP instances may be NO also because every assignment leads to an infeasible DGP instance.

We do not consider other distances than Euclidean in this survey, because our chosen application (molecules) are based on Euclidean distances (but see [31]): accordingly, we dispense with the problem name "Euclidean DGP" (EDGP), since there is no confusion. Nor do we consider imprecise versions of DGP and UDGP explicitly, where

d and δ are real intervals instead of scalars [83, 44, 32], mainly because the solution methods we discuss — all based on solving formulations with an off-the-shelf solver — already cater for this type of imprecision, also known as "experimental error" [15], by turning the strict feasibility enforced by Eq. (1) into optimization by means of a penalty function (see [83] for some formulations that explicitly cater for intervals, and [104] for a method to estimate missing distance values in the presence of intervals).

In the following we shall encode realizations of the DGP or the UDGP as $n \times K$ real matrices (the K-vector of the position of vertex $v \in V$ encoded in the v-th row of the matrix): thus, a realization can be precise, imprecise, approximate, random, or even wrong. In short, realizations will simply be whatever solution of the DGP/UDGP was found by the solution algorithm at hand.

2 Background and context

In this section we provide auxiliary and complementary information that will improve the understanding of the subsequent sections.

2.1 Complexity

The DGP (and, respectively, the UDGP) is the inverse problem of the following trivial problem: given a realization $x \in \mathbb{R}^{nK}$ compute a subset of distances adjacent to pairs of points (and, respectively, compute a subset of distance values). While the direct problems are trivial, the inverse problems are **NP**-hard. The DGP is **NP**-hard (weakly, by reduction from PARTITION to the case $K = 1$, and strongly, by reduction from 3SAT for any fixed K [105]). The UDGP is **NP**-hard even when $m = n(n-1)/2$ [66, Thm. 4.2], i.e. when all possibile distances are given, yielding a complete graph with any assignment α: by contrast, the DGP is tractable on complete graphs [108, 4]. The fact that **NP**-hardness of the DGP is proved on sparse graphs, while that of the UDGP is proved on instances corresponding to complete graphs, may be indicative of the respective application settings: DGPs are often solved on (sparse) disk graphs, while UDGPs on complete (or almost complete) lists of distance scalars[1].

While the DGP is **NP**-hard, it is only known to be in **NP** (i.e., **NP**-complete) for $K = 1$. For $K > 1$, a wholly integer input instance may give rise to a realization certificate that involves irrational components, which cannot be represented exactly in the Turing machine computation model (for $K = 1$ this does not happen). Since these irrational certificates are actually algebraic, a few attempts were made to employ different types of certificates, e.g. the minimal polynomials of the algebraic components and a rational closest to the specific desired root, but to no avail [13]. This failure, however, does not prove that the DGP is not in **NP**, and hence the problem remains open [78]. By contradiction, the UDGP has the same status. Suppose that the UDGP were in **NP** for all K: then every DGP arising from the UDGP instance given the assignment α would also be in **NP**. Since this covers all DGP instances in the DGP class, then the DGP would be entirely in **NP** for any K, which would settle the problem that is currently open.

2.2 Applications

The typical DGP applications are: synchronization of clocks in sensor network protocols [107] ($K = 1$), localization of sensor networks [20] ($K = 2$), reconstruction of the shape of proteins from distance data [27, 24] ($K = 3$), control of underwater autonomous vehicles [10] ($K = 3$), transformation of words and sentences into vectors in natural language processing [71, 55] (any K).

The typical UDGP applications are: the "partial digest" methodology in DNA sequencing[2] [110] ($K = 1$), also

[1] Even more-than-complete lists have been considered in solving UDGPs: in [54], some experiments have been conducted on a redundant list of distance scalars, i.e. having length exceeding m.

[2] Here is a summary description of the partial digest problem. Suppose one wants to determine thes sites of a DNA strand at which a certain sequence σ over A, C, G, T occurs. One employs a specific enzyme that breaks the strand at sites where σ appears, and measures the lengths of the pieces. One then solves a UDGP problem in one dimension in order to retrieve the sites at which the DNA strand was broken: those are the sites containing σ.

known as "turnpike" or "beltway" problem depending on whether the strand is an open path or a loop [109, 30]; the reconstruction of crystal structure from interatomic distances obtained from X-ray crystallography [102] ($K \in \{2,3\}$); the reconstruction of the shape of small inorganic molecules, such as e.g. nanostructures, from distance data [54, 18] ($K = 3$); and the reconstruction of the shape of proteins from distance data [15, 22] ($K = 3$).

In these lists of DGP and UDGP applications, we note immediately the preponderance of molecular applications, and the fact that the same application (shape of proteins) occurs in both. Molecular applications are frequent in DG because "looking" at a molecule, given its nano-scale, means irradiating it and actually look at the diffraction of the radiation as it passes through the molecule: either as a crystal, as in X-ray crystallography, or in solution, as in nuclear Overhauser effect spectroscopy (NOESY), or frozen in a thin layer of ice, as in Cryo-EM. The result of the analysis, the radiation spectrum, is represented as a multivariate function f in $1,2,3$ or even more dimensions. Some of the peaks of f indicate that two or more atoms of a certain type are at some given distance value (not all peaks are meaningful, however [102]).

To clarify, let us take a spectrum function $f(a,b)$ of two dimensions a,b. Supposing that a,b denoted atom IDs (e.g. $\mathrm{H}^9, \mathrm{C}^{17}$), one could simply read a peak in $f(a,b)$ as a distance between two well-defined atoms, with the distance value proportional to the integral of f over a neighbourhood including the peak. Instead, peaks depend on other entities that are correlated with distance values in more complicated ways.

- In the case of X-ray experiments on nanostructures, one can only obtain a probability of finding a certain distance between two atoms (this probability function is called *pair distribution function*, or sometimes simply PDF, in the relevant literature). Nonetheless, this yields unassigned but relatively precise and complete distance value measurements in nanostructures, which is why nanostructures are a typical UDGP application.
- In the case of NOESY experiments on proteins in solution, distance values are implied by resonance effects at some given frequencies specific to each atomic nucleus. These frequencies are known as the "chemical shifts" of an atom: a peak in chemical shift space may denote the closeness of two atoms, and their distance value can be estimated. The assignment of the values to atom pairs is also estimated, see Sect. 2.2.1 below. These estimations yield considerable errors [15], which is why the determination of the shape of proteins is an application of both the DGP and the UDGP [49, 117].

2.2.1 The case of proteins

Chemical shifts depend on the atom type and its environment. Unfortunately, many atoms in a protein have very close chemical shifts, so that, for practical purposes, these atoms share the same chemical shift. This means that a single peak $p = (a,b)$ of intensity $f(a,b)$ is assigned ambiguously to all of the atoms in the set A^p_a with the same chemical shift a, and all of the atoms in the set B^p_b with the same chemical shift b at the peak p. The intensity also depends on the number of bonds in $A^p_a \times B^p_b$ that actually occur in the protein with distance d proportional to p, which further complicates matters.

Luckily, though, proteins have a periodic backbone with a simple known structure. By a mixture of experimental and algorithmic procedures, this structure makes it possible to derive an assignment of distance values to edges: this justifies the appearance of proteins in the DGP applications list. These procedures are imperfect, however, and a systematic assignment error still occurs [15]: this requires a re-assignment phase, which justifies the appearance of proteins in the UDGP applications list.

2.3 Rigidity and flexibility

Rigidity theory likens a graph to a bar-and-joint structure, and follows the mechanical analogy mathematically. Accordingly, the graph plays a prominent part in rigidity theory, most of the results of which are rooted in the (assigned) DGP.

Given a DGP instance (K,G) where $G = (V,E,d)$, if the instance is YES it always has uncountably many different realizations, since, if $x \in \mathbb{R}^{nK}$ is a realization, $\phi(x)$ is a different realization for every nontrivial congruence ϕ of $\mathbb{R}^K$

(translation, rotations, reflections). With respect to G, a congruence fixes every distance defined by the realization of any pair of vertices:

$$\forall \phi \in \mathscr{E}(K), (u,v) \in V \times V \quad \|x_u - x_v\|_2 = \|\phi(x_u) - \phi(x_v)\|_2, \tag{3}$$

where $\mathscr{E}(K)$ is the group of congruences of $\mathbb{R}^K$. An *isometry* ψ with respect to the *framework* (G,x), on the other hand, only fixes every distance corresponding to a graph edge:

$$\forall \{u,v\} \in E \quad \|x_u - x_v\|_2 = \|\psi(x_u) - \psi(x_v)\|_2. \tag{4}$$

By Eq. (3)-(4), every congruence is an isometry, but the converse does not hold. Isometries are decomposable to the form $\psi \circ \phi$ where ϕ is a congruence. If a certain isometry ψ can only be decomposed in such a way that ϕ is the trivial congruence (identity), then ψ is irreducible. We consider the set $\mathscr{I}(G,x)$ of irreducible isometries, and among these we identify the local ones: a subset $\Psi \subset \mathscr{I}(G,x)$ consists of *local isometries* of (G,x) if for an arbitrarily small neighbourhood χ of x there is a nontrivial isometry $\psi \in \Psi$ with $\psi(x) \in \chi$. Local isometries may be partial translations and rotations (i.e. they only rotate and/or translate the realization of a nontrivial subgraph of G). Partial reflections [82, 32], on the other hand, are not local. A framework is *rigid* in $\mathbb{R}^K$ if it has no local isometries in $\mathbb{R}^K$, and *flexible* otherwise [7, §2]. This can be written formally in terms of the function that maps the realization $x \in \mathbb{R}^{nK}$ to the square distance value vector:

$$f_{G,K}(x) = (\|x_u - x_v\|_2^2 \mid \{u,v\} \in E),$$

that is, the framework (G,x) is rigid iff there is an arbitrarily small neighbourhood χ of x such that:

$$f_{G,K}^{-1}(f_{G,K}(x)) \cap \chi = f_{\bar{G}K}^{-1}(f_{\bar{G},K}(x)) \cap \chi,$$

where $\bar{G}$ is the *graph completion* of G, namely the complete graph over V. In other words, a framework is rigid if it locally behaves like a complete graph under isometries. This justifies the definition since the only isometries of complete graph frameworks are congruences.

The *rigidity matrix* $R^G(x)$ of the framework (G,x) is an $|E| \times |V|K$ matrix. Rows correspond to edges $\{u,v\}$ (with the convention that $u < v$) and columns to couples (w,k) of vertex and coordinate. The $(\{u,v\},(w,k))$-th entry of the rigidity matrix is defined as follows:

$$\forall \{u,v\} \in E, w \in V, k \leq K \quad R^G_{uv,wk}(x) = \begin{cases} x_{uk} - x_{vk} & \text{if } w = u \\ x_{vk} - x_{uk} & \text{if } w = v \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

A realization x of G is *regular* if its rigidity matrix $R^G(x)$ has maximum rank over all possible edge-weightings d of G and their corresponding realizations [77, §7.3.2]. Moreover, (G,x) is rigid in $\mathbb{R}^K$ if x is regular and the rank of $R^G(x)$ is $Kn - (K'+1)(2K-K')$, where K' is the dimension of the affine subspace spanned by x, and flexible if the rank is less than the given value [7, §3]. Moreover, (G,x) is *infinitesimally rigid* iff x is regular and (G,x) is a rigid framework.

It also turns out that frameworks of a given graph G are either almost all rigid, or almost all flexible [42]. This happens because almost all matrices have the maximum rank afforded by their size (smaller ranks occur by linear dependence, which occurs almost never). This allows us to treat rigidity and flexibility as properties of the graph rather than of the framework, which leads to *generically rigid* or *generically flexible* graphs [8]. Usually, this genericity is imposed by requiring that the components of x are algebraically independent on $\mathbb{Q}$.

The main open problem in rigidity theory is that of obtaining an exact algorithm for determining graph rigidity in $\mathbb{R}^K$ based only on the graph structure $G = (V,E)$, rather than on the edge weights [78]. Such algorithms exist for $K \in \{1,2\}$ [59, 88], but not for $K \geq 3$.

Because rigid graphs do not allow for local isometries, their number of isometric incongruent realizations is finite. Rigid graphs having a single realization (up to congruences) are known as *globally rigid*. This is particularly interesting in those DGP applications where one must reconstruct a single configuration of points in space from (assigned) distances: for example sensor network localization. Reconstructing the wrong network, even if it is consistent with the observed distances, would be useless. If a graph is globally rigid, there is no such risk.

2.3.1 Unassigned rigidity

It turns out that generic global rigidity was extended to the unassigned case in [45]: for a generic realization x of a generically globally rigid graph G, let $d = f_{G,K}(x)$. If there is a graph H with a realization y such that $f_{HK}(y)$ is a re-ordering of d, then H is isomorphic to G (i.e. H can be obtained from G by a vertex relabelling), and $x = y$ up to congruence. This result is valid for all $K \geq 2$. If G is 3-connected, it is also valid for $K = 1$. The result also implies that G, H have the same number of vertices and edges. This means that generic global rigidity depends on the edge weights only, rather than on their incidence to vertices. The condition on genericity is crucial: for nongeneric realizations there may exist cases where G, H are not isomorphic.

2.4 A mathematical programming primer

All of the methods we shall discuss in this survey are formulation-based. We are going to formulate the DGP and UDGP in many different ways by means of Mathematical Programming (MP), a declarative formal language for describing and solving optimization problems [115]. The general form of an MP formulation is:

$$\min_x \{f(x) \mid \forall i \leq p \; g_i(x) \leq 0 \wedge x \in X\}, \tag{6}$$

where:

- x is an array of *decision variables*;
- the functions $f(x)$ and $g_i(x)$ for each $i \leq p$ are represented by a mathematical expression based on a formal grammar with the usual arithmetic operators, elementary functions, and brackets;
- $f(x)$ is the *objective function* to be minimized;
- $\forall i \leq p \; g_i(x) \leq 0$ are *explicit constraints*;
- X is a set of *implicit constraints* that may be hard or inconvenient to represent, but for which there exist convenient computational methods.

We warn the reader that sometimes, to save space, MP formulations present some explicit constraints implicitly, when the explicit form has already been discussed previously.

Once a problem is represented by a MP formulation, one may look at the type of implicit constraints (e.g. nonnegative orthant, integer lattice), variables (e.g. continuous, integer, binary, mixed, matrix) and terms (e.g. linear, quadratic, polynomial, general nonlinear) involved in objective and constraints, and choose an off-the-shelf piece of software called *solver* that caters to the formulation properties. When the solver is deployed on the formulation, given sufficient time and assuming the formulation conforms to the theoretical assumptions assumed by the solver[3], it will provide one or more solutions to the problem, or report an error. Implicit constraints are handled by specific parts of the algorithm implemented by the chosen solver.

The input of an MP formulation are the mathematical expressions f, g_i (for $i \leq p$) and possibly the choice of solver given by X. The output is given by the values of the decision variables after the solution process, or the type of error returned. MP formulations may turn out to be feasible or infeasible, and bounded or unbounded. An appropriate solver may be able to prove feasibility/infeasibility, boundednes/unboundedness, and also provide a solution as an output (values of the decision variables after the solver terminates).

Solvers are implemented algorithms for solving a certain subclass of MP formulations. The different existing solvers yield a cover of the MP class by subclasses, which provide a taxonomy for MP. For the purposes of this survey, the taxonomy we make use of is: Linear Programming (LP), Semidefinite Programming (SDP), Mixed-Integer LP (MILP), Quadratically Constrained Quadratic Programming (QCQP), convex QCQP (cQCQP), Nonlinear Programming (NLP), convex NLP (cNLP), Mixed-Integer QCQP (MIQCQP), Mixed-Integer NLP (MINLP).

[3] Such assumptions, such as constraint qualifications in nonlinear problems, are sometimes computationally inefficient or impossible to verify in the Turing machine model.

Solvers may be local or global: a local solver requires a starting point (initial variable values) and reaches a close-by local optimum. A global solver gives some kind of guarantee of global optimality. Usually, local solvers for purely continuous problems are significantly faster than their global counterparts. If integer variables are involved, however, most problems become **NP**-hard, and local solvers may be as slow as global ones (in fact, with integer variables, a "local solver" often consists in running a global solver that is only allowed a given amount of computation time).

In the taxonomy above, LP is the only MP subclass in **P**, and the only subclass for which feasibility and boundedness (or their converse) can be proved by the solver with a certificate that can be verified by a Turing Machine (TM). Moreover, LP, SDP, and cQCQP are all convex MP formulations: if a local solver identifies a local optimum, it is also global by convexity. The cQCQP and SDP subclasses, therefore, have similar characteristics to the LP subclass, but within an $\varepsilon > 0$ precision limit. MILP, QCQP, MIQCQP, NLP, MINLP are all **NP**-hard; moreover, MIQCQP and MINLP are undecidable [69].

The situation with cNLP is more complicated: many cNLPs are "tame", in the sense that a local NLP solver will usually identify a local optimum which, by convexity, is also global (the "usually" caters for the possibility of the cNLP not conforming to constraint qualification). The cNLP class, however, also includes copositive reformulations of the Motzkin-Straus formulation [96] of MAX CLIQUE, a famously **NP**-hard problem, showing that cNLP is also **NP**-hard by inclusion [23]. In practice, no local NLP solver is able to deal with the implicit copositivity constraint.

An important feature of MP formulations is that they can be *reformulated*, i.e. symbolically changed, so that some aspect of the formulation remains invariant. The most common invariants are (a) the set of global optima, (b) at least one global optimum, (c) the globally optimal value, (d) a guarantee that the reformulation will yield a bound in the optimization direction of the original formulation, (e) a general approximation guarantee [68]. The reformulation used most frequently is *linearization* [73], which consists in replacing a mathematical expression $f(x)$ with a new variable y, then adjoin the *defining constraint* $y = f(x)$ to the formulation. The resulting reformulation is *exact*, i.e. all global optima of the reformulated problem can be mapped to global optima of the original problem. Linearizations are often employed as a starting point for further reformulations, usually of the *relaxation* type, i.e. a reformulation yielding a guaranteed bound in the optimization direction.

2.5 Solution methods

In this section we survey the most common solution methods for DGP and UDGP.

2.5.1 With the graph

Many DGP solution methods depend on further assumptions of rigidity. Sometimes it stems from the application, e.g. the "molecular rigidity" assumption [90] is exploited in [62] to devise a vertex order leading to an algorithmic approach. More often, certain classes of rigid graphs display an orderly structure that can be exploited algorithmically.

An example of this occurrence is given by *trilateration*, which refers to the determination of a single point at given distances from three other points on the earth surface [39]. A *trilaterative* graph $G = (V, E)$ is endowed with a vertex order such that, for each vertex v having order rank > 3, there are at least three vertices u_1, u_2, u_3, with rank less than v, such that $\{u_j, v\} \in E$ for $j \in \{1, 2, 3\}$ [38]. A *K-laterative* graph is analogous to a trilaterative one, where 3 is replaced by K [77]. The vertex order can be exploited in proofs by induction and in algorithms. For example, K-laterative graphs are generically rigid in $\mathbb{R}^K$ and generically globally rigid in $\mathbb{R}^{K-1}$. The proofs are as follows: in $K-1$ dimensions each vertex $v > K$ can be placed generically uniquely (if the DGP instance is YES); in K dimensions each vertex can be placed generically in at most two distinct positions. The placement operation reduces to solving a linear system in $K-1$ dimensions, and an easy quadratic system in K dimensions [77, Ch. 3], but there are faster methods for specific dimensions such as $K = 3$ in 3D [100, 44].

In sensor network localization, trilateration is exploited in 2D [9] to reconstruct a single network. The geometric build-up algorithm [35] exploits trilateration with $K = 4$ in 3D, while the Branch-and-Prune (BP) [79, 75, 46, 97, 99, 65, 116, 43] exploits trilateration with $K = 3$ in 3D. With general graphs, decomposition in rigid components is sometimes exploited [50, 28]. But once the rigid components have been individually realized, their synchronization

is not straightforward, and prone to errors.

Thus, most solution methods for the DGP on general graphs are mostly MP-based, which allows the practitioner to simply "model" the problem by an MP formulation, and call an appropriate solver (see Sect. 2.4 above). To aim at solving large-sized instances, on the other hand, some hand-crafted MP-based solution algorithms for solving the DGP as an NLP formulation have been devised and implemented in the past, e.g. [94, 51, 80, 87] (see [83] for more details). Some of these methods gain considerable speed-ups from combinatorial considerations, such as for example those based on *facial reduction* [21, 56, 57], which reduces the number of variables in conic programs such as SDP, often used in solving DGP and UDGP as detailed below (Sect. 4).

An interesting alternative to hand-crafted MP-based methods is to deploy random projections [70, §7.3] on DGP formulations [85]: this reduces the size of MP formulations for large DGP instances, which can therefore be solved by an off-the-shelf solver. A solution retrieval process then provides an approximate solution to the original, large-sized instance.

2.5.2 Without the graph

To the best of our knowledge, MP was only used for solving the UDGP in [30, 37, 22]. In [30], the 1-dimensional UDGP (UDGP_1) arising in the partial digest methodology was modelled as binary quadratic program, and solved approximately using SDP, with the approximation being exact for some UDGP_1 subclasses. The formulations in Eq. (11)-(12) below represent the theoretical contribution of [37]. In [22], a MINLP was used to maximize the number of satisfiable (interval) distance constraints while keeping the feasibility error of each infeasible constraint as small as possible.

There are several algorithms for solving the UDGP in one dimension (the typical setting of the "partial digest" methodology, see Sect. 2.2). The first two proposed in the literature are a pseudopolynomial algorithm and a practically faster exponential algorithms are given in [109]. An extension of the exponential algorithm to arbitrary dimensions is given in [66], and a deployment of the exponential algorithm in the noisy setting afforded by the application is given in [110]. Other papers about the partial digest problems focused on complexity, noisy data, bounds on the number of solutions, and heuristic algorithms. The most recent paper about the UDGP in one dimension is [26], which is based on the Lenstra-Lenstra-Lovász (LLL) basis reduction algorithm.

The "Tribond" algorithm was first proposed in [47]: the explanation is informal, and limited to 2D (so are the experiments), but the algorithm is potentially applicable to any dimension: in particular, it was applied to 3D molecules in [36]. Tribond is essentially a backtracking algorithm that starts from a subsequence of $(K+2)(K+1)/2$ distance values forming a consistent $K+2$ point structure (called "core") in $\mathbb{R}^K$ with a redundant edge (meaning that the removal of an edge leaves the structure rigid in $\mathbb{R}^K$), and then iteratively attempts to increase the size of the structure by looking for subsequences of $K+1$ distance values consistent with segments between $K+1$ already existing points and a single new point added to the structure. Backtracking occurs both in the core construction phase, and in the incremental phase. The algorithm terminates when the structure grows to the given number n of points. The "Liga" algorithm is an extremely effective heuristic method based on growing, selecting, and combining substructures (in a way similar to genetic algorithms) until a satisfactory structure emerges [54]. Both Tribond and Liga have been conceived with the purpose of realizing small molecules for which it is possible to obtain a rich (sometimes complete or even redundant) set of distances. While Tribond is exhaustive, and so provides some sort of guarantee, Liga is not. Both, however, provided very convincing results of interest to the physical-chemical communities.

3 MP formulations for the DGP and the UDGP

All DGP formulations are either equivalent to, or derived from, Eq. (1), and similarly for UDGP formulations. By turning the decision problem Eq. (1) into an optimization problem, most of our formulations (with the exception of push-and-pull and variants, see Sect. 3.4) relax feasibility into optimality: specifically, the DGP instance is YES if and only if the globally optimal objective function value is zero. Moreover, if the instance is NO because of slightly imprecise distance data, any realization will yield an optimal objective function value that is "reasonably" small. This

is why, in this survey, we chose to dispense from representing experimental measurement errors explicitly by intervals: one may more simply cater to these errors by accepting realizations with objective values that are slightly larger than zero, but within a given $\varepsilon > 0$.

With most DGP formulations, we can derive a corresponding UDGP formulation by means of the mapping DGP $\rightarrow$ UDGP given by $\mathscr{D} \rightarrow \mathscr{U}_<$ mentioned in Sect. 3.1 just before Example 3.4. This usually involves changing a few indices and sets in the formulation, and adding the component relative to the assignment α.

All congruences (translations, rotations, reflections) applied to a valid realization produce another valid realization: thus, all MP formulations of YES instances have in fact an uncountable number of global optima. We found that it sometimes help solvers to fix at least the translations (fixing rotations and reflections at the formulation level is harder). This can be achieved with a set of *centroid constraints*:

$$\forall k \leq K \quad \sum_{u \in V} x_u = 0. \tag{7}$$

For UDGP formulations, the quantification of the sum is on $i \leq n$.

3.1 Basic results

We make the statement about the the relative difficulty betwen UDGP and DGP (paragraph "Take the unassigned form [...]" in Sect. 1 on p. 3) more precise. In doing so, we shall introduce some notions and notations which will allow us to derive reformulations in subsequent sections.

For a function f, $\mathsf{dom} f$ is the domain and $\mathsf{ran} f$ the range of f. Note that the sequence δ can be seen as a function $[m] \rightarrow \mathbb{R}_+$.

3.1 Proposition
Let $\mathscr{U} = (K, n, L)$ be an instance of the UDGP, and $\alpha : [m] \rightarrow [n] \times [n]$ be an assignment. Consider the graph $G_\alpha = (V, E, d)$ where $V = [n]$, $E = \mathsf{ran}\alpha$, $d = \delta \circ \alpha^{-1}$ and the corresponding DGP instance $\mathscr{D}_\alpha = (K, G_\alpha)$. If $\mathscr{U}$ is YES, then $\mathscr{D}_\alpha$ is YES. If $\mathscr{U}$ is NO, $\mathscr{D}_\alpha$ is NO for all possible α.

Proof. Assume $\mathscr{U}$ is YES. The graph G_α has $[n]$ as vertex set V, and edges $\{u, v\}$ for $u < v \leq n$ whenever there is $\ell \leq m$ with $\alpha(\ell) = (u, v)$. Moreover, each edge $\{u, v\}$ is weighted by δ_ℓ such that $\alpha(\ell) = \{u, v\}$. By Eq. (2) and the definition of G_α, x also satisfies Eq. (1), i.e. $\mathscr{D}_\alpha$ is YES. Assume now that $\mathscr{U}$ is NO, and suppose, to aim at a contradiction, that there is α such that $\mathscr{D}_\alpha$ is YES: then it has a realization x for G_α, which, by Eq. (1) and by definition of G_α, is also a realization of $\mathscr{U}$, against the assumption. □

We note that Prop. 3.1 introduces a mapping from UDGP to DGP instances, namely $(\mathscr{U}, \alpha) \rightarrow \mathscr{D}_\alpha$. We also emphasize the notation G_α in order to refer to the the graph corresponding to the assignment α from a UDGP instance, as in Prop. 3.1.

3.2 Example
Consider the UDGP with input $(K = 2, n = 3, L = (3, 1, 1))$. Since $K = 2$ and $n = 3$ we want a triangle in the plane. But since the distance scalars $3, 1, 1$ violate the triangular inequality, the instance is infeasible. Observe that L is a list rather than a set of scalars, since we may need to specify scalars with a multiplicity greater than one (this is often the case in nanostructures). Moreover, by Prop. 3.1, no reconstructed graph G_α yields a feasible DGP instance. ■

3.3 Example
Consider the UDGP with input $(K = 2, n = 3, L = (\delta_1, \delta_2, \delta_3))$, which results in a triangle in the plane, with side lengths $\delta_1, \delta_2, \delta_3$ and an underlying graph

$$T_2 = (\{1,2,3\}, \{\{1,2\}, \{2,3\}, \{1,3\}\}),$$

which is the complete graph on 3 vertices. The realization does not change if the point labels change, so all of the possible assignments are valid:

- $\alpha^1 : \delta_1 \to \{1,2\}, \delta_2 \to \{2,3\}, \delta_3 \to \{1,3\}$
- $\alpha^2 : \delta_1 \to \{1,2\}, \delta_2 \to \{1,3\}, \delta_3 \to \{2,3\}$
- $\alpha^3 : \delta_1 \to \{1,3\}, \delta_2 \to \{2,3\}, \delta_3 \to \{1,2\}$
- $\alpha^4 : \delta_1 \to \{1,3\}, \delta_2 \to \{1,2\}, \delta_3 \to \{2,3\}$
- $\alpha^5 : \delta_1 \to \{2,3\}, \delta_2 \to \{1,2\}, \delta_3 \to \{1,3\}$
- $\alpha^6 : \delta_1 \to \{2,3\}, \delta_2 \to \{1,3\}, \delta_3 \to \{1,2\},$

even though all are realized by the same triangle, but with permuted vertex labels. In general, if an assignment α is computed for a UDGP instance, leading to a graph G_α, all isomorphic versions of G_α are admissible reconstructions. If all possible distances are given for the number n of points (as e.g. in this example $m = |L| = 3 = n(n-1)/2$), all $n!$ vertex permutations lead to feasible assignments, i.e. to isomorphic copies of the 3-clique graph. ■

An inverse mapping from DGP to UDGP instances is constructed as follows: let $\mathscr{D} = (K, G = (V, E, d))$ be a DGP instance. Let $<$ be any total order on E inducing the edge list $(e_1, \ldots, e_m)$. Then we can define $\delta_\ell = d_{e_\ell}$ for all $\ell \leq m$. This yields a UDGP instance $\mathscr{U}_< = (K, |V|, \delta)$ which is YES if $\mathscr{D}$ is YES. By contrast, $\mathscr{U}_<$ may be YES even though $\mathscr{D}$ is NO, as shown in Example 3.4.

3.4 Example
Consider the graph G given by the following weight function $d_{12} = 3, d_{23} = 4, d_{13} = 5, d_{14} = 2, d_{24} = 2$, shown in Fig. 1 (top left) with the correct realization: Evidently, the DGP with $K = 2$ on G is YES. Now consider instead

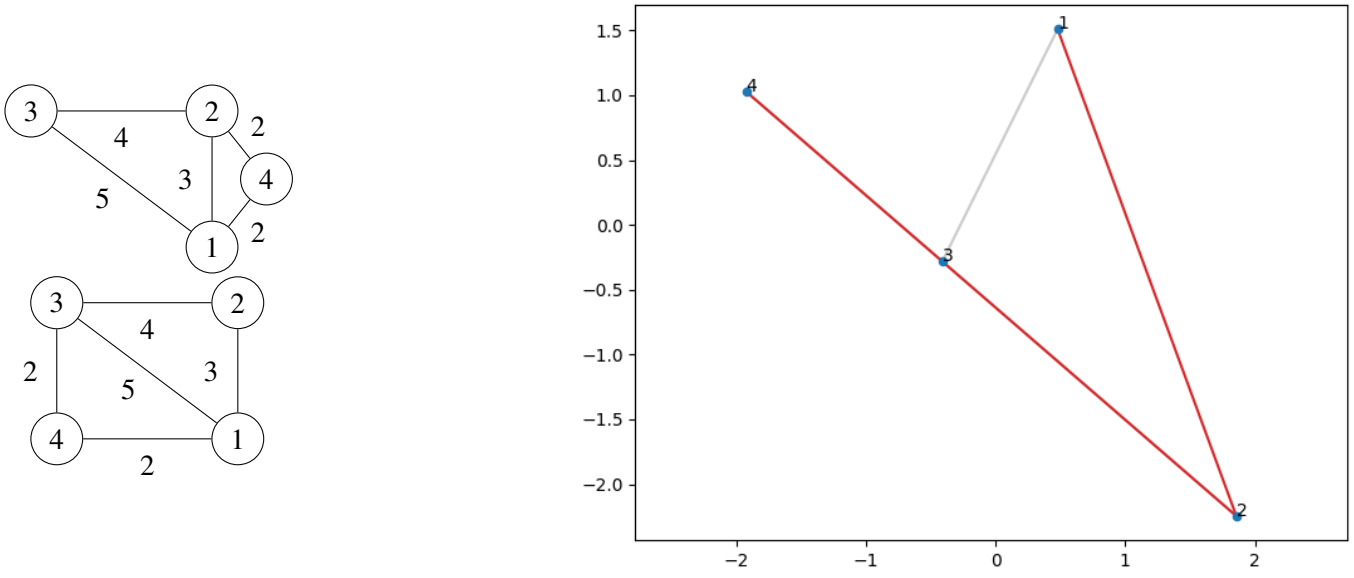


Figure 1: The same UDGP instance gives rise to (at least) three DGP instances (two YES, one NO).

the graph H defined by $d_{12} = 3, d_{23} = 4, d_{13} = 5, d_{14} = 2, d_{34} = 2$, where the distance value 2 previously assigned to $\{2,4\}$ is now assigned to $\{1,4\}$, shown in Fig. 1 (bottom left) with an incorrect realization. It is obvious that the DGP instance defined on H is NO, since there is no triangular realization for the subgraph $H[1,3,4]$ defined on distance values $5,2,2$: they do not satisfy triangular inequalities. Nonetheless, both DGPs are obtained from the same UDGP instance $\mathscr{U} = (2,4,(2,2,3,4,5))$ by means of different assignments:

$$\begin{aligned} \alpha_G &: 1 \to \{1,4\}, 2 \to \{2,4\}, 3 \to \{1,2\}, 4 \to \{2,3\}, 5 \to \{1,3\} \\ \alpha_H &: 1 \to \{1,4\}, 2 \to \{3,4\}, 3 \to \{1,2\}, 4 \to \{2,3\}, 5 \to \{1,3\}. \end{aligned}$$

Another possible solution of $\mathscr{U}$, found by a solver deployed on an MP formulation, is the assignment $\alpha' : 1 \to \{1,3\}, 2 \to \{3,4\}, 3 \to \{2,3\}, 4 \to \{1,2\}, 5 \to \{2,4\}$ leading to the realization in Fig. 1 (right), where the triangle on $2,3,4$ is flat. ■

3.2 The quartic formulation

Minimizing the sum of squared differences of the two sides of a system of equations is the most common way to solve such a system. This probably comes from the fact that, for a linear system, there exists a closed formula (linear regression); moreover, for a linear system, the corresponding optimization problem is a "tame" cNLP. The application of this technique to nonlinear systems, by contrast, gives rise to nonconvex NLPs in general, which is an **NP**-hard class. When applied to the DGP system (1), this is to be expected in view of the fact that the DGP itself is **NP**-hard, but it weakens the justification for choosing the minimization of the sum of squared differences to reformulate Eq. (1). This formulation can be traced back to [111] (and perhaps even earlier). It was tested computationally as a DGP formulation starting from [63] and in many more papers after it.

$$\min_{x\in\mathbb{R}^K} \sum_{\{u,v\}\in E} \left(\|x_u - x_v\|_2^2 - d_{uv}^2\right)^2. \tag{8}$$

The quartic formulation (8) is unconstrained. It minimizes a nonconvex multivariate polynomial of degree 4 (hence the name *quartic*). It can be solved globally with a global NLP solver, or locally with a local NLP solver starting from a given imprecise realization x' of the weighted graph defined by $\{d_{uv} \mid \{u,v\} \in E\}$.

3.2.1 The unassigned quartic

We represent the unknown assignment $\alpha : [m] \to [n] \times [n]$ by means of binary variables $y_{ij\ell}$ such that $y_{ij\ell} = 1$ iff $\alpha(\ell) = (i,j)$. The assignment properties are that (i) it is a function:

$$\forall \ell \leq m \quad \sum_{i<j\leq n} y_{ij\ell} \leq 1, \tag{9}$$

and (ii) it is injective:

$$\forall i < j \leq n \quad \sum_{\ell\leq m} y_{ij\ell} = 1. \tag{10}$$

Since these assignment constraints will be repeated often in the following, we summarize them in the set $\mathscr{A} = \{y \in \{0,1\}^{n^2m} \mid \text{Eq. (9)-(10)}\}$, and then use the implicit constraint $y \in \mathscr{A}$ to refer to the (explicit) constraints Eq. (9)-(10) together with the implicit constraint $y \in \{0,1\}^{n^2m}$.

We now modify Eq. (8) so that each term in the objective is added to the sum only if the corresponding y variable is set to 1:

$$\min_{\substack{x\in\mathbb{R}^{nK}\\ y\in\mathscr{A}}} \sum_{\substack{\ell\leq m\\ i<j\leq n}} y_{ij\ell}\left(\|x_i - x_j\|_2^2 - \delta_\ell^2\right)^2. \tag{11}$$

The quartic UDGP formulation Eq. (11) is a constrained MINLP involving polynomial functions of degree 5, which first appeared in [37]. It can be solved using a global MINLP solver. Because of the presence of the binary variables y, local solutions may be achieved by giving a resource constraint (CPU time, iterations, number of nodes) to the global solver. Using global solvers with resource constraints is referred to as "using a global solver locally".

There is an interesting, and unexpected continuous exact reformulation of Eq. (11), yielding a nonconvex NLP which can be solved locally or globally using a local or global NLP solver. This is based on the observation that, under certain conditions, maximizing a sum of squared variables defined over $[0,1]$ yields a binary vector.

$$\left.\begin{array}{rcl} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ y\in\mathscr{A},t\in\mathbb{R}}} \quad t - \sum\limits_{\substack{\ell\leq m\\ i<j\leq m}} y_{ij\ell}^2 & & \\ \sum\limits_{\substack{\ell\leq m\\ i<j\leq n}} y_{ij\ell}\left(\|x_i - x_j\|_2^2 - \delta_\ell^2\right)^2 & = & t. \end{array}\right\} \tag{12}$$

The proof that this reformulation is indeed exact is given in [37, Thm. 2]. This formulation is enticing insofar as it would allow one to solve it locally using local NLP solvers, which take a starting point and improve it (which is usually much faster than using a global solver locally on a mixed-integer formulation). However, the theorem that guarantees that the continuous y variables have binary values only applies at global optima. We also note that, at

global optima, the objective function value is not zero but $-m$. Thus, it might be harder to claim that NO instances with only slighly imprecise distances are recognizable by a slight variation on the objective function: in Eq. (12) the "error" given by the objective value represents distance errors as well as assignment errors (since a non-binary y value cannot be interpreted as an assignment).

3.3 The system formulation

The system formulation is almost equivalent to the quartic: it simply linearizes the differences under the square:

$$\left.\begin{array}{rll} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ s\in\mathbb{R}^m}} & \sum\limits_{\{u,v\}\in E} s_{uv}^2 & \\ \forall\{u,v\}\in E & \|x_u - x_v\|_2^2 & = \quad d_{uv}^2 + s_{uv}. \end{array}\right\} \tag{13}$$

The system formulation is a nonconvex QCQP which can be solved locally or globally by local or global QCQP or NLP solvers. This formulation probably first appeared in print in [70, Eq. (24)], but it was used in computational work at least since 2014.

A variant of the system formulation is based on the ℓ_1-norm to penalize the error, instead of the ℓ_2-norm:

$$\left.\begin{array}{rll} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ s\in\mathbb{R}^{2m}}} & \sum\limits_{\{u,v\}\in E} (s_{uv}^+ & + \quad s_{uv}^-) \\ \forall\{u,v\}\in E & \|x_u - x_v\|_2^2 & = \quad d_{uv}^2 + s_{uv}^+ - s_{uv}^-. \end{array}\right\} \tag{14}$$

This is actually a Quadratically Constrained Program (QCP) since the objective is linear. The QCP class is a subsclass of QCQP (since all linear forms are also trivial quadratic forms). As regards DGP and UDGP formulations, there is no substantial practical difference between the two classes, as there are no specific QCP solvers that cannot solve QCQPs too.

3.3.1 The unassigned system formulation

We refer to Sect. 3.2.1 for the definition of the assignment constraint set $\mathscr{A}$ and the binary variables y. The unassigned version of Eq. (13) is

$$\left.\begin{array}{rll} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ s\in\mathbb{R}^m, y\in\mathscr{A}}} & \sum\limits_{\substack{\ell\le m\\ i<j\le n}} s_\ell^2 & \\ \forall \ell\le m, i<j\le n & -s_{uv} - M(1-y_{ij\ell}) \le & \|x_i - x_j\|_2^2 - \delta_\ell^2 \quad \le s_{uv} + M(1-y_{ij\ell}), \end{array}\right\} \tag{15}$$

where M is a "big-M" constant that must be an upper bound to the diameter of any realization satisfying the DGP. A very slack bound $M = (\sum_{\ell\le m}\delta_\ell)^2$ is provided in [22, Prop. 2.2].

The unassigned version of the ℓ_1-norm variant in Eq. (14) is:

$$\left.\begin{array}{rll} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ s\in\mathbb{R}^{2m}, y\in\mathscr{A}}} & \sum\limits_{\substack{\ell\le m\\ i<j\le n}} (s_\ell^+ + s_\ell^-) & \\ \forall \ell\le m, i<j\le n & -s_{uv}^- - M(1-y_{ij\ell}) \le & \|x_i - x_j\|_2^2 - \delta_\ell^2 \quad \le s_{uv}^+ + M(1-y_{ij\ell}). \end{array}\right\} \tag{16}$$

In Eq. (15)-(16) the y variables are used to activate or deactivate the constraints according to whether the distance δ_ℓ is assigned to edge $\{u,v\}$ or not. We note that the term on the objective need not be multiplied by y when $y=0$ since this is taken care of by the optimization direction.

The advantage of unassigned system formulations w.r.t other UDGP formulations is that they are MIQCQPs instead of MINLPs, which allows one to deploy a larger set of solvers upon them. In fact, Eq. (16) is a Mixed-Integer QCP (MIQCP). Eq. (16) was first used in [72].

3.4 The push-and-pull formulation

The push-and-pull formulation of the DGP [32, 93] is as follows:

$$\left.\begin{array}{rll} \max\limits_{x\in\mathbb{R}^{nK}} & \sum\limits_{\{u,v\}\in E} \|x_u - x_v\|_2^2 & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \leq \quad d_{uv}^2. \end{array}\right\} \tag{17}$$

It is a nonconvex QCQP with a concave objective (maximization of a convex function) and convex quadratic constraints. It can be solved locally or globally by local or global QCQP or NLP solvers. Its name is given by the suggestion that the constraints push the realization points together, while the objective pulls them apart.

There is also a pull-and-push formulation that inverts the objective direction and the constraint senses:

$$\left.\begin{array}{rll} \min\limits_{x\in\mathbb{R}^{nK}} & \sum\limits_{\{u,v\}\in E} \|x_u - x_v\|_2^2 & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \geq \quad d_{uv}^2, \end{array}\right\} \tag{18}$$

which is not quite as useful as Eq. (17) since it is usually (practically) easier, for many local NLP solvers, to decrease a difficult objective than to satisfy difficult constraints. Nonetheless we shall see later that even Eq. (18) has a use.

It is not immediately obvious that Eq. (17) is an exact reformulation of Eq. (1). A proof of this fact is given in [93, Prop. 2.8] (the proof for Eq. (18) is analogous).

3.4.1 The unassigned push-and-pull

We refer to Sect. 3.2.1 for the definition of the assignment constraint set $\mathscr{A}$ and the binary variables y. The unassigned version of Eq. (17) is

$$\left.\begin{array}{rll} \max\limits_{\substack{x\in\mathbb{R}^{nK} \\ y\in\mathscr{A}}} & \sum\limits_{\substack{\ell\leq m \\ i<j\leq n}} y_{ij\ell}\|x_i - x_j\|_2^2 & \\ \forall \ell \leq m, i < j \leq n & \|x_i - x_j\|_2^2 & \leq \quad \delta_\ell^2 + M(1-y_{ij\ell}). \end{array}\right\} \tag{19}$$

As in the unassigned system formulations, a value of M is given in [22, Prop. 2.2]. In Eq. (19) the y variables only count assigned indices in the objective, and deactivate constraints for non-assigned indices.

Eq. (19) is not quadratic, since the objective is a cubic polynomial. It is therefore a MINLP, which can be solved with a global MINLP solver, possibly used locally. A similar formulation to Eq. (19) was given in [22].

3.5 The cycle formulation

The cycle formulation for the DGP is presented in [74]. In its native form, it decomposes the DGP into two phases: a constrained optimization problem and the solution of a linear system. The optimization problem constraint is quantified over a basis $\mathscr{B}$ [6] of the cycle space of the input graph G:

$$\left.\begin{array}{rll} \min\limits_{z\in[-\mathbf{d},\mathbf{d}]^{mK}} & \sum\limits_{\{u,v\}\in E} (\|z_{uv}\|_2^2 & - \quad d_{uv}^2)^2 \\ \forall C \in \mathscr{B}(G) & \sum\limits_{\{u,v\}\in C} z_{uv} & = \quad 0, \end{array}\right\} \tag{20}$$

where $\mathbf{d}_{uv} = (d_{uv}, \ldots, d_{uv}) \in \mathbb{R}^K$ and $z_{uv} = (z_{uv1}, \ldots, z_{uvK})$ for each $\{u,v\} \in E$. The linear system is:

$$\forall \{u,v\} \in E, k \leq K \quad x_{uk} - x_{vk} = z_{uvk}. \tag{21}$$

Once Eq. (20) is solved, Eq. (21) is a linear system of mK equations and nK unknowns, which can be solved (if z is an optimum of Eq. (20)) to retrieve the realization $x \in \mathbb{R}^{nK}$.

The proof in [74] leading to the correctness of this decomposition is long but elementary. However, the fact that the problem is decomposed in a hard part (Eq. (20)) and an easy part (Eq. (21)) does not improve computational

performances by much. We therefore propose the following exact reformulation, which integrates Eq. (21) into Eq. (20): this considerably shortens the correctness proof.

$$\left.\begin{array}{lrcll} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ z\in[-\mathbf{d},\mathbf{d}]^{mK}}} & \sum\limits_{\{u,v\}\in E}(\|z_{uv}\|_2^2 & - & d_{uv}^2)^2 & \\ \forall C\in\mathscr{B}(G) & \sum\limits_{\{u,v\}\in C} z_{uv} & = & 0 & (\dagger) \\ \forall\{u,v\}\in E & x_u - x_v & = & z_{uv}. & (\ddagger) \end{array}\right\} \tag{22}$$

3.5 Proposition
Eq. (22) *is an exact reformulation of Eq.* (1).

Proof. Let x by any solution of Eq. (1). By the *linearization constraints* (‡) we know that the objective function value of Eq. (22) is zero. Moreover, for any $k \leq K$ and any cycle $C = \{1,\ldots,c\}$ (wlog) in the graph we have:

$$\begin{array}{rcl} (x_{1k}-x_{2k})+(x_{2k}-x_{3k})+\cdots+(x_{ck}-x_{1k}) & = & \\ = x_{1k}-(x_{2k}-x_{2k})-(x_{3k}-x_{3k})+\cdots-x_{1k} & = & 0. \end{array}$$

Since every cycle can be generated linearly from the cycle basis $\mathscr{B}(G)$, the *cycle constraints* (†) are also satisfied. Conversely, let x' be a global optimum of Eq. (22). By replacing z_{uv} with $x_u - x_v$ in the objective (which we can do since x' satisfies the linearization constraints (‡) in Eq. (22)), we see that x' is also a global optimum of Eq. (8), which is therefore a solution of Eq. (1). □

In fact, the proof of Prop. 3.5 also holds without the cycle constraints, which implies that the following formulation is also correct:

$$\left.\begin{array}{lrcll} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ z\in[-\mathbf{d},\mathbf{d}]^{mK}}} & \sum\limits_{\{u,v\}\in E}(\|z_{uv}\|_2^2 & - & d_{uv}^2)^2 & \\ \forall\{u,v\}\in E & x_u - x_v & = & z_{uv}. & \end{array}\right\} \tag{23}$$

We also note that Eq. (23) is a trivial reformulation of Eq. (8) by linearization of the terms $x_u - x_v$ by the variables z_{uv}. The point of the cycle formulation is that the cycle constraints (†) tighten any relaxation of Eq. (23), leading to better performances with global NLP solvers based on spatial Branch-and-Bound (sBB) [14]. The performance of local NLP solvers deployed on Eq. (23) is impacted less clearly by the cycle constraints: for this reason, Eq. (23) deserves more in-depth study w.r.t. local NLP optimization.

We note that Eq. (20), (22), and (23) are all NLPs involving polynomials of degree 4. They can easily be reformulated based on the system (resp. push-and-pull) formulation, yielding nonconvex QCQPs with nonconvex (resp. convex) quadratic constraints. We write these reformulations for (23): the cycle constraints (†) may be added to Eq. (24)-(26) to yield QCQP reformulations analogous to (22):

$$\left.\begin{array}{lrcl} \min\limits_{\substack{x\in\mathbb{R}^{nK},s\in\mathbb{R}^m\\ z\in[-\mathbf{d},\mathbf{d}]^{mK}}} & \sum\limits_{\{u,v\}\in E} s_{uv}^2 & & \\ \forall\{u,v\}\in E & \|z_{uv}\|_2^2 & = & d_{uv}^2+s_{uv} \\ \forall\{u,v\}\in E & x_u - x_v & = & z_{uv} \end{array}\right\} \tag{24}$$

$$\left.\begin{array}{lrcl} \max\limits_{\substack{x\in\mathbb{R}^{nK}\\ z\in[-\mathbf{d},\mathbf{d}]^{mK}}} & \sum\limits_{\{u,v\}\in E} \| & z_{uv} & \|_2^2 \\ \forall\{u,v\}\in E & \|z_{uv}\|_2^2 & \leq & d_{uv}^2 \\ \forall\{u,v\}\in E & x_u - x_v & = & z_{uv}. \end{array}\right\} \tag{25}$$

Eq. (24)-(25) correspond to the ℓ_2-norm error. Another exact reformulation of Eq. (24) can be written for the ℓ_1-norm, similarly to Eq. (14):

$$\left.\begin{array}{lrcl} \min\limits_{\substack{x\in\mathbb{R}^{nK},s\in\mathbb{R}^{2m}\\ z\in[-\mathbf{d},\mathbf{d}]^{mK}}} & \sum\limits_{\{u,v\}\in E}(s_{uv}^+ & + & s_{uv}^-) \\ \forall\{u,v\}\in E & \|z_{uv}\|_2^2 & = & d_{uv}^2+s_{uv}^+-s_{uv}^- \\ \forall\{u,v\}\in E & x_u - x_v & = & z_{uv}. \end{array}\right\} \tag{26}$$

3.5.1 The unassigned cycle formulation

The cycle formulations that include the cycle constraints (†) involve the knowledge of the underlying graph, which is not available as part of the input in the UDGP. Therefore, we cannot derive unassigned formulations from Eq. (20)

and (22). We can derive the unassigned versions of the simplified cycle formulations without cycle constraints (†), namely Eq. (23)-(26). We limit our treatment to the one derived from Eq. (26) because it yields a MIQCQP instead of a general MINLP, as explained in Sect. 3.3.1.

$$\left.\begin{array}{rrcl} \min\limits_{\substack{x\in\mathbb{R}^{nK}\\ s\in\mathbb{R}^{2m},y\in\mathscr{A}}} & \sum\limits_{\substack{\ell\leq m\\ i<j\leq n}} (s^+_\ell + s^-_\ell) & & \\ \forall \ell \leq m, i<j\leq n & -s^-_{uv} - M(1-y_{ij\ell}) & \leq & \|z_{ij}\|_2^2 - \delta_\ell^2 \leq s^+_{uv} + M(1-y_{ij\ell}) \\ \forall i<j\leq n & x_i - x_j & = & z_{ij}. \end{array}\right\} \tag{27}$$

Eq. (27) is a MIQCQP that can be solved using a global MIQCQP solver (possibly used locally). To the best of our knowledge, this formulation is new.

4 Matrix relaxations and approximations

In Sect. 3 we presented eleven MP formulations for the DGP and six for the UDGP. Among the DGP formulations, seven are (MI)QC(Q)P. Among the UDGP ones, three are (MI)QC(Q)P. The interest of limiting the polynomial degree to 2 is that one can directly derive SDP relaxations of the original formulation [19, 92, 57]. From these, one can then derive further linear relaxations and approximations by means of Diagonally Dominant Programming (DDP) [33], i.e. linear programming over the primal and dual cones of diagonally dominant (DD) matrices, which is in fact a subclass of LP — for which there exist extremely fast solvers. The application of SDP/DDP to the UDGP yields mixed-integer versions of the SDP/DDP formulations of the DGP. Since DDP $\subset$ LP, we obtain Mixed-Integer SDP (MISDP) relaxations, and MILP relaxations and approximations for the UDGP.

We shall limit the application of DDP reformulations to those formulations of Sect. 3 where the quadratic terms only involve the realization variables x. These are: the ℓ_1-norm system formulations Eq. (14) and (16), and the push-and-pull/pull-and-push formulations Eq. (17)-(18).

The basic reformulation steps to obtain (MI)SDP relaxations and (MI)LP relaxations and approximations is the same, as it applies term-wise to the expression $\|x_u - x_v\|_2^2$ and to the implicit constraints.

4.1 Constructing the SDP relaxation

Consider any DGP formulation in Sect. 3, and in particular the term $\|x_u - x_v\|_2^2$ for any $\{u,v\} \in E$. We have:

$$\begin{array}{rcl} \|x_u - x_v\|_2^2 & = & \|x_u\|_2^2 + \|x_v\|_2^2 - 2\langle x_u, x_v\rangle \\ & = & \langle x_u, x_u\rangle + \langle x_v, x_v\rangle - 2\langle x_u, x_v\rangle \\ & = & X_{uu} + X_{vv} - 2X_{uv} \end{array} \tag{28}$$

by linearization of any term $\langle x_t, x_w\rangle$ (for $t,w \in V$) with the additional variable X_{tw}. There may be up to n^2 linearization variables organized in a symmetric $n\times n$ matrix X. We can now replace the nonlinear term $\|x_u - x_v\|_2^2$ by the linear term $X_{uu} + X_{vv} - 2X_{uv}$ in any DGP formulation, then add the defining constraint matrix

$$X = xx^\top. \tag{29}$$

For any of the above DGP formulations of Sect. 3, the symbolic procedure just described provides an exact reformulation. We note that such reformulations are not very convenient to solve, as satisfying Eq. (29) is a difficult task for most solvers.

Now we rewrite Eq. (29) as $X - xx^\top = 0$, then we relax this to $X - xx^\top \succeq 0$, which, using the Schur complement, reads:

$$\begin{pmatrix} \mathbf{1}_K & x^\top \\ x & X \end{pmatrix} \succeq 0. \tag{30}$$

If every occurrence of x was eliminated by the linearization process, we can simplify Eq. (30) to

$$X \succeq 0. \tag{31}$$

SDP formulations are tractable cNLPs the nonlinearity of which is exclusively in the implicit constraint $X \succeq 0$. SDPs can be solved using an SDP solver, which runs in polynomial time up to any desired $\varepsilon > 0$ precision. The issue is practical, though: there are currently no SDP solvers to address SDPs with millions of variables and constraints, which is the current situation for LP solvers.

The SDP relaxation of the DGP is a formulation that also describes the EUCLIDEAN DISTANCE MATRIX COMPLETION PROBLEM (EDMCP), which is equivalent to the POSITIVE SEMI-DEFINITE MATRIX COMPLETION PROBLEM (PSDMCP). Both are discussed extensively in [60]. Computational experiments on such formulations have been obtained since the late 1990's [5], mostly by using ad-hoc codes implementing primal-dual algorithms.

4.2 The DDP restriction

The practical inadequacy of current SDP solvers motivates the search for inner and outer polyhedral cones to approximate the SDP cone. In this section we look at the cone of DD matrices. By Gershgorin's theorem [41], every DD matrix is also positive semidefinite (PSD), while the converse does not hold. Thus, if we replace the PSD constraint $X \succeq 0$ with "X is a DD matrix" we obtain an inner approximation of an SDP formulation. There remain two questions: (i) how can we describe the DD cone explicitly, and (ii) how does the solution of a DDP help us solve the DGP?

The first question is easiest: an $n \times n$ symmetric matrix X is DD if it satisfies:

$$\forall i \leq n \quad \sum_{\substack{j \leq n \\ j \neq i}} |X_{ij}| \leq X_{ii}. \tag{32}$$

Eq. (32) is a piecewise-linear (hence nonlinear) constraint. But a linear description exists for it [3, Thm. 3.9], based on linearizing the nonlinear term $|X_{ij}|$ by the components of a new matrix variable T:

$$\forall i \leq n \quad \sum_{\substack{j \leq n \\ j \neq i}} T_{ij} \leq X_{ii} \tag{33}$$

$$-T \leq X \leq T. \tag{34}$$

We let $\mathbf{D}_n = \{X \in \mathbf{S}_n; \mid \exists T \text{ Eq. } (33)-(34)\}$, where $\mathbf{S}_n$ is the set of all $n \times n$ symmetric matrices, be the linear description of the DD cone. The DDP corresponding to a given SDP can then be derived by replacing $X \succeq 0$ with $X \in \mathbf{D}_n$.

The second question depends on the output. Since $\mathbf{D}_n \subsetneq \mathbf{S}_n^+ = \{X \in \mathbf{S}_n \mid X \succeq 0\}$, there may be feasible SDPs where the corresponding DDP is infeasible: we can sometimes help this by relaxing some of the explicit constraints. However, if the DDP is feasible, we obtain a PSD solution matrix X', which provides an interesting solution, since it can be factored (we shall see the significance of this below). On the other hand, although the DDP is a restriction of the corresponding SDP, we start from an SDP that is a relaxation of the original DGP formulation: we cannot directly infer any relationship between the original objective function value and the optimal objective value of the DDP.

DDP formulations are LPs, which can be solved with any LP solver. DDP was extensively investigated by Amir Ali Ahmadi and co-authors [91, 2, 1, 3]. These techniques were first applied to the DGP in [33].

4.3 The dual DDP relaxation

If C is a cone, its dual cone C^* is defined as $\{\psi \mid \forall \phi \in C \ \langle \phi, \psi \rangle \geq 0\}$. It turns out that the dual DD cone $\mathbf{D}_n^*$ of $\mathbf{D}_n$ is finitely generated by the matrices:

$$E_{ij}^{\pm} = (e_i \pm e_j)(e_i \pm e_j)^\top$$

for every $i, j \leq n$, where $e_1, \ldots, e_n$ is the standard basis of $\mathbb{R}^n$ [11]. In other words, we have

$$\mathbf{D}_n^* = \{X \in \mathbf{S}_n \mid \forall i, j \leq n \ \mathrm{tr}(XE_{ij}^{\pm}) \geq 0\}. \tag{35}$$

Equivalently, since each $E_{ij}^{\pm}$ is defined as the gram matrix of trivial linear combinations of basis vectors, we consider $\Delta = \{e_i \mid i \leq n\} \cup \{e_i \pm e_j \mid i < j \leq n\}$, and describe $\mathbf{D}_n^*$ as follows:

$$\mathbf{D}_n^* = \{X \in \mathbf{S}_n \mid \forall v \in \Delta \quad v^\top X v \geq 0\}. \tag{36}$$

By Eq. (35), we can easily compute an explicit form for the constraints $\mathsf{tr}(XE_{ij}^{\pm}) \geq 0$. We know that $E_{ii} = \mathsf{diag}(e_i)$ for all $i \leq n$, and that:

- E_{ij}^{+} has the single nonzero minor $\begin{pmatrix} 1_{ii} & 1_{ij} \\ 1_{ij} & 1_{jj} \end{pmatrix}$;
- E_{ij}^{-} has the single nonzero minor $\begin{pmatrix} 1_{ii} & -1_{ij} \\ -1_{ij} & 1_{jj} \end{pmatrix}$.

By inspection we have:

$$\begin{array}{rrcl} \forall i \leq n & \mathsf{tr}(XE_{ii}) & = & X_{ii} \\ \forall i,j \leq n & \mathsf{tr}(XE_{ij}^{+}) & = & X_{ii} + X_{jj} + 2X_{ij} \\ \forall i,j \leq n & \mathsf{tr}(XE_{ij}^{-}) & = & X_{ii} + X_{jj} - 2X_{ij}. \end{array}$$

Therefore, we can define $\mathbf{D}_n^*$ by explicit constraints as follows:

$$\mathbf{D}_n^* = \{X \in \mathbf{S}_n \mid \mathsf{diag}(X) \geq 0 \ \wedge \ \forall i < j \leq n \ X_{ii} + X_{jj} \pm 2X_{ij} \geq 0\}. \tag{37}$$

The representation of $\mathbf{D}_n^*$ in Eq. (36) helps us prove that the dual DDP cone is an outer approximation of the PSD cone: by Eq. (36) we have that $X \in \mathbf{D}_n^*$ if $v^\top X v \geq 0$ for $v \in \Delta \subset \mathbb{R}^n$. Since PSD matrices are all and those for which $v^\top X v$ for all $v \in \mathbb{R}^n$, the inclusion $\mathbf{S}_n^+ \subsetneq \mathbf{D}_n^*$ follows.

Dual DDP formulations belong to the LP class: they can therefore be solved by any LP solver. As for DDP, these formulations were investigated by Ahmadi and co-authors (see the citations at the end of Sect. 4.2). Their application to the DGP can be found in [70, §6.1.4].

4.4 Matrix reformulations of the DGP and UDGP

Based on the sets $\mathbf{S}_n^+$, $\mathbf{D}_n$, and $\mathbf{D}_n^*$, we can define SDP relaxations, DDP restrictions, and dual DDP relaxations for the QCQP and MIQCQP formulations in Sect. 3 where the quadratic terms only involve the x variables. Some of the DGP relaxations below have appeared in [33, 71, 70], while the rest is new. Some of the UDGP relaxations have been used in [72].

4.4.1 From the system formulation

From the DGP system formulation in Eq. (14), for all $\mathbf{X} \in \{\mathbf{S}_n^+, \mathbf{D}_n, \mathbf{D}_n^*\}$ we derive the following DGP reformulations:

$$\left.\begin{array}{rrcl} \min\limits_{\substack{X \in \mathbf{X} \\ s \in \mathbb{R}^{2m}}} & \sum\limits_{\{u,v\} \in E} (s_{uv}^{+} & + & s_{uv}^{-}) \\ \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = & d_{uv}^2 + s_{uv}^{+} - s_{uv}^{-}. \end{array}\right\} \tag{38}$$

We remark that Eq. (38) (as well as all the formulations in this section) describes *three* different formulations depending on the symbol $\mathbf{X}$ that ranges in the PSD cone $\mathbf{S}_n^+$, the DD cone $\mathbf{D}_n$, the dual DD cone $\mathbf{D}_n^*$: the first is an SDP relaxation, the second an inner LP approximation of the SDP, and the third an outer LP relaxation of the SDP. Thus, the formulations in Sect. 4.4 are actually *meta-formulations*: they become formulations only after fixing the meaning of the symbol $\mathbf{X}$.

From the UDGP system formulation in Eq. (16), for all $\mathbf{X} \in \{\mathbf{S}_n^+, \mathbf{D}_n, \mathbf{D}_n^*\}$ we derive the following UDGP reformulations:

$$\left.\begin{array}{rrcl} \min\limits_{\substack{X \in \mathbf{X}, y \in \mathscr{A} \\ s \in \mathbb{R}^{2m}}} & \sum\limits_{\substack{\ell \leq m \\ i<j \leq n}} (s_\ell^{+} + s_\ell^{-}) & & \\ \forall \left\{ \begin{array}{l} \ell \leq m \\ i < j \leq n \end{array} \right. & -s_{uv}^{-} - M(1-y_{ij\ell}) \leq & X_{ii} + X_{jj} - 2X_{ij} - \delta_\ell^2 & \leq s_{uv}^{+} + M(1-y_{ij\ell}). \end{array}\right\} \tag{39}$$

We recall that $\mathscr{A}$ describes the binary variables y and corresponding assignment constraints (see Sect. 3.2.1). The three formulations described in Eq. (39) are a MISDP (for $\mathbf{X} = \mathbf{S}_n^+$) and two MILPs (otherwise).

4.4.2 From push-and-pull formulations

From the DGP push-and-pull formulation in Eq. (17), for all $\mathbf{X} \in \{\mathbf{S}_n^+, \mathbf{D}_n, \mathbf{D}_n^*\}$ we derive the following DGP reformulations:

$$\left.\begin{array}{rll} \max\limits_{X \in \mathbf{X}} & \sum\limits_{\{u,v\} \in E} (X_{uu} + X_{vv} - 2X_{uv}) & \\ \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & \leq \quad d_{uv}^2. \end{array}\right\} \tag{40}$$

For the related pull-and-push formulation in Eq. (18) we derive:

$$\left.\begin{array}{rll} \min\limits_{X \in \mathbf{X}} & \sum\limits_{\{u,v\} \in E} (X_{uu} + X_{vv} - 2X_{uv}) & \\ \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & \geq \quad d_{uv}^2. \end{array}\right\} \tag{41}$$

Eq. (41) with $\mathbf{X} = \mathbf{D}_n$ is only motivation for the original formulation Eq. (18) (the constraints of which are concave, and therefore hard to satisfy): because of the potential feasibility issues of solving DDPs (i.e. the DDP might be infeasible even if the original SDP is feasible), Eq. (40)-(41) enlarge the feasible region both $\leq, \geq$ constraint senses: at least one of them must be feasible.

When $\mathbf{X} = \mathbf{S}_n$, we also propose a related formulation mentioned by Yinyu Ye in one of his course slides:

$$\left.\begin{array}{rll} \max\limits_{X \succeq 0} & \mathsf{tr}(X) & \\ \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} & = \quad d_{uv}^2. \end{array}\right\} \tag{42}$$

The constraints of Eq. (42) are an SDP relaxation of Eq. (1). If the original DGP instance is NO because it has no realization in $\mathbb{R}^K$, then Eq. (42) is feasible. On the other hand, if the original DGP instance is NO because the given distances cannot be realized in any dimension, then Eq. (42) is infeasible. Obviously, if the DGP instance is YES then Eq. (42) is also feasible.

For feasible cases of Eq. (42), using equations instead of inequalities (as in Eq. (40)-(41)) produces a tighter relaxation. The objective function heuristically attempts to reduce the rank of the solution, as

$$\mathsf{tr}(X) = \mathsf{tr}(P\Lambda P^\top) = \mathsf{tr}(PP^\top \Lambda) = \mathsf{tr}(\Lambda) = \sum_{u \in V} \lambda_u,$$

where $P\Lambda P^\top$ is an eigendecomposition of X, and minimizing the sum of eigenvalues should help decrease the rank of X (we shall see why this is convenient in Sect. 4.5).

We do not derive UDGP versions from any of the formulations in this section, since it would yield a product of variables (y and X) in the objective function, resulting in nonlinear programs with cone constraints (which are impractical to solve).

4.5 DGP post-processing

In this section we assume that the problem being solved is a DGP. From matrix formulations we do not obtain a realization $x' \in \mathbb{R}^{nK}$ as output, but a symmetric matrix $X' \in \mathbb{R}^{n \times n}$. If X' is PSD, as would happen for SDP and DDP, then X' is a Gram matrix of a realization, so it can be written as $X' = \xi\xi^\top$, where $\xi \in \mathbb{R}^{n \times n}$: in general, ξ can be interpreted as a realization in $\mathbb{R}^n$ instead of in $\mathbb{R}^K$. If X is the solution of a dual DDP, then $\xi \in \mathbb{C}^n$ in general. In both cases, in order to find the "closest" realization in K dimensions, we must reduce the rank of the realization points (in $\mathbb{R}^n$ or $\mathbb{C}^n$) to obtain a realization matrix in $x' \in \mathbb{R}^{nK}$.

In our past work we have considered two dimensionality reduction methodologies: Principal Component Analysis (PCA) [52, 114] and Barvinok's naive algorithm extended to K dimensions [12, 86]. PCA can be applied to the rows of ξ (the point vectors) to reduce them to the K principal components (or possibly even fewer if X', as the solution of a dual DDP, fails to be a PSD matrix). Barvinok's naive algorithm can only be applied to solutions of SDPs and DDPs (with dual DDPs one may simply hope for the best). PCA produces the K-dimensional realization x' closest to the solution of the SDP or DDP. Barvinok's naive algorithm produces, with arbitrarily high probability, a K-dimensional

realization x' that is "reasonably close to" (or just "not too far from") a realization of the original DGP. Comparative computational experiments between these two rank reduction methods can be found in [86].

Once a matrix $x' \in \mathbb{R}^{nK}$ has been computed, it can be *refined*, i.e. its realization error can be reduced, by using x' as a starting point on any one of the DGP formulations of NLP/QCQP type in Sect. 3 solved by a local NLP/QCQP solver. This yields an approximate realization x^* of the DGP.

4.5.1 A solution process for DGPs

In summary, we propose the following process for solving DGP instances too difficult or too large to be dealt with by global NLP/QCQP solvers:

1. Solve an SDP/DDP/dual DDP reformulation of the DGP instance (this should be reasonably fast), obtain an $n \times n$ symmetric matrix solution X, and factor it as $\xi\xi^\top$.
2. Reduce the rank of ξ to an $n \times K$ realization matrix x' using various dimensionality reduction methods.
3. Improve the quality of the realization x' by using it as a starting point in a local NLP solver, which will yield a good-quality realization x^* of the given DGP instance.

4.6 UDGP post-processing

Solving UDGPs poses the problem of graph reconstruction, as discussed in Sect. 3.1. Solving UDGPs globally is only possible for tiny instances, in general. Thus, the MISDP/MILP matrix reformulations of the UDGP in Eq. (39) are the only practically viable possibility to handle medium to large-sized UDGP instances. High-quality global MILP solvers can be configured to find all (or many) solutions during the search. In general, one will find solutions (X', y') where X' is the matrix solution and y' encodes the assignment α.

The first post-processing task to carry out is the reconstruction of the DGP graph G_α (see Sect. 3.1). The second post-processing task is to work out a realization $x \in \mathbb{R}^{nK}$ of the reconstructed graph. There are two possibilities: either one considers the matrix solution X', or one discards it. Paired with the graph G_α, the solution X' can be used as described in Sect. 4.5, i.e. rank reduction followed by refinement. If X' is discarded, the graph G_α defines a DGP, which can be solved using the process given in Sect. 4.5.1. The second possibility was adopted in [72], since the first gave poor quality realizations.

4.7 Two remarks over concave constraints

In this section we provide answers to two issues that arose during talks at the conference that these proceedings book relate to. Both are relative to the concave constraint that imposes a lower bound to a square Euclidean norm. While both answers are probably "folklore", to the best of our knowledge they have never appeared in print.

In the context of the (assigned) DGP and matrix formulation, we ignore pairs u, v of vertices that are not edges in the graph. A popular way to treat them is to add a "greater than or equal" constraints with respect to some upper distance threshold $\bar{d}$, whenever one is known, i.e.

$$\forall \{u, v\} \notin E \quad \|x_u - x_v\|_2^2 \geq \bar{d} \tag{43}$$

in the case of vector formulations, or

$$X_{uu} + X_{vv} - 2X_{uv} \geq \bar{d}^2 \tag{44}$$

in the case of matrix formulations. This might help avoid the typical "overclustering" effect of realization points around the origin for weakly connected vertices. We do not consider these constraints for two reasons: the first, and foremost, is that it assumes that *all* distances shorter than $\bar{d}$ have been measured, which may not be the case for proteins (and other molecules) — and even a single wrong constraint of this type is likely to change the resulting

structure considerably. The second reason is that Eq. (43) is a concave constraint, which is hard to enforce for many solvers (the linearized version in Eq. (44), by contrast, is simply a linear constraint).

It is well known that strict constraints cannot appear in MP formulations, since they yield open sets: and optima over open sets may not exist. Typically, with an upper bound $\bar{d}$ on distance values, mathematicians will want to impose

$$\forall \{u,v\} \in E \quad \|x_u - x_v\|_2^2 \leq \bar{d}$$

and

$$\forall \{u,v\} \notin E \quad \|x_u - x_v\|_2^2 > \bar{d},$$

which might make optima of any MP formulation including Eq. (4.7) in its contraints non-existent. One possible reformulation of Eq. (4.7) models set openness by formulation unboundedness. We introduce an auxiliary variable $t \geq 0$ in the formulation, and rewrite Eq. (4.7) as:

$$\forall \{u,v\} \notin E \quad \|x_u - x_v\|_2^2 \geq \bar{d} + e^{-t}. \tag{45}$$

Unboundedness in t is only possible if $\|x_u - x_v\|_2^2$ is forced to be exactly equal to $\bar{d}$, which can only happen if the bound $\bar{d}$ is too small. While Eq. (45) is nonconvex in general, its linearized version

$$\forall \{u,v\} \notin E \quad X_{uu} + X_{vv} - 2X_{uv} \geq \bar{d} + e^{-t} \tag{46}$$

is a convex constraint, which is representable using an exponential cone. If the rest of the formulation is also a conic program (such as e.g. SDP or DDP formulations), the whole problem can be solved efficiently using a conic programming solver.

5 Computational evaluation

In this section we attempt to answer the following questions.

1. Up to what size can we solve DGP/UDGP instances to global optimality in an acceptable time on a laptop?
2. For DGP/UDGP instances of various types, is it better to run a simple stochastic matheuristics (see below) around an exact formulation solved locally, or use the matrix formulation based solution process?
3. Is MP practically useful for solving realistic DGP/UDGP instances?

We shall answer the first question by finding size thresholds beyond which the exponential nature of global optimization algorithms on our MP formulations becomes limiting.

The second question deserves an explanation. A *matheuristic* is a heuristic algorithm (i.e. that does not provide exactness guarantees) based on a MP formulation. A *stochastic* algorithm uses an element of randomness during its execution. The simplest stochastic matheuristic is MultiStart (MS) [106, 58], shown in Alg. 1. We shall answer the

Algorithm 1 MultiStart

```
initialize x̄ to NaN
while resource limit not reached do
    sample random starting point x'
    solve a MP formulation locally from x', obtain x̄
    if x̄ improves on previous optimum then
        update x* ← x̄
    end if
    return x*
end while
```

second question by means of a comparison between MS and the process described in Sect. 4.5.1 (for DGP) and 4.6 (for UDGP).

We propose to answer the third question by attempting to reconstruct protein shapes from distance data (both with and without the graph).

Finally, we note that centroid constraints (Eq. (7)) have been added to all of our formulations. Although the results in [86] report a computational advantage slightly in favour of Barvinok's naive algorithm, we have chosen to use PCA as a dimensional reduction method on matrix formulation solutions, simply because it is better known.

5.1 Instances

DGP instances are organized in three families.

- A set $\mathscr{R}$ of 28 random biconnected graphs of given sparsity and size, based on a vertex set of points in the Euclidean plane $\mathbb{R}^2$, with edges generated by means of an Erdős-Renyi process over a starting Hamiltonian cycle and weighted by Euclidean distance between the corresponding points, to be realized in $K = 2$ (all of these instances are YES by construction). The set $\mathscr{R}$ is used to answer the first question, i.e. to what instance size can we solve DGPs and UDGPs to guaranteed optimality?
- A collection $\mathscr{G}$ of 309 graphs of different types and sizes, some weighted some not (i.e., with unit weight), to be realized in $K = 2$ (most of the randomly weighted versions of these instances are generically NO; some of the others are YES). The set $\mathscr{G}$ is used to answer the second question, i.e. is it better to solve DGP/UDGP formulations locally withing a MS algorithm (Alg. 1), or use matrix formulation based solution process (Sect. 4.4)?
- A set $\mathscr{P}$ of protein graphs simulating NOESY experiments with known covalent bonds and angles: in other words a set of disk graphs (of radius 5.5Å) on a vertex set of points in $\mathbb{R}^3$ (all of these instances are YES by definition). We use this set to answer the third question: are MP formulations actually useful in practice in regard to the DGP/UDGP?

5.1.1 Random euclidean graphs

More precisely, the set $\mathscr{R}$ consists of 28 random biconnected graph generated with the Erdős-Renyi model for each vertex set size $n \in \{5, 8, 10, 12, 15, 18, 20\}$, and for each sparsity parameter $p \in \{0.4, 0.8, 0.9, 0.95\}$.

5.1.2 Different graph types

The set $\mathscr{G}$ is composed as follows:

1. 18 almost k-regular graphs on n vertices (9 randomly weighted and 9 unweighted),
2. 18 random graphs on n vertices with edge generation probability p (9 randomly weighted and 9 unweighted),
3. 18 bipartite graphs on $n+n$ vertices with edge generation probability p (9 randomly weighted and 9 unweighted),
4. 18 tripartite graphs on $n+n+n$ vertices with edge generation probability p (9 weighted and 9 unweighted),
5. 10 square meshes with n^2 vertices (5 weighted and 5 unweighted),
6. 10 torus meshes with n^2 vertices as a folded-up square mesh (5 weighted and 5 unweighted),
7. 10 triangular meshes with n vertices per side (5 weighted and 5 unweighted),
8. 126 clustered graphs with k clusters on n vertices with intra-cluster edge generation probability p and inter-cluster edge generation probability q (63 weighted and 63 unweighted),
9. 6 power law graphs on n vertices where the degree of vertex i is $\lceil n\alpha i^{-\tau} \rceil$ with $\alpha \in (0,1), \tau > 0$ (3 weighted and 3 unweighted),
10. 18 chain of k-cliques on n vertices (9 weighted and 9 unweighted),
11. 10 chain of triangles on n vertices (5 weighted and 5 unweighted),

12. 18 DMDGP [81] on n vertices with k contiguous adjacent predecessors (9 weighted and 9 unweighted),
13. 5 Beeker-Glusa graphs [13]: chains of triangles with specific edge costs (weighted only),
14. 6 local graphs on n vertices with edge threshold t: vertices are n points in the plane, edges exist if Euclidean distance between two points shorter than t (weighted only),
15. 18 norm graphs on n vertices chosen as points in the plane with edge generation probability p, edges are weighted by ℓ_1 and ℓ_∞ distances between points (weighted only).

5.1.3 Protein instances

The set $\mathscr{P}$ is composed of protein graphs constructed from the Protein Data Bank (PDB) [16] in such a way as to roughly mimick the output of a NOESY experiment on an NMR machine. We selected a set of proteins (and pieces thereof) that cover a reasonable spectrum size, from small to reasonably large (see Table 1). For each of these we extracted the first available realization in the PDB and computed all of the inter-atomic distances, then we discarded those with length larger than 5.5Å. The foremost difference between these instances and those actually obtained from NOESY experiments followed by distance assignment processes is that there are no mis-assigned edges in the protein graphs in $\mathscr{P}$. We note that these graphs are sometimes disconnected. For example, tiny consists of a connected

Name	$\|V\|$	$\|E\|$
tiny	38	335
1guu-1	150	959
1guu-4000	150	968
C0030pkl	198	3247
1PPT	303	3102
1guu	427	955
100d	491	5741
3al1	681	17417
1hpv	1633	18512
il2	2098	45251
1tii	5691	69800

Table 1: Vertex and edge set sizes of protein instances in the class $\mathscr{P}$.

component of 37 atoms and a single disconnected atom (an isolated vertex in the graph); and 1guu has 277 isolated vertices (in fact it actually has only 150 connected atoms, like its kin instances 1guu-1 and 1guu-4000). We chose to keep such occurrences because graphs occurring from applications are often atypical with respect to the usual assumptions of connectedness: and benchmarks on these instances are supposed to verify the practical usefulness of our formulations.

5.1.4 UDGP versions of $\mathscr{R}$, $\mathscr{G}$, $\mathscr{P}$

All of our UDGP instances were derived from DGP ones by discarding the graph structure: we only keep K, n, and the list L of distance values from the graph edges.

5.2 Hardware and software

All tests have been carried out on an Intel architecture server with: 2 Intel Xeon Platinum 8362 CPUs at 2.80GHz, each with 32 cores with hyperthreading, for a total of 128 cores; and 2TB RAM.

The software system consists of a set of coordinated Python 3 [112] scripts, bash scripts, and AMPL [40] code. This system calls LP, MILP, SDP, (local and global) NLP, MISDP, and MINLP solvers as follows: CPLEX 22.1.1 [53] for LP, Gurobi 10.0.1 [48] for MILP, nonconvex NLP, and MINLP, IPOPT 3.4.11 [25] for solving NLP locally, SCS

3.2.3 [101] for SDP, Pajarito [89] for MISDP (called from a Julia [17] script, and using Gurobi and Mosek [95] as subsolver). The MILP/MINLP solver is deployed with a CPU time limit of 1800s. The LP, SDP, and MISDP solvers are used without specific configurations. The local NLP solver is used within a MS algorithm limited to 5 iterations unless specified otherwise. All solvers were run with their default configurations, aside from a more frequent display and the time limit set to 1800s.

The scripts of our software system were run in parallel by means of the Slurm Workload Manager 20.11.9. The number of CPUs employed in each run depends on the deployed solver (which is the most time-consuming task of the script): among the above solvers only CPLEX and Gurobi use multiple CPUs by default, and they make their own decisions on how many CPUs they employ. The rest of the solvers run on one CPU only (aside of course from solvers that use CPLEX or Gurobi as subsolver). The CPU times we record in our experiments are wall-clock times recorded by the Python scripts launching the solvers.

We remark that there is an option for providing IPOPT with a termination based on CPU time limit, but the verification for this type of termination is only carried out in a certain outer phase of the algorithm. This means that IPOPT can (and often does) exceed the given CPU time limit by arbitrary amounts. We therefore decided to refrain from imposing a time limit on IPOPT in our computational experiments. This explains the fact that our reported CPU times may dramatically exceed the default CPU time limit.

5.3 DGP tests and results

We have tested the following formulations:

1. cycle (Eq. (22)),
2. cyclesimple (Eq. (23)),
3. cycpushpull (Eq. (25) with cycle constraints (†)),
4. cycsimplepushpull (Eq. (25)),
5. cycsimplesys1 (Eq. (26)),
6. cycsimplesys2 (Eq. (24)),
7. cycsys1 (Eq. (26) with (†)),
8. cycsys2 (Eq. (24) with (†)),
9. pushpull (Eq. (17)),
10. pullpush (Eq. (18)),
11. quartic (Eq. (8)),
12. system1 (Eq. (14)),
13. system2 (Eq. (13)),

both by themselves, and used as refinement steps to:

1. an SDP matrix formulation, i.e. a variant of Eq. (42) with a modified objective

$$\sum_{\{u,v\}\in E} (X_{uu} + X_{vv} - 2X_{uv}) + 0.1\,\mathsf{tr}(X),$$

   which was heuristically found to perform slightly better on protein instances[4]: we note that Eq. (42) is infeasible on NO instances of the DGP where the graph cannot be realized in any dimension;

2. the corresponding DDP and dualDDP polyhedral approximations that replace the PSD cone $\mathbf{S}_n^+$ with the DDP and dualDDP cones $\mathbf{D}_n, \mathbf{D}_n^*$; we also relaxed the equations $X_{uu} + X_{vv} - 2X_{uv} = d_{uv}^2$ to $\geq$-inequalities in the DDP to prevent an excessive number of infeasibilities.

[4] With the constraints of Eq. (42), the term $\sum_{\{u,v\}\in E}(X_{uu} + X_{vv} - 2X_{uv})$ in the objective is actually equal to the constant $\sum_{uv} d_{uv}^2$, and so it should be irrelevant. But not every SDP or local NLP solver always ensures feasibility at every step: this depends on the reformulations and algorithms it implements. We think that this fact might give this formulation the slight empirical advantage we observed in previously conducted experiments.

We present our computational results grouped in various ways:

- by approximate vertex set size (graphs grouped by $|V|$ closest to multiples of 10);
- by approximate edge set size (graphs grouped by $|E|$ closest to multiples of 50);
- by approximate edge density $\frac{|E|}{V(V-1)/2}$ (graphs grouped by density values closest to multiples of $1/10$);
- by graph type (only for the class $\mathscr{G}$), where randomly weighted graph types have their name prefixed by 'W');
- by formulation type.

The accuracy of our DGP results was described by *mean distance error* (mde), *largest distance error* (lde), and the algorithmic performance by seconds of CPU time. For a realization $x \in \mathbb{R}^{nK}$ of a DGP instance (K,G) where $G = (V,E,d)$, we have:

$$\mathsf{mde}(x) = \sum_{\{u,v\}\in E} \left| \|x_u - x_v\|_2^2 - d_{uv}^2 \right| \tag{47}$$

$$\mathsf{lde}(x) = \max_{\{u,v\}\in E} \left| \|x_u - x_v\|_2^2 - d_{uv}^2 \right|. \tag{48}$$

5.3.1 The Euclidean graph collection $\mathscr{R}$

The point of this graph collection (Sect. 5.1.1) is to provide a testbed of small graphs (all of which are YES instances of the DGP) for answering our first question concerning the DGP: how far can we go up in size and yet obtain a DGP realization with an algorithmic guarantee that the realization is precise, at least up to an $\varepsilon > 0$ tolerance and in a reasonable amount of time? We consider an optimality tolerance of 10^{-6}, and a "resonable amount of time" to mean 1800s of CPU time. We solve these instances with the Gurobi solver.

We only consider the exact formulations cycle, cyclesimple, cycpushpull, cycsimplepushpull, cycsimplesys1, cycsimplesys2, cycsys1, cycsys2, pullpush, pushpull, quartic, system1, system2 (see Sect. 5.3). We do not consider any of the matrix formulations (SDP, DDP, dual DDP) because none of them is exact.

We present average results grouped by approximate vertex cardinality, and its corresponding bar plot figure, in Table 2. The results are shown in terms of mde, lde, CPU time. These data show that guaranteed globally optimal

$\approx \lvert V \rvert$	mde	lde	CPU
10	0.0001	0.0003	192.63
20	0.3884	1.8223	1356.60

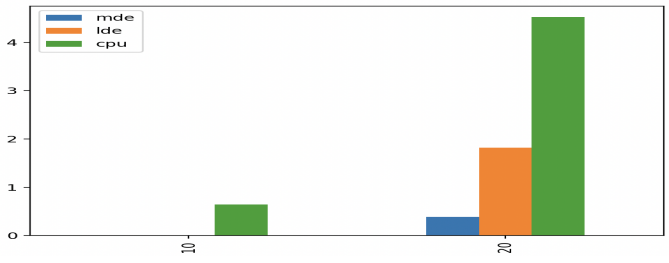


Table 2: Average results on approximate vertex cardinality and the corresponding bar plot for the graph class $\mathscr{R}$ (the CPU time column was scaled by 1/300).

solutions may only be found in 1800s of CPU time up to $|V| = 15$.

In Table 3 (left), we present average results grouped by formulation type. The corresponding bar plot figure is shown in Table 3 (right). These data show that guaranteed optimal solutions in 1800s are more likely to happen with as few edges as possible: very likely with 50 edges, and very unlikely with 200.

In Table 4 (left), we present average results grouped by edge density. The corresponding bar plot figure is shown in Table 4 (right). Guaranteed optimal solutions in 1800s are most likely with edge densities around 0.6-0.7.

$\approx \lvert E \rvert$	mde	lde	CPU
50	0.0085	0.0678	294.37
100	0.1711	0.9917	1162.36
150	0.3331	1.6304	1594.27
200	1.1886	4.9158	1794.13

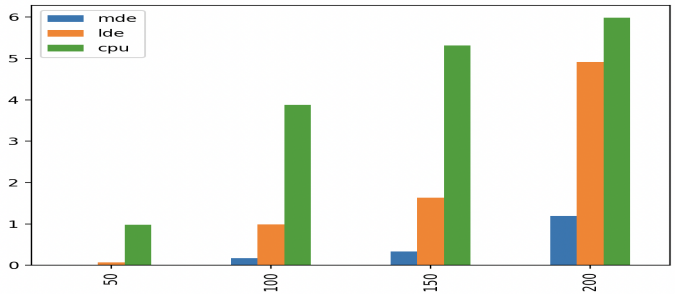


Table 3: Average results on approximate edge cardinality and the corresponding bar plot for the graph class $\mathscr{R}$ (the CPU time column was scaled by 1/300).

$\approx$ **density**	mde	lde	CPU
0.4	0.3945	2.3032	1532.25
0.5	0.3024	1.8276	1160.63
0.6	0.0001	0.0004	373.80
0.7	0.0000	0.0001	1062.62
0.8	0.3705	1.7659	1233.24
0.9	0.2539	0.9882	880.90
1.0	0.1654	0.7392	609.35

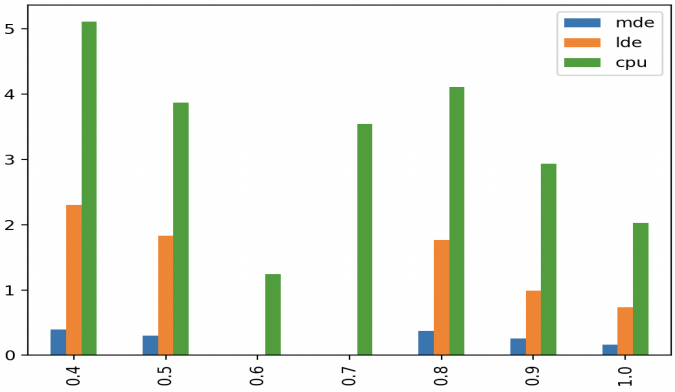


Table 4: Average results on approximate edge density and the corresponding bar plot for the graph class $\mathscr{R}$ (the CPU time column was scaled by 1/300).

formulation	mde	lde	CPU
cycle	0.0046	0.0197	806.88
cyclesimple	0.1074	0.4928	1042.94
cycpushpull	0.0007	0.0037	1256.00
cycsimplepushpull	0.0013	0.0067	1451.29
cycsimplesys1	0.0000	0.0002	276.27
cycsimplesys2	0.0861	0.3885	930.46
cycsys1	0.0000	0.0001	453.94
cycsys2	0.0447	0.3032	771.03
pullpush	2.2092	9.7720	668.97
pushpull	0.0001	0.0002	1353.12
quartic	0.2173	1.3483	808.64
system1	0.0000	0.0002	490.75
system2	0.2141	1.2033	840.52

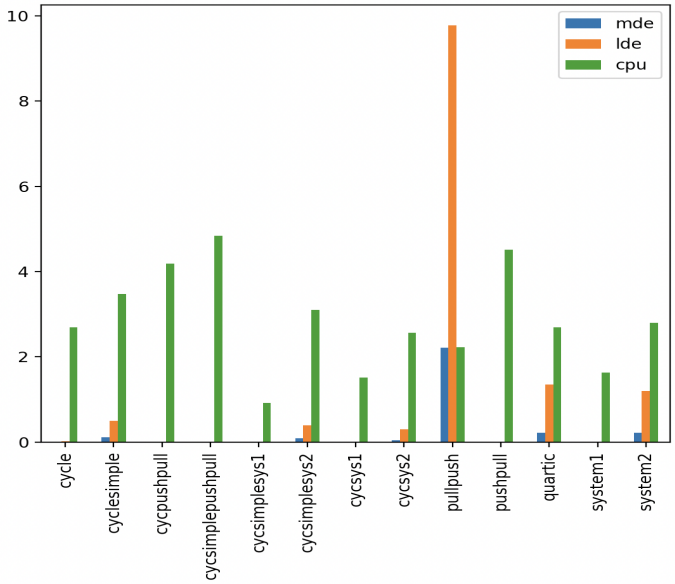


Table 5: Average results on formulation types for the graph class $\mathscr{R}$ (the CPU time column was scaled by 1/300).

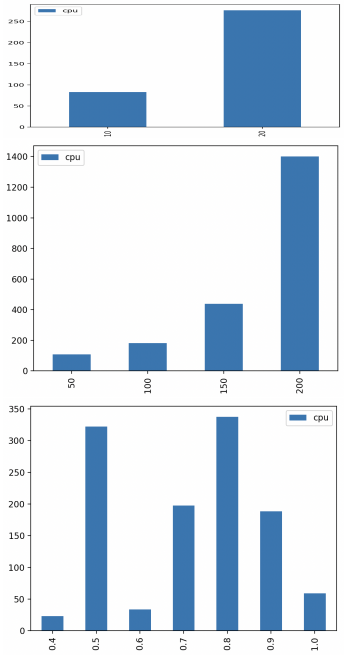
In Table 5 (left), we present average results grouped by formulation type. The corresponding bar plot figure is shown in Table 5 (right). The formulations that were able to yield guaranteed optimal solutions in 1800s to precision 10^{-4} on average were: cycsimplesys1, pushpull, system1. The formulations cycpushpull and cycsimplepushpull went up to precision 10^{-3} on average. We note that pullpush was the worst-performing formulation by far (this is consistent with the observation below Eq. (18)).

We now restrict the analysis to the cases where the global nonconvex NLP solver reached termination within the

$\approx \lvert V \rvert$	CPU
10	82.91
20	276.23

$\approx \lvert E \rvert$	CPU
50	108.30
100	181.33
150	438.50
200	1400.09

$\approx$ **dens.**	CPU
0.4	22.96
0.5	322.11
0.6	33.85
0.7	197.63
0.8	337.40
0.9	188.36
1.0	58.97



formulation	CPU
cycle	163.99
cyclesimple	30.93
cycpushpull	398.56
cycsimple-pushpull	369.86
cycsimplesys1	92.64
cycsimplesys2	59.92
cycsys1	83.10
cycsys2	199.12
pullpush	137.02
pushpull	403.17
quartic	64.25
system1	53.70
system2	120.11

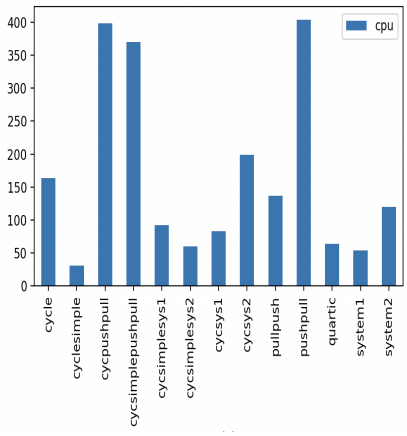


Table 6: Average CPU time for naturally terminating (instance,formulation) pairs in $\mathscr{R}$.

allotted time limit (1800s), which happened in 206 (instance, formulation) pairs. The average mde for these cases is 0.66×10^{-6}, the average lde is 0.0002, the average CPU time is $\approx$ 140s. All of these pairs have mde and lde between 0 and 0.0006, which we do not report, as they are essentially zero, and indicate a correct realization. The average CPU times grouped by vertex/edge cardinality, graph density, and formulation type are given in Table 6. While the picture does not change too much for $|V|$, $|E|$, and edge density, the best performing formulations are different in this test with respect to the previous test: the best performing formulation was cyclesimple, followed by system1, quartic, and cycsimplesys2.

5.3.2 The 309-graph collection $\mathscr{G}$

We use this graph collection (Sect. 5.1.2) to establish whether it is preferable to solve DGP instances with local NLP solvers on non-matrix DGP formulations within a MS algorithm, or to use the matrix formulation solution process. The test is such that we need only group results by formulation type.

In Table 7, we present average results grouped by formulation type. The corresponding bar plot figure is shown in Fig. 2. The best performing formulations in terms of solution quality are sdp_pca_χ with

$$\chi \in \{\text{quartic}, \text{cyclesimple}, \text{system2}\};$$

in terms of CPU time we have cyclesimple, quartic, system2, and dualddp_pca_χ with χ as above. An important remark is that the best formulations in terms of CPU time are also very good in terms of solution quality. Among these, six are matrix formulations of SDP and dualDDP types, and three are non-matrix. Our answer to the second question in relation to DGP is therefore that the matrix formulation process is generally better, but the refinement step is crucially important.

Since the instance family $\mathscr{G}$ contains many graph types, we also present average results grouped by formulation type in Table 8 (left), and the corresponding bar plot figure in Table 8 (right).

5.3.3 The protein graph collection $\mathscr{P}$

We attempt to reconstruct the shape of proteins from a partial set of inter-atomic distances with their adjacencies (Sect. 5.1.3) by using the methods that appear more promising from earlier DGP experiments (Sect. 5.3.1-5.3.2),

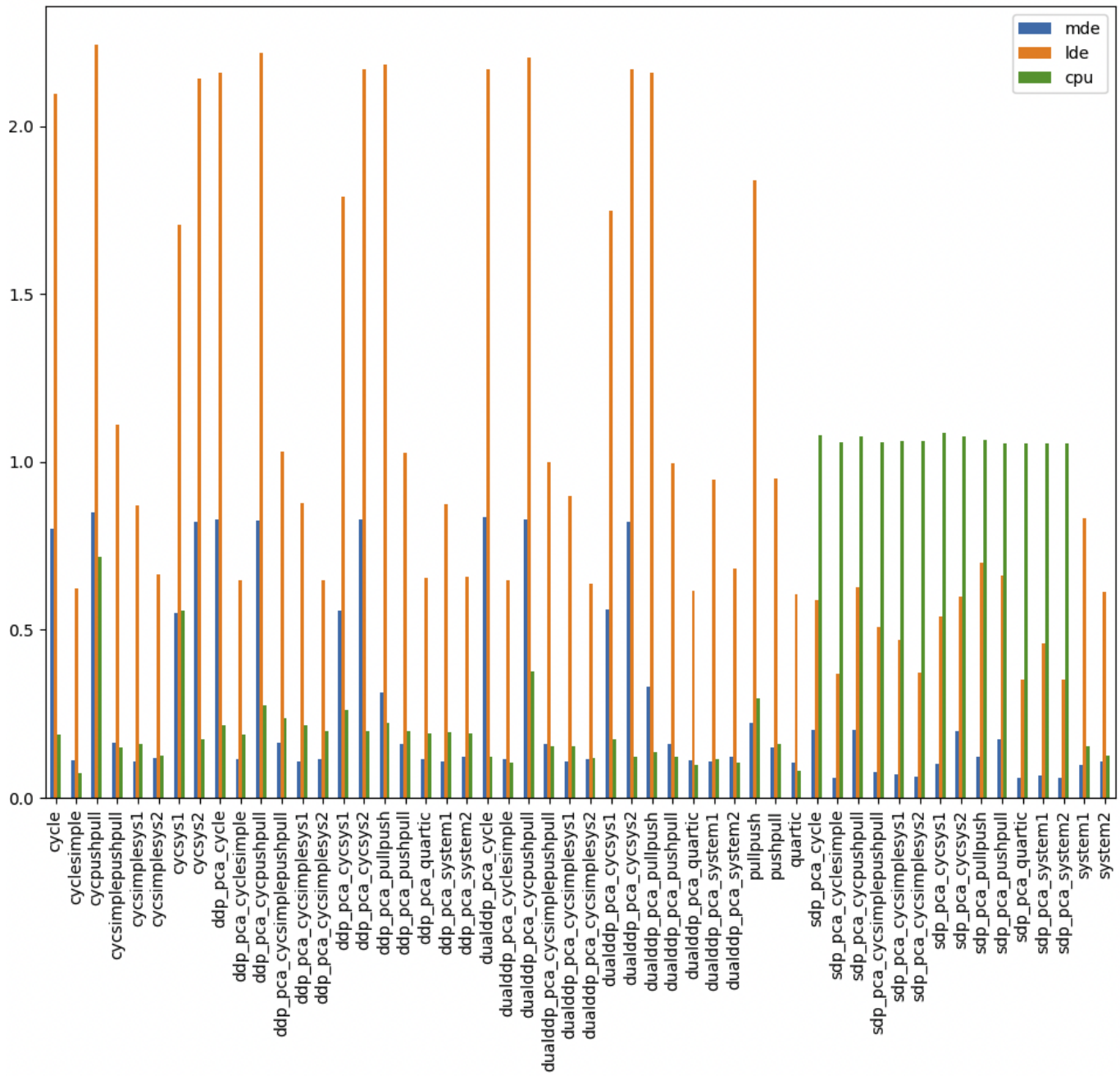


Figure 2: The bar plot for the graph class $\mathscr{G}$ corresponding to Table 7.

namely: the non-matrix formulations cyclesimple, quartic, system2 and the matrix formulations based on SDP and dual DDP followed by PCA, with refinement step carried out using the non-matrix formulations above. Since SDP solvers fail with larger instances (from 3al upwards, see Table 5.1.3), we replaced SDP by DDP in such cases. For non-matrix formulations we used IPOPT within the MS algorithm (Alg. 1) with a 10 iterations limit.

We first present results about the best formulation per protein instance. By "best" we mean the best trade-off between mde and lde. When no solution dominates the others over both measures the choice was made by the authors of this survey, based on their past experience.

The results grouped by formulation type (independently of the instance) are given in Table 10. The best formulation as regards solution quality (mde, lde) is system2, which is also the worst for CPU time (but CPU time is not a crucial measure for the purpose of finding the structure of proteins). The formulation quartic is the closest competitor. We note that, however, the variance of mde, lde over all of the tested formulations is small. The fastest formulation is dualddp_pca_quartic, with ddp_pca_quartic the closest competitor. The CPU time variance is small for all formu-

formulation	mde	lde	CPU
cycle	0.8001	2.0958	0.19
cyclesimple	0.1128	0.6248	0.07
cycpushpull	0.8492	2.2438	0.72
cycsimplepushpull	0.1635	1.1107	0.15
cycsimplesys1	0.1088	0.8713	0.16
cycsimplesys2	0.1190	0.6669	0.13
cycsys1	0.5517	1.7059	0.56
cycsys2	0.8206	2.1401	0.18
ddp_pca_cycle	0.8278	2.1597	0.22
ddp_pca_cyclesimple	0.1157	0.6470	0.19
ddp_pca_cycpushpull	0.8269	2.2188	0.28
ddp_pca_cycsimplepushpull	0.1633	1.0303	0.24
ddp_pca_cycsimplesys1	0.1090	0.8762	0.21
ddp_pca_cycsimplesys2	0.1154	0.6478	0.20
ddp_pca_cycsys1	0.5578	1.7894	0.26
ddp_pca_cycsys2	0.8273	2.1707	0.20
ddp_pca_pullpush	0.3141	2.1826	0.22
ddp_pca_pushpull	0.1621	1.0275	0.20
ddp_pca_quartic	0.1143	0.6536	0.19
ddp_pca_system1	0.1084	0.8753	0.20
ddp_pca_system2	0.1207	0.6601	0.19
dualddp_pca_cycle	0.8348	2.1682	0.12
dualddp_pca_cyclesimple	0.1141	0.6480	0.10
dualddp_pca_cycpushpull	0.8297	2.2055	0.38
dualddp_pca_cycsimplepushpull	0.1614	1.0001	0.15
dualddp_pca_cycsimplesys1	0.1083	0.8970	0.15
dualddp_pca_cycsimplesys2	0.1136	0.6361	0.12
dualddp_pca_cycsys1	0.5596	1.7468	0.17
dualddp_pca_cycsys2	0.8235	2.1685	0.12
dualddp_pca_pullpush	0.3314	2.1606	0.14
dualddp_pca_pushpull	0.1616	0.9969	0.12
dualddp_pca_quartic	0.1104	0.6177	0.10
dualddp_pca_system1	0.1083	0.9467	0.12
dualddp_pca_system2	0.1206	0.6822	0.10
pullpush	0.2235	1.8404	0.29
pushpull	0.1516	0.9501	0.16
quartic	0.1042	0.6053	0.08
sdp_pca_cycle	0.2006	0.5890	1.08
sdp_pca_cyclesimple	0.0607	0.3683	1.06
sdp_pca_cycpushpull	0.2027	0.6268	1.08
sdp_pca_cycsimplepushpull	0.0773	0.5090	1.06
sdp_pca_cycsimplesys1	0.0705	0.4720	1.06
sdp_pca_cycsimplesys2	0.0625	0.3718	1.06
sdp_pca_cycsys1	0.1021	0.5414	1.09
sdp_pca_cycsys2	0.2004	0.6002	1.08
sdp_pca_pullpush	0.1215	0.6993	1.07
sdp_pca_pushpull	0.1744	0.6633	1.06
sdp_pca_quartic	0.0593	0.3524	1.06
sdp_pca_system1	0.0663	0.4607	1.06
sdp_pca_system2	0.0581	0.3515	1.06
system1	0.0986	0.8314	0.15
system2	0.1077	0.6118	0.13

Table 7: Average results on formulation types for the graph class $\mathscr{G}$.

lations but quartic and system2, which took considerably longer to solve. The best trade-off overall between quality and CPU time is ddp_pca_system2.

graph type	mde	lde	CPU
Walmostreg	0.0916	0.4283	0.15
Wbipartite	0.0008	0.0460	1.31
Wcliquechain	0.2607	0.9838	0.15
Wcluster	0.2021	0.7913	0.19
Wdmdgp	0.2680	1.0036	0.15
Wmesh	0.0106	0.1382	0.13
Wpowerlaw	0.0571	0.4401	0.11
Wrandom	0.0755	0.4857	0.14
Wtorus	0.0501	0.3761	0.10
Wtriangle	0.1551	0.6157	0.06
Wtrichain	0.0363	0.1330	0.04
Wtripartite	0.0086	0.1551	4.23
almostreg	0.0410	0.2685	0.14
beeker_glusa	0.0000	0.0000	0.04
bipartite	0.0000	0.0000	0.88
cliquechain	0.2419	1.0672	0.17
cluster	0.1713	0.7952	0.18
dmdgp	0.2512	1.0721	0.17
local	0.1436	0.6596	0.26
mesh	0.0000	0.0000	0.09
norm	3.3560	12.0053	0.57
powerlaw	0.0086	0.2692	0.18
random	0.0464	0.2732	0.13
torus	0.0070	0.0659	0.08
triangle	0.0853	0.4461	0.06
trichain	0.0034	0.0269	0.04
tripartite	0.0000	0.0000	2.10

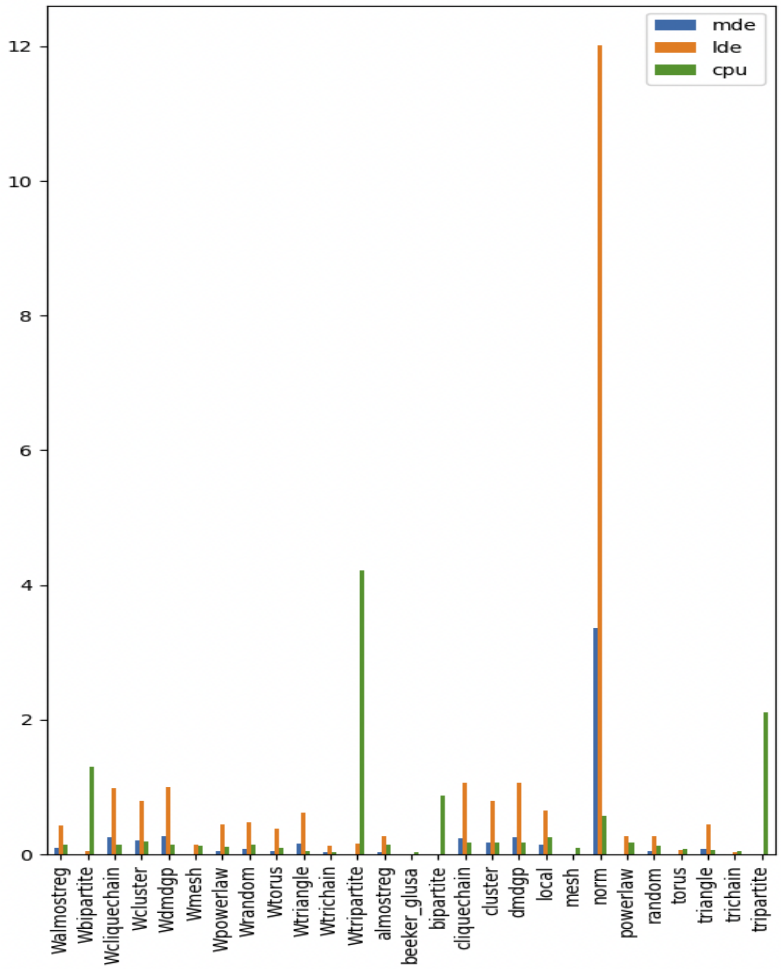


Table 8: Average results on graph types and the corresponding bar plot for the graph class $\mathscr{G}$.

instance	formulation	mde	lde	CPU
tiny	dualddp_cyclesimple	0.00	0.00	0.19
1guu-1	dualddp_system2	0.06	1.05	0.83
1guu-400	system2	0.08	0.97	2.70
C0030pkl	dualddp_cyclesimple	0.04	1.32	5.68
1PPT	quartic	0.26	2.58	8.48
1guu	sdp_system2	0.05	0.87	8476.58
100d	cyclesimple	0.33	3.06	13.50
3al1	system2	0.04	2.28	498.07
1hpv	ddp_quartic	0.40	3.62	214.49
il2	ddp_quartic	0.03	4.36	1262.47
1tii	dualddp_quartic	0.43	4.10	2928.25

Table 9: Best formulations per instance of graph class $\mathscr{P}$. The SDP formulation was replaced by the DDP formulation in the lower half due to excessive size.

5.4 UDGP tests and results

We have tested the following exact formulations on the set of UDGP instances obtained as explained in Sect. 5.1.4:

1. ucycsimplesys1 (Eq. (27)),
2. upushpull (Eq. (19)),

formulation	mde	lde	CPU
cyclesimple	0.2343	2.9293	532.43
quartic	0.1710	2.5200	2795.65
system2	0.1518	2.4399	6512.35
sdp_cyclesimple	0.1777	2.1073	2593.07
sdp_quartic	0.2339	2.1764	2738.82
sdp_system2	0.1740	2.1560	2958.58
ddp_cyclesimple	0.2859	3.9973	1280.88
ddp_quartic	0.2460	4.1536	1024.87
ddp_system2	0.2800	4.2591	2036.27
dualddp_cyclesimple	0.2013	2.5855	437.30
dualddp_quartic	0.2273	2.7961	370.53
dualddp_system2	0.2334	2.7907	634.17

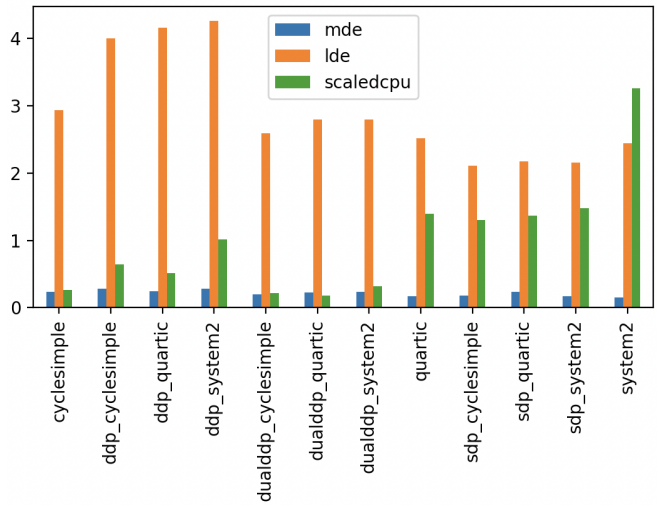


Table 10: Average results (over all instances) on formulation types for the graph class $\mathscr{P}$.

3. uquartic (Eq. (11)),
4. uquarticcont (Eq. (12)),
5. usystem1 (Eq. (16)),
6. usystem2 (Eq. (15)),

and the approximate matrix reformulations:

1. misdp_usystem1 (Eq. (39) with $\mathbf{X} = \mathbf{S}_n^+$),
2. middp_usystem1 (Eq. (39) with $\mathbf{X} = \mathbf{D}_n$),
3. midualddp_usystem1 (Eq. (39) with $\mathbf{X} = \mathbf{D}_n^*$).

We present our computational results grouped in the same way as for the DGP results (Sect. 5.3). The quality of the realization is measured by mde and lde, as in Sect. 5.3.

While there is no intrinsic measure of the assignment α, all our UDGP instances are generated from a DGP one (by losing the graph and keeping the distance values), so we can evaluate the difference between the graph G_α reconstructed from α and the original graph G that gave rise to the UDGP instance. This measure, called gphsim, is based on an evaluation of label-independent topological similarity of graphs if G, G_α have the same number of nodes, and on a comparison of the (zero-padded) spectra of the Laplacian matrices [29] of G, G_α otherwise (we let $\mathsf{spectrumLaplacian}(G)$ be the vector of eigenvalues of the Laplacian matrix of G); it makes use of the normalized adjacency matrix $\widehat{\mathsf{adj}}(G)$ of a graph G, which is its adjacency matrix scaled by its matrix norm. The measure gphsim is computed as in Alg. 2. It has values in $[-1, 1]$, with $\texttt{gphsim} = 1$ if G, G_α are isomorphic graphs.

Accordingly, for UDGP results we report mde, lde and gphsim. Only the first two measures attest to the success of the solution algorithm (when close to zero), while a gphsim measure significantly lower than 1 might simply attest to many different graphs compatible with the given distance values.

5.4.1 The Euclidean graph collection $\mathscr{R}$

As mentioned in Sect. 5.3.1 for DGP instances, we use this benchmark to establish to what size we can hope to solve UDGP instances to guaranteed optimality. Tolerance and time are as in the DGP case (10^{-6} and 1800s, see Sect. 5.3.1). As in the DGP case, given that these tests aim at establishing the size for which we may be able to solve such problems to optimality, we only tested the exact formulations ucycsimplesys1, upushpull, uquartic, uquarticcont, usystem1, usystem2.

Algorithm 2 The graph similarity measure gphsim

if $|V(G)| = |V(G_\alpha)|$ **then**
  gphsim $= 0$
  if G, G_α have matching degree sequences **then**
    gphsim $\leftarrow$ gphsim $+ 1$
    if G, G_α have matching triangle sequences **then**
      gphsim $\leftarrow$ gphsim $+ 1$
      if G, G_α have matching clique sequences **then**
        gphsim $\leftarrow$ gphsim $+ 1$
        if G, G_α are isomorphic **then**
          gphsim $\leftarrow$ gphsim $+ 1$
        end if
      end if
    end if
  end if
  gphsim $\leftarrow$ gphsim$/4$
  if gphsim < 1 **then**
    gphsim $\leftarrow \frac{1}{2}$gphsim $+ \frac{1}{2}\mathsf{tr}(\widehat{\mathsf{adj}}(G)\,\widehat{\mathsf{adj}}(H))$
  end if
else
  # assume $|V(G)| < |V(G_\alpha)|$ without loss of generality
  $s_G = \mathsf{spectrumLaplacian}(G)$ # eigenvalues in decreasing order
  pad s_G with $V(G_\alpha) - V(G)$ tailing zeros
  $s_{G_\alpha} = \mathsf{spectrumLaplacian}(G_\alpha)$
  normalize s_G, s_H
  gphsim $= \langle s_G, s_{G_\alpha} \rangle$
end if

$\approx \lvert V \rvert$	mde	lde	CPU
5	0.1533	0.6015	1037.35
8	2.2117	5.6512	1059.07
10	1.6241	5.8019	1263.91
12	3.3561	9.8532	669.65
15	4.2756	13.3832	1000.90
18	4.1293	12.4107	1297.91
20	7.0485	19.2524	1443.99

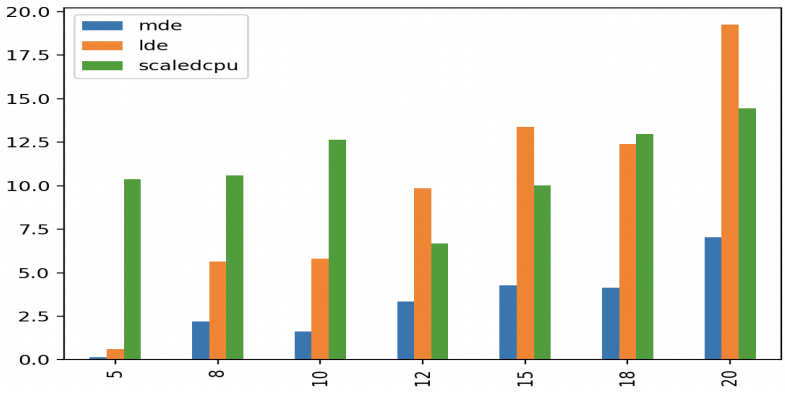


Table 11: Average results on $|V|$ for the graph class $\mathscr{R}$ on UDGP.

The results are shown in Tables 11-15. The only story they tell is that the realization errors are always large, even when the reconstructed graph G_α is isomorphic to the original graph G (Table 14).

A more in-depth look at the results reveals that the global MINLP solver terminated naturally on 40 (instance, MINLP formulation) pairs out of the 152 pairs tested. Of these, only 6 yielded mde and lde measures within $O(10^{-3})$ (involving the smallest instance sizes and mainly the upushpull formulation) and only 3 within $O(10^{-5})$: (euclid-5_0.9,upushpull), (euclid-8_0.4,upushpull), (euclid-8_0.4,system1).

In particular, the fact that a global MINLP solver terminates naturally on 40 (instances, formulation) pairs, but only on 6 does it find an optimum that looks close enough to a global optimum, denotes a high level of degeneration and a general lack of constraint qualification conditions in the local optimization algorithms implemented by the local NLP subsolver(s) within the global MINLP solver [113, 67]. We therefore conclude that the current state of affairs

$\approx \lvert E \rvert$	mde	lde	CPU
50	1.6256	4.8607	1044.06
100	4.3829	13.6181	1000.95
150	4.5666	13.9273	1317.59
200	6.6131	16.7588	1406.54

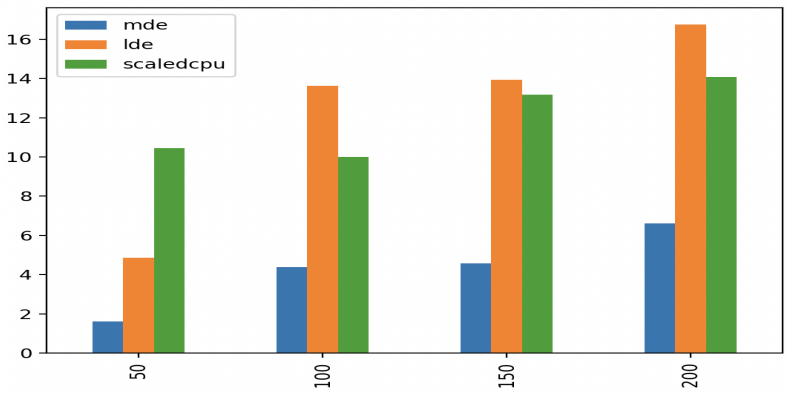


Table 12: Average results on $\lvert E \rvert$ for the graph class $\mathscr{R}$ on UDGP.

$\approx$ **density**	mde	lde	CPU
0.4	2.4903	8.0731	987.70
0.5	4.9176	15.1659	901.58
0.6	0.5023	2.1405	1088.01
0.7	4.3175	14.0233	790.11
0.8	3.4730	10.0472	1306.70
0.9	3.4126	9.6798	1231.16
1.0	2.8122	8.0143	1060.42

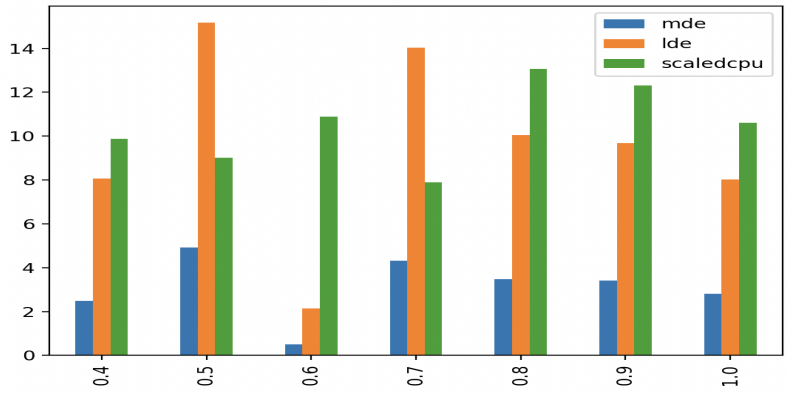


Table 13: Average results on edge density for the graph class $\mathscr{R}$ on UDGP.

$\approx$ `gphsim`	mde	lde	CPU
0.1	0.4060	3.0809	1804.01
0.2	3.0043	9.5707	755.69
0.3	3.2969	10.5904	1105.91
0.4	4.2320	12.1688	1278.52
1.0	2.1472	5.7603	1045.87

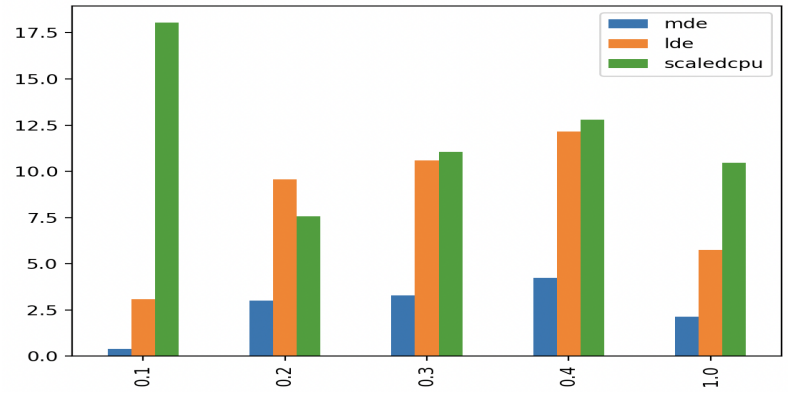


Table 14: Average results on graph similarity (`gphsim`) for the graph class $\mathscr{R}$ on UDGP.

with MINLP formulations and their global solvers is not mature enough for the UDGP. The only usable formulation is continuous nonconvex NLP uquarticcont.

5.4.2 The 309-graph collection $\mathscr{G}$

As in the DGP case (see Sect. 5.3.2), we use this benchmark to verify whether it is better to use the matrix formulation process or not. This test, however, involves some differences with respect to the DGP case.

1. We established in Sect. 5.4.1 that non-matrix MINLP formulations of the UDGP are ill-behaved to the point of being next to useless, even when the solver achieves a natural termination to what should be a global optimum

formulation	mde	lde	CPU
ucycsimplesys1	1.1592	5.0889	1620.06
upushpull	4.7066	12.3713	993.11
uquartic	5.5397	13.5655	1195.73
uquarticcont	0.2852	1.4838	59.50
usystem1	2.9467	11.0602	1368.15
usystem2	4.3407	12.0299	1301.94

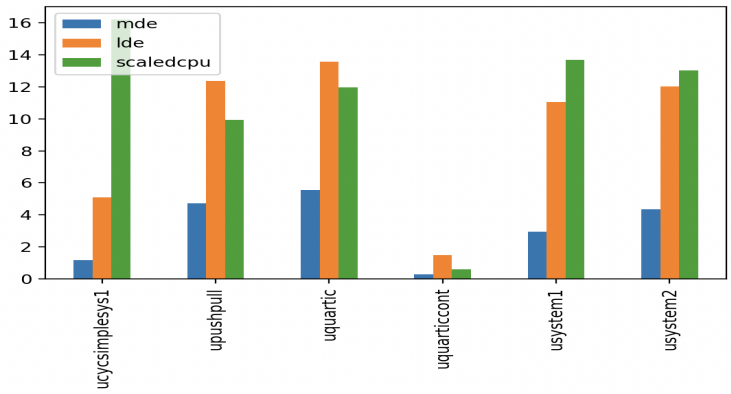


Table 15: Average results on formulation type for the graph class $\mathscr{R}$ on UDGP.

(but which most of the times is not). This leaves uquarticcont as the only non-matrix formulation. We solve it using a local NLP solver within the MS algorithm (Alg. 1), since a global nonconvex NLP solver would be too time-consuming.

2. Local optima of MINLP may be as hard to find (both theoretically and practically) as global optima, which prevents the use of "refinement" in practice. Again, this leaves uquarticcont (a nonconvex NLP) as the only possible alternative in the area of non-matrix formulations of the UDGP: since quarticcont is continuous, local optima can be found rapidly. The downside is that the y variables are likely to attain non-integral values at local optima (X', y'), which prevents the construction of the graph G_α.

3. As mentioned in Sect. 4.6, we have two possible matrix formulation processes. The first uses the matrix solution X' followed by rank reduction and refinement over the reconstructed graph G_α. The second discards X' and solves the DGP instance (K, G_α). In the first case, refinement only involves a single call to a local NLP solver from the starting point x' obtained using rank reduction. In the second case we can use any NLP solver with any DGP formulation. Given point 2. above, we focus on the second process: we use matrix formulations only in order to reconstruct G_α, and then solve the resulting DGP instance.

The test we run therefore consists in comparing:

A. results on uquarticcont solved using a a local NLP solver (IPOPT) within MS, and

B. results from matrix formulations yielding a DGP instance (K, G_α) solved using the quartic formulation (chosen as the best non-matrix formulation in Sect. 5.3.2) by means of the local NLP solver IPOPT within MS (5 iterations) acting on the quartic formulation.

This comparison is reported in Table 16, with average performance measures aggregated by formulation type. We denote the mixed-integer SDP (MISDP) formulation used in the UDGP by umisdp, the mixed-integer DDP (MIDDP) matrix formulation by umiddp, and the mixed-integer dual DDP matrix formulation by umidualddp. The best performance in both solution quality and CPU time is given by umiddp_quartic.

We also look at other result aggregations: by vertex cardinality (Table 17), by edge cardinality (Table 18), and by graph similarity (Table 19). We recall that the graph similarity score gphsim measures the success of the graph reconstruction step from distance values in UDGPs (see Alg. 2).

As in Sect. 5.3.2, for $\mathscr{G}$ we also present results aggregations according to graph type in Table 20, obtaining results close to those of Table 8.

5.4.3 The protein graph collection $\mathscr{P}$

We attempt to reconstruct the shape of proteins from a partial set of inter-atomic distances without their adjacencies (see Sect. 5.1.3 and 5.1.4) by using the methods that appear more promising from earlier UDGP experiments

formulation	mde	lde	CPU
umiddp	0.0360	0.2149	690.43
umidualddp	0.1387	0.6687	1791.72
umisdp	0.0914	0.4381	1862.70
uquarticcont	0.0889	0.3045	17416.50

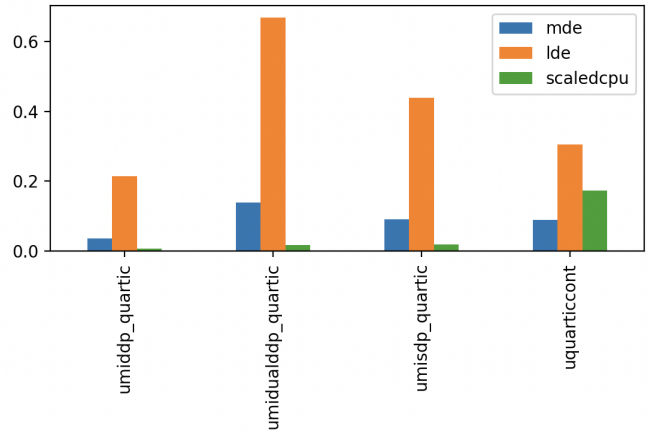


Table 16: Average results on formulation type for the graph class $\mathscr{G}$ on UDGP.

$\approx \lvert V \rvert$	mde	lde	CPU
3	0.0000	0.0000	8.82
4	0.0000	0.0000	8.84
5	0.0070	0.0140	8.85
6	0.0098	0.0471	9.53
7	0.0137	0.0325	9.11
9	0.0074	0.0304	11.54
10	0.0093	0.0447	10.26
15	0.0097	0.0671	462.55
16	0.0034	0.0333	238.49
20	0.0431	0.2903	395.46
21	0.0111	0.0784	486.29
25	0.0045	0.0339	331.86
28	0.0065	0.0605	681.86
35	0.1135	0.5495	1851.65
36	0.0032	0.0318	358.64
40	0.0000	0.0006	43.73
49	0.0049	0.0622	1886.85
50	0.1943	0.7452	7485.38
60	0.0005	0.0156	370.49
70	0.0012	0.0209	1420.22
100	0.0093	0.1128	17656.13
105	0.0114	0.1419	18530.49
150	0.0757	0.3191	227135.50

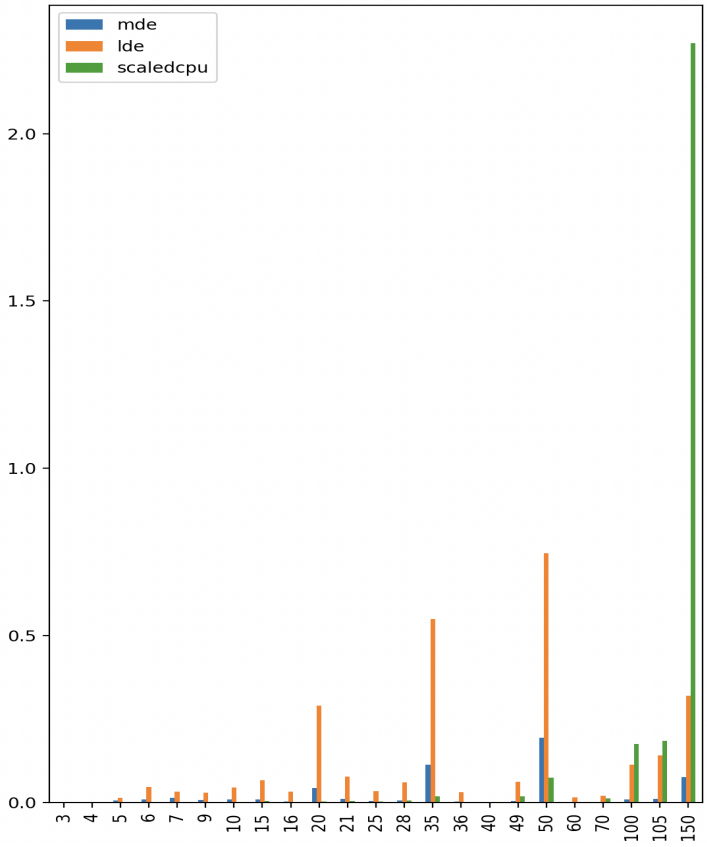


Table 17: Average results on $\lvert V \rvert$ for the graph class $\mathscr{G}$ on UDGP.

(Sect. 5.4.1-5.4.2), namely: carrying out the graph reconstruction using the mixed-integer DDP and dual DDP matrix formulations, and then solving the resulting DGP instance using IPOPT within the MS algorithm (Alg. 1) with a 5 iterations limit. IPOPT was deployed on the quartic formulation.

Some attempts with the usual time limit (1800s) imposed on the (matrix) MILP formulations only yielded solutions for the tiny instance. We therefore removed the largest protein instances from our benchmark, and only focused on a subset of six protein instances (from tiny up to 1guu in Table 1). We allowed Gurobi a maximum of 12h of CPU time. Even so, with the exception of tiny, we only obtained solutions from the mixed-integer dual DDP formulation, which therefore becomes, *de facto*, the only eligible formulation to successfully obtain approximate solutions of the UDGP on protein instances.

$\approx \lvert E \rvert$	mde	lde	CPU
50	0.0055	0.0416	150.87
100	0.0108	0.0937	991.03
150	0.0279	0.2748	6839.26
200	0.0985	0.6401	3502.28
250	0.0238	0.1488	34620.31
300	0.0481	0.2441	26749.03
350	1.2046	6.4760	5869.55
400	0.0885	0.3493	12939.00
450	0.0449	0.1929	16695.82
500	0.0762	0.3332	21461.61
550	1.0753	4.1404	15175.19
650	2.4117	9.8750	39603.21
700	1.3523	7.3373	39099.54
1100	7.4280	22.6607	23115.13
1150	4.7297	13.1730	23067.89

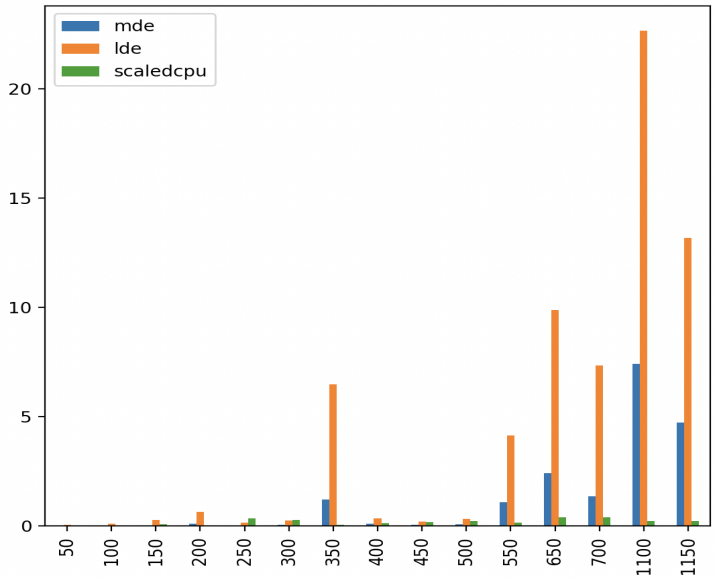


Table 18: Average results on $\lvert E \rvert$ for the graph class $\mathscr{G}$ on UDGP.

$\approx$ gphsim	mde	lde	CPU
0.0	0.0062	0.0530	3741.30
0.1	0.0149	0.1102	4205.63
0.2	0.1318	0.6932	6163.91
0.3	0.0074	0.1082	241.70
0.4	2.2244	7.6193	9435.18
0.5	0.0025	0.0075	18.30
0.6	0.0070	0.0943	25506.97
0.7	0.0000	0.0000	121338.65
0.8	0.0080	0.0995	9913.92
0.9	0.0113	0.0968	11824.05
1.0	0.0459	0.2785	1437.42

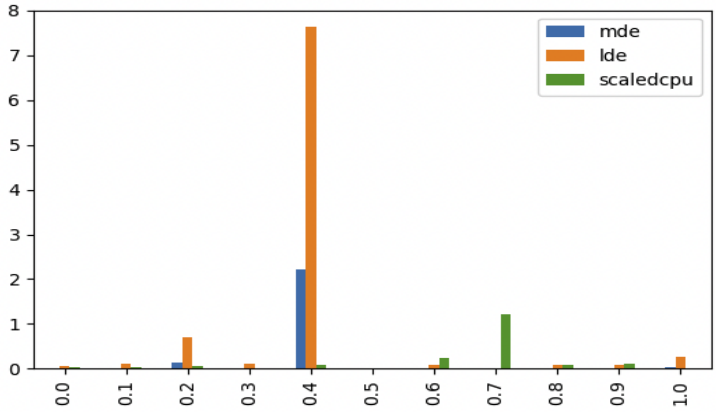


Table 19: Average results on graph similarity (gphsim) for the graph class $\mathscr{G}$ on UDGP.

We remark that, in [37], the uquarticcont formulation in Eq. (12) was successfully used to solve random UDGP instances (generated similarly to [61]) with up to 400 atoms in just over 13h of CPU time by the Baron [103] solver. But instances generated from PDB information (Sect. 5.1.3) are generally more difficult — the random instances from [61] have more and better distributed solutions.

Given the dearth of instances of this benchmark there is no need for aggregated and averaged results. We give full results in Table 21. The results in table 21 appear to indicate that the MIDDP formulations applied to proteins are either infeasible or have few solutions that are hard to find. Instead, dual DDP relaxations are always feasible whenever the originating SDP relaxation is feasible, and so are their mixed-integer counterparts, provided that the given distance values are compatible with an ℓ_2 metric: this may well be the reason that the umidualddp matrix formulation is the only one that scores some success on protein instances.

And yet, dual DDPs (both continuous and mixed-integer) generally produce indefinite matrix solutions, which, after dimensional reduction, contain considerable error. Even if we only employ the umidualddp formulation to reconstruct the graph from the values of the assignment variables y, we would expect an indefinite matrix to carry more reconstruction error than a PSD one. And, indeed, most of the larger instances have disappointingly large mde and lde error measures. While a part of this error can be attributed to the MS algorithm configured with only 5 iterations, another is certainly due to poor graph reconstruction. That no instance displays a graph reconstruction similarity (gphsim) value of exactly 1.0 (i.e., a perfect reconstruction) is to be expected: there will be many graphs

graph type	mde	lde	CPU
Walmostreg	0.0121	0.1218	1780.42
Wbipartite	0.0070	0.0895	9740.23
Wcliquechain	0.0361	0.2257	3281.06
Wcluster	0.0364	0.2062	4254.55
Wdmdgp	0.0360	0.2195	3533.00
Wmesh	0.0041	0.0558	729.02
Wpowerlaw	0.0077	0.0914	708.66
Wrandom	0.0114	0.1228	1669.00
Wtorus	0.0176	0.1138	1275.43
Wtriangle	0.0200	0.1250	638.01
Wtrichain	0.0074	0.0212	8.97
Wtripartite	0.0399	0.2537	63168.87
almostreg	0.0000	0.0000	283.50
beeker_glusa	0.0110	0.0333	9.02
bipartite	0.0000	0.0000	3006.49
cliquechain	0.0000	0.0000	1377.71
cluster	0.0034	0.0107	2742.12
dmdgp	0.0000	0.0000	1909.70
local	0.0058	0.0476	707.85
mesh	0.0000	0.0000	113.17
norm	1.3154	5.6693	9234.31
powerlaw	0.0000	0.0000	116.94
random	0.0000	0.0000	256.40
torus	0.0000	0.0000	327.64
triangle	0.0000	0.0000	22.26
trichain	0.0000	0.0000	9.05
tripartite	0.0000	0.0000	42806.12

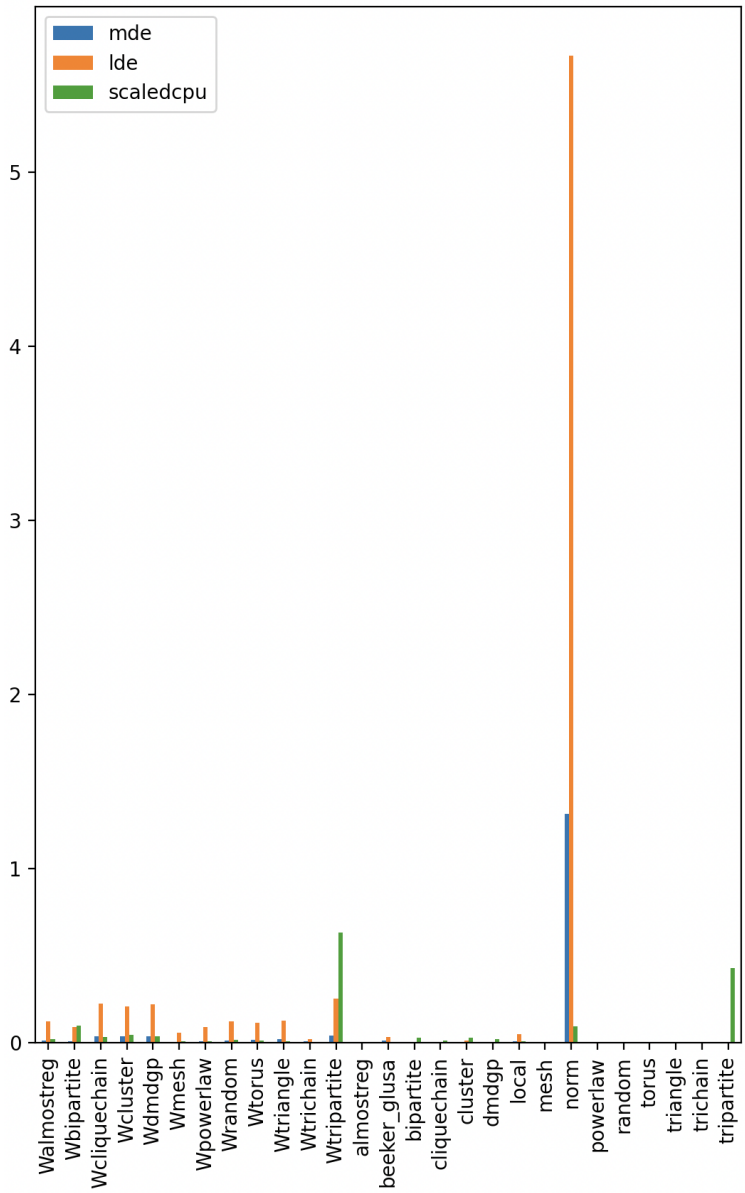


Table 20: Average results on graph types and the corresponding bar plot for the graph class $\mathscr{G}$ on UDGP.

instance	formulation	gphsim	mde	lde	CPU
tiny	umiddp	0.989	0.000	0.006	2257.76
tiny	umidualddp	0.216	0.154	1.962	58.53
1guu-1	umidualddp	0.917	0.126	1.513	335825.94
1guu-4000	umidualddp	0.929	6.429	10.300	4143.67
C0030pkl	umidualddp	0.891	7.912	11.704	25833.85
1PPT	umidualddp	0.034	10.579	14.236	20248.91
1guu	umidualddp	0.619	13.307	16.749	14289.60

Table 21: Results for (part of) the graph class $\mathscr{G}$ on UDGP.

compatible with the same distance values. But not all of these graphs will be realizable in $K = 3$ dimensions. We can see this in the tiny instance, the only one that could be solved by both umiddp and umidualddp. The umiddp solution has a gphsim score that is very close to 1.0, and almost zero mde and lde error measures. On the contrary, the umidualddp solution has a gphsim score of 0.216 and considerably large error measures given the small instance size (38 atoms).

The only encouraging result is 1guu-1, the size of which is nontrivial (150 atoms). The umidualddp formulation provided a gphsim score of 0.917, and IPOPT, taking over 92h of CPU time, was able to find a solution with tolerable error measures. The rest of the results show error measures that denote bad graph reconstructions and realizations.

6 Conclusion

In this survey we have surveyed MP formulation-based methods for solving distance geometry problems, even when the input is a list of distance values instead of a weighted graph (i.e., the distances are not assigned to graph edges). The computational benchmarks established that while formulation-based methodologies are useful to solve even fairly large DGP instances (derived from protein data) in 3D, similar methodologies are not able to solve UDGP instances of the same size.

References

[1] A. Ahmadi and G. Hall. Sum of squares basis pursuit with linear and second order cone programming. In H. Harrington, M. Omar, and M. Wright, editors, *Algebraic and Geometric Methods in Discrete Mathematics*, volume 685 of *Contemporary Mathematics*, pages 27–54. AMS, Providence, RI, 2017.

[2] A. Ahmadi and A. Majumdar. Dsos and sdsos optimization: more tractable alternatives to sum of squares and semidefinite optimization. Technical Report 1706.02586v1, arXiv, 2017.

[3] A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: More tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebra and Geometry*, 3(2):193–230, 2019.

[4] J. Alencar, C. Lavor, and L. Liberti. Realizing euclidean distance matrices by sphere intersection. *Discrete Applied Mathematics*, 256:5–10, 2019.

[5] A. Alfakih, A. Khandani, and H. Wolkowicz. Solving Euclidean distance matrix completion problems via semidefinite programming. *Computational Optimization and Applications*, 12:13–30, 1999.

[6] E. Amaldi, L. Liberti, F. Maffioli, and N. Maculan. Edge-swapping algorithms for the minimum fundamental cycle basis problem. *Mathematical Methods of Operations Research*, 69:205–223, 2009.

[7] L. Asimow and B. Roth. The rigidity of graphs. *Transactions of the AMS*, 245:279–289, 1978.

[8] L. Asimow and B. Roth. The rigidity of graphs II. *Journal of Mathematical Analysis and Applications*, 68:171–190, 1979.

[9] J. Aspnes, T. Eren, D. Goldenberg, S. Morse, W. Whiteley, R. Yang, B. Anderson, and P. Belhumeur. A theory of network localization. *IEEE Transactions on Mobile Computing*, 5(12):1663–1678, 2006.

[10] A. Bahr, J. Leonard, and M. Fallon. Cooperative localization for autonomous underwater vehicles. *International Journal of Robotics Research*, 28(6):714–728, 2009.

[11] G. Barker and D. Carlson. Cones of diagonally dominant matrices. *Pacific Journal of Mathematics*, 57(1):15–32, 1975.

[12] A. Barvinok. Measure concentration in optimization. *Mathematical Programming*, 79:33–53, 1997.

[13] N. Beeker, S. Gaubert, C. Glusa, and L. Liberti. Is the distance geometry problem in NP? In Mucherino et al. [98], pages 85–94.

[14] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4):597–634, 2009.

[15] B. Berger, J. Kleinberg, and T. Leighton. Reconstructing a three-dimensional model with arbitrary errors. *Journal of the ACM*, 46(2):212–235, 1999.

[16] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I.N. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acid Research*, 28:235–242, 2000.

[17] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[18] S. Billinge, P. Duxbury, D. Gonçalves, C. Lavor, and A. Mucherino. Assigned and unassigned distance geometry: Applications to biological molecules and nanostructures. *4OR*, 14:337–376, 2016.

[19] P. Biswas. *Semidefinite programming approaches to distance geometry problems*. PhD thesis, Stanford University, 2007.

[20] P. Biswas, T. Lian, T. Wang, and Y. Ye. Semidefinite programming based algorithms for sensor network localization. *ACM Transactions in Sensor Networks*, 2:188–220, 2006.

[21] J. Borwein and H. Wolkowicz. Facial reduction for a cone-convex programming problem. *Journal of the Australian Mathematical Society A*, 30:369–180, 1981.

[22] M. Bruglieri, R. Cordone, and L. Liberti. Maximum feasible subsystems of distance geometry constraints. *Journal of Global Optimization*, 83:29–47, 2022.

[23] S. Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming A*, 120:479–495, 2009.

[24] A. Cassioli, B. Bordeaux, G. Bouvier, A. Mucherino, R. Alves, L. Liberti, M. Nilges, C. Lavor, and T. Malliavin. An algorithm to enumerate all possible protein conformations verifying a set of distance constraints. *BMC Bioinformatics*, 16:23–38, 2015.

[25] COIN-OR. *Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT*, 2006.

[26] R. Connelly, S. Gortler, and L. Theran. Reconstruction in one dimension from unlabeld Euclidean lengths. *Combinatorica*, to appear.

[27] G. Crippen and T. Havel. *Distance Geometry and Molecular Conformation*. Wiley, New York, 1988.

[28] M. Cucuringu, A. Singer, and D. Cowburn. Eigenvector synchronization, graph ridigity and the molecule problem. *Information and Inference: a journal of the IMA*, 1:21–67, 2012.

[29] D. Cvetković, P. Rowlinson, and S. Simić. *An introduction to the theory of graph spectra*. CUP, Cambridge, 2010.

[30] T. Dakić. *On the turnpike problem*. PhD thesis, Simon Fraser University, 2000.

[31] C. D'Ambrosio and L. Liberti. Distance geometry in linearizable norms. In F. Nielsen and F. Barbaresco, editors, *Geometric Science of Information*, volume 10589 of *LNCS*, pages 830–838, Berlin, 2017. Springer.

[32] C. D'Ambrosio, Ky Vu, C. Lavor, L. Liberti, and N. Maculan. New error measures and methods for realizing protein graphs from distance data. *Discrete and Computational Geometry*, 57(2):371–418, 2017.

[33] G. Dias and L. Liberti. Diagonally dominant programming in distance geometry. In R. Cerulli, S. Fujishige, and R. Mahjoub, editors, *International Symposium in Combinatorial Optimization*, volume 9849 of *LNCS*, pages 225–236, New York, 2016. Springer.

[34] I. Dokmanić, R. Parhizkar, J. Ranieri, and M. Vetterli. Euclidean distance matrices: Essential theory, algorithms and applications. *IEEE Signal Processing Magazine*, 1053-5888:12–30, Nov. 2015.

[35] Q. Dong and Z. Wu. A linear-time algorithm for solving the molecular distance geometry problem with exact inter-atomic distances. *Journal of Global Optimization*, 22:365–375, 2002.

[36] P. Duxbury, L. Granlund, S. Gujarathi, P. Juhás, and S. Billinge. The unassigned distance geometry problem. *Discrete Applied Mathematics*, 204:117–132, 2016.

[37] P. Duxbury, C. Lavor, L. Liberti, and L. de Salles-Neto. Unassigned distance geometry and molecular conformation problems. *Journal of Global Optimization*, 83:73–82, 2022.

[38] T. Eren, D. Goldenberg, W. Whiteley, Y. Yang, A. Morse, B. Anderson, and P. Belhumeur. Rigidity, computation, and randomization in network localization. *IEEE*, pages 2673–2684, 2004.

[39] B. Fang. Trilateration and extension to global positioning system navigation. *Journal of Guidance, Control, and Dynamics*, 9(6):715–717, 1986.

[40] R. Fourer and D. Gay. *The AMPL Book*. Duxbury Press, Pacific Grove, 2002.

[41] S. Gershgorin. über die Abgrenzung der Eigenwerte einer Matrix. *Zvesti Akademii Nauk SSSR. Otdelenie Fizicheskikh i Matematicheskikh Nauk*, 6:749–754, 1931.

[42] H. Gluck. Almost all simply connected closed surfaces are rigid. In A. Dold and B. Eckmann, editors, *Geometric Topology*, volume 438 of *Lecture Notes in Mathematics*, pages 225–239, Berlin, 1975. Springer.

[43] D. Gonçalves, C. Lavor, L. Liberti, and M. Souza. A new algorithm for the ^{k}DMDGP subclass of distance geometry problems with exact distances. *Algorithmica*, 83:2400–2426, 2021.

[44] D. Gonçalves, A. Mucherino, C. Lavor, and L. Liberti. Recent advances on the interval distance geometry problem. *Journal of Global Optimization*, 69:525–545, 2017.

[45] S. Gortler, A. Healy, and D. Thurston. Characterizing generic global rigidity. *American Journal of Mathematics*, 132(4):897–939, 2010.

[46] W. Gramacho, A. Mucherino, C. Lavor, and N. Maculan. A parallel BP algorithm for the discretizable distance geometry problem. In *Proceedings of the Workshop on Parallel Computing and Optimization*, pages 1756–1762, Piscataway, 2012. IEEE.

[47] S. Gujarathi, C. Farrow, C. Glosser, L. Granlund, and P. Duxbury. *Ab initio* reconstruction of complex Euclidean networks in two dimensions. *Physical Review E*, 89:053311, 2014.

[48] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[49] T. Havel and K. Wüthrich. An evaluation of the combined use of nuclear magnetic resonance and distance geometry for the determination of protein conformations in solution. *Journal of Molecular Biology*, 182(2):281–294, 1985.

[50] B. Hendrickson. The molecule problem: exploiting structure in global optimization. *SIAM Journal on Optimization*, 5:835–857, 1995.

[51] L. Hoai An. Solving large scale molecular distance geometry problems by a smoothing technique via the gaussian transform and d.c. programming. *Journal of Global Optimization*, 27:375–397, 2003.

[52] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933.

[53] IBM. *ILOG CPLEX 22.1 User's Manual*. IBM, 2022.

[54] P. Juhás, D. Cherba, P. Duxbury, W. Punch, and S. Billinge. *Ab initio* determination of solid-state nanostructure. *Nature*, 440(30):655–658, 2006.

[55] S. Khalife, D. Gonçalves, and Leo Liberti. Distance geometry for word representations and applications. *Journal of Computational Mathematics and Data Science*, 6:100073, 2023.

[56] N. Krislock. *Semidefinite Facial Reduction for Low-Rank Euclidean Distance Matrix Completion*. PhD thesis, University of Waterloo, 2010.

[57] N. Krislock and H. Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. *SIAM Journal on Optimization*, 20:2679–2708, 2010.

[58] S. Kucherenko and Yu. Sytsko. Application of deterministic low-discrepancy sequences in global optimization. *Computational Optimization and Applications*, 30(3):297–318, 2004.

[59] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, 1970.

[60] M. Laurent. A tour d'horizon on positive semidefinite and Euclidean distance matrix completion problems. In *Topics in Semidefinite and Interior-Point Methods*, volume 18 of *Fields Institute for Research in Mathematical Sciences: Communication*, Providence, RI, 1998. American Mathematical Society.

[61] C. Lavor. On generating instances for the molecular distance geometry problem. In Liberti and Maculan [84], pages 405–414.

[62] C. Lavor, L. Liberti, B. Donald, B. Worley, B. Bardiaux, T. Malliavin, and M. Nilges. Minimal NMR distance information for rigidity of protein graphs. *Discrete Applied Mathematics*, 256:91–104, 2019.

[63] C. Lavor, L. Liberti, and N. Maculan. Computational experience with the molecular distance geometry problem. In J. Pintér, editor, *Global Optimization: Scientific and Engineering Case Studies*, pages 213–225. Springer, Berlin, 2006.

[64] C. Lavor, L. Liberti, N. Maculan, and A. Mucherino. Recent advances on the discretizable molecular distance geometry problem. *European Journal of Operational Research*, 219:698–706, 2012.

[65] C. Lavor, L. Liberti, and A. Mucherino. The *interval* Branch-and-Prune algorithm for the discretizable molecular distance geometry problem with inexact distances. *Journal of Global Optimization*, 56:855–871, 2013.

[66] P. Lemke, S. Skiena, and W. Smith. Reconstructing sets from interpoint distances. In B. Aronov and *et al.*, editors, *Discrete and Computational Geometry*, volume 25 of *Algorithms and Combinatorics*, pages 597–631, Berlin, 2003. Springer.

[67] L. Liberti. Writing global optimization software. In Liberti and Maculan [84], pages 211–262.

[68] L. Liberti. Reformulations in mathematical programming: Definitions and systematics. *RAIRO-RO*, 43(1):55–86, 2009.

[69] L. Liberti. Undecidability and hardness in mixed-integer nonlinear programming. *RAIRO-Operations Research*, 53:81–109, 2019.

[70] L. Liberti. Distance geometry and data science. *TOP*, 28:271–339, 2020.

[71] L. Liberti. A new distance geometry method for constructing word and sentence vectors. In *Companion Proceedings of the Web Conference (DL4G Workshop)*, volume 20 of *WWW*, New York, 2020. ACM.

[72] L. Liberti. Unassigned distance geometry and the Buckminsterfullerene. AIRO-ODS 2024 Conference, Cham, accepted. Springer.

[73] L. Liberti, S. Cafieri, and F. Tarissan. Reformulations in mathematical programming: A computational approach. In A. Abraham, A.-E. Hassanien, P. Siarry, and A. Engelbrecht, editors, *Foundations of Computational Intelligence Vol. 3*, number 203 in Studies in Computational Intelligence, pages 153–234. Springer, Berlin, 2009.

[74] L. Liberti, G. Iommazzo, C. Lavor, and N. Maculan. Cycle-based formulations in distance geometry. *Open Journal of Mathematical Optimization*, 4:art. 1, 16p., 2023.

[75] L. Liberti and C. Lavor. On a relationship between graph realizability and distance matrix completion. In V. Kostoglou, G. Arabatzis, and L. Karamitopoulos, editors, *Proceedings of BALCOR*, volume I, pages 2–9, Thessaloniki, 2011. Hellenic OR Society.

[76] L. Liberti and C. Lavor. Six mathematical gems in the history of distance geometry. *International Transactions in Operational Research*, 23:897–920, 2016.

[77] L. Liberti and C. Lavor. *Euclidean Distance Geometry: An Introduction*. Springer, New York, 2017.

[78] L. Liberti and C. Lavor. Open research areas in distance geometry. In A. Migalas and P. Pardalos, editors, *Open Problems in Optimization and Data Analysis*, volume 141 of *SOIA*, pages 183–223. Springer, New York, 2018.

[79] L. Liberti, C. Lavor, and N. Maculan. A branch-and-prune algorithm for the molecular distance geometry problem. *International Transactions in Operational Research*, 15:1–17, 2008.

[80] L. Liberti, C. Lavor, N. Maculan, and F. Marinelli. Double variable neighbourhood search with smoothing for the molecular distance geometry problem. *Journal of Global Optimization*, 43:207–218, 2009.

[81] L. Liberti, C. Lavor, N. Maculan, and A. Mucherino. Euclidean distance geometry and applications. *SIAM Review*, 56(1):3–69, 2014.

[82] L. Liberti, C. Lavor, and A. Mucherino. The discretizable molecular distance geometry problem seems easier on proteins. In Mucherino et al. [98], pages 47–60.

[83] L. Liberti, C. Lavor, A. Mucherino, and N. Maculan. Molecular distance geometry methods: from continuous to discrete. *International Transactions in Operational Research*, 18:33–51, 2010.

[84] L. Liberti and N. Maculan, editors. *Global Optimization: from Theory to Implementation*. Springer, Berlin, 2006.

[85] L. Liberti, B. Manca, and P.-L. Poirion. Random projections for the distance geometry problem. In M. Noy et al., editor, *Proceedings of the workshop Discrete Mathematics Days*, Santander, 2022. Universidad de Cantabria.

[86] L. Liberti and K. Vu. Barvinok's naive algorithm in distance geometry. *Operations Research Letters*, 46:476–481, 2018.

[87] R. Lima and J. Martinez. Solving molecular distance geometry problems using a continuous optimization approach. In Mucherino et al. [98], pages 213–224.

[88] L. Lovász and Y. Yemini. On generic rigidity in the plane. *SIAM Journal on Algebraic and Discrete Methods*, 3(1):91–98, 1982.

[89] M. Lubin, E. Yamangil, R. Bent, and J.P. Vielma. Extended formulations in mixed-integer convex programming. In Q. Louveaux and M. Skutella, editors, *Integer Programming and Combinatorial Optimization (Proceedings of IPCO16)*, number 9682 in LNCS, pages 102–113, New York, 2016. Springer.

[90] P. Luisi. Molecular conformational rigidity: An approach to quantification. *Naturwissenschaften*, 64:569–574, 1977.

[91] A. Majumdar, A. Ahmadi, and R. Tedrake. Control and verification of high-dimensional systems with dsos and sdsos programming. In *Conference on Decision and Control*, volume 53, pages 394–401, Piscataway, 2014. IEEE.

[92] A. Man-Cho So and Y. Ye. Theory of semidefinite programming for sensor network localization. *Mathematical Programming B*, 109:367–384, 2007.

[93] L. Mencarelli, Y. Sahraoui, and L. Liberti. A multiplicative weights update algorithm for MINLP. *EURO Journal on Computational Optimization*, 5:31–86, 2017.

[94] J. Moré and Z. Wu. Global continuation for distance geometry problems. *SIAM Journal of Optimization*, 7(3):814–846, 1997.

[95] Mosek ApS. *The `mosek` manual, Version 10*, 2024.

[96] T. Motzkin and E. Straus. Maxima for graphs and a new proof of a theorem of Turán. *Canadian Journal of Mathematics*, 17:533–540, 1965.

[97] A. Mucherino, C. Lavor, and L. Liberti. Exploiting symmetry properties of the discretizable molecular distance geometry problem. *Journal of Bioinformatics and Computational Biology*, 10:1242009(1–15), 2012.

[98] A. Mucherino, C. Lavor, L. Liberti, and N. Maculan, editors. *Distance Geometry: Theory, Methods, and Applications*. Springer, New York, 2013.

[99] A. Mucherino, C. Lavor, T. Malliavin, L. Liberti, M. Nilges, and N. Maculan. Influence of pruning devices on the solution of molecular distance geometry problems. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *LNCS*, pages 206–217, Berlin, 2011. Springer.

[100] A. Mucherino, L. Liberti, and C. Lavor. `MD-jeep`: an implementation of a branch-and-prune algorithm for distance geometry problems. In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software*, volume 6327 of *LNCS*, pages 186–197, New York, 2010. Springer.

[101] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Operator splitting for conic optimization via homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[102] A. Patterson. A direct method for the determination of the components of interatomic distances in crystals. *Zeitschrift für Kristallographie*, 90:517–542, 1935.

[103] N.V. Sahinidis and M. Tawarmalani. *BARON 7.2.5: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2005.

[104] A. Báez Sánchez and C. Lavor. On the estimation of unknown distances for a class of Euclidean distance matrix completion problems with interval data. *Linear Algebra and its Applications*, 592:287–305, 2020.

[105] J. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. *Proceedings of 17th Allerton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[106] F. Schoen. Two-phase methods for global optimization. In P.M. Pardalos and H.E. Romeijn, editors, *Handbook of Global Optimization*, volume 2, pages 151–177. Kluwer Academic Publishers, Dordrecht, 2002.

[107] A. Singer. Angular synchronization by eigenvectors and semidefinite programming. *Applied and Computational Harmonic Analysis*, 30:20–36, 2011.

[108] M. Sippl and H. Scheraga. Cayley-Menger coordinates. *Proceedings of the National Academy of Sciences*, 83:2283–2287, 1986.

[109] S. Skiena, W. Smith, and P. Lemke. Reconstructing sets from interpoint distances. In *Proceedingsof the sixth ACM Symposium on Computational Geometry*, volume 6 of *SOCG*, pages 332–339, New York, 1990. ACM.

[110] S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56(2):275–294, 1994.

[111] Y. Takane, F. Young, and J. De Leeuw. Nonmetric individual differences in multidimensional scaling: an alternating least squares method with optimal scaling features. *Psychometrika*, 42:7–67, 1977.

[112] G. van Rossum and *et al. Python Language Reference, version 3*. Python Software Foundation, 2019.

[113] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[114] Wikipedia. Principal component analysis, 2019. [Online; accessed 190726].

[115] H.P. Williams. *Model Building in Mathematical Programming*. Wiley, Chichester, 4th edition, 1999.

[116] B. Worley, F. Delhommel, F. Cordier, T. Malliavin, B. Bardiaux, N. Wolff, M. Nilges, C. Lavor, and L. Liberti. Tuning interval branch-and-prune for protein structure determination. *Journal of Global Optimization*, 72:109–127, 2018.

[117] K. Wüthrich. NMR studies of structure and function of biological macromolecules (Nobel lecture). *Angewandte Chemie*, 42:3340–3363, 2003.