TOP (invited survey, to appear in 2020, Issue 2) manuscript No.
(will be inserted by the editor)

Distance Geometry and Data Science

Leo Liberti

Dedicated to the memory of Mariano Bellasio (1943-2019).

Received: date / Accepted: date

Abstract Data are often represented as graphs. Many common tasks in data science are based on distances between entities. While some data science methodologies natively take graphs as their input, there are many more that take their input in vectorial form. In this survey we discuss the fundamental problem of mapping graphs to vectors, and its relation with mathematical programming. We discuss applications, solution methods, dimensional reduction techniques and some of their limits. We then present an application of some of these ideas to neural networks, showing that distance geometry techniques can give competitive performance with respect to more traditional graph-to-vector mappings.

Keywords Euclidean distance · Isometric embedding · Random projection · Mathematical Programming · Machine Learning · Artificial Neural Networks

Contents

This research was partly funded by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement n. 764759 ETN "MINOA".

L. Liberti
LIX CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, France
E-mail: liberti@lix.polytechnique.fr

1 Introduction

This survey is about the application of Distance Geometry (DG) techniques to problems in Data Science (DS). More specifically, data are often represented as graphs, and many methodologies in data science require vectors as input. We look at the fundamental problem in DG, namely that of reconstructing vertex positions from given edge lengths, in view of using its solution methods in order to produce vector input for further data processing.

The organization of this survey is based on a "storyline". In summary, we want to exhibit alternative competitive methods for mapping graphs to vectors in order to analyse graphs using Machine Learning (ML) methodologies requiring vectorial input. This storyline will take us through fairly different subfields of mathematics, Operations Research (OR) and computer science. This survey does not provide exhaustive literature reviews in all these fields. Its purpose (and usefulness) rests in communicating the main idea sketched above, rather than serving as a reference for a field of knowledge. It is nonetheless a survey because, limited to the scope of its purpose, it aims at being informative and also partly educational, rather than just giving the minimal notions required to support its goal.

Here is a more detailed account of our storyline. We first introduce DG, some of its history, its fundamental problem and its applications. Then we

motivate the use of graph representations for several types of data. Next, we discuss some of the most common tasks in data science (e.g. classification, clustering) and the related methodologies (unsupervised and supervised learning). We introduce some robust and efficient algorithms used for embedding general graphs in vector spaces. We present some dimensional reduction operations, which are techniques for replacing sets X of high-dimensional vectors by lower-dimensional ones X', so that some of the properties of X are preserved at least approximately in X'. We discuss the instability of distances on randomly generated vectors and its impact on distance-based algorithms. Finally, we present an application of much of the foregoing theory: we train an Artificial Neural Network (ANN) on many training sets, so as to learn several given clusterings on sentences in natural language. Some training sets are generated using traditional methods, namely incidence vectors of short sequences of consecutive words in the corpus dictionary. Other training sets are generated by representing sentences by graphs and then using a DG method to encode these graphs into vectors. It turns out that some of the DG-generated training sets have competitive performances with the traditional methods. While the empirical evidence is too limited to support any general conclusion, it might invite more research on this topic.

The survey is interspersed with eight theorems with proofs. Aside from Thm. 8 about distance instability, the proof of which is taken almost verbatim from the original source [26], the proofs from the other theorems are not taken from any source. This does not mean that the theorems and their proofs are actually original. The theorems are usually quite easy to prove. Their proofs are reasonably short, and, we hope, easy to follow. There are several reasons for the presence of these theorems in this survey: (a) we have not found them stated and proved clearly anywhere else, and we wish we had during our research work (Thm. 1-4); (b) their proofs showcase some point we deem important about the underlying theory (Thm. 7-8); (c) they give some indication of the proof techniques involved in the overarching field (Thm. 6-7); (d) they justify a precise mathematical statement for which we found no citation (Thm. 5). While there may be some original mathematical results in this survey, e.g. Eq. (35) and the corresponding Thm. 5 (though something similar might be found in Henry Wolkowicz' work) as well as the computational comparison in Sect. 7.3.2, we believe that the only truly original part is the application of DG techniques to constructing training sets of ANNs in Sect. 9. Sect. 4, about representing data by graphs, may also contain some new ideas to Mathematical Programming (MP) readers, although everything we wrote can be easily reconstructed from existing literature, though some of which might perhaps be rather exotic to MP readership.

The rest of this paper is organized as follows. In Sect. 2 we give a brief introduction to the field of MP, considered as a formal language for optimization. In Sect. 3 we introduce the field of DG. In Sect. 4 we give details on how to represent four types of data as graphs. In Sect. 5 we introduce methods for clustering on vectors as well as directly on graphs. In Sect. 6 we present many methods for realizing graphs in Euclidean spaces, most of which are

based on MP. In Sect. 7 we present some dimensional reduction techniques. In Sect. 8 we discuss the distance instability phenomenon, which may have a serious negative inpact on distance-based algorithms. In Sect. 9 we present a case-in-point application of natural language clustering by means of an ANN, and discuss how the DG techniques can help construct the input part of the training set.

2 Mathematical Programming

Many of the methods discussed in this survey are optimization methods. Specifically, they belong to MP, which is a field of optimization sciences and OR. While most of the readers of this paper should be familiar with MP, the interpretation we give to this term is more formal than most other treatments, and we therefore discuss it in this section.

2.1 Syntax

MP is a formal language for describing optimization problems. The valid sentences of this language are the MP *formulations*. Each formulation consist of an array p of *parameter* symbols (which encode the problem input), an array x of n *decision variable* symbols (which will contain the solution), an *objective function* $f(p, x)$ with an optimization direction (either min or max), a set of *explicit constraints* $g_i(p, x) \leq 0$ for all $i \leq m$, and some *implicit constraints*, which impose that x should belong to some implicitly described set X. For example, some of the variables should take integer values, or should belong to the non-negative orthant, or to a positive semidefinite (psd) cone. The typical MP formulation is as follows:

$$\left.\begin{array}{rr} \mathrm{opt}_x & f(p,x) \\ \forall i \leq m & g_i(p,x) \leq 0 \\ & x \in X. \end{array}\right\} \tag{1}$$

It is customary to define MP formulations over explicitly closed feasible sets, in order to prevent issues with feasible formulations which have infima or suprema but no optima. This prevents the use of strict inequality symbols in the MP language.

2.2 Taxonomy

MP formulations are classified according to syntactical properties. We list the most important classes:

- if f, g_i are linear in x and X is the whole space, Eq. (1) is a Linear Program (LP);

- if f, g_i are linear in x and $X = \{0,1\}^n$, Eq. (1) is a Binary Linear Program (BLP);
- if f, g_i are linear in x and X is the whole space intersected with an integer lattice, Eq. (1) is a Mixed-Integer Linear Program (MILP);
- if f is a quadratic form in x, g_i are linear in x, and X is the whole space, Eq. (1) is a Quadratic Program (QP); if f is convex, then it is a convex QP (cQP);
- if f is linear in x and g_i are quadratic forms in x, and X is the whole space or a polyhedron, Eq. (1) is a Quadratically Constrained Program (QCP); if g_i are convex, it is a convex QCP (cQCP);
- if f and g_i are quadratic forms in x, and X is the whole space or a polyhedron, Eq. (1) is a Quadratically Constrained Quadratic Program (QCQP); if f, g_i are convex, it is a convex QCQP (cQCQP);
- if f, g_i are (possibly) nonlinear functions in x, and X is the whole space or a polyhedron, Eq. (1) is a Nonlinear Program (NLP); if f, g_i are convex, it is a convex NLP (cNLP);
- if x is a symmetric matrix of decision variables, f, g_i are linear, and X is the set of all psd matrices, Eq. (1) is a Semidefinite Program (SDP);
- if we impose some integrality constraints on any decision variable on formulations from the classes QP, QCQP, NLP, SDP, we obtain their respective mixed-integer variants MIQP, MIQCQP, MINLP, MISDP.

This taxonomy is by no means complete (see [107, §3.2] and [191]).

2.3 Semantics

As in all formal languages, sentences are given a meaning by replacing variable symbols with other mathematical entities. In the case of MP, its semantics is assigned by an algorithm, called *solver*, which looks for a numerical solution $x^* \in \mathbb{R}^n$ having some optimality properties and satisfying the constraints. For example, BLPs such as Eq. (19) can be solved by the CPLEX solver [87]. This allows users to solve optimization problems just by "modelling" them (i.e. describing them as a MP formulation) instead of having to invent a specific solution algorithm. As a formal descriptive language, MP was shown to be Turing-complete [109,120].

2.4 Reformulations

It is always the case that infinitely many formulations have the same semantics: this can be seen in a number of trivial ways, such as e.g. multiplying some constraint $g_i \leq 0$ by any positive scalar in Eq. (1). This will produce an uncountable number of different formulations with the same feasible and optimal set.

Less trivially, this property is precious insofar as solvers perform more or less efficiently on different (but semantically equivalent) formulations. More

generally, a symbolic transformation on an MP formulation for which one can provide some guarantees on the consequent changes on the feasible or optimal set is called a *reformulation* [107, 111, 110].

Three types of reformulation guarantees will appear in this survey:

- the *exact* reformulation: the optima of the reformulated problem can be mapped easily back to those of the original problem;
- the *relaxation*: the optimal objective function value of the reformulated problem provides a bound (in the optimization direction) on the optimal objective function value of the original problem;
- the *approximating* reformulation: a sequence of formulations based on a parameter which also appears in a "guarantee statement" (e.g. an inequality providing a bound on the optimal objective function value of the original problem); when the parameter tends to infinity, the guarantee proves that formulations in the sequence get closer and closer to an exact reformulation or to a relaxation.

Reformulations are only useful when they can be solved more efficiently than the original problem. Exact reformulations are important because the optima of the original formulation can be retrieved easily. Relaxations are important in order to evaluate the quality of solutions of heuristic methods which provide solutions without any optimality guarantee; moreover, they are crucial in Branch-and-Bound (BB) type solvers (such as e.g. CPLEX). Approximating reformulations are important to devise approximate solution methods for MP problems.

There are some trivial exact reformulations which guarantee that Eq. (1) is much more general than it would appear at first sight: for example, inequality constraints can be turned into equality constraints by the addition of slack or surplus variables; equality constraints can be turned to inequality constraints by listing the constraint twice, once with $\leq$ sense and once with $\geq$ sense; minimization can be turned to maximization by the equation $\min f = -\max -f$ [111, §3.2].

2.4.1 Linearization

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

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

These two reformulation techniques are often used in sequence: one identifies problematic nonlinear terms, linearizes them, and then relaxes the defining constraints. Carrying this out recursively for every term in an NLP [131], and only relaxing the nonlinear defining constraints yields an LP relaxation of an NLP [169, 172, 23].

3 Distance Geometry

DG refers to a foundation of geometry based on the concept of distances instead of those of points and lines (Euclid) or point coordinates (Descartes). The axiomatic foundations of DG were first laid out in full generality by Menger [134], and later organized and systematized by Blumenthal [30]. A *metric space* is a triplet $(\mathbb{X}, \mathbb{D}, d)$, where $\mathbb{X}$ is an abstract set, $\mathbb{D} \subseteq \mathbb{R}_+$, and d is a binary relation $d : \mathbb{X} \times \mathbb{X} \to \mathbb{D}$ obeying the metric axioms:

1. $\forall x, y \in \mathbb{X} \quad d(x, y) = 0 \leftrightarrow x = y$ (identity);
2. $\forall x, y \in \mathbb{X} \quad d(x, y) = d(y, x)$ (symmetry);
3. $\forall x, y, z \in \mathbb{X} \quad d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality).

Based on these notions, one can define sequences and limits (calculus), as well as open and closed sets (topology). For any triplet x, y, z of distinct elements in $\mathbb{X}$, y is *between* x and z if $d(x, y) + d(y, z) = d(x, z)$. This notion of *metric betweenness* can be used to characterize convexity: a subset $\mathbb{Y} \subseteq \mathbb{X}$ is *metrically convex* if, for any two points $x, z \in \mathbb{Y}$, there is at least one point $y \in \mathbb{Y}$ between x and z. The fundamental notion of invariance in metric spaces is that of congruence: two metric spaces $\mathbb{X}, \mathbb{Y}$ are *congruent* if there is a mapping $\mu : \mathbb{X} \to \mathbb{Y}$ such that for all $x, y \in \mathbb{X}$ we have $d(x, y) = d(\mu(x), \mu(y))$.

The word "isometric" is often used as a synonym of "congruent" in many contexts, e.g. with *isometric embeddings* (Sect. 6.2.2). In this survey, we mostly use "isometric" in relation to mappings from graphs to sets of vectors such that the weights of the edges are the same as the length of the segments between the vectors corresponding to the adjacent vertices. In other words, "isometric" is mostly used for partially defined metric spaces — only the distances corresponding to the graph edges are considered.

While a systematization of the axioms of DG were formulated in the twentieth century, DG is pervasive throughout the history of mathematics, starting with Heron's theorem (computing the area of a triangle given the side lengths) [67], going on to Euler's conjecture on the rigidity of (combinatorial) polyhedra [84], Cauchy's creative proof of Euler's conjecture for strictly convex polyhedra [42], Cayley's theorem for inferring point positions from determinants of distance matrices [43], Maxwell's analysis of the stiffness of frames [130], Henneberg's investigations on rigidity of structures [83], Gödel's fixed point theorem for showing that a tetrahedron with nonzero volume can be embedded isometrically (with geodetic distances) on the surface of a sphere [77], Menger's systematization of DG [135], yielding, in particular, the concept of the Cayley-Menger determinant (an extension of Heron's theorem to any dimension, which was used in many proofs of DG theorems), up to Connelly's disproof of Euler's conjecture [49]. A fuller account of many of these achievements is given in [113]. An extension of Gödel's theorem on the sphere embedding in any finite dimension appears in [122].

3.1 The distance geometry problem

Before the widespread use of computers, the main applied problem of DG was to congruently embed finite metric spaces in some vector space. The first mention of the need for isometric embeddings using only a partial set of distances probably appeared in [196]. This need arose from wireless sensor networks: by estimating a set of distances for pairs of sensors which are close enough to establish peer-to-peer communication, is it possible to recover the position for all sensors in the network? Note that (a) distances can be recovered from peer-to-peer communicating pairs by monitoring the amount of battery required to exchange data; and (b) the positions for the sensors are in $\mathbb{R}^K$, with $K = 2$ (usually) or $K = 3$ (sometimes).

Thus we can formulate the main problem in DG.

> Distance Geometry Problem (DGP): given an integer $K > 0$ and a simple undirected graph $G = (V, E)$ with an edge weight function $d : E \to \mathbb{R}_+$, determine whether there exists a *realization* $x : V \to \mathbb{R}^K$ such that:
> $$\forall \{u, v\} \in E \quad \|x(u) - x(v)\| = d(u, v). \tag{2}$$

We let $n = |V|$ and $m = |E|$ in the following.

We can re-state the DGP as follows: given a weighted graph G and the dimension K of a vector space, draw G in $\mathbb{R}^K$ so that each edge is drawn as a straight segment of length equal to its weight. We remark that the realization x, defined as a function, is usually represented as an $n \times K$ matrix $x = (x_{uk} \mid u \in V \wedge k \leq K)$, which may also be seen as an element of $\mathbb{R}^{nK}$.

Notationally, we usually write x_u, x_v and d_{uv}. If the norm used in Eq. (2) is ℓ_2, then the above equation is usually squared, so it becomes a multivariate polynomial of degree two:

$$\forall \{u, v\} \in E \quad \|x_u - x_v\|_2^2 = d_{uv}^2. \tag{3}$$

While most of the distances in this paper will be Euclidean, we shall also mention the so-called *linearizable norms* [51], i.e. ℓ_1 and ℓ_∞, because they can be described using linear forms. We also remark that the input of the DGP can also be represented by a *partial* $n \times n$ *distance matrix* D where only the entries d_{uv} corresponding to $\{u, v\} \in E$ are specified.

Many more notions about the DGP can be found in [117,114].

3.2 Number of solutions

A DGP instance may have no solutions if the given distances do not define a metric, a finite number of solutions if the graph is rigid, or uncountably many solutions if the graph is flexible.

Restricted to the ℓ_2 norm, there are several different notions of rigidity. We only define the simplest, which is easiest to explain intuitively: if we consider the graph as a representation of a joint-and-bar framework, a graph is flexible

if the framework can move (excluding translations and rotations) and rigid otherwise. The formal definition of rigidity of a graph $G = (V, E)$ involves: (a) a mapping D from a realization $x \in \mathbb{R}^{nK}$ to the partial distance matrix

$$\mathsf{D}(x) = (\|x_u - x_v\| \mid \{u, v\} \in E);$$

and (b) the completion $\mathsf{K}(G)$ of G, defined as the complete graph on V. We want to say that G is rigid if, were we to move x ever so slightly (excluding translations and rotations), $\mathsf{D}(x)$ would also vary accordingly. We formalize this idea indirectly: a graph is *rigid* if the realizations in a neighbourhood χ of x corresponding to changes in $\mathsf{D}(x)$ are equal to those in the neighbourhood $\bar{\chi}$ of a realization $\bar{x}$ of $\mathsf{K}(G)$ [114, Ch. 7]. We note that realizations $\bar{x} \in \bar{\chi}$ correspond to small variations in $\mathsf{D}(\mathsf{K}(G))$: this definition makes sense because $\mathsf{K}(G)$ is a complete graph, which implies that its distance matrix has no variable components that can change, and hence $\bar{\chi}$ may only contain congruences.

We obtain the following formal characterization of rigidity [16]:

$$\mathsf{D}^{-1}(\mathsf{D}(x)) \cap \chi = \mathsf{D}^{-1}(\mathsf{D}(\bar{x})) \cap \bar{\chi}. \tag{4}$$

Uniqueness of solution (modulo congruences) is sometimes a necessary feature in applications. Many different sufficient conditions to uniqueness have been found [117, §4.1.1]. By way of example as concerns the number of DGP solutions in graphs, a complete graph has at most one solution modulo congruences, as remarked above. It was proved in [115] that protein backbone graphs have a realization set having power of two cardinality with probability 1. As shown in Fig. 1 (bottom row), a cycle graph on four vertices has uncountably many solutions.

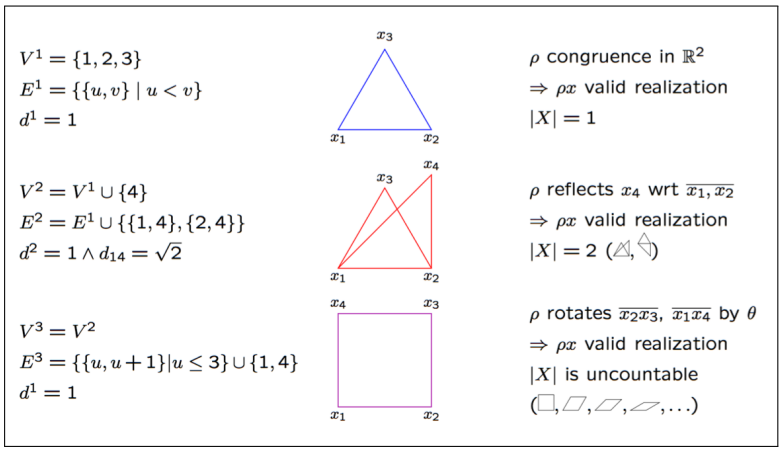

Fig. 1 Instances with one, two, and uncountably many realizations.

On the other hand, the remaining possibility of an infinite but countably many realizations of a DGP instance cannot happen, as shown in Thm. 1.

This result is a corollary of a well-known theorem of Milnor's. It was noted informally in [117, p. 27] without details; we provide a proof here.

Theorem 1 *No DGP instance may have an infinite but countable number of solutions.*

Proof Eq. (3) is a system of m quadratic equations associated with the instance graph G. Let $X \subseteq \mathbb{R}^{nK}$ be the variety associated to Eq. (3). Now suppose X is countable: then no connected component of X may contain uncountably many elements. By the notion of connectedness, this implies that every connected component is an isolated point in X. If X is countable, it must contain a countable numbers of connected components. By [139], the number of connected components of X is finite; in particular, it is bounded by $O(3^{nK})$. Hence the number of connected components of X is finite. Since each is an isolated point, i.e. a single realization of G, $|X|$ is finite. □

3.3 Applications

The DGP is an inverse problem with many applications to science and engineering.

3.3.1 Engineering

When $K = 1$ a typical application is that of clock synchronization [168]. Network protocols for wireless sensor networks are designed so as to save power in communication. When synchronization and battery usage are key, the peer-to-peer communications needed to exchange the timestamp can be limited to the exchange of a single scalar, i.e. the time (or phase) difference. The problem is then to retrieve the absolute times of all of the clocks, given some of the phase differences. This is equivalent to a DGP on the time line, i.e. in a single dimension. We already sketched above the problem of Sensor Network Localization (SNL) in $K \in \{2, 3\}$ dimensions. In $K = 3$ we also have the problem of controlling fleets of Underwater Autonomous Vehicles (UAV), which requires the localization of each UAV in real-time [17, 171].

3.3.2 Science

An altogether different application in $K = 3$ is the determination of protein structure from Nuclear Magnetic Resonance (NMR) experiments [194]: proteins are composed of a linear backbone and some side-chains. The backbone determines a total order on the backbone atoms, by which follow some properties of the protein backbone graph. Namely, the distances from vertex i to vertices $i-1$ and $i-2$ in the order are known almost exactly because of chemical information, and the distance between vertex i and vertex $i-3$ is known approximately because of NMR output. Moreover, some other distances (with longer index difference) may also be known because of NMR — typically, when

the protein folds and two atoms from different folds happen to be close to each other. If we suppose all of these distances are known exactly, we obtain a subclass of DGP which is called Discretizable Molecular DGP (DMDGP). The structure of the graph of a DMDGP instance is such that vertex i is adjacent to its three immediate predecessors in the order: this yields a graph which consists of a sequence of embedded cliques on 4 vertices, the edges of which are called *discretization edges*, with possibly some extra edges called *pruning edges*.

If we had to realize this graph with $K = 2$, we could use *trilateration* [66]: given three points in the plane, compute the position of a fourth point at known distance from the three given points. Trilateration gives rise to a system of equations which has either no solution (if the distance values are not a metric) or a unique solution, since three distances in two dimensions are enough to disambiguate translations, rotations and reflections. Due to the specific nature of the DMDGP graph structure, it would suffice to know the positions of the first three vertices in the order to be able to recursively compute the positions of all other vertices. With $K = 3$, however, there remains one degree of freedom which yields an uncertainty: the reflection.

We can still devise a combinatorial algorithm which, instead of finding a unique solution in $n - K$ trilateration steps, is endowed with back-tracking over reflections. Thus, the DMDGP can be solved completely (meaning that all incongruent solutions can be found) in worst-case exponential time by using the Branch-and-Prune (BP) algorithm [116]. The DMDGP has other very interesting symmetry properties [121], which allow for an *a priori* computation of its number of solutions [115], as well as for generating all of the incongruent solutions from any one of them [144]; moreover, it turns out that BP is Fixed-Parameter Tractable (FPT) on the DMDGP [118].

3.3.3 Machine Learning

So far, we have only listed applications where K is fixed. The focus of this survey, however, is a case where K may vary: if we need to map graphs to vectors in view of preprocessing the input of a ML methodology, we may choose a dimension K appropriate to the methodology and application at hand. See Sect. 9 for an example.

3.4 Complexity

3.4.1 Membership in **NP**

The DGP is clearly a decision problem, and one may ask whether it is in **NP**. As stated above, with real number input in the edge weight function, it is clear that it is not, since the Turing computation model cannot be applied. We therefore consider its rational equivalent, where $d : E \to \mathbb{Q}_+$, and ask the same question. It turns out that, for $K > 1$, we do not know whether the DGP is in

NP: the issue is that the solutions of sets of quadratic polynomials over $\mathbb{Q}$ may well be algebraic irrational. We therefore have the problem of establishing that a realization matrix x with algebraic component verifies Eq. (3) in polynomial time. While some compact representations of algebraic numbers exist [109, §2.3], it is not known how to employ them in the polynomial time verification of Eq. (3). Negative results for the most basic representations of algebraic numbers were derived in [22].

On the other hand, it is known that the DGP is in **NP** for $K = 1$: as this case reduces to realizing graphs on a single real line, the fact that all of the given distances are in $\mathbb{Q}$ means that the distance between any two points on the line is rational: therefore, if one point is rational, then all the others can be obtained as sums and differences of this one point and a set of rational values, which implies that there is always a rational realization. Naturally, verifying whether a rational realization verifies Eq. (3) can be carried out in polynomial time.

3.4.2 **NP***-hardness*

It was proved in [162] that the DGP is **NP**-hard, even for $K = 1$ (reduction from Partition to the DGP on simple cycle graphs, see a detailed proof in [114, §2.4.2]), and hence actually **NP**-complete for $K = 1$. In the same paper [162], with more complicated gadgets it was also shown that the DGP is **NP**-hard for each fixed K and with edge weights restricted to taking values in $\{1, 2\}$ (reduction from 3Sat).

A sketch of an adaptation of the reduction to cycle graphs is given in [197] for DMDGP graphs, showing that they are an **NP**-hard subclass of the DGP. A full proof following a similar idea can be found in [103].

4 Representing data by graphs

It may be obvious to most readers that data can be naturally represented by graphs. This is immediately evident whenever data represent similarities or dissimilarities between entities in a vertex set V. In this section we make this intuition more explicit for a number of other relevant cases.

4.1 Processes

The description of a process, be it chemical, electric/electronic, mechanical, computational, logical or otherwise, is practically always based on a directed graph, or *digraph*, $G = (N, A)$. The set of nodes N represents the various stages of the process, while the arcs in A represent transitions between stages.

Formalizations of this concept may possibly be first ascribed to the organization of knowledge proposed by Aristotle into genera and differences, commonly represented with a tree (a class of digraphs). While no graphical representation of this tree ever came to us from Aristotelian times, the commentator

Porphyry of Tyre (3rd century AD) did refer to a representation which was actually drawn as a tree (at least since the 10th century [178]). Many interesting images can be found in `last-tree.scottbot.net/illustrations/`, see e.g. Fig. 2.

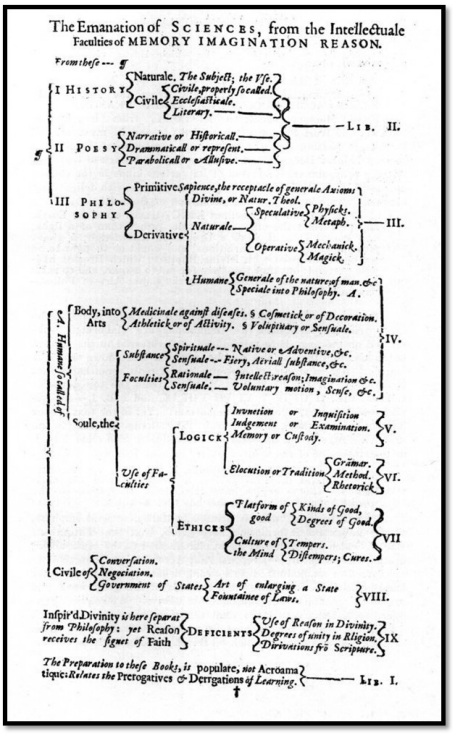

Fig. 2 A tree diagram from F. Bacon's *Advancement of Learning*, Oxford 1640.

A general treatment of process diagrams in mechanical engineering is given in [75]. Bipartite graphs with two node classes representing operations and materials have been used in process network synthesis in chemical engineering [73]. Circuit diagrams are a necessary design tool for any electrical and electronic circuit [167]. Software flowcharts (i.e. graphical description of computer programs) have been used in the design of software so pervasively that one of the most important results in computer science, namely the Böhm-Jacopini's theorem on the expressiveness of universal computer languages, is based on a formalization of the concept of flowchart [31]. The American National Standards Institute (ANSI) set standards for flowcharts and their symbols in the 1960s. The International Organization for Standardization (ISO) adopted the ANSI symbols in 1970 [187]. The *cyclomatic number* $|E| - |V| + 1$ of a graph, namely the size of a cycle basis of the cycle space, was adopted as a measure of process graph complexity very early (see [149, 59, 37, 13] and [99, §2.3.4.1]).

An evalution of flowcharts to process design is the Unified Modelling Language (UML) [147], which was mainly conceived to aid the design of software-based systems, but was soon extended to much more general processes. With respect to flowcharts, UML also models interactions between software systems and hardware systems, as well as with system users and stakeholders. When it is applied to software, UML is a semi-formal language, in the sense that it can automatically produce a set of header files with the description of classes and other objects, ready for code development in a variety of programming languages [108].

4.2 Text

One of the foremost issues in linguistics is the formalization of the rules of grammar in natural languages. On the one hand, text is scanned linearly, word by word. On the other hand, the sense of a sentence becomes apparent only when sentences are organized as trees [45]. This is immediately evident in the computer parsing of formal languages, with a "lexer" which carries out the linear scanning, and a "parser" which organizes the lexical tokens in a *parsing tree* [106]. The situation is much more complicated for natural languages, where no rule of grammar is ever absolute, and any proposal for overarching principles has so many exceptions that it is hard to argue in their favor [142].

The study of natural languages is usually split into syntax (how the sentence is organized), semantics (the sense conveyed by the sentence) and pragmatics (how the context when the sentence is uttered influences the meaning, and the impact that the uttered sentence has on the context itself) [143]. The current situation is that we have been able to formalize rules for natural language syntax (namely turning a linear text string into a parsing tree) fairly well, using probabilistic parsers [126] as well as supervised ML [48]. We are still far from being able to successfully formalize semantics. Semiotics suggested many ways to assign semantics to sentences [65], but none of these is immediately and easily implementable as a computer program.

Two particularly promising suggestions are the organization of knowledge into an evolving encyclopedia, and the representation of the sense of words in a "space" with "semantic axes" (e.g. "good/bad", "white/black", "left/right"...). The first suggestion yielded organized corpora such as WordNet [138], which is a tree representation of words, synonyms and their semantical relations, not unrelated to a Porphyrian tree (Sect. 4.1). There is still a long way to go before the second is successfully implemented, but we see in the Google Word Vectors [137] the start of a promising path (although even easy semantical interpretations, such as analogies, are apparently not so well reflected in these word vectors, despite the publicity [97]).

For pragmatics, the situation is even more dire; some suggestions for representing knowledge and cognition w.r.t. the state of the world are given in [140]. See [185] for more information.

Insofar as graphs are concerned, syntax is organized into tree graphs, and semantics is often organized in corpora that are also trees, or directed acyclic graphs (DAGs), e.g. WordNet and similar.

4.2.1 Graph-of-words

In Sect. 9 we consider a graph representation of sentences known as the *graph-of-words* [159]. Given a sentence s represented as a sequence of words $s = (s_1, \ldots, s_m)$, an n-gram is a subsequence of n consecutive words of s. Each sentence obviously has at most $(m - n + 1)$ n-grams. In a graph-of-words $G = (V, E)$ of order n, V is the set of words in s; two words have an edge only if they appear in the same n-gram; the weight of the edge is equal to the number of n-grams in which the two words appear. This graph may also be enriched with semantic relations between the words, obtained e.g. from WordNet.

4.3 Databases

The most common form of data collection is a database; among the existing database representations, one of the most popular is the tabular form used in spreadsheets and relational databases.

A *table* is a rectangular array A with n rows (the records) and m columns (the features), which is (possibly only partially) filled with values. Specifically, each feature column must have values of the same type (when present). If A_{rf} is filled with a value, we denote this $\mathsf{def}(r, f)$, for each record index r and feature index f. We can represent this array via a bipartite graph $B = (R, F, E)$ where R is the set of records, F is the set of features, and there is an edge $\{r, f\} \in E$ if the (r, f)-th component A_{rf} of A is filled. A label function ℓ assigns the value A_{rf} to the edge $\{r, f\}$. While this is an edge-labelled graph, the labels (i.e. the contents of A) may not always be interpretable as edge weights — so this representation is not yet what we are looking for.

We now assume that there is a symmetric function $d_f : A_{\cdot,f} \times A_{\cdot,f} \to \mathbb{R}_+$ defined over elements of the column $A_{\cdot,f}$: since all elements in a column have the same type, such functions can always be defined in practice. We note that d_f is undefined whenever one of the two arguments is not filled with a value. We can then define a composite function $d : R \times R \to \mathbb{R}_+$ as follows:

$$\forall r \neq s \in R \quad d(r,s) = \begin{cases} \sum\limits_{\substack{f \in F \\ \mathsf{def}(r,f) \wedge \mathsf{def}(s,f)}} d_f(A_{rf}, A_{sf}) \\ \text{undefined if } \exists f \in F \ (\neg\mathsf{def}(r,f) \vee \neg\mathsf{def}(s,f)). \end{cases} \tag{5}$$

Next, we define a graph $G = (R, E')$ over the records R, where

$$E' = \{ \{r, s\} \mid r \neq s \in R \wedge d(r, s) \text{ is defined} \},$$

weighted by the function $d : E' \to \mathbb{R}_+$ defined in Eq. (5). We call G the *database distance graph*. Analysing this graph yields insights about record distributions, similarity and differences.

4.4 Abductive inference

According to [64], there are three main modes of rational thought, corresponding to three different permutations of the concepts "hypothesis" (call this H), "prediction" (call this P), "observation" (call this O). Each of the three permutations singles out a pair of concepts and a remaining concept. Specifically:

1. *deduction*: $\mathrm{H} \wedge \mathrm{P} \rightarrow \mathrm{O}$;
2. *(scientific) induction*: $\mathrm{O} \wedge \mathrm{P} \rightarrow \mathrm{H}$;
3. *abduction*: $\mathrm{H} \wedge \mathrm{O} \rightarrow \mathrm{P}$.

Take for example the most famous syllogism about Socrates being mortal:

- H: "all humans are mortal";
- P: "Socrates is human";
- O: "Socrates is mortal".

The syllogism is an example of deduction: we are given H and P, and deduce O. Note also that deduction is related to *modus ponens*: if we call A the class of all humans and B the class of all mortals, and let s be the constant denoting Socrates, the syllogism can be restated as $(\forall x\ A(x) \subseteq B(x) \wedge A(s)) \rightarrow B(s)$. Deduction infers truths (propositional logic) or provable sentences (first-order and higher-order logic), and is mostly used by logicians and mathematicians.

Scientific induction[1] exploits observations and verifies predictions in order to derive a general hypothesis: if a large quantity of predictions is verified, a general hypothesis can be formulated. In other words, given O and P we infer H. Scientific induction can never provide proofs in sufficiently expressive logical universes, no matter the amount of observations and verified predictions. Any false prediction, however, disproves the hypothesis [154]. Scientific induction is about causality; it is mostly used by physicists and other natural scientists.

Abduction [62] infers educates guesses about a likely state of a known universe from observed facts: given H and O, we infer P. According to [132],

> Deductions lead from rules and cases to facts — the conclusions. Inductions lead toward truth, with less assurance, from cases and facts, toward rules as generalizations, valid for bound cases, not for accidents. Abductions, the apagoge of Aristotle, lead from rules and facts to the hypothesis that the fact is a case under the rule.

According to [64] it can be traced back to Peirce [151], who cited Aristotle as a source. The author of [156] argues that the precise Aristotelian source cited by Peirce fails to make a valid reference to abduction; however, he also concedes that there are some forms of abduction foreshadowed by Aristotle in the texts where he defines definitions.

Let us see an example of abduction. Sherlock Holmes is called on a crime scene where Socrates lies dead on his bed. After much evidence is collected and a full-scale investigation is launched, Holmes ponders some possible hypotheses: for example, all rocks are dead. The prediction that is logically consistent with this hypothesis and the observation that Socrates is dead would be that

[1] Not to be confused with *mathematical induction*.

Socrates is a rock. After some unsuccessful tests using Socrates' remains as a rock, Holmes eliminates this possibility. After a few more untenable suggestions by Dr. Watson, Holmes considers the hypothesis that all humans are mortal. The logically consistent prediction is that Socrates is a man, which, in a dazzling display of investigating abilities, Holmes finds it to be exactly the case. Thus Holmes brilliantly solves the mystery, while Lestrade was just about ready to give up in despair. Abduction is about plausibility; it is the most common type of human inference.

Abduction is also the basis of learning: after witnessing a set of facts, and postulating hypotheses which link them together, we are able to make predictions about the future. Abductions also can, and in fact often turn out to, be wrong, e.g.:

- H: all beans in the bag are white;
- O: there is a white bean next to the bag;
- P: the bean was in the bag.

The white bean next to the bag, however, might have been placed there before the bag was even in sight. With this last example, we note that abductions are inferences often used in statistics. For an observation O, a set $\mathcal{H}$ of hypotheses and a set of possible predictions $\mathcal{P}$, we must evaluate

$$\forall \mathrm{H} \in \mathcal{H}, \mathrm{P} \in \mathcal{P} \quad p_{\mathrm{HP}} = \mathbb{P}(\mathrm{O} \mid \mathrm{O,H\ abduce\ P}),$$

and then choose the pair (H,P) having largest probability p_{HP} (see a simplified example in Fig. 3).

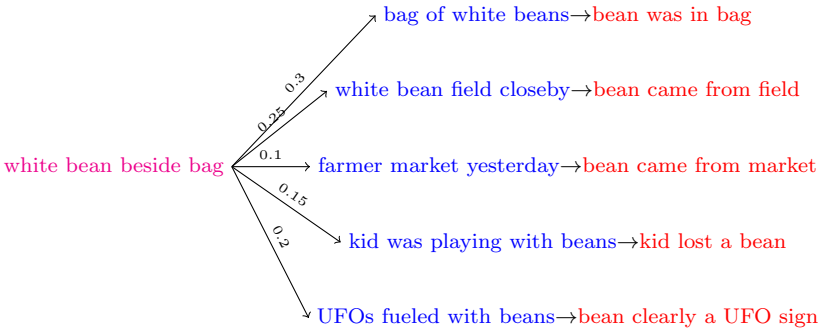

Fig. 3 Evaluating probabilities in abduction. From left to right, observation O abduces the inference H→P.

When more than one observation is collected, one can also compare distributions to make more plausible predictions, see Fig. 4. Abduction appears close to the kind of analysis often required by data scientists.

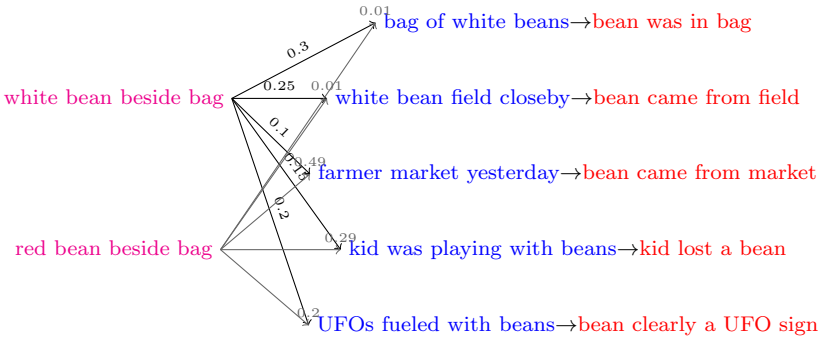

Fig. 4 Probability distributions over abduction inferences assigned to observations.

4.4.1 The abduction graph

We now propose a protocol for modelling good predictions from data, by means of an *abduction graph*. We consider:

- a set $\mathcal{O}$ of observations O;
- a set $\mathcal{I} \subseteq \mathcal{H} \times \mathcal{P}$ of abductive premises, namely pairs (H, P).

First, we note that different elements of $\mathcal{I}$ might be logically incompatible (e.g. there may be contradictory sets of hypotheses or predictions). We must therefore extract a large set of logically compatible subsets of $\mathcal{I}$. Consider the relation $\sim$ on $\mathcal{I}$ with $h \sim k$ meaning that $h, k \in \mathcal{I}$ are logically compatible. This defines a graph $(\mathcal{I}, \sim)$. We then find the largest (or at least large enough) clique $\bar{\mathcal{I}}$ in $(\mathcal{I}, \sim)$.

Next, we define probability distributions p^{O} on $\bar{\mathcal{I}}$ for each $\mathrm{O} \in \mathcal{O}$. We let $E = \{\{\mathrm{O}, \mathrm{O}'\} \mid \delta(p^{\mathrm{O}}, p^{\mathrm{O}'}) \leq \delta_0\}$, where δ evaluates dissimilarities between probability distributions, e.g. δ could be the Kullback-Leibler (KL) divergence [100], and δ_0 a given threshold. Thus E defines a relation on $\mathcal{O}$ if $p^{\mathrm{O}}, p^{\mathrm{O}'}$ are sufficiently similar. We can finally define the graph $\mathcal{F} = (\mathcal{O}, E)$, with edges weighted by δ.

If we think of Sherlock Holmes again, the abduction graph encodes sets of clues compatible with the most likely consistent explanations.

5 Common data science tasks

DS refers to practically every task or problem defined over large amounts of data. Even problems in **P**, and sometimes even those for which there exist linear time algorithms, may take too long when confronted with huge-scale instances. We are not going to concern ourselves here with evaluation problems (such as computing means, variances, higher-order moments or other statistical measures), which are the realm of statistics, but rather with decision problems. In particular, it appears that a very common family of decision problems solved

on large masses of data are those that help people make sense of the data themselves: in other words, classification and clustering.

There is no real functional distinction between the two, as both aim at partitioning the data into a relatively small number of subsets. However, "classification" usually refers to the problem of assigning class labels to data elements, while "clustering" indicates a classification based on the concept of similarity or distance, meaning that similar data elements should be in the same class. This difference is usually more evident in the algorithmic description: classification methods tend to exploit information inherent to elements, while clustering methods consider information relative to pairs of elements. In the rest of this paper, we shall adopt a functional view, and simply refer to "clustering" to indicate both classification and clustering.

Given a set P of n entities and some pairwise similarity function $\delta : P \times P \to \mathbb{R}_+$, clustering aims at finding a set of k subsets $C_1, \ldots, C_k \subseteq P$ such that each cluster contains as many similar entities, and as few dissimilar entities, as possible. Cluster analysis — as a field — grew out of statistics in the course of the second half of the 20th century, encouraged by the advances in computing power. But some early forms of cluster analysis may also be attributed to earlier scientists (e.g. Aristotle, Buffon, Cuvier, Linné [82]).

We note that "clustering on graphs" may refer to two separate tasks.

A. Cluster the vertices of a given graph.
B. Cluster the graphs in a given set.

Both may arise depending on the application at hand. The proposed DG techniques for realizing graphs into vector spaces apply to both of these tasks (see Sect. 9.4.2).

As mentioned above, this paper focuses on transforming graphs into vectors so as to be able to use vector-based methods for classification and clustering. We shall first survey some of these methods. We shall then mention some methods for classifying/clustering graphs directly (i.e. without needing to transform them into vectors first).

5.1 Clustering on vectors

Methods for classification and clustering on vectors are usually seen as part of ML. They are partitioned into unsupervised and supervised learning methods. The former are usually based on some similarity or dissimilarity measure defined over pairs of elements. The latter require a *training set*, which they exploit in order to find a set of optimal parameter values for a parametrized "model" of the data.

5.1.1 The k-means algorithm

The k-means algorithm is a well-known heuristic for solving the following problem [12].

Minimum Sum-of-Squares Clustering (MSSC). Given an integer $k > 0$ and a set $P \subset \mathbb{R}^m$ of n vectors, find a set $\mathcal{C} = \{C_1, \ldots, C_k\}$ of subsets of P such that the function

$$f(\mathcal{C}) = \sum_{j \leq k} \sum_{x \in C_j} \|x - \mathsf{centroid}(C_j)\|_2^2 \tag{6}$$

is minimum, where

$$\mathsf{centroid}(C_j) = \frac{1}{|C_j|} \sum_{x \in C_j} x. \tag{7}$$

It is interesting to note that the MSSC problem can also be seen as a discrete analogue of the problem of partitioning a body into smaller bodies having minimum sum of moments of inertia [170].

The k-means algorithm improves a given initial clustering $\mathcal{C}$ by means of the two following operations:

1. compute centroids $c_j = \mathsf{centroid}(C_j)$ for each $j \leq k$;
2. for any pair of clusters $C_h, C_j \in \mathcal{C}$ and any point $x \in C_h$, if x is closer to c_j than to c_h, move x from C_h to C_j.

These two operations are repeated until the clustering $\mathcal{C}$ no longer changes. Since the only decision operation (i.e. operation 2) is effective only if it decreases $f(\mathcal{C})$, it follows that k-means is a local descent algorithm. In particular, this very simple analysis offers no guarantee on the approximation of the objective function. For more information on the k-means algorithm, see [29].

k-means is an unsupervised learning technique [91], insofar as it does not rest on a data model with parameters to be estimated prior to actually finding clusters. Moreover, the number "k" of clusters must be known *a priori*.

5.1.2 Artificial Neural Networks

An ANN is a parametrized model for representing an unknown function. Like all such models, it needs data in order to estimate suitable values for the parameters: this puts ANNs in the category of supervised ML. An ANN consists of two MP formulations defined over a graph and a training set.

An ANN is formally defined as a triplet $\mathcal{N} = (G, T, \phi)$, where:

- $G = (V, A)$ is a directed graph, with a node weight function $b : V \to \mathbb{R}$ (threshold at a node), and an edge weight function $w : A \to \mathbb{R}$ (weight on an arc); moreover, a subset $I \subset V$ of *input nodes* with $|I| = n$ and a subset $O \subset V$ of *output nodes* with $|O| = k$ are given in G;
- $T = (X, Y)$ is the training set, where $X \subset \mathbb{R}^n$ (input set), $Y \subset \mathbb{R}^k$ (output set), and $|X| = |Y|$;
- $\phi : \mathbb{R} \to \mathbb{R}$ is the *activation function* (many common activation functions map injectively into $[0, 1]$).

The two MP formulations assigned to an ANN describe the *training problem* and the *evaluation problem*. In the training problem, appropriate values for b, w are found using T. In the evaluation problem, a given input vector in $\mathbb{R}^n$ (usually not part of the input training set X) is mapped to an output vector in $\mathbb{R}^k$. The training problem decides values for the ANN parameters when seen as a model for an unknown function mapping the training input X to the training output Y. After the model is trained, it can be evaluated on new (unseen) input.

In the following, we use standard notation on graphs. For a node $i \in V$ we let $N^-(i) = \{j \in V \mid (j,i) \in A\}$ be the *inward star* and $N^+(i) = \{j \in V \mid (i,j) \in A\}$ be the *outward star* of i. For undirected graphs $G = (V, E)$, we let $N(i) = \{j \in V \mid \{i,j\} \in E\}$ be the *star* of i. Moreover, for a tensor $s_{i_1,\ldots,i_r}$, where $i_j \in I_j$ for each $j \leq r$, we denote a *slice* of s, defined by subsets $J_j \subseteq I_j$ for some $j \leq r$, by $s[J_1] \cdots [J_r]$.

We discuss the evaluation phase first. Given values for w, b and an input vector $x \in \mathbb{R}^n$, we decide a node weight function u over V as follows:

$$u_I = x \tag{8}$$

$$\forall j \in V \smallsetminus I \quad u_j = \phi\Big(\sum_{i \in N^-(j)} w_{ij} u_i + b_j\Big). \tag{9}$$

We remark that Eq. (9) is not an optimization but a decision problem. Nonetheless, it is a MP formulation (formally with zero objective function). After solving Eq. (9), one retrieves in particular $u[O]$, which correspond to an output vector in $u[0] = y \in \mathbb{R}^k$. When G is acyclic, this decision problem reduces to a simple computation, which "propagates" the values of u from the input nodes and forward through the network until they reach the output nodes. If G is not acyclic, different solution methods must be used [14, 70, 79].

The *training problem* is given in Eq. (10). We let N be the index set for the training pairs (x, y) in T (we recall that $|X| = |Y|$), and introduce a 2-dimensional tensor v of decision variables indexed by N and V.

$$\left.\begin{array}{rl} \min\limits_{w,b,v} \mathsf{dist}(v[N][O], Y) & \\ \forall t \in N & v_t[I] = X \\ \forall t \in N, j \in V \smallsetminus I & v_{tj} = \phi_j\Big(\sum\limits_{i \in N^-(j)} w_{ij} v_{ti} + b_j\Big), \end{array}\right\} \tag{10}$$

where $\mathsf{dist}(A, B)$ is a dissimilarity function taking dimensionally consistent tensor arguments A, B, which becomes closer to zero as A and B get closer. The solution of the training problem yields optimal values w^*, b^* for the arc weights and node biases.

The training problem is in general a nonconvex optimization problem (because of the products between w and v, and of the ϕ functions occurring in equations), which may have multiple global optima: finding them with state-of-the-art methods might require exponential time. For specific types of graphs and choices of objective function $\mathsf{dist}(\cdot,\cdot)$, the training problem may turn out to be convex. For example, if G is a DAG, $V = I \dot\cup O$, the induced subgraphs

$G[I]$ and $G[O]$ are empty (i.e. they have no arcs), the activation functions are all sigmoids $\phi(z) = (1 + \exp(-z))^{-1}$, and $\mathsf{dist}(\cdot, \cdot)$ is the negative logarithm of the likelihood functions

$$\prod_{t \in N} \phi(w^\top x^t + b_i)^{y_t} (1 - \phi(w^\top x^t + b_i))^{1-y_t}$$

summed over all output nodes $i \in O$, then it can be shown that the training problem is convex [94,166].

In contemporary treatments of ANNs, the underlying graph G is almost always assumed to be a DAG. In modern Application Programming Interfaces (API), the acyclicity of G is enforced by recursively replacing v_{tj} with the corresponding expression in $\phi(\cdot)$.

Most algorithms usually solve Eq. (10) only locally and approximately. Usually, they employ a technique called Stochastic Gradient Descent (SGD) [34]. This is a form of gradient descent where, at each iteration, the gradient of a multivariate function is estimated by partial gradients with respect to a randomly chosen subset of variables [141, p. 100].

The functional definition of an optimum for the training problem Eq. 10 is poorly understood, as finding precise local (or global) optima is considered "overfitting". In other words, global or almost global optima of Eq. (10) lead to evaluations which are possibly perfect for pairs in the training set, but unsatisfactory for yet unseen input. Currently, finding "good" optima of ANN training problems is mostly based on experience, although a considerable effort is under way in order to reach a sound definition of optimum [57,198,80,46].

The main reason why ANNs are so popular today is that they have proven hugely successful at image recognition [79], and also extremely good at accomplishing other tasks, including natural language processing [48]. Many efficient applications of ANNs to complex tasks involve interconnected networks of ANNs of many different types [25].

ANNs originated from an attempt to simulate neuronal activity in the brain: should the attempt prove successful, it would realize the old human dream of endowing a machine with human intelligence [193]. While ANNs today display higher precision than humans in some image recognition tasks, they may also be easily fooled by a few appropriately positioned pixels of different colors, which places the realization of "human machine intelligence" still rather far in the future — or even unreachable, e.g. if Penrose's hypothesis of quantum activity in the brain influencing intelligence at a macroscopic level holds [152]. For more information about ANNs, see [164,79].

5.2 Clustering on graphs

While we argue in this paper that DG techniques allow the use of vector clustering methods to graph clustering, there also exist methods for clustering on graphs directly. We discuss two of them, both applicable to the task of clustering vertices of a given graph (Task A on p. 19).

5.2.1 Spectral clustering

Consider a connected graph $G = (V, E)$ with an edge weight function $w : E \to \mathbb{R}_+$. Let A be the *adjacency matrix* of G, with $A_{ij} = w_{ij}$ for all $\{i, j\} \in E$, and $A_{ij} = 0$ otherwise. Let Δ be the diagonal *weighted degree matrix* of G, with $\Delta_{ii} = \sum_{j \neq i} A_{ij}$ and $\Delta_{ij} = 0$ for all $i \neq j$. The *Laplacian* of G is defined as $L = \Delta - A$.

Spectral clustering aims at finding a minimum balanced cut $U \subset V$ in G by looking at the spectrum of the Laplacian of G. For now, we give the word "balanced" only an informal meaning: it indicates the fact that we would like clusters to have approximately the same cardinality (we shall be more precise below). Removing the *cutset* $\delta(U)$ (i.e. the set of edges between U and $V \setminus U$) from G yields a two-way partitioning of V. If $|\delta(U)|$ is minimum over all possible cuts U, then the two sets $U, V \setminus U$ should both intuitively induce subgraphs $G[U]$ and $G[V \setminus U]$ having more edges than those in $\delta(U)$. In other words, the criterion we are interested in maximizes the intra-cluster edges of the subgraphs of G induced by the cluster while minimizing the inter-cluster edges of the corresponding cutsets.

We remark that each of the two partitions can be recursively partitioned again. A recursive clustering by two-way partitioning is a general methodology which is part of a family of *hierarchical* clustering methods [163]. So the scope of this section is not limited to generating two clusters only.

For simplicity, we only discuss the case with unit edge weights, although the generalization to general weights is not problematic. Thus, Δ_{ii} is the degree of vertex $i \in V$. We model a balanced partition $\{B, C\}$ corresponding to a minimum cut by means of decision variables $x_i = 1$ if $i \in B$ and $x_i = -1$ if $i \in C$, for each $i \leq n$, with $n = |V|$. Then $f(x) = \frac{1}{4} \sum_{\{i,j\} \in E} (x_i - x_j)^2$ counts the number of intercluster edges between B and C. We have:

$$\begin{aligned} 4f(x) &= \sum_{\{i,j\} \in E} (x_i^2 + x_j^2) - 2 \sum_{\{i,j\} \in E} x_i x_j = \sum_{\{i,j\} \in E} 2 - \sum_{i,j \leq n} x_i a_{ij} x_j = \\ &= 2|E| - x^\top A x = \sum_{i \leq n} x_i d_i x_i - x^\top A x = x^\top (\Delta - A) x = x^\top L x, \end{aligned}$$

whence $f(x) = \frac{1}{4} x^\top L x$. We can therefore obtain cuts with minimum $|\delta(B)|$ by minimizing $f(x)$.

We can now give a more precise meaning to the requirement that partitions are *balanced*: we require that x must satisfy the constraint

$$\sum_{i \leq n} x_i = 0. \tag{11}$$

Obviously, Eq. (11) only ensures equal cardinality partitions on graphs having an even number of vertices. However, we relax the integrality constraints $x \in \{-1, 1\}^n$ to $x \in [-1, 1]^n$, so $\sum_{i \leq n} x_i = 0$ is applicable to any graph. With this relaxation, the values of x might be fractional. We shall deal with this issue

by rounding them to $\{-1, 1\}$ after obtaining the solution. We also note that the constraint

$$x^\top x = \|x\|_2^2 = n \tag{12}$$

holds for $x \in \{-1, 1\}^n$, and so it provides a strengthening of the continuous relaxation to $x \in [-1, 1]^n$. We therefore obtain a relaxed formulation of the minimum balanced two-way partitioning problem as follows:

$$\left.\begin{array}{rl} \min\limits_{x \in [-1,1]^n} & \frac{1}{4} x^\top L x \\ \text{s.t.} & \mathbf{1}^\top x = 0 \\ & \|x\|_2^2 = n. \end{array}\right\} \tag{13}$$

We remark that, by construction, L is a *diagonally dominant* (dd) symmetric matrix with non-negative diagonal, namely it satisfies

$$\forall i \leq n \quad L_{ii} \geq \sum_{j \neq i} |L_{ij}| \tag{14}$$

(in fact, L satisfies Eq. (14) at equality). Since all dd matrices are also psd [186], $f(x)$ is a convex function. This means that Eq. (13) is a cQP, which can be solved at global optimality in polynomial time [175].

By [68], there is another polynomial time method for solving Eq. (14), which is generally more efficient than solving a cQP in polynomial time using a Nonlinear Programming (NLP) solver. This method concerns the second-smallest eigenvalue of L (called *algebraic connectivity*) and its corresponding eigenvector. Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the ordered eigenvalues of L and $u_1, \ldots, u_n$ be the corresponding eigenvectors, normalized so that $\|u_i\|_2^2 = n$ for all $i \leq n$. It is known that $u_1 = \mathbf{1}$, $\lambda_1 = 0$ and, if G is connected, $\lambda_2 > 0$ [136,32]. By the definition of eigenvalue and eigenvector, we have

$$\forall i \leq n \quad Lu_i = \lambda_i u_i \quad \Rightarrow \quad {u_i}^\top L u_i = \lambda_i {u_i}^\top u_i = \lambda_i \|u_i\|_2^2 = \lambda_i n. \tag{15}$$

Because of the orthogonality of the eigenvectors, if $i \geq 2$ we have $u_i u_1 = 0$, which implies $u_2 \mathbf{1} = 0$ (i.e. u_2 satisfies Eq. (11)). We recall that eigenvectors are normalized so that $\|u_i\|_2^2 = n$ for all $i \leq n$ (in particular, u_2 satisfies Eq. (12)). By Eq. (15), since $\lambda_1 = 0$, λ_2 yields the smallest nontrivial objective function value $\frac{n}{4}\lambda_2$ with solution $\bar{x} = u_2$, which is therefore a solution of Eq. (13).

Theorem 2 *The eigenvector u_2 corresponding to the second smallest eigenvalue λ_2 of the graph Laplacian L is an optimal solution to Eq.* (13)*.*

Proof Since the eigenvectors $u_1, \ldots, u_n$ are an orthogonal basis of $\mathbb{R}^n$, we can express an optimal solution as $\bar{x} = \sum_i c_i u_i$. Thus,

$$x^\top L x = \sum_{i,j} c_i c_j u_i^\top Q u_j = \sum_{i,j} c_i c_j \lambda_j u_i^\top u_j = n \sum_{i>1} c_i^2 \lambda_i. \tag{16}$$

The last equality in Eq. (16) follows because $Lu_i = \lambda_i u_i$ for all $i \leq n$, $u_i^\top u_j = 0$ for each $i \neq j$, and $\lambda_1 = 0$. Since $u_1 = \mathbf{1}$ and by eigenvector orthogonality,

letting $\mathbf{1}^\top \bar{x} = 0$ yields $c_1 = 0$. Lastly, requiring $\|\bar{x}\|_2 = n$, again by eigenvector orthogonality, yields

$$\begin{aligned} \big\| \sum_{i>1} c_i u_i \big\|_2^2 &= \big\langle \sum_{i>1} c_i u_i, \sum_{j>1} c_j u_j \big\rangle = \sum_{i,j>1} c_i c_j \langle u_i, u_j \rangle \\ &= \sum_{i>1} c_i^2 \|u_i\|_2^2 = n \sum_{i>1} c_i^2 = n. \end{aligned} \tag{17}$$

After replacing c_i^2 by y_i in Eq. (16)-(17), we can reformulate Eq. (13) as

$$n \min \big\{ \sum_{i>1} \lambda_i y_i \mid \sum_{i>1} y_i = 1 \wedge y \geq 0 \big\},$$

which is equivalent to finding the convex combination of $\lambda_2, \ldots, \lambda_n$ with smallest value. Since $\lambda_2 \leq \lambda_i$ for all $i > 2$, the smallest value is achieved at $y_2 = 1$ and $y_i = 0$ for all $i > 2$. Hence $\bar{x} = u_2$ as claimed. □

Normally, the components of $\bar{x}$ obtained this way are not in $\{-1, 1\}$. We round $\bar{x}_i$ to its closest value in $\{-1, 1\}$, breaking ties in such a way as to keep the bisection balanced. We then obtain a practically efficient approximation of the minimum balanced cut.

5.2.2 Modularity clustering

Modularity, first introduced in [146], is a measure for evaluating the quality of a clustering of vertices in a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_+$ on the edges. We let $n = |V|$ and $m = |E|$. Given a vertex clustering $\mathcal{C} = (C_1, \ldots, C_k)$, where each $C_i \subseteq V$, $C_i \cap C_j = \varnothing$ for each $i \neq j$, and $\bigcup_i C_i = V$, the *modularity* of $\mathcal{C}$ is the proportion of edges in E that fall within a cluster minus the expected proportion of the same quantity if edges were distributed at random while keeping the vertex degrees constant. This definition is not so easy to understand, so we shall assume for simplicity that $w_{uv} = 1$ for all $\{u, v\} \in E$ and $w_{uv} = 0$ otherwise. We give a more formal definition of modularity, and comment on its construction.

The "fraction of the edges that fall within a cluster" is

$$\frac{1}{m} \sum_{i \leq k} \sum_{\substack{u,v \in C_k \\ \{u,v\} \in E}} 1 = \frac{1}{2m} \sum_{\substack{i \leq k \\ (u,v) \in (C_k)^2}} w_{uv}$$

where $w_{uv} = w_{vu}$ turns out to be the (u, v)-th component of the $n \times n$ symmetric incidence matrix of the edge set E in $V \times V$ — thus we divide by $2m$ rather than m in the right hand side (RHS) of the above equation. The "same quantity if edges were distributed at random while keeping the vertex degrees constant" is the probability that a pair of vertices u, v belongs to the edge set of a random graph on V. If we were computing this probability over random graphs sampled uniformly over all graphs on V with m edges, this probability would be $1/m$; but since we only want to consider graphs with the same

degree sequence as G, the probability is $\frac{|N(u)|\,|N(v)|}{2m}$ [105]. Here is an informal explanation: given vertices u, v, there are $k_u = |N(u)|$ "half-edges" out of u, and $k_v = |N(v)|$ out of v, which could come together to form an edge between u and v (over a total of $2m$ "half-edges"). Thus we obtain a modularity

$$\mu(\mathcal{C}) = \frac{1}{2m} \sum_{\substack{(u,v)\in C^2 \\ C\in\mathcal{C}}} (w_{uv} - k_u k_v/(2m))$$

for the clustering $\mathcal{C}$.

We now introduce binary variables x_{uv} which have value 1 if $u, v \in V$ are in the same cluster, and 0 otherwise. This allows us to rewrite the modularity as:

$$\begin{aligned} \mu(x) &= \frac{1}{2m} \sum_{u\neq v\in V} (w_{uv} - k_u k_v/(2m)) x_{uv} \\ &= \frac{1}{m} \sum_{u<v\in V} (w_{uv} - k_u k_v/(2m)) x_{uv}. \end{aligned} \tag{18}$$

Following [10], we can reformulate the modularity maximization problem to a clique partitioning problem with the following formulation:

$$\left.\begin{array}{rr} \max & \mu(x) \\ \forall 1 \le i < j < k \le n & x_{ij} + x_{jk} - x_{ik} \le 1 \\ \forall 1 \le i < j < k \le n & x_{ij} - x_{jk} + x_{ik} \le 1 \\ \forall 1 \le i < j < k \le n & -x_{ij} + x_{jk} + x_{ik} \le 1 \\ \forall 1 \le i < j \le n & x_{ij} \in \{0,1\}, \end{array}\right\} \tag{19}$$

which is a BLP formulation. The weighted variant of this problem yields a formulation like Eq. (19) where w are the edge weights and $k_u = \sum_{\{u,v\}\in E} w_{uv}$ for all $v \neq u$ in V. Another variant for graphs including loops and multiple edges is described in [39]. We note that, by Eq. (19), maximizing modularity does not require the number of clusters to be known *a priori*.

There is a large literature about modularity maximization and its solution methods: for a survey, see [71, §VI]. Solution methods based on MP are of particular interest to the topics of this survey. A BLP formulation similar to Eq. (19) was proposed in [38]. Another BLP formulation with different sets of decision variables (requiring the number of clusters to be known *a priori*) was proposed in [195]. Some column generation approaches, which scale better in size w.r.t. previous formulations, were proposed in [10]. Some MP based heuristics are discussed in [40, 41, 11].

6 Robust solution methods for the DGP

In this section we discuss some solution methods for the DGP which can be extended to deal with cases where distances are uncertain, noisy or wrong. Most of the methods we present are based on MP. We also discuss a different (non-MP based) class of methods in Sect. 6.2, in view of their computational efficiency.

6.1 Mathematical programming based methods

DGP solution methods based on MP are robust to noisy or wrong data because MP allows for: (a) modification of the objective and constraints; (b) adjoining of side constraints. Moreover, although we do not review these here, there are MP-based methodologies for ensuring robustness of solutions [24], probabilistic constraints [153], and scenario-based stochasticity [28], which can be applied to the formulations in this section.

6.1.1 Unconstrained quartic formulation

A system of equations such as Eq. (3) is itself a MP formulation with objective function identically equal to zero, and $X = \mathbb{R}^{nK}$. It therefore belongs to the QCP class. In practice, solvers for this class perform rather poorly when given Eq. (3) as input [102]. Much better performances can be obtained by solving the following unconstrained formulation:

$$\min \sum_{\{u,v\}\in E} \big(\|x_u - x_v\|_2^2 - d_{uv}^2\big)^2. \tag{20}$$

We note that Eq. (20) consists in the minimization of a polynomial of degree four. It belongs to the class of nonconvex NLP formulations. In general, this is an **NP**-hard class [109], which is not surprising, as it formulates the DGP which is itself an **NP**-hard problem. Very good empirical results can be obtained on the DGP by solving Eq. (20) with a local NLP solver (such as e.g. IPOPT [47] or SNOPT [76]) from a good starting point [102]. This is the reason why Eq. (20) is very important: it can be used to "refine" solutions obtained with other methods, as it suffices to let such solutions be starting points given to a local solver acting on Eq. (20).

Even if the distances d_{uv} are noisy or wrong, optimizing Eq. (20) can yield good approximate realizations. If the uncertainty on the distance values is modelled using an interval $[d_{uv}^L, d_{uv}^U]$ for each edge $\{u, v\}$, the following function [119] can be optimized instead of Eq. (20):

$$\min \sum_{\{u,v\}\in E} \big(\max(0, (d_{uv}^L)^2 - \|x_u - x_v\|_2^2) + \max(0, \|x_u - x_v\|_2^2 - (d_{uv}^U)^2)\big). \tag{21}$$

The DGP variant where distances are intervals instead of values is known as the INTERVAL DGP (iDGP) [78,104].

Note that Eq. (21) involves binary max functions with two arguments. Relatively few MP user interfaces/solvers would accept this function. To overcome this issue, we linearize (see Sect. 2.4.1) the two max terms by two sets of added decision variables y, z, and obtain

$$\left.\begin{array}{rr} \min \sum_{\{u,v\}\in E} (y_{uv} + z_{uv}) & \\ \forall \{u,v\} \in E \qquad \|x_u - x_v\|_2^2 \geq (d_{uv}^L)^2 - y_{uv} & \\ \forall \{u,v\} \in E \qquad \|x_u - x_v\|_2^2 \leq (d_{uv}^U)^2 + z_{uv} & \\ y, z \geq 0, & \end{array}\right\} \tag{22}$$

which follows from Eq. (21) because of the objective function direction, and because $a \geq \max(b, c)$ is equivalent to $a \geq b \wedge a \geq c$. We note that Eq. (22) is no longer an unconstrained quartic, however, but a QCP. It expresses a minimization of penalty variables to the quadratic inequality system

$$\forall \{u, v\} \in E \quad (d^L_{uv})^2 \leq \|x_u - x_v\|_2^2 \leq (d^U_{uv})^2. \tag{23}$$

We also note that many local NLP solvers take very arbitrary functions in input (such as functions expressed by computer code), so the reformulation Eq. (22) may be unnecessary when only locally optimal solutions of Eq. (21) are needed.

6.1.2 Constrained quadratic formulations

We propose two formulations in this section. The first is derived directly from Eq. (3):

$$\left.\begin{array}{r} \min \sum\limits_{\{u,v\} \in E} s_{uv}^2 \\ \forall \{u, v\} \in E \; \|x_u - x_v\|_2^2 = d_{uv}^2 + s_{uv}. \end{array}\right\} \tag{24}$$

We note that Eq. (24) is a QCQP formulation. Similarly to Eq. (22) it uses additional variables to penalize feasibility errors w.r.t. (3). Differently from Eq. (22), however, it removes the need for two separate variables to model slack and surplus errors. Instead, s_{uv} is unconstrained, and can therefore take any value. The objective, however, minimizes the sum of the squares of the components of s. In practice, Eq. (24) performs much better than Eq. (3); on average, the performance is comparable to that of Eq. (20). We remark that Eq. (24) has a convex objective function but nonconvex constraints.

The second formulation we propose is an exact reformulation of Eq. (20). First, we replace the minimization of squared errors by absolute values, yielding

$$\min \sum_{\{u,v\} \in E} \big| \|x_u - x_v\|_2^2 - d_{uv}^2 \big|,$$

which clearly has the same set of global optima as Eq. (20). We then rewrite this similarly to Eq. (22) as follows:

$$\left.\begin{array}{rr} \min \sum\limits_{\{u,v\} \in E} (y_{uv} + z_{uv}) & \\ \forall \{u, v\} \in E & \|x_u - x_v\|_2^2 \geq d_{uv}^2 - y_{uv} \\ \forall \{u, v\} \in E & \|x_u - x_v\|_2^2 \leq d_{uv}^2 + z_{uv} \\ & y, z \geq 0, \end{array}\right\}$$

which, again, does not change the global optima. Next, we note that we can fix $z_{uv} = 0$ without changing global optima, since they all have the property that $z_{uv} = 0$. Now we replace y_{uv} in the objective function by $d_{uv}^2 - \|x_u - x_v\|_2^2$, which we can do without changing the optima since the first set of constraints reads $y_{uv} \geq d_{uv}^2 - \|x_u - x_v\|_2^2$. We can discard the constant d_{uv}^2 from the

objective, since adding constants to the objective does not change optima, and change $\min -f$ to $-\max f$, yielding:

$$\left.\begin{array}{rl} \max & \sum\limits_{\{u,v\}\in E} \|x_u - x_v\|_2^2 \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 \leq d_{uv}^2, \end{array}\right\} \tag{25}$$

which is a QCQP known as the "push-and-pull" formulation of the DGP, since the constraints ensure that x_u, x_v are pushed closer together, while the objective attempts to pull them apart [133, §2.2.1].

Contrariwise to Eq. (24), Eq. (25) has a nonconvex (in fact, concave) objective function and convex constraints. Empirically, this often turns out to be somewhat easier than tackling the reverse situation. The theoretical justification is that finding a feasible solution in a nonconvex set is a hard task in general, whereas finding local optima of a nonconvex function in a convex set is tractable: the same cannot be said for global optima, but in practice one is often satisfied with "good" local optima.

6.1.3 Semidefinite programming

SDP is linear optimization over the cone of psd matrices, which is convex: if A, B are two psd matrices, $C = \alpha A + (1-\alpha)B$ is psd for $\alpha \in [0,1]$. Suppose there is $x \in \mathbb{R}^n$ such that $x^\top C x < 0$. Then $\alpha x^\top A x + (1-\alpha) x^\top B x < 0$, so $0 \leq \alpha x^\top A x < -(1-\alpha) x^\top B x \leq 0$, i.e. $0 < 0$, which is a contradiction, hence C is also psd, as claimed. Therefore, SDP is a subclass of cNLP.

The SDP formulation we propose is a relaxation of Eq. (3). First, we write $\|x_u - x_v\|_2^2 = \langle x_u, x_u\rangle + \langle x_v, x_v\rangle - 2\langle x_u, x_v\rangle$. Then we linearize all of the scalar products by means of additional variables X_{uv}:

$$\begin{aligned} \forall \{u,v\} \in E \quad X_{uu} + X_{vv} - 2X_{uv} &= d_{uv}^2 \\ X &= xx^\top. \end{aligned}$$

We note that $X = xx^\top$ constitutes the whole set of defining constraints $X_{uv} = \langle x_u, x_v\rangle$ (for each $u, v \leq n$) introduced by the linearization procedure (Sect. 2.4.1).

The relaxation we envisage does not entirely drop the defining constraints, as in Sect. 2.4.1. Instead, it relaxes them from $X - xx^\top = 0$ to $X - xx^\top \succeq 0$. In other words, instead of requiring that all of the eigenvalues of the matrix $X - xx^\top$ are zero, we simply require that they should be ≥ 0. Moreover, since the original variables x do not appear anywhere else, we can simply require $X \succeq 0$, obtaining:

$$\left.\begin{array}{rl} \forall \{u,v\} \in E \; X_{uu} + X_{vv} - 2X_{uv} &= d_{uv}^2 \\ X &\succeq 0. \end{array}\right\} \tag{26}$$

The SDP relaxation in Eq. (26) has the property that it provides a solution $\bar{X}$, which is an $n \times n$ symmetric matrix. Spectral decomposition of $\bar{X}$ yields $P \Lambda P^\top$, where P is a matrix of eigenvectors and $\Lambda = \mathsf{diag}(\lambda)$ where λ is a

vector of eigenvalues of $\bar{X}$. Since $\bar{X}$ is psd, $\lambda \geq 0$, which means that $\sqrt{\Lambda}$ is a real matrix. Therefore, by setting $Y = P\sqrt{\Lambda}$ we have that

$$YY^\top = (P\sqrt{\Lambda})(P\sqrt{\Lambda})^\top = P\sqrt{\Lambda}\sqrt{\Lambda}P^\top = P\Lambda P^\top = \bar{X},$$

which implies that $\bar{X}$ is the Gram matrix of Y. Thus we can take Y to be a realization satisfying Eq. (3). The only issue is that Y, as an $n \times n$ matrix, is a realization in n dimensions rather than K. Naturally, $\mathsf{rk}(Y) = \mathsf{rk}(\bar{X})$ need not be equal to n, but could be lower; in fact, in order to find a realization of the given graph, we would like to find a solution $\bar{X}$ with rank at most K. Imposing this constraint is equivalent to asking that $X = xx^\top$ (which have been relaxed in Eq. (26)).

We note that Eq. (26) is a pure feasibility problem. Every SDP solver, however, also accepts an objective function as input. In absence of a "natural" objective in a pure feasibility problem, we can devise one to heuristically direct the search towards parts of the psd cone which we believe might contain "good" solutions. A popular choice is

$$\begin{aligned} \min \mathsf{tr}(X) = \min \mathsf{tr}(P\Lambda P^\top) = \min \mathsf{tr}(PP^\top\Lambda) = \\ = \min \mathsf{tr}(PP^{-1}\Lambda) = \min \lambda_1 + \cdots + \lambda_n, \end{aligned}$$

where tr is the trace, the first equality follows by spectral decomposition (with P a matrix of eigenvectors and Λ a diagonal matrix of eigenvalues of X), the second by commutativity of matrix products under the trace, the third by orthogonality of eigenvectors, and the last by definition of trace. This aims at minimizing the sum of the eigenvalues of X, hoping this will decrease the rank of $\bar{X}$.

For the DGP applied to protein conformation (Sect. 3.3.2), the objective function

$$\min \sum_{\{u,v\} \in E} (X_{uu} + X_{vv} - 2X_{uv})$$

was empirically found to be a good choice [61, §2.1]. More (unpublished) experimentation showed that the scalarization of the two objectives:

$$\min \sum_{\{u,v\} \in E} (X_{uu} + X_{vv} - 2X_{uv}) + \gamma \mathsf{tr}(X), \tag{27}$$

with γ in the range $O(10^{-2})$-$O(10^{-3})$, is a good objective function for solving Eq. (26) when it is applied to protein conformation.

In the majority of cases, solving SDP relaxations does not yield solution matrices with rank K, even with objective functions such as Eq. (27). We discuss methods for constructing an approximate rank K realization from $\bar{X}$ in Sect. 7.

SDP is one of those problems which is not known to be in **P** (nor **NP**-complete) in the Turing machine model. It is, however, known that SDPs can be solved in polynomial time up to a desired error tolerance $\epsilon > 0$, with the complexity depending on $\frac{1}{\epsilon}$ as well as the instance size. Currently, however,

the main issue with SDP is technological: state-of-the art solvers do not scale all that well with size. One of the reasons is that K is usually fixed (and small) with respect to n, so the while the original problem has $O(n)$ variables, the SDP relaxation has $O(n^2)$. Another reason is that the Interior Point Method (IPM), which often features as a "state of the art" SDP solver, has a relatively high computational complexity [155]: a "big oh" notation estimate of $O(\max(m,n)mn^{2.5})$ is given in Bubeck's blog at ORFE, Princeton.[2]

6.1.4 Diagonally dominant programming

In order to address the size limitations of SDP, we employ some interesting linear approximations of the psd cone proposed in [125,5]. An $n \times n$ real symmetric matrix X is diagonally dominant (dd) if

$$\forall i \leq n \quad \sum_{j \neq i} |X_{ij}| \leq X_{ii}. \tag{28}$$

As remarked in Sect. 5.2.1, it is well known that every dd matrix is also psd, while the converse may not hold. Specifically, the set of dd matrices form a sub-cone of the cone of psd matrices [18].

The interest of dd matrices is that, by linearization of the absolute value terms, Eq. (28) can be reformulated so it becomes linear: we introduce an added matrix T of decision variables, then write:

$$\forall i \leq n \quad \sum_{j \neq i} T_{ij} \leq X_{ii} \tag{29}$$

$$-T \leq X \leq T, \tag{30}$$

which are linear constraints equivalent to Eq. (28) [5, Thm. 10]. One can see this easily whenever $X \geq 0$ or $X \leq 0$. Note that

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

$$\forall i \leq n \quad X_{ii} \geq \sum_{j \neq i} T_{ij} \geq \sum_{j \neq i} -X_{ij}$$

follow directly from Eq. (29)-(30). Now one of the RHSs is equal to $\sum_{j \neq i} |X_{ij}|$, which implies Eq. (28). For the general case, the argument uses the extreme points of Eq. (29)-(30) and elimination of T by projection.

We can now approximate Eq. (26) by the pure feasibility LP:

$$\left.\begin{array}{rr} \forall \{u,v\} \in E \; X_{uu} + X_{vv} - 2X_{uv} = d_{uv}^2 \\ \forall i \leq n \qquad \sum_{j \neq i} T_{ij} \leq X_{ii} \\ -T \leq X \leq T, \end{array}\right\} \tag{31}$$

[2] blogs.princeton.edu/imabandit/2013/02/19/orf523-ipms-for-lps-and-sdps/

which we call a *diagonally dominant program* (DDP). As in Eq. (26), we do not explicitly give an objective function, since it depends on the application. Since the DDP in Eq. (31) is an inner approximation of the corresponding SDP in Eq. (26), the DDP feasible set is a subset of that of the SDP. This situation yields both an advantage and a disadvantage: any solution $\tilde{X}$ of the DDP is psd, and can be obtained at a smaller computational cost; however, the DDP might be infeasible even if the corresponding SDP is feasible (see Fig. 5, left). In order to decrease the risk of infeasibility of Eq. (31), we relax

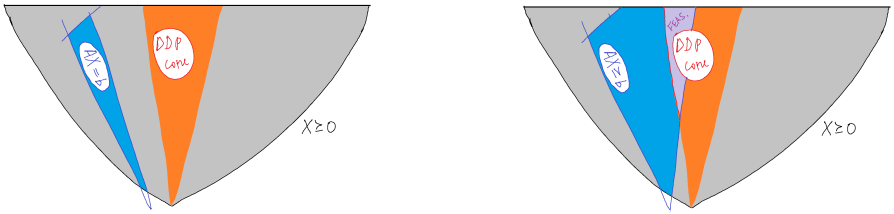

Fig. 5 On the left, the DDP is infeasible even if the SDP is not; on the right, a relaxed set of constraints makes the DDP feasible.

the equation constraints to inequality, and impose an objective as in the push-and-pull formulation Eq. (25):

$$\left.\begin{array}{rrr} \max & \sum\limits_{\{u,v\}\in E} (X_{uu} + X_{vv} - 2X_{uv}) & \\ \forall \{u,v\} \in E & X_{uu} + X_{vv} - 2X_{uv} \leq d_{uv}^2 & \\ \forall i \leq n & \sum\limits_{j \neq i} T_{ij} \leq X_{ii} & \\ & -T \leq X \leq T. & \end{array}\right\} \tag{32}$$

This makes the DDP feasible set larger, which means it is more likely to be feasible (see Fig. 5, right). Eq. (32) was successfully tested on protein graphs in [61].

If C is any cone in $\mathbb{R}^n$, the *dual cone* C^* is defined as:

$$C^* = \{y \in \mathbb{R}^n \mid \forall x \in C \ \langle x, y\rangle \geq 0\}.$$

Note that the dual cone contains the set of vectors making a non-obtuse angle with all of the vectors in the original (primal) cone. We can exploit the dual dd cone in order to provide another DDP formulation for the DGP which turns out to be an outer approximation. Outer approximations have symmetric advantages and disadvantages w.r.t. the inner ones: if the original SDP is feasible, than the outer DDP approximation is also feasible (but the DDP may be feasible even if the SDP is not); however, the solution $\tilde{X}$ we obtain from the DDP need not be a psd matrix. Some computational experience related to [161] showed that it often happens that more or less half of the eigenvalues of $\tilde{X}$ are negative.

We now turn to the actual DDP formulation related to the dual dd cone. A cone C of $n \times n$ real symmetric matrices is *finitely generated* by a set $\mathcal{X}$ of matrices if:

$$\forall X \in C \; \exists \delta \in \mathbb{R}_+^{|\mathcal{X}|} \quad X = \sum_{x \in \mathcal{X}} \delta_x x x^\top.$$

It turns out [18] that the dd cone is finitely generated by

$$\mathcal{X}_{\mathsf{dd}} = \{e_i \mid i \leq n\} \cup \{e_i \pm e_j \mid i < j \leq n\},$$

where $e_1, \ldots, e_n$ is the standard orthogonal basis of $\mathbb{R}^n$. This is proved in [18] by showing that the following rank-one matrices are extreme rays of the dd cone:

- $E_{ii} = \mathsf{diag}(e_i)$, where $e_i = (0, \ldots, 0, 1_i, 0, \ldots, 0)^\top$;
- E_{ij}^+ has a minor $\begin{pmatrix} 1_{ii} & 1_{ij} \\ 1_{ji} & 1_{jj} \end{pmatrix}$ and is zero elsewhere;
- E_{ij}^- has a minor $\begin{pmatrix} 1_{ii} & -1_{ij} \\ -1_{ji} & 1_{jj} \end{pmatrix}$ and is zero elsewhere,

and, moreover, that the extreme rays are generated by the standard basis vectors as follows:

$$\begin{aligned} \forall i \leq n \quad & E_{ii} = e_i e_i^\top \\ \forall i < j \leq n \quad & E_{ij}^+ = (e_i + e_j)(e_i + e_j)^\top \\ \forall i < j \leq n \quad & E_{ij}^- = (e_i - e_j)(e_i - e_j)^\top. \end{aligned}$$

This observation allowed Ahmadi and his co-authors to write the DDP formulation Eq. (32) in terms of the extreme rays $E_{ii}, E_{ij}^\pm$ [5], and also to define a column generation algorithms over them [4].

If a matrix cone is finitely generated, the dual cone has the same property. Let $\mathbb{S}_n$ be the set of real symmetric $n \times n$ matrices; for $A, B \in \mathbb{S}_n$ we define an inner product $\langle A, B \rangle = A \bullet B \triangleq \mathsf{tr}(AB^\top)$.

Theorem 3 *Assume C is finitely generated by $\mathcal{X}$. Then C^* is also finitely generated. Specifically, $C^* = \{Y \in \mathbb{S}_n \mid \forall x \in \mathcal{X} \; (Y \bullet x x^\top \geq 0)\}$.*

Proof By assumption, $C = \{X \in \mathbb{S}_n \mid \exists \delta \in \mathbb{R}_+^{|\mathcal{X}|} \; X = \sum_{x \in \mathcal{X}} \delta_x x x^\top\}$.

($\Rightarrow$) Let $Y \in \mathbb{S}_n$ be such that, for each $x \in \mathcal{X}$, we have $Y \bullet x x^\top \geq 0$. We are going to show that $Y \in C^*$, which, by definition, consists of all matrices Y such that for all $X \in C$, $Y \bullet X \geq 0$. Note that, for all $X \in C$, we have $X = \sum_{x \in \mathcal{X}} \delta_x x x^\top$ (by finite generation). Hence $Y \bullet X = \sum_x \delta_x Y \bullet x x^\top \geq 0$ (by definition of Y), whence $Y \in C^*$.

($\Leftarrow$) Suppose $Z \in C^* \setminus \{Y \mid \forall x \in \mathcal{X} \; (Y \bullet x x^\top \geq 0)\}$. Then there is $\mathcal{X}' \subset \mathcal{X}$ such that for any $x \in \mathcal{X}'$ we have $Z \bullet x x^\top < 0$. Consider any $Y = \sum_{x \in \mathcal{X}'} \delta_x x x^\top \in C$ with $\delta \geq 0$. Then $Z \bullet Y = \sum_{x \in \mathcal{X}'} \delta_x Z \bullet x x^\top < 0$, so $Z \notin C^*$, which is a contradiction. Therefore $C^* = \{Y \mid \forall x \in \mathcal{X} \; (Y \bullet x x^\top \geq 0)\}$ as claimed. □

We are going to exploit Thm. 3 in order to derive an explicit formulation of the following DDP formulation based on the dual cone C^*_{dd} of the dd cone C_{dd} finitely generated by $\mathcal{X}_{\mathsf{dd}}$:

$$\left.\begin{aligned} \forall \{u,v\} \in E \; X_{uu} + X_{vv} - 2X_{uv} = d^2_{uv} \\ X \in C^*_{\mathsf{dd}}. \end{aligned}\right\}$$

We remark that $X \bullet vv^\top = v^\top X v$ for each $v \in \mathbb{R}^n$. By Thm. 3, $X \in C^*_{\mathsf{dd}}$ can be restated as $\forall v \in \mathcal{X}_{\mathsf{dd}} \; v^\top X v \geq 0$. We obtain the following LP formulation:

$$\left.\begin{aligned} \max \sum_{\{u,v\} \in E} (X_{uu} + X_{vv} - 2X_{uv}) \\ \forall \{u,v\} \in E \quad X_{uu} + X_{vv} - 2X_{uv} = d^2_{uv} \\ \forall v \in \mathcal{X}_{\mathsf{dd}} \quad v^\top X v \geq 0. \end{aligned}\right\} \tag{33}$$

With respect to the primal DDP, the dual DDP formulation in Eq. (33) provides a very tight bound to the objective function value of the push-and-pull SDP formulation Eq. (25). On the other hand, the solution $\bar{X}$ is usually far from being a psd matrix.

6.2 Fast high-dimensional methods

In Sect. 6.1 we surveyed methods based on MP, which are very flexible, insofar as they can accommodate side constraints and noisy data, but computationally demanding. In this section we discuss two very fast, yet robust, methods for embeddings graphs in Euclidean spaces.

6.2.1 Incidence vectors

The simplest, and most naive methods for mapping graphs into vectors are given by exploiting various incidence information in the graph structure. By contrast, the resulting embeddings are unrelated to Eq. (3).

Given a simple graph $G = (V, E)$ with $|V| = n$, $|E| = m$ and edge weight function $w : E \to \mathbb{R}_+$, we present two approaches: one which outputs an $n \times n$ matrix, and one which outputs a single vector in $\mathbb{R}^K$ with $K = \frac{1}{2}n(n-1)$.

1. For each $u \in V$, let $x_u = (x_{uv} \mid v \in V) \in \mathbb{R}^n$ be the incidence vector of $N(u)$ on V, i.e.:
$$\forall u \in V \quad x_{uv} = \begin{cases} w_{uv} & \text{if } \{u,v\} \in E \\ 0 & \text{otherwise.} \end{cases}$$
2. Let $K = \frac{1}{2}n(n-1)$, and $x^E = (x_e \mid e \in E) \in \mathbb{R}^K$ be the incidence vector of the edge set E into the set $\{\{i,j\} \mid i < j \leq n\}$, i.e.:
$$x_e = \begin{cases} w_e & \text{if } e \in E \\ 0 & \text{otherwise.} \end{cases}$$

Both embeddings can be obtained in $O(n^2)$ time. Both embeddings are very high dimensional. So that they may be useful in practice, it is necessary to post-process them using dimensional reduction techniques (see Sect. 7).

6.2.2 The universal isometric embedding

This method, also called *Fréchet embedding*, is remarkable in that it maps any finite metric space congruently into a set of vectors in the ℓ_∞ norm [101, §6]. No other norm allows exact congruent embeddings in vector spaces [129]. The Fréchet embedding provided the foundational idea for several other probabilistic approximate embeddings in various other norms and dimensions [35, 124].

Theorem 4 *Given any finite metric space (X, d), where $|X| = n$ and d is a distance function defined on X, there exists an embedding $\rho : X \to \mathbb{R}^n$ such that $(\rho(X), \ell_\infty)$ is congruent to (X, d).*

This theorem is surprising because of its generality in conjunction with the exactness of the result: it works on *any* (finite) metric space. The "magic hat" out of which we shall pull the vectors in $\rho(X)$ is simply the only piece of data we are given, namely the distance matrix of X. More precisely the i-th element of X is mapped to the vector corresponding to the i-th column of the distance matrix.

Proof Let $\mathsf{D}(X)$ be the distance matrix of (X, d), namely $\mathsf{D}_{ij}(X) = (d(x_i, x_j))$ where $X = \{x_1, \ldots, x_n\}$. We denote $d(x_i, x_j) = d_{ij}$ for brevity. For any $j \leq n$ we let $\rho(x_j) = \delta_j$, where δ_j is the j-th column of $\mathsf{D}(X)$. We have to show that $\|\rho(x_i) - \rho(x_j)\|_\infty = d_{ij}$ for each $i < j \leq n$. By definition of the ℓ_∞ norm, for each $i < j \leq n$ we have

$$\|\rho(x_i) - \rho(x_j)\|_\infty = \|\delta_i - \delta_j\|_\infty = \max_{k \leq n} |\delta_{ik} - \delta_{jk}| = \max_{k \leq n} |d_{ik} - d_{jk}|. \quad (*)$$

By the triangular inequality on (X, d), for $i < j \leq n$ and $k \leq n$ we have:

$$\begin{aligned} & d_{ik} \leq d_{ij} + d_{jk} \quad \wedge \quad d_{jk} \leq d_{ij} + d_{ik} \\ \Rightarrow\ & d_{ik} - d_{jk} \leq d_{ij} \quad \wedge \quad d_{jk} - d_{ik} \leq d_{ij} \\ \Rightarrow\ & |d_{ik} - d_{jk}| \leq d_{ij}; \end{aligned}$$

since these inequalities are valid for each k, by $(*)$ we have:

$$\|\rho(x_i) - \rho(x_j)\|_\infty \leq \max_k d_{ij} = d_{ij}, \quad (\dagger)$$

where the last equality follows because d_{ij} does not depend on k. Now we note that the maximum of $|d_{ik} - d_{jk}|$ over k must exceed the value of the same expression when either of the terms d_{ik} or d_{jk} is zero, i.e. when $k \in \{i, j\}$, since, when $k = i$, then $|d_{ik} - d_{jk}| = |d_{ii} - d_{ji}| = d_{ij}$, and the same holds when $k = j$. Hence,

$$\max_{k \leq n} |d_{ik} - d_{jk}| \geq d_{ij}. \quad (\ddagger)$$

By $(*)$, $(\dagger)$ and $(\ddagger)$, we finally have:

$$\forall i < j \leq n \quad \|\rho(x_i) - \rho(x_j)\|_\infty = d_{ij}$$

as claimed. □

We remark that Thm. 4 is only applicable when $\mathsf{D}(X)$ is a distance matrix, which corresponds to the case of a graph G edge-weighed by d being a complete graph. We address the more general case of any (connected) simple graph $G = (V, E)$, corresponding to a partially defined distance matrix, by completing the matrix using the shortest path metric (this distance matrix completion method was used for the Isomap heuristic, see [173,112] and Sect. 7.1.1):

$$\forall \{i, j\} \notin E \quad d_{ij} = \mathsf{shortest\_path\_length}_G(i, j). \tag{34}$$

In practice, we can compute the lengths of all shortest paths in G by using the Floyd-Warshall algorithm, which runs in $O(n^3)$ time (but in practice it is very fast).

This method yields a realization of G in ℓ_∞^n, which is a high-dimensional embedding. It is necessary to post-process it using dimensional reduction techniques (see Sect. 7).

6.2.3 Multidimensional scaling

The literature on Multidimensional Scaling (MDS) is extensive [50,33], and many variants exist. The basic version, called *classic MDS*, aims at finding an approximate realization of a partial distance matrix. In other words, it is a heuristic solution method for the

> Euclidean Distance Matrix Completion Problem (EDMCP). Given a simple undirected graph $G = (V, E)$ with an edge weight function $w : E \to \mathbb{R}_+$, determine whether there exists an integer $K > 0$ and a realization $x : V \to \mathbb{R}^K$ such that Eq. (3) holds.

The difference between EDMCP and DGP may appear diminutive, but it is in fact very important. In the DGP the integer K is part of the input, whereas in the EDMCP it is part of the output. This has a large effect on worst-case complexity: while the DGP is **NP**-hard even when only an ε-approximate realization is sought [162, §5], ε-approximate realizations of EDMCPs can be found in polynomial time by solving an SDP [7]. Consider the following matrix:

$$\Delta(E, d) = \begin{cases} w_{ij}^2 & \text{if } \{i, j\} \in E \\ d_{ij} & \text{otherwise,} \end{cases}$$

where $d = (d_{ij} \mid \{i, j\} \notin E)$ is a vector of decision variables, and $J = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, with $\mathbf{1}$ being the all-one vector. Then the following formulation is valid for the EDMCP:

$$\left.\begin{array}{ll} \min\limits_{d,T,G} & \mathbf{1} \bullet T \\ & -T \leq G + \frac{1}{2} J\, \Delta(E, d)\, J \leq T \\ & G \succeq 0, \end{array}\right\} \tag{35}$$

where $\mathbf{1}$ is the $n \times n$ all-one matrix.

Theorem 5 *The SDP in Eq.* (35) *correctly models the EDMCP.*

By "correctly models" we mean that the solution of the EDMCP can be obtained in polynomial time from the solution of the SDP in Eq. (35).

Proof First, we remark that, given a realization $x : V \to \mathbb{R}^n$, its Gram matrix is $G = xx^\top$, and its squared Euclidean distance matrix (EDM) is

$$D^2 = (\|x_u - x_v\|_2^2 \mid u \leq n \wedge v \leq n) \in \mathbb{R}^{n \times n}.$$

Next, we recall that

$$G = I_n - \frac{1}{2} J D^2 J \tag{36}$$

by [56] (after [165] — see [113, §7] for a direct proof). Now we note that minimizing $\mathbf{1} \bullet T$ subject to $-T \leq G + \frac{1}{2} J \Delta(E,d) J \leq T$ is an exact reformulation of

$$\min_{G,d} \|G - (-1/2) J \Delta(E,d) J)\|_1, \quad (*)$$

since $\mathbf{1} \bullet T = \sum_{i,j} T_{ij}$, and T is used to "sandwich" the argument of the ℓ_1 norm in $(*)$.

We also recall another basic fact of linear algebra: a matrix is Gram if and only if it is psd: hence, requiring $G \succeq 0$ forces G to be a Gram matrix. Consequently, if the optimal objective function value of Eq. (35) is zero with corresponding solution d^*, T^*, G^*, then $\mathsf{tr}(T^*) = 0 \Rightarrow T^* = 0 \Rightarrow (*)$. Moreover, G^* is a Gram matrix, so $\Delta(E,d^*)$ is its corresponding EDM. Lastly, the realization x^* corresponding to the Gram matrix G^* can be obtained by spectral decomposition of $G^* = P \Lambda P^\top$, which yields $x^* = P\sqrt{\Lambda}$: this implies that the EDMCP instance is YES. Otherwise $T^* \neq 0$, which means that the EDMCP instance is NO (otherwise there would be a contradiction on $\mathsf{tr}(T^*) > 0$ being optimal). □

The practically useful corollary to Thm. (5) is that solving Eq. (35) provides an approximate solution x^* even if $\Delta(E,d)$ cannot be completed to an EDM.

Classic MDS is an efficient heuristic method for finding an approximate realization of a partial distance matrix $\Delta(E,d)$. It works as follows:

1. complete $\Delta(E,d)$ to an approximate EDM $\tilde{D}^2$ using the shortest-path metric (Eq. (34));
2. let $\tilde{G} = I_n - \frac{1}{n} J \tilde{D}^2 J$;
3. let $P \tilde{\Lambda} P^\top$ be the spectral decomposition of $\tilde{G}$;
4. if $\tilde{\Lambda} \geq 0$ then, by Eq. (36), $\tilde{D}^2$ is a EDM, with corresponding (exact) realization $\tilde{x} = P\sqrt{\Lambda}$;
5. otherwise, let $\Lambda^+ = \mathsf{diag}((\max(\lambda, 0) \mid \lambda \in \Lambda))$: then $\tilde{x} = P\sqrt{\Lambda^+}$ is an approximate realization of $\tilde{D}^2$.

Note that both Eq.(35) and classic MDS determine K as part of the output, i.e. K is the rank of the realization (respectively x^* and $\tilde{x}$).

7 Dimensional reduction techniques

Dimensional reduction techniques reduce the dimensionality of a set of vectors according to different criteria, which may be heuristic, or give some (possibly probabilistic) guarantee of keeping some quantity approximately invariant. They are necessary in order to make many of the methods in Sect. 6 useful in practice.

7.1 Principal component analysis

Principal Component Analysis (PCA) is one of the foremost dimensional reduction techniques. It is ascribed to Harold Hotelling[3] [86].

Consider an $n \times m$ matrix X consisting of n data row vectors in $\mathbb{R}^m$, and let $K < m$ be a given integer. We want to find a change of coordinates for X such that the first component has largest variance over the transformed vectors, the second component has second-largest variance, and so on, until the K-th component. The other components can be neglected, as the variance of the data in those directions is low.

The usual geometric interpretation of PCA is to take the smallest enclosing ellipsoid $\mathcal{E}$ for X: then the required coordinate change maps component 1 to the line parallel to the largest radius of $\mathcal{E}$, component 2 to the line parallel to the second-largest radius of $\mathcal{E}$, and so on until component K (see Fig. 6). The statistical interpretation of PCA looks for the change of coordinates which makes the data vectors be uncorrelated in their components. Fig. 6 should give an intuitive idea about why this interpretation corresponds with the ellipsoid of the geometric interpretation. The cartesian coordinates in Fig. 6 are certainly correlated, while the rotated coordinates look far less correlated. The zero correlation situation corresponds to a perfect ellipsoid. An ellipsoid is described by the equation $\sum_{j \leq n} \left(\frac{x_j}{r_j}\right)^2 = 1$, which has no mixed terms $x_i x_j$ contributing to correlation. Both interpretations are well (and formally) argued in [180, §2.1].

The interpretation we give here is motivated by DG, and related to MDS (Sect. 6.2.3). PCA can be seen as a modification of MDS which only takes into account the K (nonnegative) principal components. Instead of Λ^+ (step 5 of the MDS algorithm), PCA uses a different diagonal matrix Λ^{pca}: the i-th diagonal component is

$$\Lambda^{\mathsf{pca}}_{ii} = \begin{cases} \max(\Lambda_{ii}, 0) & \text{if } i \leq K \\ 0 & \text{otherwise,} \end{cases} \tag{37}$$

[3] A young and unknown George Dantzig had just finished his presentation of LP to an audience of "big shots", including Koopmans and Von Neumann. Harold Hotelling raised his hand, and stated: "but we all know that the world is nonlinear!", thereby obliterating the simplex method as a mathematical curiosity. Luckily, Von Neumann answered on Dantzig's behalf and in his defence [53].

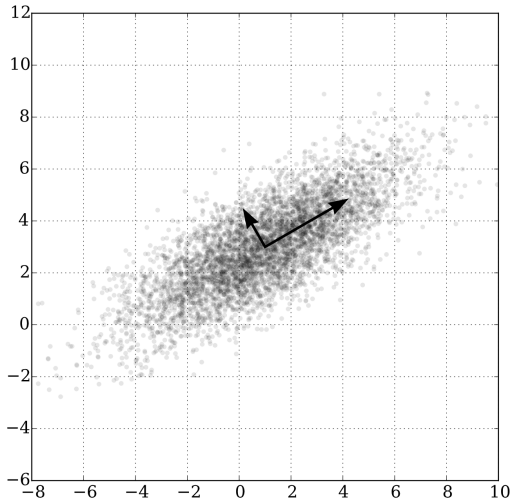

Fig. 6 Geometric interpretation of PCA (image from [188]).

where $P\Lambda P^\top$ is the spectral decomposition of $\tilde{G}$. In this interpretation, when given a partial distance matrix and the integer K as input, PCA can be used as an approximate solution method for the DGP.

On the other hand, the PCA algorithm is most usually considered as a method for dimensionality reduction, so it has a data matrix X and an integer K as input. It is as follows:

1. let $\tilde{G} = XX^\top$ be the $n \times n$ Gram matrix of the data matrix X;
2. let $P\tilde{\Lambda}P^\top$ be the spectral decomposition of $\tilde{G}$;
3. return $\tilde{x} = P\sqrt{\Lambda^{\mathsf{pca}}}$.

Then $\tilde{x}$ is an $n \times K$ matrix, where $K < n$. The i-th row vector in $\tilde{x}$ is a dimensionally reduced representation of the i-th row vector in X.

There is an extensive literature on PCA, ranging over many research papers, dedicated monographs and textbooks [188,93,180]. Among the variants and extensions, see [58,160,55,8,60].

7.1.1 Isomap

One of the most interesting applications of PCA is possibly the Isomap algorithm [173], already mentioned above in Sect. 6.2.2, which is able to use PCA in order to perform a nonlinear dimensional reduction from the original dimension m to a given target dimension K, as follows.

1. Form a connected graph $H = (V, E)$ with the column indices $1, \ldots, n$ of X as vertex set V: determine a threshold value τ such that, for each column vector x_i in X (for $i \leq n$), and for each x_j in X such that $\|x_i - x_j\|_2 \leq \tau$, the edge $\{i, j\}$ is in the edge set E; the graph H should be as sparse as possible but also connected.

2. Complete H using the shortest path metric (Eq. (34)).
3. Use PCA in the MDS interpretation mentioned above: interpret the completion of (V, E) as a metric space, construct its (approximate) EDM $\tilde{D}$, compute the corresponding (approximate) Gram matrix $\tilde{G}$, compute the spectral decomposition of $\tilde{G}$, replace its diagonal eigenvalue matrix Λ as in Eq. (37), and return the corresponding K-dimensional vectors.

Intuitively, Isomap works well because in many practical situations where a set X of points in $\mathbb{R}^m$ are close to a (lower) K-dimensional manifold, the shortest path metric is likely to be a better estimation of the Euclidean distance in $\mathbb{R}^K$ than the Euclidean distance in $\mathbb{R}^m$, see [173, Fig. 3].

7.2 Barvinok's naive algorithm

By Eq. (26), we can solve an SDP relaxation of the DGP and obtain an $n \times n$ psd matrix solution $\bar{X}$ which, in general, will not have rank K (i.e., it will not yield an $n \times K$ realization matrix, but rather an $n \times n$ one). In this section we shall derive a dimensionality reduction algorithm to obtain an approximation of $\bar{X}$ which has the correct rank K.

7.2.1 Quadratic Programming feasibility

Barvinok's naive algorithm [20, §5.3] is a probabilistic algorithm which can find an approximate vector solution $x' \in \mathbb{R}^n$ to a system of quadratic equations

$$\forall i \leq m \quad x^\top Q^i x = a_i, \tag{38}$$

where the Q^i are $n \times n$ symmetric matrices, $a \in \mathbb{R}^m$, $x \in \mathbb{R}^n$, and m is polynomial in n. The analysis of this algorithm provides a probabilistic bound on the maximum distance that x' can have from the set of solutions of Eq. (38). Thereafter, one can run a local NLP solver with x' as a starting point, and obtain a hopefully good (approximate) solution to Eq. (38). We note that this algorithm is still not immediately applicable to the our setting where K could be different from 1: we shall address this issue in Sect. 7.2.4.

Barvinok's naive algorithm solves an SDP relaxation of Eq. (38), and then retrieves a certain randomized vector from the solution:

1. form the SDP relaxation

$$\forall i \leq m \; (Q^i \bullet X = a_i) \wedge X \succeq 0 \tag{39}$$

   of Eq. (38) and solve it to obtain $\bar{X} \in \mathbb{R}^{n \times n}$;
2. let $T = \sqrt{\bar{X}}$, which is a real matrix since $\bar{X} \succeq 0$ (T can be obtained by spectral decomposition, i.e. $\bar{X} = P\Lambda P^\top$ and $T = P\sqrt{\Lambda}$);
3. let y be a vector sampled from the multivariate normal distribution $\mathsf{N}^n(0, 1)$;
4. compute and return $x' = Ty$.

The analysis provided in [20] shows that $\exists c > 0$ and an integer $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$

$$\mathsf{P}\left(\forall i \leq m \quad \mathsf{dist}(x', \mathcal{X}_i) \leq c\sqrt{\|\bar{X}\|_2 \ln n}\right) \geq 0.9. \tag{40}$$

In Eq. (40), $\mathsf{P}(\cdot)$ denotes the probability of an event,

$$\mathsf{dist}(b, B) = \inf_{\beta \in B} \|b - \beta\|_2$$

is the Euclidean distance between the point b and the set B, and c is a constant that only depends on $\log_n m$. We note that the term $\sqrt{\|\bar{X}\|_2}$ in Eq. (40) arises from T being a factor of $\bar{X}$. We note also that 0.9 follows from assigning some arbitrary value to some parameter — i.e. 0.9 can be increased as long as the problem size is large enough.

For cases of Eq. (38) where one of the quadratic equations is $\|x\|_2^2 = 1$ (namely, the solutions of Eq. (38) must belong to the unit sphere), it is noted in [20, Eg. 5.5] that, if $\bar{X}$ is "sufficiently generic", then it can be argued that $\|\bar{X}\|_2 = O(1/n)$, which implies that the bounding function $c\sqrt{\bar{X}_2 \ln n} \to 0$ as $n \to \infty$. This, in turn, means that x' converges towards a feasible solution of the original problem in the limit.

7.2.2 Concentration of measure

The term $\ln n$ in Eq. (40) arises from a phenomenon of high-dimensional geometry called "concentration of measure".

We recall that a function $f : \mathcal{X} \to \mathbb{R}$ is *Lipschitz* if there is a constant $M > 0$ s.t. for any $x, y \in \mathcal{X}$ we have $|f(x) - f(y)| < M\|x - y\|_2$. A measure space $(\mathcal{X}, \mu)$ has the *concentration of measure* property if for any Lipschitz function f, there are constants $C, c > 0$ such that:

$$\forall \varepsilon > 0 \quad \mathsf{P}(|f(x) - \mathsf{E}_\mu(f)| > \varepsilon \mid x \in \mathcal{X}) \leq C\, e^{-c\varepsilon^2} \tag{41}$$

where $\mathsf{E}_\mu(f) = \int_{\mathcal{X}} f(x) d\mu$. In other words, $\mathcal{X}$ has measure concentration if for any Lipschitz function f, its discrepancy from its mean value is small with arbitrarily high probability. It turns out that the Euclidean space $\mathbb{R}^n$ with the Gaussian density measure $\phi(x) = (2\pi)^{n/2} e^{-\|x\|_2^2/2}$ has measure concentration [21, §5.3].

Measure concentration is interesting in view of applications since, given any large enough closed subset A of $\mathcal{X}$, its ε-neighbourhood

$$A(\varepsilon) = \{x \in \mathcal{X} \mid \mathsf{dist}(x, A) \leq \varepsilon\} \tag{42}$$

contains almost the whole measure of $\mathcal{X}$. More precisely, if $(\mathcal{X}, \mu)$ has measure concentration and $A \subset \mathcal{X}$ is closed, for any $p \in (0, 1)$ there is a $\varepsilon_0(p) > 0$ such that [123, Prop. 2]:

$$\forall \varepsilon \geq \varepsilon_0(p) \quad \mu(A(\varepsilon)) > 1 - p. \tag{43}$$

Eq. (43) is useful for applications because it defines a way to analyse probabilistic algorithms. For a random point sampled in $(\mathcal{X}, \mu)$ that happens to be in A on average, Eq. (43) ensures that it is unlikely that it should be far from A. This can be used to bound errors, as Barvinok did with his naive algorithm. Concentration of measure is fundamental in data science, insofar as it may provide algorithmic analyses to the effect that some approximation errors decrease in function of the increasing instance size.

7.2.3 Analysis of Barvinok's algorithm

We sketch the main lines of the analysis of Barvinok's algorithm (see [19, Thm. 5.4] or [123, §3.2] for a more detailed proof). We let $\mathcal{X} = \mathbb{R}^n$ and $\mu(x) = \phi(x)$ be the Gaussian density measure. It is easy to show that

$$\mathsf{E}_\mu(x^\top Q^i x \mid x \in \mathcal{X}) = \mathsf{tr}(Q^i)$$

for each $i \leq m$, and, from this, that given the factorization $\bar{X} = TT^\top$, that

$$\mathsf{E}_\mu(x^\top T^\top Q^i Tx \mid x \in \mathcal{X}) = \mathsf{tr}(T^\top Q^i T) = \mathsf{tr}(Q^i \bar{X}) = Q^i \bullet \bar{X} = a_i.$$

This shows that, for any $y \sim \mathsf{N}^n(0, 1)$, the average of $y^\top T^\top Q^i Ty$ is a_i.

The analysis then goes on to show that, for some $y \sim \mathsf{N}^n(0, 1)$, it is unlikely that $y^\top T^\top Q^i Ty$ should be far from a_i. It achieves this result by defining the sets $A_i^+ = \{x \in \mathbb{R}^n \mid x^\top Q^i x \geq a_i\}$, $A_i^- = \{x \in \mathbb{R}^n \mid x^\top Q^i x \leq a_i\}$, and their respective neighbourhoods $A_i^+(\varepsilon)$, $A_i^-(\varepsilon)$. Using a technical lemma [123, Lemma 4] it is possible to apply Eq. (43) to $A_i^+(\varepsilon)$ and $A_i^-(\varepsilon)$ to argue for concentration of measure. Applying the union bound it can be shown that their intersection $A_i(\varepsilon)$ is the neighbourhood of $A_i = \{x \in \mathbb{R}^n \mid x^\top Q^i x = a_i\}$. Another application of the union bound to all the sets $A_i(\varepsilon)$ yields the result [123, Thm. 5].

We note that concentration of measure proofs often have this structure: (a) prove that a certain event holds on average; (b) prove that the discrepancy from average gets smaller and/or more unlikely with increasing size. Usually proving (a) is easier than proving (b).

7.2.4 Applicability to the DGP

The issue with trying to apply Barvinok's naive algorithm to the DGP is that we should always assume $K = 1$ by Eq. (38). To circumvent this issue, we might represent an $n \times K$ realization matrix as a vector in $\mathbb{R}^{nK}$ by stacking its columns (or concatenating its rows). This, on the other hand, would require solving SDPs with $nK \times nK$ matrices, which is prohibitive because of size.

Luckily, Barvinok's naive algorithm can be very easily extended to arbitrary values of K. We replace Step 3 by:

3*b*. let y be an $n \times K$ matrix sampled from $\mathsf{N}^{n \times K}(0, 1)$.

The corresponding analysis needs some technical changes [123], but the overall structure is the same as the case $K = 1$. The obtained bound replaces $\sqrt{\ln n}$ in Eq. (40) with $\sqrt{\ln nK}$.

In the DGP case, the special structure of the matrices Q^i (for i ranging over the edge set E) makes it possible to remove the factor K, so we retrieve the exact bound of Eq. (40). As noted in Sect. 7.2.1, if the DGP instance is on a sphere [122], this means that $x' = Ty$ converges to an exact realization with probability 1 in the limit of $n \to \infty$. Similar bounds to Eq. (40) were also derived for the iDGP case [123].

Barvinok also described concentration of measure based techniques for finding low-ranking solutions solutions of the SDP in Eq. (39) (see [19] and [21, §6.2]), but these do not allow the user to specify an arbitrary rank K, so they only apply to the EDMCP.

7.3 Random projections

Random projections are another dimensionality reduction technique exploiting high-dimensional geometry properties and, in particular, the concentration of measure phenomenon (Sect. 7.2.2). They are more general than Barvinok's naive algorithm (Sect. 7.2) in that they apply to sets of vectors in some high-dimensional Euclidean space $\mathbb{R}^n$ (with $n \gg 1$). These sets are usually finite and growing polynomially with instance sizes [176], but they may also be infinite [192], in which case the technical name used is *subspace embeddings*.

7.3.1 The Johnson-Lindenstrauss Lemma

The foremost result in RPs is the celebrated Johnson-Lindenstrauss Lemma (JLL) [92]. For a set of vectors $\mathcal{X} \subset \mathbb{R}^n$ with $|\mathcal{X}| = \ell$, and an $\varepsilon \in (0, 1)$ there is a $k = O(\frac{1}{\varepsilon^2} \ln \ell)$ and a mapping $f : \mathcal{X} \to \mathbb{R}^k$ such that:

$$\forall x, y \in \mathcal{X} \quad (1 - \varepsilon)\|x - y\|_2 \leq \|f(x) - f(y)\|_2 \leq (1 + \varepsilon)\|x - y\|_2. \tag{44}$$

The proof of this result [92, Lemma 1] is probabilistic, and show that an f satisfying Eq. (44) exists with some nonzero probability.

Later and more modern proofs (e.g. [54]) clearly point out that f can be a linear operator represented by a $k \times n$ matrix T, each component of which can be sampled from a *subgaussian distribution*. This term refers to a random variable $\mathfrak{V}$ for which there are constants C, c s.t. for each $t > 0$ we have

$$\mathsf{P}(|\mathfrak{V}| > t) \leq C\, e^{-ct^2}.$$

In particular, the Gaussian distribution is also subgaussian. Then the probability that a randomly sampled T satisfies Eq. (44) can be shown to exceed $1/\ell$. The union bound then provides an estimate on the number of samplings of T necessary to guarantee Eq. (44) with a desired probability.

Some remarks are in order.

1. Computationally Eq. (44) is applied to some given data as follows: given a set $\mathcal{X}$ of ℓ vectors in $\mathbb{R}^n$ and some error tolerance $\varepsilon \in (0,1)$, find an appropriate $k = O(\frac{1}{\varepsilon^2}\ln\ell)$, construct the $k \times n$ RP T by sampling each of its components from $\mathsf{N}(0, \frac{1}{\sqrt{k}})$, then define the set $T\mathcal{X} = \{Tx \mid x \in X\}$. By the JLL, $T\mathcal{X}$ is *approximately congruent* to $\mathcal{X}$ in the sense of Eq. (44); however, $T\mathcal{X} \subset \mathbb{R}^k$ whereas $\mathcal{X} \subset \mathbb{R}^n$, and, typically, $k \ll n$.
2. The computation of an appropriate k would appear to require an estimation of the constant in the expression $O(\frac{1}{\varepsilon^2}\ln\ell)$. Values computed theoretically are often so large as to make the technique useless in practice. As far as we know, this constant has only been done empirically in some cases [177], ending up with an estimation of the constant at 1.8 (which is the value we employed in most of our experiments).
3. The term $\frac{1}{\sqrt{k}}$ is the standard deviation of the normal distribution from which the components of T must be sampled. It corresponds to a scaling of the vectors in $T\mathcal{X}$ induced by the loss in dimensions (see Thm. 6).
4. In the expression $O(\frac{1}{\varepsilon^2}\ln\ell)$, the logarithmic term is the one that counts for analysis purposes, but in practice ε^{-2} can be large. Our advice is to take $\varepsilon \in (0.1, 0.2)$ and then fine-tune ε according to results.
5. Surprisingly, the target dimension k is independent of the original dimension n.
6. Even if the data in $\mathcal{X}$ is sparse, $T\mathcal{X}$ ends up being dense. Different classes of sparse RPs have been investigated [2,95] in order to tackle this issue. A simple algorithm [52, §5.1] consists in initializing T as the $k \times n$ zero matrix, and then only fill components using samples from $\mathsf{N}(0, \frac{1}{\sqrt{kp}})$ with some given probability p. The value of p corresponds to the density of T. In general, and empirically, it appears that the larger n and ℓ are, the sparser T can be.
7. Obviously, a Euclidean space of dimension k can embed at most k orthogonal vectors. An easy, but surprising corollary of the JLL is that as many as $O(2^k)$ approximately orthogonal vectors can fit in $\mathbb{R}^k$. This follows by [182, Prop. 1] applied to the standard basis $S = \{e_1, \ldots, e_n\}$ of $\mathbb{R}^n$: we obtain $\forall i < j \leq n\ (-\varepsilon \leq \langle Te_i, Te_j\rangle - e_ie_j \leq \varepsilon)$, which implies $|\langle Te_i, Te_j\rangle| \leq \varepsilon$ with $TS \subset \mathbb{R}^k$ and $k = O(\ln n)$. Therefore TS is a set of $O(2^k)$ almost orthogonal vectors in $\mathbb{R}^k$, as claimed.
8. Typical applications of RPs arise in clustering databases of large files (e.g. e-mails, images, songs, videos), performing basic tasks in ML (e.g. k-means [36], k-nearest neighbors (k-NN) [90], robust learning [15] and more [88]), and approximating large MP formulations (e.g. LP, QP, see Sect. 7.3.3).
9. The JLL seems to suggest that most of the information encoded by the congruence of a set of vectors can be maintained up to an ε tolerance in much smaller dimensional spaces. This is not true for sets of vectors in low dimensions. For example, with $n \in \{2, 3\}$ a few attempts immediately show that RPs yield sets of projected vectors which are necessarily incongruent with the original vectors.

In this paper, we do not give a complete proof of the JLL, since many different ones have already been provided in research articles [92,54,89,6,95, 128,9] and textbooks [176,129,96,179]. We only prove the first part of the proof, namely the easy result that RPs preserve norms on average. This provides an explanation for the variance $1/k$ of the distribution from which the components of T are sampled.

Theorem 6 *Let T be a $k \times n$ RP sampled from $\mathsf{N}(0, \frac{1}{\sqrt{k}})$, and $u \in \mathbb{R}^n$; then $\mathsf{E}(\|Tu\|_2^2) = \|u\|_2^2$.*

Proof We prove the claim for $\|u\|_2 = 1$; the result will follow by scaling. For each $i \leq k$ we define $v_i = \sum_{j \leq n} T_{ij} u_j$. Then $\mathsf{E}(v_i) = \mathsf{E}\big(\sum_{j \leq m} T_{ij} u_j\big) = \sum_{j \leq m} \mathsf{E}(T_{ij}) u_j = 0$. Moreover,

$$\mathsf{Var}(v_i) = \sum_{j \leq m} \mathsf{Var}(T_{ij} u_j) = \sum_{j \leq m} \mathsf{Var}(T_{ij}) u_j^2 = \sum_{j \leq m} \frac{u_j^2}{k} = \frac{1}{k} \|u\|^2 = \frac{1}{k}.$$

Now, $\frac{1}{k} = \mathsf{Var}(v_i) = \mathsf{E}(v_i^2 - (\mathsf{E}(v_i))^2) = \mathsf{E}(v_i^2 - 0) = \mathsf{E}(v_i^2)$. Hence

$$\mathsf{E}(\|Tu\|^2) = \mathsf{E}(\|v\|^2) = \mathsf{E}\big(\sum_{i \leq k} v_i^2\big) = \sum_{i \leq k} \mathsf{E}(v_i^2) = \sum_{i \leq k} \frac{1}{k} = 1,$$

as claimed. □

7.3.2 Approximating the identity

If T is a $k \times n$ RP where $k = O(\varepsilon^{-2} \ln n)$, both $TT^\top$ and $T^\top T$ have some relation with the identity matrices I_k and I_n. This is a lesser known phenomenon, so it is worth discussing it here in some detail.

We look at $TT^\top$ first. By [199, Cor. 7] for any $\epsilon \in (0, \frac{1}{2})$ we have

$$\|\frac{1}{n} T\, T^\top - I_k\|_2 \leq \varepsilon$$

with probability at least $1 - \delta$ as long as $n \geq \frac{(k+1)\ln(2k/\delta)}{\mathcal{C}\varepsilon^2}$, where $\mathcal{C} \geq \frac{1}{4}$ is a constant.

In Table 1 we give values of $\|s\, TT^\top - I_d\|_2$ for $s \in \{1/n, 1/d, 1\}$, $n \in \{1000, 2000, \ldots, 10000\}$ and $d = \lceil \ln(n)/\epsilon^2 \rceil$ where $\epsilon = 0.15$. It is clear that

	n									
s	1e3	2e3	3e3	4e3	5e3	e3	7e3	8e3	9e3	1e4
$1/n$	9.72	7.53	6.55	5.85	5.36	5.01	4.71	4.44	4.26	4.09
$1/d$	5e1	1e2	1.5e2	2e2	2.5e2	3e2	3.5e2	3.9e2	4.4e2	4.8e2
1	2e5	4e5	6e5	8e5	1e6	1.2e6	1.4e6	1.6e6	1.8e6	2e6

Table 1 Values of $\|sTT^\top - I_d\|$ in function of s, n.

the error decreases as the size increases only in the case $s = \frac{1}{n}$. This seems to indicate that the scaling is a key parameter in approximating the identity.

Let us now consider the product $T^\top T$. It turns out that, for each fixed vector x not depending on T, the matrix $T^\top T$ behaves like the identity w.r.t. x.

Theorem 7 *Given any fixed $x \in \mathbb{R}^n$, $\epsilon \in (0,1)$ and a RP $P \in \mathbb{R}^{d\times n}$, there is a universal constant $\mathcal{C}$ such that*

$$-\mathbf{1}\varepsilon \leq T^\top Tx - x \leq \mathbf{1}\varepsilon. \tag{45}$$

with probability at least $1 - 4e^{\mathcal{C}\epsilon^2 d}$.

Proof By definition, for each $i \leq n$ we have $x_i = \langle e_i, x\rangle$, where e_i is the i-th unit coordinate vector. By elementary linear algebra we have $\langle e_i, T^\top Tx\rangle = \langle Te_i, Tx\rangle$. By [52, Lemma 3.1], for $i \leq n$ we have

$$\langle e_i, x\rangle - \epsilon\|x\|_2 \leq \langle Te_i, Tx\rangle \leq \langle e_i, x\rangle + \epsilon\|x\|$$

with arbitrarily high probability, which implies the result. □

One might be tempted to infer from Thm. 7 that $T^\top T$ "behaves like the identity matrix" (independently of x). This is generally false: Thm. 7 only holds for a given (fixed) x.

In fact, since T is a $k \times n$ matrix with $k < n$, $T^\top T$ is a square symmetric psd $n \times n$ matrix with rank k, hence $n-k$ of its eigenvalues are zero — and the nonzero eigenvalues need not have value one. On the other hand, $T^\top T$ looks very much like a slightly perturbed identity, on average, as shown in Table 2.

n	diagonal	off-diag
500	1.00085	0.00014
1000	1.00069	0.00008
1500	0.99991	-0.00006
2000	1.00194	0.00005
2500	0.99920	-0.00004
3000	0.99986	-0.00000
3500	1.00044	0.00000
4000	0.99693	0.00000

Table 2 Average values of diagonal and off-diagonal components of $T^\top T$ in function of n, where T is a $k \times n$ RP with $k = O(\varepsilon^{-2}\ln n)$ and $\epsilon = 0.15$.

7.3.3 Using RPs in MP

Random projections have mostly been applied to probabilistic approximation algorithms. By randomly projecting their (vector) input, one can execute algorithms with lower-dimensional vector more efficiently. The approximation guarantee is usually derived from the JLL or similar results.

A line of research about applying RPs to MP fromulations has been started in [183,182,181,52]. Whichever algorithm one may choose in order to solve the MP, the RP properties guarantee an approximation on optimality and/or feasibility. Thus, this approach leads to stronger/more robust results with respect to applying RPs to algorithmic input.

Linear and integer feasibility problems (i.e. LP and MILP formulations without objective function) are investigated in [183] from a purely theoretical points of view. The effect of RPs on LPs (with nonzero objective) are investigated in [182], both theoretically and computationally. Specifically, the randomly projected LP formulation is shown to have bounded feasibility error and an approximation guarantee on optimality. The computational results suggest that the range of practical application of this technique starts with relatively small LPs (thousands of variables/constraints). In both [183,182] we start from a (MI)LP in standard form

$$\mathcal{P} \equiv \min\{c^\top x \mid Ax = b \wedge x \geq 0 \wedge x \in X\}$$

(where $X = \mathbb{R}^n$ or $\mathbb{Z}^n$ respectively), and obtain a randomly projected formulation under the RP $T \sim \mathsf{N}^{n\times k}(0, \frac{1}{\sqrt{k}})$ with the form

$$T\mathcal{P} \equiv \min\{c^\top x \mid TAx = Tb \wedge x \geq 0 \wedge x \in X\},$$

i.e. T reduces the number of constraints in $\mathcal{P}$ to $O(\ln n)$, which can therefore be solved more efficiently.

The RP technique in [181,52] is different, insofar as it targets the number of variables. In [52] we consider a QP of the form:

$$\mathcal{Q} \equiv \max\{x^\top Qx + c^\top x \mid Ax \leq b\},$$

where Q is $n \times n$, $c \in \mathbb{R}^n$, A is $m \times n$, and $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$. This is projected as follows:

$$T\mathcal{Q} \equiv \max\{u^\top \bar{Q}x + \bar{c}^\top u \mid \bar{A}u \leq b\},$$

where $\bar{Q} = TQT^\top$ is $k \times k$, $\bar{A} = AT^\top$ is $m \times k$, $\bar{c} = Tc$ is in $\mathbb{R}^k$, and $u \in \mathbb{R}^k$. In [181] we consider a QCQP $\mathcal{Q}'$ like $\mathcal{Q}$ but subject to a ball constraint $\|x\|_2 \leq 1$. In the projected problem $T\mathcal{Q}'$, this is replaced by a ball constraint $\|u\|_2 \leq 1$. Both [52,181] are both theoretical and computational. In both cases, the number of variables of the projected problem is $O(\ln n)$.

In applying RPs to MPs, one solves the smaller projected problems in order to obtain an answer concerning the corresponding original problems. In most cases one has to devise a way to retrieve a solution for the original problem using the solution of the projected problem. This may be easy or difficult depending on the structure of the formulation and the nature of the RP.

8 Distance instability

Most of the models and methods in this survey are based on the concept of distance: usually Euclidean, occasionally with other norms. The k-means algorithm (Sect. 5.1.1) is heavily based on Euclidean distances in Step 2 (p. 20), where the reassignment of a point to a cluster is carried out based on proximity: in particular, one way to implement Step 2 is to solve a 1-nearest neighbor problem. The training of an ANN (Sect. 5.1.2) repeatedly solves a minimum distance subproblem in Eq. (10). In spectral clustering (Sect. 5.2.1) we have a Euclidean norm constraint in Eq. (12). All DGP solution methods (Sect. 6), with the exception of incidence vectors (Sect. 6.2.1), are concerned with distances by definition. PCA (Sect. 7.1), in its interpretation of a modified MDS, can be seen as another solution method for the DGP. Barvinok's naive algorithm (Sect. 7.2) is a dimensional reduction method for SDPs the analysis of which is based on a distance bound; moreover, it was successfully applied to the DGP [123]. The RP-based methods discussed in Sect. 7.3 have all been derived from the JLL (Sect. 7.3.1), which is a statement about the Euclidean distance. We also note that the focus of this survey is on typical DS problems, which are usually high-dimensional.

It is therefore absolutely essential that all of these methods should be able to take robust decisions based on comparing distance values computed on pairs of high-dimensional vectors. It turns out, however, that smallest and largest distances $D_{\mathsf{min}}, D_{\mathsf{max}}$ of a random point $Z \in \mathbb{R}^n$ to a set of random points $X_1, \ldots, X_\ell \subset \mathbb{R}^n$ are almost equal (and hence, difficult to compare) as $n \to \infty$ under some reasonable conditions. This holds for any distribution used to sample Z, X_i. This result, first presented in [26] and subsequently discussed in a number of papers [85,3,72,63,157,127,69], appears to jeopardize all of the material presented in this survey, and much more beyond. The phenomenon leading to the result is known as *distance instability* and *concentration of distances*.

8.1 Statement of the result

Let us look at the exact statement of the distance instability result.

First, we note that the points $Z, X_1, \ldots, X_\ell$ are not given points in $\mathbb{R}^n$ but rather multivariate random variables with n components, so distance instability is a purely statistical statement rather than a geometric one. We consider

$$\begin{aligned} Z &= (Z_1, \ldots, Z_n) \\ \forall i \leq \ell \quad X_i &= (X_{i1}, \ldots, X_{in}), \end{aligned}$$

where $Z_1, \ldots, Z_n$ are random variables with distribution $\mathcal{D}_1$; $X_{11}, \ldots, X_{\ell n}$ are random variables with distribution $\mathcal{D}_2$; and all of these random variables are independently distributed.

Secondly, $D_{\mathsf{min}}, D_{\mathsf{max}}$ are functions of random variables:

$$D_{\mathsf{min}} = \min\{\mathsf{dist}(Z, X_i) \mid i \leq \ell\} \tag{46}$$
$$D_{\mathsf{max}} = \max\{\mathsf{dist}(Z, X_i) \mid i \leq \ell\}, \tag{47}$$

and are therefore random variables themselves. In the above, dist denotes a function mapping pairs of points in $\mathbb{R}^n$ to a non-negative real number, which makes distance instability a very general phenomenon. Specifically, dist need not be a distance at all.

Third, we now label every symbol with an index m, which will be used to compute limits for $m \to \infty$: Z^m, X^m, $\mathcal{D}_1^m$, $\mathcal{D}_2^m$, D_{min}^m, D_{max}^m, dist^m. We shall see that the proof of the distance instability result is wholly syntactical: its steps are very simple and follow from basic statistical results. In particular, we can see m as an abstract parameter under which we shall take limits, and the proof will hold. Since the proof holds independently of the value of n, it also holds if we assume that $m = n$, i.e. if we give m the interpretation of dimensionality of the Euclidean space embedding the points. While this assumption is not necessary for the proof to hold, it may simplify its understanding: $m = n$ makes the proof somewhat less general, but it gives the above indexing a more concrete meaning. Specifically, $Z, X, \mathcal{D}, D, \mathsf{dist}$ are points, distributions, extreme distance values and a distance function in dimension m, and the limit $m \to \infty$ is a limit taken on increasing dimension.

Fourth, the "reasonable conditions" referred to above for the distance instability result to hold are that there is a constant $p > 0$ such that

$$\exists i \leq \ell \quad \lim_{m \to \infty} \mathsf{Var}\left(\frac{(\mathsf{dist}(Z^m, X_i^m))^p}{\mathsf{E}((\mathsf{dist}(Z^m, X_i^m))^p)}\right) = 0. \tag{48}$$

A few remarks on Eq. (48) are in order.

(a) The existential quantifier simply encodes the fact that the X_i are all identically distributed, so a statement involving variance and expectation of quantities depending on the X_i random variables holds for all $i \leq \ell$ if it holds for just one X_i.
(b) The constant p simply gives more generality to the result, but plays no role whatsoever in the proof; it can be used in order to simplify computations when dist is an ℓ_p norm.
(c) The fraction term in Eq. (48) measures a spread relative to an expectation. Requiring that the limit of this relative spread goes to zero for increasing dimensions looks like an asymptotic concentration requirement (hence the alternative name "distance concentration" for the distance instability phenomenon). Considering the effect of concentration of measure phenomena in high dimensions (Sect. 7.2.2), distance instability might now appear somewhat less surprising.

With these premises, we can state the distance instability result.

Theorem 8 *If D^m_{min} and D^m_{max} are as in Eq.* (46)-(47) *and satisfy Eq.* (48)*, then, for any $\varepsilon > 0$, we have*

$$\lim_{m\to\infty} \mathsf{P}\left(D^m_{\mathsf{max}} \leq (1+\varepsilon)D^m_{\mathsf{min}}\right) = 1. \tag{49}$$

Thm. 8 basically states that closest and farthest neighbors of Z are indistinguishable up to an ε. If the closest and farthest are indistinguishable, trying to discriminate between the closest and the second closest neighbors of a given point might well be hopeless due to floating point errors (note that this discrimination occurs at each iteration of the well known k-means algorithm). This is why distance instability is sometimes cited as a reason for convergence issues in k-means [74].

8.2 Related results

In [26], several scenarios are analyzed to see where distance instability occurs — even if some of the requirement of distance instability are relaxed [26, §3.5] — and where it does not [26, §4]. Among the cases where distance instability does not apply, we find the case where the data points X are well separated and the case where the dimensionality is implicitly low. Among the cases where it does apply, we find k-NN: in their experiments, the authors of [26] find that k-NN becomes unstable already in the range $n \in \{10, 20\}$ dimensions. Obviously, the instability of k-NN propagates to any algorithm using k-NN, such as k-means.

Among later studies, [85] proposes an alternative definition of dist where high-dimensional points are projected into lower dimensional spaces. In [85], the authors study the impact of distance instability on different ℓ_p norms, and concludes that smallest values of p lead to more stable norms; in particular, quasinorms with $0 < p < 1$ are considered. Some counterexamples are given against a generalization of this claim for quasinorms in [72]. In [63], the converse of Thm. 8 is proved, namely that Eq. (48) follows from Eq. (49): from this fact, the authors find practically relevant cases where Eq. (48) is not verified, and propose them as "good" examples of where k-means can help. In [127], the authors propose multiplicative functions dist and show that they are robust w.r.t. distance instability. In [157], distance instability is related to "hubness", i.e. the number of times a point appears among the k nearest neighbors of other points. In [69], an empirical study is provided which shows how to show an appropriate ℓ_p norm that should avoid distance instability w.r.t. hubness.

8.3 The proof

The proof of the instability theorem can be found in [26]. We repeat it here to demonstrate the fact that it is "syntactical": every step follows from the previous ones by simple logical inference. There is no appeal to any results other

than convergence in probability, Slutsky's theorem, and a simple corollary as shown below. The proof does not pass from object language to meta-language, nor does it require exotic interpretations of symbols in complicated contexts. Although one may find this result surprising, there appears to be no reason to doubt it, and no complication in the proof warranting sophisticated interpretations. The only point worth re-stating is that this is a result about probability distributions, not about actual instances of real data.

Lemma 1 *Let* $\{B^m \mid m \in \mathbb{N}\}$ *be a sequence of of random variables with finite variance. Assume that* $\lim_{m\to\infty} \mathsf{E}(B^m) = b$ *and that* $\lim_{m\to\infty} \mathsf{Var}(B^m) = 0$. *Then*

$$\forall \varepsilon > 0 \quad \lim_{m\to\infty} \mathsf{P}(\|B^m - b\| \leq \varepsilon) = 1. \tag{50}$$

A random variable sequence satisfying Eq. (50) is said to *converge in probability* to b. This is denoted $B^m \to_{\mathsf{P}} b$.

Lemma 2 (Slutsky's theorem [190]) *Let* $\{B^m \mid m \in \mathbb{N}\}$ *be a sequence of random variables, and* $g : \mathbb{R} \to \mathbb{R}$ *be a continuous function. If* $B^m \to_{\mathsf{P}} b$ *and* $g(b)$ *exists, then* $g(B^m) \to_{\mathsf{P}} g(b)$.

Corollary 1 *If* $\{A^m \mid m \in \mathbb{N}\}$ *and* $\{B^m \mid m \in \mathbb{N}\}$ *are sequences of random variables such that* $A^m \to_{\mathsf{P}} a$ *and* $B^m \to_{\mathsf{P}} b \neq 0$, *then* $\frac{A^m}{B^m} \to_{\mathsf{P}} \frac{a}{b}$.

Proof of Thm. 8. Let $\mu_m = \mathsf{E}((d^m(Z^m, X_i^m))^p)$. We note that μ_m is independent of i since all X_i^m are identically distributed.

We claim $V_m = \frac{(d^m(Z^m,X_i^m))^p}{\mu_m} \to_{\mathsf{P}} 1$:

- we have $\mathsf{E}(V_m) = 1$ since it is a random variable over its mean: hence, trivially, $\lim_m \mathsf{E}(V_m) = 1$;
- by the hypothesis of the theorem (Eq. (48)), $\lim_m \mathsf{Var}(V_m) = 0$;
- by Lemma 1, $V_m \to_{\mathsf{P}} 1$, which establishes the claim.

Now, let $\mathbf{V}^m = (V_m \mid i \leq \ell)$. By the claim above, we have $\mathbf{V}^m \to_{\mathsf{P}} \mathbf{1}$. Now by Lemma 2 we obtain $\min(\mathbf{V}^m) \to_{\mathsf{P}} \min(\mathbf{1}) = 1$ and, similarly, $\max(\mathbf{V}^m) \to_{\mathsf{P}} 1$. By Cor. 1, $\frac{\max(\mathbf{V}^m)}{\min(\mathbf{V}^m)} \to_{\mathsf{P}} 1$. Therefore,

$$\frac{D^m_{\mathsf{max}}}{D^m_{\mathsf{min}}} = \frac{\mu_m \max(\mathbf{V}^m)}{\mu_m \min(\mathbf{V}^m)} \to_{\mathsf{P}} 1.$$

By definition of convergence in probability, we have

$$\forall \varepsilon > 0 \quad \lim_{m\to\infty} \mathsf{P}(|D^m_{\mathsf{max}}/D^m_{\mathsf{min}} - 1| \leq \varepsilon) = 1.$$

Moreover, since $\mathsf{P}(D^m_{\mathsf{max}} \geq D^m_{\mathsf{min}}) = 1$, we have

$$\mathsf{P}(D^m_{\mathsf{max}} \leq (1+\varepsilon)D^m_{\mathsf{min}}) = \mathsf{P}(D^m_{\mathsf{max}}/D^m_{\mathsf{min}} - 1 \leq \varepsilon) = \mathsf{P}(|D^m_{\mathsf{max}}/D^m_{\mathsf{min}} - 1| \leq \varepsilon) = 1.$$

The result follows by taking the limit as $m \to \infty$. □

8.4 In practice

In Fig. 7, we show how ε (Eq. (49)) varies with increasing dimension n (recall we assume $m = n$) between 1 and 10000. It is clear that ε decreases very

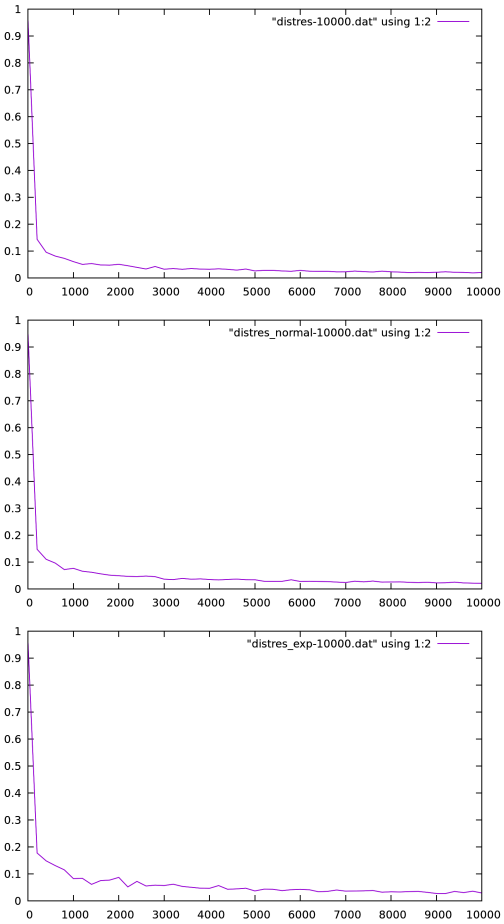

Fig. 7 Plots of ε versus n for the uniform distribution on $[0, 1]$ (left), $\mathsf{N}(0, 1)$ (center), the exponential distribution with parameter 1 (right).

rapidly towards zero, and then reaches its asymptotic value more slowly. On the other hand, ε is the distortion between minimum and maximum distance values; most algorithms need to discriminate between smallest and second smallest distance values.

Most of the papers listed in Sect. 8.2 include empirical tests which illustrate the impact and limits of the distance instability phenomenon.

9 An application to neural networks

In this last section we finally show how several concepts explained in this survey can be used conjunctively. We shall consider a natural language processing task (Sect. 4) where we cluster some sentences (Sect. 5) using an ANN (Sect. 5.1.2) with different training sets $T = (X, Y)$. We compare ANN performances depending on the training set used.

The input set X is a vector representation of the input sentences. The output set Y is a vectorial representation of cluster labels: we experiment with (a) clusterings obtained by running k-means (Sect. 5.1.1) on the input sets, and (b) a clustering found by a modularity maximization heuristic (Sect. 5.2.2). All of these clusterings are considered "ground truth" sets Y we would like our ANN to learn to associate to various types of input vector sets X representing the sentences. The sentences to be clustered are first transformed into graphs (Sect. 4.2), and then into vectors (Sect. 6), which then undergo dimensionality reduction (Sect. 7).

Our goal is to compare the results obtained by the same ANN with different vector representations for the same text: most notably, the comparison will confront how well or poorly input vector sets can predict the ground truth outputs. We will focus specifically on a comparison of the well-known incidence vectors (Sect. 6.2.1) embeddings w.r.t. the newly proposed DGP methods we surveyed in Sect. 6.

In our implementations, all our code was developed using Python 3 [158].

9.1 Performance measure

We are going to measure the performance quality of the error of an ANN, which is based on a comparison of its output with the ground truth that the ANN is supposed to learn. Using the notation of Sect. 5.1.2, if the ANN output for a given input $x \in \mathbb{R}^n$ consists of a vector $y \in \mathbb{R}^k$, and if the ground truth corresponding to x is $z \in \mathbb{R}^k$, then we define the error as the *loss* function:

$$\mathsf{loss}(y, z) = \|y - z\|_2. \tag{51}$$

An ANN $\mathcal{N} = (G, T, \phi)$ is usually evaluated over many (input,output) pairs. Let $\hat{X} \subset \mathbb{R}^n$ and $\hat{Y} \subset \mathbb{R}^k$ be, respectively, a set of input vectors and the corresponding set of output vectors evaluated by the trained ANN. Let $\hat{Z}$ be a set of ground truth vectors corresponding to $\hat{X}$, and assume $|\hat{X}| = |\hat{Y}| = |\hat{Z}| = q$. The cumulative loss measure evaluated on the *test set* $(\hat{X}, \hat{Z})$ is then

$$\mathsf{loss}(\mathcal{N}) = \frac{1}{q} \sum_{\substack{y \in \hat{Y} \\ z \in \hat{Z}}} \mathsf{loss}(y, z). \tag{52}$$

9.2 A Natural Language Processing task

Clustering of sentences in a text is a common task in Natural Language Processing. We considered "On the duty of civil disobedience" by H.D. Thoreau [174,184]. This text is stored in an ASCII file `walden.txt` which can be obtained from `archive.org`. The file is 661146 bytes long, organized in 10108 lines and 116608 words. The text was parsed into sentences using basic methods from NLTK [27] under Python 3. Common words, stopwords, punctuation and unusual characters were removed. After "cleaning", the text was reduced to 4083 sentences over a set of 11431 "significant" words (see Sect. 9.2.1).

As mentioned above, we want to train our ANN to learn different types of clusterings:

- (***k-means***) obtained by running the the k-means unsupervised clustering algorithm (Sect. 5.1.1) over the different vector representations of the sentences in the text;
- (***sentence graph***) obtained by running a modularity clustering heuristic (Sect. 5.2.2) on a graph representation of the sentences in the document (see Sect. 9.2.2).

These clusterings are used as ground truths, and provide the output part of the training sets to be used by the ANN, as well as of the test sets for measuring purposes (Sect. 9.1). See Sect. 9.4.1 for more information on the construction of these clusterings.

9.2.1 Selecting the sentences

We constructed two sets of sentences.

- **The large sentence set.** Each sentence in `walden.txt` was mapped to an incidence vector of 3-grams in $\{0,1\}^{48087}$, i.e. a dictionary of 48087 3-grams over the text. In other words, 48087 3-grams were found in the text, then each sentence was mapped to a vector having 1 at component i iff the i-th 3-gram was present in the sentence. Since some sentences had fewer than 3 significant words, only 3940 sentences remained in the sentence set S, which was therefore represented as a 3940×48087 matrix $\bar{S}$ with components in $\{0,1\}$.
- **The small sentence set.** It turns out that most of the 3-grams in the set S only appear a single time. We selected a subset $S' \subset S$ of sentences having 3-grams appearing in at least two sentences. It turns out that $|S'| = 245$, and the total number of 3-grams appearing more than once is 160. S' is therefore naturally represented as a 245×160 matrix $\bar{S}'$ with components in $\{0,1\}$.

We constructed training sets (Sect. 9.4) for each of these two sets.

9.2.2 Construction of a sentence graph

In this section we describe the method used to construct a sentence graph $G^{\mathsf{s}} = (S, E)$ from the text, which is used to produce a ground truth for the (***sentence graph***) type. G^{s} is then clustered using the greedy modularity clustering heuristic in the Python library `networkX` [81].

Each sentence in the text is encoded into a weighted graph-of-word (see Sect. 4.2.1) over 3-grams, with edges $\{u, v\}$ weighted by the number c_{uv} of 3-grams where the two words u, v appear. The union of the graph-of-words for the sentences (contracting repeated words to a single vertex) yields a weighted graph-of-word G^{w} for the whole text.

The graph $G^{\mathsf{w}} = (W, F)$ is then "projected" onto the set S of sentences as follows. We define the logical proposition $P(u, v, s, t)$ to mean $(u \in s \wedge v \in t) \vee (v \in s \wedge u \in t)$ for words u, v and sentences s, t. The edge set E of G^{s} is then defined by the following implication:

$$\forall \{u, v\} \in F,\ s, t \in S \qquad P(u, v, s, t) \rightarrow \{s, t\} \in E.$$

In other words, s, t form an edge in E if two words u, v in s, t (respectively) or t, s form an edge in F. For each edge $\{s, t\} \in E$, the weight w_{st} is given by:

$$w_{st} = \sum_{\substack{\{u,v\} \in F \\ P(u,v,s,t)}} c_{uv},$$

with edge weights meaning similarity.

9.3 The ANN

We consider a very simple ANN $\mathcal{N} = (G, T, \phi)$. In the terminology of Sect. 5.1.2, the underlying digraph $G = (V, A)$ is tripartite with $V = V_1 \dot\cup V_2 \dot\cup V_3$. The "input layer" V_1 has n nodes, where n is the dimensionality of the input vector set X. The "output layer" V_3 has a single node. The "hidden layer" V_2 has a constant number of nodes (20 in our experiments). The training set T is discussed in Sect. 9.4. We adopt the piecewise-linear mapping known as *rectified linear unit* (ReLu) [189] for the activation functions ϕ in V_2, and a traditional sigmoid function for the single node in V_3. Both types of activation functions map into $[0, 1]$.

We implemented $\mathcal{N}$ using the Python library `keras` [44], which is a high-level API running over TensorFlow [1]. The default configuration was chosen for all layers. We used the Adam solver [98] in order to train the network. Each training set was split in three parts: 35% of the vectors were used for training, 35% for *validation* (a training phase used for deciding values of any model parameter aside from v, b, w, if any exist, and/or for deciding when to stop the training phase), and 30% for testing. The performance of the ANN is measured using the loss function in Eq. (52).

9.4 Training sets

Our goal is to compare training sets $T = (X, Y)$ where the vectors in X were constructed in a variety of ways, and the vectors in Y were obtained by running k-means (Sect. 5.1.1) on the vectors in X.

In particular, we consider input sets $X(\sigma, \mu, \rho)$ where:

- $\sigma \in \Sigma = \{S', S\}$ is the sentence set: $\sigma = S'$ corresponds to the small set with 245 sentences, $\sigma = S$ corresponds to the large set with 3940 sentences;
- $\mu \in M = \{\mathsf{inc}, \mathsf{uie}, \mathsf{qrt}, \mathsf{sdp}\}$ is the method used to map sentences to vectors: inc are the incidence vectors (Sect. 6.2.1), uie is the universal isometric embedding (Sect. 6.2.2), qrt is the unconstrained quartic (Sect. 6.1.1), sdp is the SDP (Sect. 6.1.3);
- $\rho \in R = \{\mathsf{pca}, \mathsf{rp}\}$ is the dimensional reduction method used: pca is PCA (Sect. 7.1), rp are RPs (Sect. 7.3).

The methods in M were all implemented using Python 3 with some well known external libraries (e.g. `numpy`, `scipy`). Specifically, qrt was implemented using the Ipopt [47] NLP solver, and sdp was implemented using the SCS [148] SDP solver. As for the dimensional reduction methods in R, the PCA implementation of choice was the probabilistic PCA algorithm implemented in the Python library `scikit-learn` [150]. The RPs we chose were the simplest: each component of the RP matrices was sampled from an appropriately scaled zero-mean Gaussian distribution (Thm. 6).

9.4.1 The output set

The output set Y should naturally contain discrete values, namely the labels of the h clusters $\{1, 2, \ldots, h\}$ in the ground truth clusterings. We map these values to scalars in $[0, 1]$ (or, according to Sect. 5.1.2, to k-dimensional vectors with $k = 1$) as follows. We divide the range $[0, 1]$ into $h - 1$ equal sub-intervals of length $1/(h-1)$, and hence h discrete values in $[0, 1]$. Then we assign labels to sub-intervals endpoints: label j is mapped to $(j-1)/(h-1)$ (for $1 \leq j \leq h$).

As mentioned above, we consider two types of output sets:

- (***k-means***) for each input set $X(\sigma, \mu, \rho)$ we obtained an output set $Y(\sigma, \mu, \rho)$ using k-means (Sect. 5.1.1) implementation in `scikit-learn` [150] on the vectors in X, for each sentence set $\sigma \in \Sigma$, method $\mu \in M$, and dimensional reduction method $\rho \in R$;
- (***sentence graph***) for each sentence set $\sigma \in \Sigma$ we constructed a sentence graph as detailed in Sect. 9.2.2.

9.4.2 Realizations to vectors

The inc method (Sect. 6.2.1) is the only one (in our benchmark) that can natively map sentences of various lengths into vectors all having the same number of components.

For all other methods in $M \setminus \{\mathsf{inc}\}$, we loop over sentences (in small/large sets S', S). For each sentence we construct its graph-of-words (Sect. 4.2.1). We then realize it in some arbitrary dimensional Euclidean space $\mathbb{R}^K$ (specifically, we chose $K = 10$) using uie, qrt, sdp. At this point, we are confronted with the following difficulty: a realization of a graph G with p vertices in $\mathbb{R}^K$ is a $p \times K$ matrix, and we have as many graphs G as we have sentences, with p varying over the number of unique words in the sentences (i.e. the cardinalities of the vertex sets of the graphs-of-words).

In order to reduce all of these differently-sized realizations to vectors having the same dimension, we follow the following procedure. Given realizations $\{x^i \in \mathbb{R}^{p_i \times K} \mid i \in \sigma\}$, where σ is the set of sentences (for $\sigma \in \Sigma$) and x^i realizes the graph-of-word of sentence $i \in \sigma$,

1. we stack the columns of x^i so as to obtain a single vector $\hat{x}^i \in \mathbb{R}^{p_i K}$ for each $i \in \sigma$;
2. we let $\hat{n} = \max_i p_i K$ be the maximum dimensionality of the stacked realizations;
3. we pad every realization vector $\hat{x}^i$ shorter than $\hat{n}$ with zeros to achieve dimension $\hat{n}$ for stacked realization vectors;
4. we form the $s \times \hat{n}$ matrix $\hat{X}$ having $\hat{x}^i$ as its i-th row (for $i \in \sigma$ and with $s = |\sigma|$);
5. we reduce the dimensionality of $\hat{X}$ to an $s \times n$ matrix X with pca or rp.

9.5 Computational comparison

We discuss the details of our training sets, a validation test, and the comparison tests.

9.5.1 Training set statistics

In Table 3 we report the dimensionalities of the vectors in the input parts $X(\sigma, \mu, \rho)$ of the training sets, as well as the number of clusters in the output sets $Y(\sigma, \mu, \rho)$ of the (***k-means***) class. We recall that the number of clusters

Dimensionality of input vectors								
μ	$\lvert\sigma\rvert = 245$				$\lvert\sigma\rvert = 3940$			
ρ	inc	uie	qrt	sdp	inc	uie	qrt	sdp
pca	3	159	244	200	3	10	400	400
rp	100	248	248	248	373	373	373	373
original	*160*	*1140*	*1140*	*1140*	*48087*	*1460*	*1460*	*1460*
Number of clusters to learn								
pca	4	3	11	6	3	8	9	14
rp	4	3	7	5	3	9	16	14

Table 3 Training set statistics for $X(\sigma, \mu, \rho)$ and corresponding output sets in the (***k-means***) class.

was found with k-means in the `scikit-learn` implementation. The choice of 'k' corresponds to the smallest number of clusters giving a nontrivial clustering (with "trivial" meaning having a cluster of zero cardinality, or too close to zero relative to the set size, only possibly allowing some outlier clusters with a single element). Some more remarks follow.

- For $\rho = \mathsf{pca}$ we employed the smallest dimension such that the residual variance in the neglected components was almost zero; this ranges from 3 to 244 in Table 3. For the two cases where the dimensionality reduction was set to 400 (qrt and sdp in the large sentence set S), the residual variance was nonzero.
- It is interesting that for $\mu = \mathsf{uie}$ we have higher projected dimensionality (248) in the small set S' than in the large set S (10): this depends on the fact that the large set has more easily distinguishable clusters (8 found by k-means) than the small set (only 3 found by k-means). The dimension of $X(\mathsf{inc}, \mathsf{pca}, S)$ is smaller (3) than that of $X(\mathsf{uie}, \mathsf{pca}, S)$ (10) even though the original number of dimensions of the former (48087) vastly exceeds that of the latter (1460) for the same reason.
- The training sets $X(\sigma, \mathsf{inc}, \mathsf{pca})$ are the smallest-dimensional ones (for $\sigma \in \{S', S\}$): they are also "degenerate", in the sense that the vectors in a given clusters are all equal; the co-occurrence patterns of the incidence vectors conveyed relatively little information to this vectorial sentence representation.
- The RP-based dimensionality reduction method yields the same dimensionality (373) of $X(\mu, \mathsf{rp}, S)$ for $\mu \in M$. This occurs because the target dimensionality in RP depends on the number of vectors, which is the same for all methods (3940), rather than on the number of dimensions (see Sect. 7.3).

There is one output set in the (***sentence graph***) class for each $\sigma \in \Sigma$. For $\sigma = S'$ we have $|V| = 245$, $|E| = 28519$, and 230 clusters, with the first 5 clusters having $6, 5, 4, 3, 2$ elements, and the rest having a single element. For $\sigma = S$ we have $|V| = 3940$, $|E| = 7173633$, and 3402 clusters, with the first 10 clusters having $161, 115, 62, 38, 34, 29, 19, 16, 14, 11$ elements, and the rest having fewer than 10 elements.

9.5.2 Comparison tests

We first report the comparative results of the ANN on

$$T = (X(\sigma, \mu_1, \rho_1), Y(\sigma, \mu_2, \rho_2))$$

for $\sigma \in \Sigma$, $\mu_1, \mu_2 \in M$, $\rho_1, \rho_2 \in R$. The sums in the rightmost columns of Table 4 are only carried out on terms obtained with an input vector generation method μ_1 different from the method μ_2 used to obtain the ground truth clustering via k-means (since we want to compare methods). The results corresponding to cases where $\mu_1 = \mu_2$ are emphasized in italics in the table. The best performance sums are emphasized in boldface, and the worst are shown in grey.

Training set inputs	Training set outputs									
	μ	inc	inc	uie	uie	qrt	qrt	sdp	sdp	sum
	ρ	pca	rp	pca	rp	pca	rp	pca	rp	$\mu' \neq \mu$
	$\lvert\sigma\rvert$	245								
	inc pca	*0.061*	*0.042*	0.059	0.013	0.094	0.108	0.064	0.025	**0.363**
	inc rp	*0.005*	*0.010*	0.055	0.015	0.104	0.109	0.065	0.025	0.373
	uie pca	0.271	0.052	*0.070*	*0.169*	0.233	0.201	0.127	0.111	0.995
	uie rp	0.093	0.026	*0.094*	*0.076*	0.191	0.236	0.079	0.117	0.976
	qrt pca	0.082	0.067	0.105	0.047	*0.084*	*0.133*	0.071	0.087	0.459
	qrt rp	0.057	0.068	0.059	0.053	*0.162*	*0.073*	0.095	0.055	0.387
	sdp pca	0.106	0.063	0.067	0.022	0.106	0.135	*0.058*	*0.034*	0.499
	sdp rp	0.095	0.065	0.093	0.021	0.103	0.139	*0.074*	*0.018*	0.516
	$\lvert\sigma\rvert$	3940								
	inc pca	*0.052*	*0.013*	0.068	0.027	0.106	0.164	0.079	0.161	0.605
	inc rp	*0.001*	*0.000*	0.067	0.028	0.106	0.167	0.080	0.159	0.607
	uie pca	0.063	0.022	*0.020*	*0.016*	0.124	0.201	0.070	0.127	0.607
	uie rp	0.061	0.023	*0.024*	*0.023*	0.131	0.190	0.072	0.126	0.603
	qrt pca	0.063	0.022	0.36	0.023	*0.038*	*0.218*	0.079	0.159	**0.382**
	qrt rp	0.062	0.024	0.047	0.025	*0.120*	*0.035*	0.076	0.164	0.398
	sdp pca	0.063	0.021	0.023	0.024	0.126	0.195	*0.033*	*0.149*	0.452
	sdp rp	0.059	0.021	0.025	0.024	0.121	0.176	*0.083*	*0.037*	0.426

Table 4 Comparison tests on output sets of (***k-means***) class.

According to Table 4, for the small sentence set the best method is inc, but qrt and sdp are not far behind; the only really imprecise method is uie. For the large sentence set the best method is qrt, with sdp not far behind; both inc, uie are imprecise.

Training inputs	Training set outputs								
	μ	inc	inc	uie	uie	qrt	qrt	sdp	sdp
	ρ	pca	rp	pca	rp	pca	rp	pca	rp
	$\lvert\sigma\rvert$	245							
		0.107	0.108	0.196	0.184	0.129	0.151	0.109	0.122
	$\lvert\sigma\rvert$	3940							
		0.097	0.098	0.124	0.119	0.136	0.113	0.114	0.106

Table 5 Comparison tests on output sets of (***sentence graph***) class.

In Table 5, which has a similar format as Table 4, we report results on training sets

$$\bar{T} = (X(\sigma, \mu, \rho), \bar{Y}(\sigma))$$

for $\sigma \in \Sigma$, $\mu \in M$, $\rho \in R$, where $\bar{Y}(\sigma)$ are output sets of the (***sentence graph***) class. For the small set, inc is the best method (independently of ρ), with ($\mu = \mathsf{sdp}, \rho = \mathsf{pca}$) following very closely, and, in general, sdp and qrt still being acceptable; uie is the most imprecise method. For the large set inc is againt the best method, with ($\mu = \mathsf{sdp}, \rho = \mathsf{rp}$) following closely. While the other method do not excel, the performance difference between all methods is less remarkable than with the small set.

10 Conclusion

We have surveyed some of the concepts and methodologies of distance geometry which are used in data science. More specifically, we have looked at algorithms (mostly based on mathematical programming) for representing graphs as vectors as a pre-processing step to performing some machine learning task requiring vectorial input.

We started with brief introductions to mathematical programming and distance geometry. We then showed some ways to represent data by graphs, and introduced clustering on vectors and graphs. Following, we surveyed robust algorithms for realizing weighted graphs in Euclidean spaces, where the robustness is with respect to errors or noise in the input data. It turns out that most of these algorithms are based on mathematical programming. Since some of these algorithms output high-dimensional vectors and/or high-rank matrices, we also surveyed some dimensional reduction techniques. We also discussed a result about the instability of distances with respect to randomly generated points.

The guiding idea in this survey is that, when one is confronted with clustering on graphs, then distance geometry allows the use of many supervised and unsupervised clustering techniques based on vectors. To demonstrate the applicability of this idea, we showed that vectorial representations of graphs obtained using distance geometry offer competitive performances when training an artificial neural network. While we do not think that our limited empirical analysis allows any definite conclusion, we hope that it will entice more research in this area.

Acknowledgements

I am grateful to J.J. Salazar, the Editor-in-Chief of TOP, for inviting me to write this survey. This work would not have been possible without the numerous co-authors with whom I pursued my investigations in distance geometry, among which I will single out the longest-standing: C. Lavor, N. Maculan, A. Mucherino. I have first heard of concentration of measure as I passed by D. Malioutov's office at the T.J. Watson IBM Research laboratory: the door was open, the Johnson-Lindenstrauss lemma was mentioned, and I could not refrain from interrupting the conversation and asking for clarification,

as I thought it must be a mistake; incredibly, it was not, and I am grateful to Dmitry Malioutov for hosting the conversation I eavesdropped on. I very thankful to the co-authors who helped me investigate random projections, in particular P.L. Poirion and K. Vu, without whom none of our papers would have been possible. I learned about the existence of the distance instability result thanks to N. Gayraud, who suggested it to me as I expressed puzzlement at the poor quality of k-means clusterings during my talk. I am very grateful to S. Khalife and M. Escobar for reading the manuscript and making insightful comments.

References

1. Abadi, M., *et al.*: TensorFlow: Large-scale machine learning on heterogeneous systems. `http://tensorflow.org/` (2015). Software available from tensorflow.org
2. Achlioptas, D.: Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of Computer and System Sciences **66**, 671–687 (2003)
3. Aggarwal, C., Hinneburg, A., Keim, D.: On the surprising behavior of distance metrics in high dimensional space. In: J.V. den Bussche, V. Vianu (eds.) Proceedings of ICDT, *LNCS*, vol. 1973, pp. 420–434. Springer, Berlin (2001)
4. Ahmadi, A., Jungers, R., Parrilo, P., Roozbehani, M.: Joint spectral radius and path-complete graph Lyapunov functions. SIAM Journal on Optimization and Control (to appear)
5. Ahmadi, A., Majumdar, A.: 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)
6. Ailon, N., Chazelle, B.: Approximate nearest neighbors and fast Johnson-Lindenstrauss lemma. In: Proceedings of the Symposium on the Theory Of Computing, *STOC*, vol. 06. ACM, Seattle (2006)
7. Alfakih, A., Khandani, A., Wolkowicz, H.: Solving Euclidean distance matrix completion problems via semidefinite programming. Computational Optimization and Applications **12**, 13–30 (1999)
8. Allen, G.: Sparse higher-order principal components analysis. In: N. Lawrence, M. Girolami (eds.) Proceedings of the International Conference on Artificial Intelligence and Statistics, *Proceedings of Machine Learning Research*, vol. 22, pp. 27–36. PMLR, La Palma (2012). URL `http://proceedings.mlr.press/v22/allen12.html`
9. Allen-Zhu, Z., Gelashvili, R., Micali, S., Shavit, N.: Sparse sign-consistent Johnson-Lindenstrauss matrices: Compression with neuroscience-based constraints. Proceedings of the National Academy of Sciences **111**(47), 16872–16876 (2014)
10. Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Perron, S., Liberti, L.: Column generation algorithms for exact modularity maximization in networks. Physical Review E **82**(4), 046112 (2010)
11. Aloise, D., Caporossi, G., Hansen, P., Liberti, L., Perron, S., Ruiz, M.: Modularity maximization in networks by variable neighbourhood search. In: D. Bader, P. Sanders, D. Wagner (eds.) Graph Partitioning and Graph Clustering, *Contemporary Mathematics*, vol. 588, pp. 113–127. American Mathematical Society, Providence, RI (2013)
12. Aloise, D., Hansen, P., Liberti, L.: An improved column generation algorithm for minimum sum-of-squares clustering. Mathematical Programming A **131**, 195–220 (2012)
13. Amaldi, E., Liberti, L., Maffioli, F., Maculan, N.: Edge-swapping algorithms for the minimum fundamental cycle basis problem. Mathematical Methods of Operations Research **69**, 205–223 (2009)
14. Anderson, J.: An introduction to neural networks. MIT Press, Cambridge, MA (1995)
15. Arriaga, R., Vempala, S.: An algorithmic theory of learning: Robust concepts and random projection. Machine Learning **63**, 161–182 (2006)
16. Asimow, L., Roth, B.: The rigidity of graphs. Transactions of the American Mathematical Society **245**, 279–289 (1978)

17. Bahr, A., Leonard, J., Fallon, M.: Cooperative localization for autonomous underwater vehicles. International Journal of Robotics Research **28**(6), 714–728 (2009)
18. Barker, G., Carlson, D.: Cones of diagonally dominant matrices. Pacific Journal of Mathematics **57**(1), 15–32 (1975)
19. Barvinok, A.: Problems of distance geometry and convex properties of quadratic maps. Discrete and Computational Geometry **13**, 189–202 (1995)
20. Barvinok, A.: Measure concentration in optimization. Mathematical Programming **79**, 33–53 (1997)
21. Barvinok, A.: A Course in Convexity. No. 54 in Graduate Studies in Mathematics. American Mathematical Society, Providence, RI (2002)
22. Beeker, N., Gaubert, S., Glusa, C., Liberti, L.: Is the distance geometry problem in NP? In: Mucherino et al. [145], pp. 85–94
23. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software **24**(4), 597–634 (2009)
24. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust Optimization. Princeton University Press, Princeton, NJ (2009)
25. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems, *NIPS*, vol. 19, pp. 153–160. MIT Press, Cambridge, MA (2007)
26. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: C. Beeri, P. Buneman (eds.) Proceedings of ICDT, *LNCS*, vol. 1540, pp. 217–235. Springer, Heidelberg (1998)
27. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python. O'Reilly, Cambridge (2009)
28. Birge, J., Louveaux, F.: Introduction to Stochastic Programming. Springer, New York (2011)
29. Blömer, J., Lammersen, C., Schmidt, M., Sohler, C.: Theoretical analysis of the k-means algorithm: A survey. In: L. Kliemann, P. Sanders (eds.) Algorithm Engineering, *LNCS*, vol. 9220, pp. 81–116. Springer, Cham (2016)
30. Blumenthal, L.: Theory and Applications of Distance Geometry. Oxford University Press, Oxford (1953)
31. Böhm, C., Jacopini, G.: Flow diagrams, turing machines and languages with only two formation rules. Communications of the ACM **9**(5), 366–371 (1966)
32. Bollobás, B.: Modern Graph Theory. Springer, New York (1998)
33. Borg, I., Groenen, P.: Modern Multidimensional Scaling, second edn. Springer, New York (2010)
34. Bottou, L.: Stochastic gradient descent tricks. In: G. Montavon, *et al.* (eds.) Neural Networks: Tricks of the trade, *LNCS*, vol. 7700, pp. 421–436. Springer, Berlin (2012)
35. Bourgain, J.: On Lipschitz embeddings of finite metric spaces in Hilbert space. Israel Journal of Mathematics **52**(1-2), 46–52 (1985)
36. Boutsidis, C., Zouzias, A., Drineas, P.: Random projections for k-means clustering. In: Advances in Neural Information Processing Systems, NIPS, pp. 298–306. NIPS Foundation, La Jolla (2010)
37. Brambilla, A., Premoli, A.: Rigorous event-driven (red) analysis of large-scale nonlinear rc circuits. IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications **48**(8), 938–946 (2001)
38. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Transactions on Knowledge and Data Engineering **20**(2), 172–188 (2008)
39. Cafieri, S., Hansen, P., Liberti, L.: Loops and multiple edges in modularity maximization of networks. Physical Review E **81**(4), 46102 (2010)
40. Cafieri, S., Hansen, P., Liberti, L.: Locally optimal heuristic for modularity maximization of networks. Physical Review E **83**(056105), 1–8 (2011)
41. Cafieri, S., Hansen, P., Liberti, L.: Improving heuristics for network modularity maximization using an exact algorithm. Discrete Applied Mathematics **163**, 65–72 (2014)
42. Cauchy, A.L.: Sur les polygones et les polyèdres. Journal de l'École Polytechnique **16**(9), 87–99 (1813)

43. Cayley, A.: A theorem in the geometry of position. Cambridge Mathematical Journal **II**, 267–271 (1841)
44. Chollet, F., *et al.*: Keras. `https://keras.io` (2015)
45. Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, Cambridge, MA (1965)
46. Choromanska, A., Henaff, M., Mathieu, M., Arous, G.B., LeCun, Y.: The loss surfaces of multilayer networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, *AISTATS*, vol. 18. JMLR, San Diego (2015)
47. COIN-OR: Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT (2006)
48. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural Language Processing (almost) from scratch. Journal of Machine Learning Research **12**, 2461–2505 (2011)
49. Connelly, R.: A counterexample to the rigidity conjecture for polyhedra. Publications Mathématiques de l'IHES **47**, 333–338 (1978)
50. Cox, T., Cox, M.: Multidimensional Scaling. Chapman & Hall, Boca Raton (2001)
51. D'Ambrosio, C., Liberti, L.: Distance geometry in linearizable norms. In: F. Nielsen, F. Barbaresco (eds.) Geometric Science of Information, *LNCS*, vol. 10589, pp. 830–838. Springer, Berlin (2017)
52. D'Ambrosio, C., Liberti, L., Poirion, P.L., Vu, K.: Random projections for quadratic programming. Tech. Rep. 2019-7-7322, Optimization Online (2019)
53. Dantzig, G.: Reminiscences about the origins of linear programming. In: A. Bachem, M. Grötschel, B. Korte (eds.) Mathematical Programming: the state of the art. Springer, Berlin (1983)
54. Dasgupta, S., Gupta, A.: An elementary proof of a theorem by Johnson and Lindenstrauss. Random Structures and Algorithms **22**, 60–65 (2002)
55. D'Aspremont, A., Bach, F., Ghaoui, L.E.: Approximation bounds for sparse principal component analysis. Mathematical Programming B **148**, 89–110 (2014)
56. Dattorro, J.: Convex Optimization and Euclidean Distance Geometry. $\mathcal{M}\epsilon\beta oo$, Palo Alto (2015)
57. Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., Bengio, Y.: Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Advances in Neural Information Processing Systems, NIPS, pp. 2933–2941. NIPS Foundation, La Jolla (2014)
58. Demartines, P., Hérault, J.: Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. IEEE Transactions on Neural Networks **8**(1), 148–154 (1997)
59. Deo, N., Prabhu, G., Krishnamoorthy, M.: Algorithms for generating fundamental cycles in a graph. ACM Transactions on Mathematical Software **8**(1), 26–42 (1982)
60. Dey, S., Mazumder, R., Molinaro, M., Wang, G.: Sparse principal component analysis and its ℓ_1-relaxation. Tech. Rep. 1712.00800v1, arXiv (2017)
61. Dias, G., Liberti, L.: Diagonally dominant programming in distance geometry. In: R. Cerulli, S. Fujishige, R. Mahjoub (eds.) International Symposium in Combinatorial Optimization, *LNCS*, vol. 9849, pp. 225–236. Springer, New York (2016)
62. Douven, I.: Abduction. In: E. Zalta (ed.) The Stanford Encyclopedia of Philosophy, summer 2017 edn. Metaphysics Research Lab, Stanford University (2017)
63. Durrant, R., Kabán, A.: When is 'nearest neighbour' meaningful: A converse theorem and implications. Journal of Complexity **25**, 385–397 (2009)
64. Eco, U.: Horns, hooves, insteps. Some hypotheses on three kinds of abduction. In: U. Eco, T. Sebeok (eds.) Dupin, Holmes, Peirce. The Sign of Three. Indiana University Press, Bloomington (1983)
65. Eco, U.: Semiotics and the Philosophy of Language. Indiana University Press, Bloomington, IN (1984)
66. Eren, T., Goldenberg, D., Whiteley, W., Yang, Y., Morse, A., Anderson, B., Belhumeur, P.: Rigidity, computation, and randomization in network localization. IEEE pp. 2673–2684 (2004)
67. Euler, L.: Continuatio fragmentorum ex adversariis mathematicis depromptorum: II Geometria, 97. In: P. Fuss, N. Fuss (eds.) Opera postuma mathematica et physica anno 1844 detecta, vol. I, pp. 494–496. Eggers & C., Petropolis (1862)

68. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Mathematical Journal **23**(2), 298–305 (1973)
69. Flexer, A., Schnitzer, D.: Choosing ℓ_p norms in high-dimensional spaces based on hub analysis. Neurocomputing **169**, 281–287 (2015)
70. Floreano, D.: Manuale sulle Reti Neurali. Il Mulino, Bologna (1996)
71. Fortunato, S.: Community detection in graphs. Physics Reports **486**(3-5), 75–174 (2010)
72. François, D., Wertz, V., Verleysen, M.: The concentration of fractional distances. IEEE Transactions on Knowledge and Data Engineering **19**(7), 873–886 (2007)
73. Friedler, F., Huang, Y., Fan, L.: Combinatorial algorithms for process synthesis. Computers and Chemical Engineering **16**(1), 313–320 (1992)
74. Gayraud, N.: Public remark (2017). During the workshop *Le Monde des Mathématiques Industrielles* at INRIA Sophia-Antipolis (MOMI17)
75. Gilbreth, F., Gilbreth, L.: Process charts: First steps in finding the one best way to do work. In: Proceedings of the Annual Meeting. American Society of Mechanical Engineers, New York (1921)
76. Gill, P.: User's guide for SNOPT version 7.2. Systems Optimization Laboratory, Stanford University, California (2006)
77. Gödel, K.: On the isometric embeddability of quadruples of points of r_3 in the surface of a sphere. In: S. Feferman, J. Dawson, S. Kleene, G. Moore, R. Solovay, J. van Heijenoort (eds.) Kurt Gödel: Collected Works, vol. I, pp. (1933b) 276–279. Oxford University Press, Oxford (1986)
78. Gonçalves, D., Mucherino, A., Lavor, C., Liberti, L.: Recent advances on the interval distance geometry problem. Journal of Global Optimization (accepted)
79. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA (2016)
80. Haeffele, B., Vidal, R.: Global optimality in neural network training. In: Proceedings of the conference in Computer Vision and Pattern Recognition, CVPR, pp. 4390–4398. IEEE, Piscataway (2017)
81. Hagberg, A., Schult, D., Swart, P.: Exploring network structure, dynamics, and function using `NetworkX`. In: G. Varoquaux, T. Vaught, J. Millman (eds.) Proceedings of the 7th Python in Science Conference (SciPy2008), pp. 11–15. Pasadena, CA (2008)
82. Hansen, P., Jaumard, B.: Cluster analysis and mathematical programming. Mathematical Programming **79**, 191–215 (1997)
83. Henneberg, L.: Die Graphische Statik der starren Systeme. Teubner, Leipzig (1911)
84. Heron: Metrica, vol. I. Alexandria ($\sim$50AD)
85. Hinneburg, A., Aggarwal, C., Keim, D.: What is the nearest neighbor in high dimensional spaces? In: Proceedings of the Conference on Very Large Databases, *VLDB*, vol. 26, pp. 506–515. Morgan Kaufman, San Francisco (2000)
86. Hotelling, H.: Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology **24**(6), 417–441 (1933)
87. IBM: ILOG CPLEX 12.8 User's Manual. IBM (2017)
88. Indyk, P.: Algorithmic applications of low-distortion geometric embeddings. In: Foundations of Computer Science, *FOCS*, vol. 42, pp. 10–33. IEEE, Washington, DC (2001)
89. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Symposium on the Theory Of Computing, *STOC*, vol. 30, pp. 604–613. ACM, New York (1998)
90. Indyk, P., Naor, A.: Nearest neighbor preserving embeddings. ACM Transactions on Algorithms **3**(3), Art. 31 (2007)
91. Jain, A., Murty, M., Flynn, P.: Data clustering: a review. ACM Computing Surveys **31**(3), 264–323 (1999)
92. Johnson, W., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. In: G. Hedlund (ed.) Conference in Modern Analysis and Probability, *Contemporary Mathematics*, vol. 26, pp. 189–206. American Mathematical Society, Providence, RI (1984)
93. Jolliffe, I.: Principal Component Analysis, 2nd edn. Springer, Berlin (2010)
94. Jordan, M.: Why the logistic function? a tutorial discussion probabilities and neural networks. Tech. Rep. Computational Cognitive Science TR 9503, MIT (1995)

95. Kane, D., Nelson, J.: Sparser Johnson-Lindenstrauss transforms. Journal of the ACM **61**(1), 4 (2014)
96. Kantor, I., Matoušek, J., Šámal, R.: Mathematics++: Selected topics beyond the basic courses. No. 75 in Student Mathematical Library. American Mathematical Society, Providence, RI (2015)
97. Khalife, S., Liberti, L., Vazirgiannis, M.: Geometry and analogies: a study and propagation method for word representation. In: Statistical Language and Speech Processing, *SLSP*, vol. 7 (2019)
98. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of ICLR. San Diego (2015)
99. Knuth, D.: The Art of Computer Programming, Part I: Fundamental Algorithms, 3rd edn. Addison-Wesley, Reading, MA (1997)
100. Kullback, S., Leibler, R.: On information and sufficiency. Annals of Mathematical Statistics **22**(1), 79–86 (1951)
101. Kuratowski, C.: Quelques problèmes concernant les espaces métriques non-séparables. Fundamenta Mathematicæ **25**, 534–545 (1935)
102. Lavor, C., Liberti, L., Maculan, N.: Computational experience with the molecular distance geometry problem. In: J. Pintér (ed.) Global Optimization: Scientific and Engineering Case Studies, pp. 213–225. Springer, Berlin (2006)
103. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: The discretizable molecular distance geometry problem. Computational Optimization and Applications **52**, 115–146 (2012)
104. Lavor, C., Liberti, L., Mucherino, A.: The *interval* Branch-and-Prune algorithm for the discretizable molecular distance geometry problem with inexact distances. Journal of Global Optimization **56**, 855–871 (2013)
105. Lehmann, S., Hansen, L.: Deterministic modularity optimization. European Physical Journal B **60**, 83–88 (2007)
106. Levine, R., Mason, T., Brown, D.: Lex and Yacc, second edn. O'Reilly, Cambridge (1995)
107. Liberti, L.: Reformulations in mathematical programming: Definitions and systematics. RAIRO-RO **43**(1), 55–86 (2009)
108. Liberti, L.: Software modelling and architecture: Exercises. Ecole Polytechnique, `www.lix.polytechnique.fr/~liberti/swarchex.pdf` (2010)
109. Liberti, L.: Undecidability and hardness in mixed-integer nonlinear programming. RAIRO-Operations Research **53**, 81–109 (2019)
110. Liberti, L., Cafieri, S., Savourey, D.: Reformulation optimization software engine. In: K. Fukuda, J. van der Hoeven, M. Joswig, N. Takayama (eds.) Mathematical Software, *LNCS*, vol. 6327, pp. 303–314. Springer, New York (2010)
111. Liberti, L., Cafieri, S., Tarissan, F.: Reformulations in mathematical programming: A computational approach. In: A. Abraham, A.E. Hassanien, P. Siarry, A. Engelbrecht (eds.) Foundations of Computational Intelligence Vol. 3, no. 203 in Studies in Computational Intelligence, pp. 153–234. Springer, Berlin (2009)
112. Liberti, L., D'Ambrosio, C.: The Isomap algorithm in distance geometry. In: C. Iliopoulos, S. Pissis, S. Puglisi, R. Raman (eds.) Proceedings of 16th International Symposium on Experimental Algorithms (SEA), *LIPICS*, vol. 75, pp. 5:1–5:13. Dagstuhl Publishing, Schloss Dagstuhl (2017)
113. Liberti, L., Lavor, C.: Six mathematical gems in the history of distance geometry. International Transactions in Operational Research **23**, 897–920 (2016)
114. Liberti, L., Lavor, C.: Euclidean Distance Geometry: An Introduction. Springer, New York (2017)
115. Liberti, L., Lavor, C., Alencar, J., Abud, G.: Counting the number of solutions of ^{k}DMDGP instances. In: F. Nielsen, F. Barbaresco (eds.) Geometric Science of Information, *LNCS*, vol. 8085, pp. 224–230. Springer, New York (2013)
116. Liberti, L., Lavor, C., Maculan, N.: A branch-and-prune algorithm for the molecular distance geometry problem. International Transactions in Operational Research **15**, 1–17 (2008)
117. Liberti, L., Lavor, C., Maculan, N., Mucherino, A.: Euclidean distance geometry and applications. SIAM Review **56**(1), 3–69 (2014)
118. Liberti, L., Lavor, C., Mucherino, A.: The discretizable molecular distance geometry problem seems easier on proteins. In: Mucherino et al. [145], pp. 47–60

119. Liberti, L., Lavor, C., Mucherino, A., Maculan, N.: Molecular distance geometry methods: from continuous to discrete. International Transactions in Operational Research **18**, 33–51 (2010)
120. Liberti, L., Marinelli, F.: Mathematical programming: Turing completeness and applications to software analysis. Journal of Combinatorial Optimization **28**(1), 82–104 (2014)
121. Liberti, L., Masson, B., Lavor, C., Lee, J., Mucherino, A.: On the number of realizations of certain Henneberg graphs arising in protein conformation. Discrete Applied Mathematics **165**, 213–232 (2014)
122. Liberti, L., Swirszcz, G., Lavor, C.: Distance geometry on the sphere. In: J. Akiyama, *et al.* (eds.) JCDCG^2, *LNCS*, vol. 9943, pp. 204–215. Springer, New York (2016)
123. Liberti, L., Vu, K.: Barvinok's naive algorithm in distance geometry. Operations Research Letters **46**, 476–481 (2018)
124. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica **15**(2), 215–245 (1995)
125. Majumdar, A., Ahmadi, A., Tedrake, R.: Control and verification of high-dimensional systems with dsos and sdsos programming. In: Conference on Decision and Control, vol. 53, pp. 394–401. IEEE, Piscataway (2014)
126. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA (1999)
127. Mansouri, J., Khademi, M.: Multiplicative distance: a method to alleviate distance instability for high-dimensional data. Knowledge and Information Systems **45**, 783–805 (2015)
128. Matoušek, J.: On variants of the Johnson-Lindenstrauss lemma. Random Structures and Algorithms **33**, 142–156 (2008)
129. Matoušek, J.: Lecture notes on metric embeddings. Tech. rep., ETH Zürich (2013)
130. Maxwell, J.: On the calculation of the equilibrium and stiffness of frames. Philosophical Magazine **27**(182), 294–299 (1864)
131. McCormick, G.: Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. Mathematical Programming **10**, 146–175 (1976)
132. McCulloch, W.: What is a number, that a man may know it, and a man, that he may know a number? General Semantics Bulletin **26-27**, 7–18 (1961)
133. Mencarelli, L., Sahraoui, Y., Liberti, L.: A multiplicative weights update algorithm for MINLP. EURO Journal on Computational Optimization **5**, 31–86 (2017)
134. Menger, K.: Untersuchungen über allgemeine Metrik. Mathematische Annalen **100**, 75–163 (1928)
135. Menger, K.: New foundation of Euclidean geometry. American Journal of Mathematics **53**(4), 721–745 (1931)
136. Merris, R.: Laplacian matrices of graphs: A survey. Linear Algebra and its Applications **198**, 143–176 (1994)
137. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (eds.) Advances in Neural Information Processing Systems, *NIPS*, vol. 26, pp. 3111–3119. NIPS Foundation, La Jolla (2013)
138. Miller, G.: Wordnet: A lexical database for English. Communications of the ACM **38**(11), 39–41 (1995)
139. Milnor, J.: On the Betti numbers of real varieties. Proceedings of the American Mathematical Society **15**, 275–280 (1964)
140. Minsky, M.: The society of mind. Simon & Schuster, New York (1986)
141. Moitra, A.: Algorithmic aspects of Machine Learning. Cambridge University Press, Cambridge (2018)
142. Moro, A.: The boundaries of Babel. MIT Press, Cambridge, MA (2008)
143. Morris, C.: Signs, Language and Behavior. Prentice-Hall, New York (1946)
144. Mucherino, A., Lavor, C., Liberti, L.: Exploiting symmetry properties of the discretizable molecular distance geometry problem. Journal of Bioinformatics and Computational Biology **10**, 1242009(1–15) (2012)
145. Mucherino, A., Lavor, C., Liberti, L., Maculan, N. (eds.): Distance Geometry: Theory, Methods, and Applications. Springer, New York (2013)

146. Newman, M., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69**, 026113 (2004)
147. Object Management Group: Unified modelling language: Superstructure, v. 2.0. Tech. Rep. formal/05-07-04, OMG (2005)
148. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: Operator splitting for conic optimization via homogeneous self-dual embedding. Journal of Optimization Theory and Applications **169**(3), 1042–1068 (2016)
149. Paton, K.: An algorithm for finding a fundamental set of cycles of a graph. Communications of the ACM **12**(9), 514–518 (1969)
150. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
151. Peirce, C.: Illustrations of the logic of science, part 6: Induction, Deduction, and Hypothesis. Popular Science Monthly **13**, 470–482 (1878)
152. Penrose, R.: The emperor's new mind. Penguin, New York (1989)
153. Pfeffer, A.: Practical Probabilistic Programming. Manning Publications, Shelter Island, NY (2016)
154. Popper, K.: The Logic of Scientific Discovery. Hutchinson, London (1968)
155. Potra, F., Wright, S.: Interior-point methods. Journal of Computational and Applied Mathematics **124**, 281–302 (2000)
156. Proni, G.: Is there abduction in Aristotle? Peirce, Eco, and some further remarks. Ocula **17**, 1–14 (2016)
157. Radovanović, M., Nanopoulos, A., Ivanović, M.: Hubs in space: Popular nearest neighbors in high-dimensional data. Journal of Machine Learning Research **11**, 2487–2531 (2010)
158. Rossum, G.V., *et al.*: Python Language Reference, version 3. Python Software Foundation (2019)
159. Rousseau, F., Vazirgiannis, M.: Graph-of-word and TW-IDF: new approach to ad hoc IR. In: Proceedings of CIKM. ACM, New York (2013)
160. Saerens, M., Fouss, F., Yen, L., Dupont, P.: The principal components analysis of a graph, and its relationships to spectral clustering. In: J.F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (eds.) Proceedings of the European Conference in Machine Learning (ECML), *LNAI*, vol. 3201, pp. 371–383. Springer, Berlin (2004)
161. Salgado, E., Scozzari, A., Tardella, F., Liberti, L.: Alternating current optimal power flow with generator selection. In: J. Lee, G. Rinaldi, R. Mahjoub (eds.) Combinatorial Optimization (Proceedings of ISCO 2018), *LNCS*, vol. 10856, pp. 364–375 (2018)
162. Saxe, J.: Embeddability of weighted graphs in k-space is strongly NP-hard. Proceedings of 17th Allerton Conference in Communications, Control and Computing pp. 480–489 (1979)
163. Schaeffer, S.: Graph clustering. Computer Science Review **1**, 27–64 (2007)
164. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117 (2015). DOI 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE]
165. Schoenberg, I.: Remarks to Maurice Fréchet's article "Sur la définition axiomatique d'une classe d'espaces distanciés vectoriellement applicable sur l'espace de Hilbert". Annals of Mathematics **36**(3), 724–732 (1935)
166. Schumacher, M., Roßner, R., Vach, W.: Neural networks and logistic regression: Part I. Computational Statistics & Data Analysis **21**, 661–682 (1996)
167. Seshu, S., Reed, M.: Linear Graphs and Electrical Networks. Addison-Wesley, Reading, MA (1961)
168. Singer, A.: Angular synchronization by eigenvectors and semidefinite programming. Applied and Computational Harmonic Analysis **30**, 20–36 (2011)
169. Smith, E., Pantelides, C.: A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. Computers & Chemical Engineering **23**, 457–478 (1999)
170. Steinhaus, H.: Sur la division des corps matériels en parties. Bulletin de l'Académie Polonaise des Sciences Cl. III **4**(12), 801–804 (1956)

171. Tabaghi, P., Dokmanić, I., Vetterli, M.: On the move: Localization with kinetic Euclidean distance matrices. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, Piscataway (2019)
172. Tawarmalani, M., Sahinidis, N.: Global optimization of mixed integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**, 563–591 (2004)
173. Tenenbaum, J., de Silva, V., Langford, J.: A global geometric framework for nonlinear dimensionality reduction. Science **290**, 2319–2322 (2000)
174. Thoreau, H.: Resistance to civil government. In: E. Peabody (ed.) Æsthetic papers. J. Wilson, Boston, MA (1849)
175. Vavasis, S.: Nonlinear Optimization: Complexity Issues. Oxford University Press, Oxford (1991)
176. Vempala, S.: The Random Projection Method. No. 65 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, RI (2004)
177. Venkatasubramanian, S., Wang, Q.: The Johnson-Lindenstrauss transform: An empirical study. In: Algorithm Engineering and Experiments, *ALENEX*, vol. 13, pp. 164–173. SIAM, Providence, RI (2011)
178. Verboon, A.: The medieval tree of Porphyry: An organic structure of logic. In: A. Worm, P. Salonis (eds.) The Tree. Symbol, Allegory and Structural Device in Medieval Art and Thought, *International Medieval Research*, vol. 20, pp. 83–101. Brepols, Turnhout (2014)
179. Vershynin, R.: High-dimensional geometry. Cambridge University Press, Cambridge (2018)
180. Vidal, R., Ma, Y., Sastry, S.: Generalized Principal Component Analysis. Springer, New York (2016)
181. Vu, K., Poirion, P.L., D'Ambrosio, C., Liberti, L.: Random projections for quadratic programs over a Euclidean ball. In: A. Lodi, *et al.* (eds.) Integer Programming and Combinatorial Optimization (IPCO), *LNCS*, vol. 11480, pp. 442–452. Springer, New York (2019)
182. Vu, K., Poirion, P.L., Liberti, L.: Random projections for linear programming. Mathematics of Operations Research **43**(4), 1051–1071 (2018)
183. Vu, K., Poirion, P.L., Liberti, L.: Gaussian random projections for euclidean membership problems. Discrete Applied Mathematics **253**, 93–102 (2019)
184. Wikipedia: Civil disobedience (thoreau) (2019). URL `en.wikipedia.org/wiki/Civil_Disobedience_(Thoreau)`. [Online; accessed 190804]
185. Wikipedia: Computational pragmatics (2019). URL `en.wikipedia.org/wiki/Computational_pragmatics`. [Online; accessed 190802]
186. Wikipedia: Diagonally dominant matrix (2019). URL `en.wikipedia.org/wiki/Diagonally_dominant_matrix`. [Online; accessed 190716]
187. Wikipedia: Flowchart (2019). URL `en.wikipedia.org/wiki/Flochart`. [Online; accessed 190802]
188. Wikipedia: Principal component analysis (2019). URL `en.wikipedia.org/wiki/Principal_component_analysis`. [Online; accessed 190726]
189. Wikipedia: Rectifier (neurl networks) (2019). URL `en.wikipedia.org/wiki/Rectifier_(neural_networks)`. [Online; accessed 190807]
190. Wikipedia: Slutsky's theorem (2019). URL `en.wikipedia.org/wiki/Slutsky\%27s_theorem`. [Online; accessed 190802]
191. Williams, H.: Model Building in Mathematical Programming, 4th edn. Wiley, Chichester (1999)
192. Woodruff, D.: Sketching as a tool for linear algebra. Foundations and Trends in Theoretical Computer Science **10**(1-2), 1–157 (2014)
193. ben Judah of Worms, E.: Sodei razayya (XII-XIII Century). [Earliest account of how to create a Golem]
194. Wüthrich, K.: Protein structure determination in solution by nuclear magnetic resonance spectroscopy. Science **243**, 45–50 (1989)
195. Xu, G., Tsoka, S., Papageorgiou, L.: Finding community structures in complex networks using mixed integer optimisation. European Physical Journal B **60**, 231–239 (2007)

196. Yemini, Y.: The positioning problem — a draft of an intermediate summary. In: Proceedings of the Conference on Distributed Sensor Networks, pp. 137–145. Carnegie-Mellon University, Pittsburgh (1978)
197. Yemini, Y.: Some theoretical aspects of position-location problems. In: Proceedings of the 20th Annual Symposium on the Foundations of Computer Science, pp. 1–8. IEEE, Piscataway (1979)
198. Yun, C., Sra, S., Jadbabaie, A.: Global optimality conditions for deep neural networks. In: Proceedings of the 6th International Conference on Learning Representations. ICLR, La Jolla, CA (2018)
199. Zhang, L., Mahdavi, M., Jin, R., Yang, T., Zhu, S.: Recovering the optimal solution by dual random projection. In: S. Shalev-Shwartz, I. Steinwart (eds.) Conference on Learning Theory (COLT), *Proceedings of Machine Learning Research*, vol. 30, pp. 135–157. `jmlr.org` (2013)