HAL
open science

Referenced Vertex Ordering Problem: Theory, Applications and Solution Methods

Jérémy Omer, Antonio Mucherino

To cite this version:

Jérémy Omer, Antonio Mucherino. Referenced Vertex Ordering Problem: Theory, Applications and Solution Methods. 2020. hal-02509522v1

HAL Id: hal-02509522

https://hal.science/hal-02509522v1

Preprint submitted on 16 Mar 2020 (v1), last revised 4 Nov 2021 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

REFERENCED VERTEX ORDERING PROBLEM: THEORY, APPLICATIONS, AND SOLUTION METHODS

JÉRÉMY OMER & ANTONIO MUCHERINO

Abstract. We introduce the *referenced vertex ordering problem* (REVORDER) as a combinatorial decision problem generalizing several vertex ordering problems that already appeared in the scientific literature under different guises. In other words, REVORDER is a generic problem with several possible extensions corresponding to various real-life applications. Previous works show that REVORDER has a polynomial complexity, whereas its optimization version, denoted in our work as MIN REVORDER, is **NP**-hard. We give a survey of methods and algorithms that can be applied to the solution of MIN REVORDER, and we develop a new enumeration scheme for its solution. Our theoretical analysis of this scheme yields several pruning techniques aimed at the reduction of the number of enumeration nodes. We then discuss how upper and lower bounds can be computed during the enumeration to design a branch-and-bound algorithm. Finally, we validate the branch-and-bound by conducting a large set of computational experiments on instances coming from various real-life applications. Our results highlight that the newly introduced pruning techniques allow for major reductions of computational cost, with a constant solution quality. In result, our branch-and-bound outperforms other existing solution methods: among 180 instances with 60 vertices, it solves 179 instances to optimality whereas the best existing method is only able to solve 109 of them. Moreover, our tests show that our algorithm can solve medium-scale instances including up to 500 vertices, which opens the perspective of handling new real-life problems. Our implementation of the branch-and-bound algorithm, together with all instances we have used, is publicly available on GitLab.

1. Introduction

Consider the problem of an undergraduate or graduate student organizing her University program. Courses given at a second semester can be attended only when the classes planned in the first semester have been attended, but not *all* the possible classes offered in the first semester are actually necessary. Even though the problem that every student needs to solve is very close to *scheduling*, the fact that only a subset of classes is to be selected by the students makes this problem essentially different from classical scheduling.

Consider now the same student makes a successful academic carrier, and some years later she is faced with the problem of organizing the scientific program of a conference. Again, any researcher in operational research would initially think of scheduling. However, this particular event is supposed to gather experts from different disciplines, and some tutorials are therefore planned to make it possible that all participants are in fact able to follow the scientific presentations. In order to avoid to devote the first conference day entirely to tutorials, it is desirable to organize the program so that, before a scientific talk is scheduled, a *minimal* number of tutorials, related to scientific prerequisites, has already been given. This would allow the experts in other disciplines to have acquired a minimal knowledge on the topic. When possible, the maximization of the number of *reference* tutorials might be attempted, but *without* requiring that *all* tutorials on the topic are given before the scientific talk.

These two simple examples illustrate the problem that is the main focus of this paper. We point out that this problem is not new and has appeared already in the

Key words. Vertex ordering, Distance geometry, Branch-and-bound, Integer programming, Cutting planes.

scientific literature: the reader is referred to Section 2 for some details about the corresponding applications.

1.1. **Formal definition of the problem and notations.** The problem is formally represented by a simple undirected graph $G = (V, E)$, where the number of vertices is given by $|V| = n$. Denoting as $[\![a, b]\!] := \{a, a+1, \ldots, b\}$ for any $a, b \in \mathbb{Z}$, any bijective function

$$\sigma : V \longrightarrow [\![1, n]\!],$$

defines a total *vertex order* of G: given one vertex of $v \in V$, σ associates to v one integer value in $[\![1, n]\!]$, and reciprocally. For $v \in V$, we also say that the integer $\sigma(v)$ is the *rank* of v in σ.

We are interested in vertex orders satisfying some specific connectivity properties. For their definitions, we introduce the following notations.

- The *neighborhood* of a vertex v is denoted as $\mathcal{N}(v) := \{u : \{u, v\} \in E\}$, and the degree of v, $|\mathcal{N}(v)|$, is denoted as $\mathrm{d}^\circ(v)$;
- If σ is a vertex order, then $u \in V$ is called a *reference in* σ of $v \in V$ if and only if $u \in \mathcal{N}(v)$ and $\sigma(u) < \sigma(v)$.
- The set of references of v in σ is denoted as $R_\sigma(v)$. To simplify notations, we will denote the set of references of the vertex with rank $i \in [\![1, n]\!]$, $\sigma^{-1}(i)$, as $R_\sigma(i)$.

As already mentioned in the introductory examples, we wish to find vertex orders where each vertex has a required number, L, of references. Since the first L vertices cannot satisfy this constraint, it is necessary to define subsets of at least L *initial vertices*, $\mathcal{S}$, that can be positioned at the start of the order without fulfilling this requirement. In the remainder, this decision problem will be referred to as REVORDER and it is formally defined as follows.

Definition 1.1. (The Referenced Vertex Ordering problem (REVORDER))
Given a simple directed graph $G = (V, E)$, a positive integer L and $\mathcal{S} \subset 2^V$, is there a vertex order σ such that there is $S \in \mathcal{S}$ with $|S| \geq L$ and

$$\begin{cases} \sigma(S) = [\![1, |S|]\!] \\ \forall i \in [\![|S|+1, n]\!], \quad |R_\sigma(i)| \geq L. \end{cases}$$

Such a vertex order is called a *referenced order*, and the set of referenced orders of G is denoted as $\Sigma(\mathcal{S}, L)$.

In addition to the requirements defining a referenced order, some applications may also justify that we consider an ideal number of references $U \geq L$. In such case, while it is imposed that every vertex other than the initial ones has at least L references, it is also required that a maximum number of vertices have at least U references. For a given referenced order σ with initial set S, will then employ the following additional notations:

- for all $v \notin S$, $\delta_\sigma(v) \in \{0, 1\}$ indicates that v lacks references, i.e., $\delta_\sigma(v) = 1$ if and only if $|R_\sigma(v)| \leq U - 1$;
- if vertex v is such that $\delta_\sigma(v) = 1$, v will be called a *partially-referenced* vertex. In contrast, if $\delta_\sigma(v) = 0$, v is called a *fully-referenced* vertex.

The above naturally yields the following optimization counterpart of REVORDER, which will be the focus of this article.

Definition 1.2. (Minimum Referenced Vertex Ordering Problem (MIN REVORDER))
Given a simple undirected graph $G = (V, E)$, two positive integers L and $U \geq L$

and an initial set $S \in \mathcal{S}$ with $|S| \geq L$, find an optimal solution to the following optimization problem:

$$\text{MIN REVORDER}: \begin{cases} \min & \sum_{v \in V} \delta_\sigma(v) \\ \text{s. t.} & \sigma \in \Sigma(\mathcal{S}, L) \\ & |R_\sigma(v)| \leq U - 1 \implies \delta_\sigma(v) = 1. \end{cases}$$

1.2. **State of the art and contribution statement.** One important feature of REVORDER and MIN REVORDER is that they share a generic and rather simple form. This opens the perspective of using these formulations for several different real-life applications, as illustrated in the introductory examples. In Section 2, we support this statement by describing how MIN REVORDER emerges as a subproblem in the solution of some distance geometry problems, and we discuss a particular case of the interdiction problem where the adversary problem can be modeled as a MIN REVORDER.

In the context of distance geometry, previous research has been focused on a particular class of referenced orders, called *discretization orders*, where $\mathcal{S}$ is the set of L-cliques of the graph. The associated decision problem is the *Discretization Vertex Order problem* (DVOP) [11]. Lavor et al [8] have proposed a greedy algorithm that solves DVOP in polynomial time for any fixed value of L. It was successfully used in some applications related to distance geometry [5, 17]. In Section 4.2, we adapt this greedy search for finding a referenced order, or to prove that none exists (see Algorithm 2). This implies that REVORDER is in **P**. In contrast, it was proved in [18] that a particular case of MIN REVORDER is **NP**-Complete for any fixed integer values of $L \geq 0$ and $U \geq L + 1$, unless $L = 0$ and $U = 1$. The authors of [18] also established that, even for $L = 1$, the greedy search does not approximate the optimal value of this problem within a constant factor. Moreover, recent works dedicated to the exact solution of special cases of MIN REVORDER have also reported that several methods based on integer programming (IP), constraint programming and decomposition techniques were unable to deal with instances containing as few as 60 vertices within a reasonable computation time [17, 13].

With a view to overcoming these limits, we first improve one of the IP formulations, previously presented in [17] for a variant of MIN REVORDER. We develop new cutting planes whose separations are based on the analysis of cliques and cycles of the subgraph of G induced by low-degree vertices. We then propose a new enumeration scheme which builds upon the completion greedy algorithm introduced in [8]. For a faster and more accurate solution of the problem, we build a branch-and-bound framework based on this enumeration scheme. We also design several pruning techniques that are based on dominance and symmetry arguments.

We assess the performance of our branch-and-bound algorithm by a thorough comparison with four existing approaches: two cutting plane algorithms, one compact IP formulation and one constraint programming formulation. The computational experiments are carried out on a large benchmark including random artificial instances, and instances representing possible applications in structural biology, sensor network location and interdiction problems. These experiments allow for the study of the impact of the value of U, which has been set to $L + 1$ in all previous works due to the needs of the considered application. The experiments show that the new branch-and-bound widely outperforms existing methods and even allows for the solution of medium-scale instances where the graph has 500 vertices and $U = L + 1$. The codes used for the experiments (see Section 6), as well as all our instances, are available on GitLab.

The remainder of the article is organized as follows. We expose two applications of MIN REVORDER in Section 2. We sketch a state-of-the art of existing IP formulations, and develop new cutting planes in Section 3. In Section 4, we give a detailed presentation of the existing greedy completion algorithm, and we describe an enumeration scheme based on the same ideas to solve MIN REVORDER. The branch-and-bound algorithm based on this enumeration scheme is then developed in Section 5, where we show that several dominance rules and symmetries can be used to alleviate the computational effort. Finally, in Section 6, we assess our methodological contribution through numerical comparisons with the best existing methods.

2. Applications of REVORDER and MIN REVORDER

2.1. **Discretization of distance geometry graphs.** REVORDER appears as a fundamental pre-processing step for the solution of *distance geometry problems* (DGPs) [10]. The DGP consists in finding a realization in a K-dimensional Euclidean space of a simple edge-weighted undirected graph so that distances between realized vertices correspond to the weights on the corresponding edges.

Although the search space of the DGP is continuous in general, there exists a subclass of DGPs where it can be discretized and represented as a tree. The layer k of this search tree is associated with the vertex $v \in V$ such that $\sigma^{-1}(v) = k$: the nodes belonging to layer k all contain potential positions for the vertex v.

In order to verify whether an instance of the DGP is discretizable or not, the existence of a special vertex order on the vertices of V needs to be verified [15]. In the literature devoted to DGP, this special vertex order is named a *discretization order*, and it turns out that the search for discretization order (the DVOP mentioned above) is a particular case of REVORDER.

The subclass of DGPs that include only discretizable instances is referred to as the Discretizable DGP (DDGP) [15]. The DDGP can be efficiently solved by employing a branch-and-prune algorithm [12], which performs a complete enumeration on the feasible branches of the search tree. This is made possible by the assumptions satisfied by a discretization order: for every vertex v having a rank larger than K, there exist at least K references for v, so that the feasible positions for v can be computed by intersecting the K spheres centered in the reference vertices and having as radius the known distance to v.

When working with the DDGP, one is therefore interested in finding referenced orders where $L = K$. In this version of REVORDER, every initial set $S \in \mathcal{S}$ (those that appear at the beginning of the order) must also form a K-clique. This is necessary because a unique realization (modulo translations and rotations) needs be identified for the initial K vertices of the graph. This allows to fix the reference coordinate system where the remainder of the graph realization is eventually constructed. Moreover, when a vertex v has more than K references, then not only the discretization is allowed, but additional distance information is associated to v, so that a certain number of vertex positions in the tree can be immediately discarded. In other words, the total number of tree nodes can be a priori reduced when more than L reference vertices are associated to a given vertex v [5]. In fact, a vertex v with more than $L = K$ reference vertices implies that at least $K + 1$ spheres are involved in the intersection which provides the set of possible positions for v. Since the intersection of K spheres in a K-dimensional space gives at most two singletons with probability 1, the additional sphere allows to select one of these two singletons or show that the intersection is empty [8]. The benefit is evident in this context, since the implication is that there is no branching at layer v of the search tree. As a consequence, a particular case of MIN REVORDER where $U = L + 1$ has

been considered recently in [17] and [13] with the aim of selecting the most promising discretization order. This optimization problem has been called MIN DOUBLE, because two positions may have to be enumerated for partially-referenced vertices.

2.2. **The interdiction problem.** Another problem that can be found in the scientific literature and that is related to MIN REVORDER is the *interdiction problem*. This is a two-player game where, in its basic version, only two initial actions are allowed to the two players [6]. In this game, the graph G represents a network that Player 1 has the task to protect from the possible attacks performed by Player 2. Player 1 plays first, by selecting a set W of vertices to protect: the attacker will never be able to take control over (or influence) such protected nodes. Player 2 can then perform his attack. The attack is performed by choosing a set S of vertices to influence. No more actions at this point are then possible for both players. The influence is then propagated from S to vertices of $V \setminus W$ as soon as some non-influenced vertex has at least U influenced neighbors. Stated otherwise, we determine the influenced vertices by finding the largest subgraph of $G \setminus W$ that admits a referenced order starting with S and where the minimum number of references is U.

Suppose now that Player 2 does have the possibility to interfere with the propagation process, when necessary. The influence rule, based on the structure of a referenced order implies that the propagation stops when every non-influenced vertex has less than U influenced neighbors. It would be natural though to consider that Player 2 can keep spending resources to influence vertices in order to prevent the propagation from stopping, in particular if some vertices are close from being influenced (i.e. the number of neighbors is *almost* equal to U). This motivates to consider a variant of the problem, where two given values, L and U are considered in the propagation of influence. From a given set S of vertices initially influenced, a vertex becomes influenced if it has U influenced neighbors; otherwise, it may become influenced if it has at least L vertices at the expense of an additional action of Player 2. In this variant, we can remark that, once the the initial action of Player 1 has been executed, the solution of MIN REVORDER provides the minimum number of actions that Player 2 would need to influence all vertices in $V \setminus W$.

3. Integer programming formulations

In the following, recall that the literature about the search of optimal discretization orders can be easily extended to MIN REVORDER. In this section, we draw our attention on the IP formulations that were proposed for MIN DOUBLE [17, 13]. In [17], two compact formulations have been developed where binary variables indicate if u is a predecessor of v in the order, for all pairs of vertices u and v. Then one model guarantees that the variables describe a vertex ordering by considering transitivity constraints while the other uses constraints similar to those that appear in the Miller-Tucker-Zemlin formulation of the travelling salesman problem [14]. The authors then show that both formulations are outperformed by a cutting plane procedure where cycle constraints are iteratively added to the model. In [13], Bodur and MacNeil present a third compact formulation, VERTEXRANK, where binary variables indicate whether a vertex v is at rank r in the order for all $v \in V$ and all $r \in [\![1, n]\!]$. They also develop a decomposition scheme, WITNESS, where they introduce the notion of *witness*, which can be defined as a necessary reference. This means that partially and fully-referenced vertices have respectively exactly L and U witnesses. The decomposition yields a cutting plane algorithm that is also based on the separation of cycle constraints. One important difference between the models developed in [17] and in [13] is that the former enumerate all

the feasible initial cliques and include one extra binary variable per clique, whereas the latter add constraints ensuring that the first vertices form a clique.

In this section, we will focus on the cycle-constrained extended formulation of [17] which appeared to be the best performing IP approach in our tests. We first generalize this formulation to MIN REVORDER, and we develop two sets of valid inequalities based on cliques and cycles in the subgraph of G induced by low degree vertices. In Section 4, we will see how the initial sets can be analysed to yield other valid inequalities.

Our computational experiments will also consider VERTEXRANK and WITNESS: the reader is referred to the Appendix for details.

3.1. **Cycle-constrained extended formulation.** The IP model describes a referenced order σ with three sets of binary variables. Those indicating which vertex is partially-referenced in σ were already defined as $\delta_\sigma(v) \in \{0,1\}$ in Section 1.1. The other two sets of variables are as follows.

- for $\{u,v\} \in E$, $x_{u,v} = 1$ if and only if $\sigma(u) < \sigma(v)$;
- for $S \in \mathcal{S}, \kappa_S = 1$ if and only if $\sigma(S) = [\![1, |S|]\!]$.

Generalizing the cycle-constrained extended formulation of [17] to MIN REVORDER, we then get:

$$\begin{aligned}
\mathrm{IP}^{\mathrm{CCG}} : \min \quad & \sum_{v \in V} \delta_\sigma(v), \\
\text{s. t.} \quad & \sum_{(u,v) \in A^C} x_{u,v} \leq \left|A^C\right| - 1, \ \forall C \in \mathcal{C}, && (3.1)\\
& \sum_{S \in \mathcal{S}} \kappa_S = 1, && (3.2)\\
& \sum_{u \in \mathcal{N}(v)} x_{u,v} + \sum_{S \in \mathcal{K} : v \in S} U \kappa_S \geq L + (U-L)(1 - \delta_\sigma(v)), \ \forall v \in V, && (3.3)\\
& x \in \{0,1\}^{2|E|}, \kappa \in \{0,1\}^{|\mathcal{K}|}.
\end{aligned}$$

Constraints (1) ensure that x defines an acyclic orientation of G. Constraint (2) states that exactly one set is chosen among $\mathcal{S}$. Constraints (3) ensure that if a vertex is not in the initial set of vertices, then it must have at least L references if partially-referenced, and at least U if fully-referenced.

The above model includes one constraint per directed cycle of G, which can result in an IP with exponential number of constraints with respect to the cardinality of V. For this reason, the authors of [17] include the constraints (1) in a cutting plane algorithm. The algorithm starts by considering only the cycles whose lengths are at most three in (1). A new cycle constraint is then added when a new incumbent containing a cycle is found during the solution. This overall procedure will be referred to as CCG in the remainder.

3.2. **Low degree clique and cycle cutting planes.** $\mathrm{IP}^{\mathrm{CCG}}$ can be strengthened if we are able to identify subgraphs of G that will necessarily contain partially-referenced vertices in any referenced order. This will be the case in cliques and cycles that contain vertices with degrees close to U.

We first investigate the cliques of G. We intend to show that for some cliques, there cannot be a referenced order where every vertex is fully-referenced. For this, we can always look at the ideal situation where every vertex of a clique, $\mathcal{K}$, is among the last $|\mathcal{K}|$ vertices of a referenced order. In this case, an optimal ordering of $\mathcal{K}$ yields the maximum number of fully-referenced vertices in $\mathcal{K}$.

Procedure `maxfull(`$\mathcal{K}$`)`, whose pseudo-code is given by Algorithm 1, is based on this idea. In the description of the algorithm, vertices of $\mathcal{K}$ are iteratively added to

```
function maxfull(K):
    MF_K ← 0, O ← ∅;
    while O ≠ K do
        if max_{u∈K\O}{d°(u) + |O|} ≥ U + |K| − 1 then
            v ∈ argmax_{u∈K\O}{d°(u)};
            MF_K ← MF_K+1;
        else
            v ∈ argmin_{u∈K\O}{d°(u)};
        O ← O ∪ {v};
    return MF_K ;
```

Algorithm 1: Computation of an upper bound on the number of fully-referenced vertices in a clique

a set O of ordered vertices. At each iteration, either one vertex can be ordered so that it has at least U references or we choose the vertex with lowest degree.

Proposition 3.1. *Let $\mathcal{K}$ be a clique of G, then at most* `maxfull(`$\mathcal{K}$`)` *vertices of $\mathcal{K}$ are fully-referenced in any referenced order of G.*

Proof. We show the result by induction on the size of $\mathcal{K}$. If $|\mathcal{K}| = 1$, i.e., $\mathcal{K} = \{v\}$ for some $v \in V$, then `maxfull(`$\mathcal{K}$`)` $= 0$ if $\mathrm{d}^\circ(v) \leq U - 1$ and `maxfull(`$\mathcal{K}$`)` $= 1$ otherwise. If $\mathrm{d}^\circ(v) \leq U - 1$, v cannot be fully-referenced, so the result is true for $|\mathcal{K}| = 1$.

Suppose now that the proposition is true for any clique with size $p \geq 1$ and let $\mathcal{K}$ be a $(p+1)$-clique. Let $v \in \mathcal{K}$ be the first vertex included in O during the execution of `maxfull(`$\mathcal{K}$`)`. The execution of `maxfull(`$\mathcal{K}$`)` reduces to the inclusion of v in O followed by the execution of `maxfull(`$\mathcal{K} \setminus \{v\}$`)`. As a consequence, we have

$$\begin{cases} \texttt{maxfull(}\mathcal{K}\texttt{)}=\texttt{maxfull(}\mathcal{K} \setminus \{v\}\texttt{)}+1, & \text{if } \max\limits_{u \in \mathcal{K}}\{\mathrm{d}^\circ(u)\} \geq U + |\mathcal{K}| - 1, \\ \texttt{maxfull(}\mathcal{K}\texttt{)}=\texttt{maxfull(}\mathcal{K} \setminus \{v\}\texttt{)}, & \text{otherwise.} \end{cases}$$

From the induction hypothesis, no more than `maxfull(`$\mathcal{K} \setminus \{v\}$`)` vertices can be fully-referenced in $\mathcal{K} \setminus \{v\}$. Since $\mathcal{K}$ contains one more vertex than $\mathcal{K} \setminus \{v\}$, it is only natural that no more than `maxfull(`$\mathcal{K} \setminus \{v\}$`)` vertices could be fully-referenced in $\mathcal{K}$, so we only need to study the case where $\max_{u \in \mathcal{K}}\{\mathrm{d}^\circ(u)\} < U + |\mathcal{K}| - 1$.

Assume that $\max_{u \in \mathcal{K}}\{\mathrm{d}^\circ(u)\} < U + |\mathcal{K}| - 1$. Let σ be a referenced order and w be the vertex of $\mathcal{K}$ with smallest rank in σ. Since $\mathrm{d}^\circ(w) \leq \max_{u \in \mathcal{K}}\{\mathrm{d}^\circ(u)\} < U + |\mathcal{K}| - 1$ and $\sigma(u) \geq \sigma(w), \forall u \in \mathcal{K}$, then $|R_\sigma(w)| \leq U - 1$. The induction hypothesis thus yields that σ has at most `maxfull(`$\mathcal{K} \setminus \{w\}$`)` fully-referenced vertices among $\mathcal{K}$. Moreover, by design of Algorithm 1, we know that $v \in \operatorname{argmin}_{u \in \mathcal{K}}\{\mathrm{d}^\circ(u)\}$, so $\mathrm{d}^\circ(w) \geq \mathrm{d}^\circ(v)$. As a consequence, one can easily verify that

$$\texttt{maxfull(}\mathcal{K} \setminus \{w\}\texttt{)} \leq \texttt{maxfull(}\mathcal{K} \setminus \{v\}\texttt{)} \ .$$

Hence, σ has less than `maxfull(`$\mathcal{K} \setminus \{v\}$`)`$=$`maxfull(`$\mathcal{K}$`)` fully-referenced vertices. □

Notice in particular that for a p-clique $\mathcal{K}$, Proposition 3 implies that if $\max_{v \in \mathcal{K}}\{\mathrm{d}^\circ(v)\} \leq U + p - 2$, then at most $p - 1$ vertices of $\mathcal{K}$ will be fully-referenced in any referenced order.

Proposition 3 suggests that a given vertex is fully-referenced only when it is neither partially-referenced, nor in the initial set of vertices. For any clique $\mathcal{K}$ of G, we thus get the following valid inequality.

$$\sum_{v \in \mathcal{K}} (1 - \delta_\sigma(v)) + \sum_{v \in \mathcal{K}} \sum_{S \in \mathcal{S}: v \in S} \kappa_S \leq \texttt{maxfull(}\mathcal{K}\texttt{)}. \tag{3.4}$$

We wish to extend the above clique-cuts to cycles containing low degree vertices. Without further assumptions on the graph induced by the cycle we consider, the problem of finding the maximum number of fully-referenced vertices in the cycle is as hard as MIN REVORDER for $L = 0$. Given that the latter is known to be **NP**-Hard for all $U \geq 2$ [18], we know that it is impossible to design a polynomial greedy algorithm that will return the maximum number of fully-referenced vertices in any cycle. Instead, we show below that under a condition on the degrees of the vertices, it is guaranteed that at least one vertex will not be fully-referenced. This allows to focus on a potentially much smaller set of cycles.

Proposition 3.2. *Let $C = \{V^C, E^C\}$ be a cycle of G such that $\mathrm{d}^\circ(v) \leq U + 1, \forall v \in V^C$. Then any referenced order of G has at most $|C| - 1$ fully-referenced vertices among those of V^C.*

Proof. Let σ be a referenced order of G. By definition of a cycle, every vertex of C has at least two neighbors in C. The first vertex of C in σ cannot have any reference among the other vertices of C. So, if it has at most $U + 1$ neighbors, it has at most $U - 1$ references in σ, meaning that it is not fully-referenced. □

As a consequence, for any cycle $C = (V^C, E^C)$ such that $\mathrm{d}^\circ(v) \leq U+1, \forall v \in V^C$, the following is a valid inequality for $\mathrm{IP}^{\mathrm{CCG}}$.

$$\sum_{v \in V^C} (1 - \delta_\sigma(v)) + \sum_{v \in V^C} \sum_{S \in \mathcal{S}: v \in S} \kappa_S \leq \left|V^C\right| - 1. \tag{3.5}$$

4. A NEW ENUMERATION SCHEME FOR MIN REVORDER

For a more concise and insightful description of the proposed solution algorithm, we will start in Section 4.1 with a series of definitions related to the iterative construction of a referenced order. Then, in Section 4.2, we will briefly present an adaptation to REVORDER of the greedy algorithm initially proposed in [8] for DVOP. The basic idea behind our enumeration scheme will be detailed in Section 4.3, while Section 4.4 will describe how it can be used to solve MIN REVORDER.

4.1. **Preliminary definitions.**

Definition 4.1 (Incomplete orders)**.** A vertex order σ is an *incomplete referenced order* of G if there is $W \subseteq V$ such that σ is a referenced order of the subgraph of G induced by W, $G[W]$. We then denote as $\mathrm{PreIm}(\sigma)(= W)$, the preimage of σ, and $|\sigma| (= |W|)$ the number of ordered vertices in σ.

We then extend the definition of a "reference" so that it also makes sense for an incomplete order.

Definition 4.2 (References in incomplete orders)**.** Let $\{u, v\} \in E$ and σ be an incomplete referenced order of G. Vertex u is a *reference* of v in σ if and only if $u \in \mathrm{PreIm}(\sigma)$ and

- $v \in \mathrm{PreIm}(\sigma)$ and $\sigma(u) < \sigma(v)$, or
- $v \notin \mathrm{PreIm}(\sigma)$.

Similarly to a referenced order, we denote as $|R_v(\sigma)|$ the number of references of v in σ and set $\delta_\sigma(v) = 1$ if and only if $|R_v(\sigma)| \geq U$. Finally, the objective value of σ is given by

$$\Delta(\sigma) = \sum_{v \in \mathrm{PreIm}(\sigma)} \delta_\sigma(v).$$

Definition 4.3 (Extensions and candidates)**.** Let σ and $\overline{\sigma}$ be two incomplete referenced orders of G. We say that $\overline{\sigma}$ is an *extension* of σ if it starts like σ, i.e.,

$$\forall v \in \mathrm{PreIm}(\sigma) : \overline{\sigma}(v) = \sigma(v).$$

We then say that we extend σ with vertex $v \notin \mathrm{PreIm}(\sigma)$ if we assign the next available rank to v. More formally, $\overline{\sigma}$ is the extension of σ with v if and only if

$$\begin{cases} \mathrm{PreIm}(\overline{\sigma}) = \mathrm{PreIm}(\sigma) \cup \{v\}, \\ \forall u \in \mathrm{PreIm}(\sigma),\ \overline{\sigma}(u) = \sigma(u), \\ \overline{\sigma}(v) = |\sigma| + 1. \end{cases}$$

We denote as $[\sigma|v]$ the extension of σ with v.

Vertex v is a *valid candidate* for the extension of σ if and only if $v \notin \mathrm{PreIm}(\sigma)$ and $|R_v(\sigma)| \geq L$. It is a *partial candidate* if $L \leq |R_v(\sigma)| < U$ and a *full candidate* if $|R_v(\sigma)| \geq U$. The sets of partial and full candidates are respectively denoted as $\mathrm{Partial}(\sigma)$ and $\mathrm{Full}(\sigma)$.

Definition 4.4 (Completions)**.** If an extension of an incomplete order σ is a referenced order (meaning that it orders all the vertices of V), then it is a *completion* of σ.

We denote as $\Sigma(\sigma)$ the set of possible completions of σ. If $\Sigma(\sigma) \neq \emptyset$, a referenced order σ_V is an *optimal completion* of σ if $\sigma_V \in \mathrm{argmin}_{\overline{\sigma} \in \Sigma(\sigma)}\{\Delta(\overline{\sigma})\}$. The cost of an optimal completion of σ is denoted as $\Delta^*(\sigma)$, and we set $\Delta^*(\sigma) = +\infty$ if $\Sigma(\sigma) = \emptyset$.

Since the vertices of an initial set $S \in \mathcal{S}$ can be ordered in any way with no impact on the completion of the order, we will simply refer to completions of S to talk about completions of any ordering of S. We say that S is *feasible* if it admits a completion and *infeasible* otherwise. In the remainder of the article, we discuss methods that start with some initial set in $\mathcal{S}$ and incrementally extend it until a completion is built. At every step of the procedures, we consider only valid candidates for the extension. Hence, when extending an incomplete order with a vertex v, it will be implicit that v is a valid candidate unless explicitly stated.

4.2. **Greedy search for a referenced order.** As already mentioned in Section 2.1, a greedy algorithm was initially proposed for finding discretization orders in the context of distance geometry [8]. The algorithm constructs the order starting from a given initial clique in $\mathcal{O}(n\log(n) + m)$. A certificate of non-existence is delivered when no discretization order starting with this clique exists.

The close relationship between discretization and referenced orders makes it easy to adapt this greedy algorithm for REVORDER. We give the pseudo-code of this greedy search in Algorithm 2.

```
input  : An initial set S ∈ 𝒮
output: A completion of S or the proof that none exist
σ(S) ← {1, ..., |S|};
while |σ| < n do
    Let v ∈ argmax_{u∉PreIm σ}{|R_σ(u)|} ;
    R_max ← |R_σ(v)|;
    if R_max < L then
        stop: no referenced order order starts with S;
    else
        σ ← [σ|v] // extend σ with v
return σ;
```

Algorithm 2: Greedy completion algorithm.

Notice that the initial clique S is preselected, so that a complete enumeration of the solution set can actually be achieved only by invoking the greedy algorithm for every possible initial clique [5]. Moreover, the key choice in the greedy algorithm design is that it always chooses the new candidate vertex with largest number of references (see step 3 of Algorithm 2). The clique preselection, together with the way new candidate vertices are selected, can lead the greedy algorithm to find suboptimal solutions [17]. We thus develop a new enumeration scheme that explores every relevant choice when selecting the next vertex candidates (instead of making one arbitrary decision).

4.3. **The enumeration scheme.** In this section, we study in deeper details the properties of valid and full candidates during the construction of a referenced order. The results of this study motivate the different components of our enumeration scheme.

4.3.1. *Propagation of full candidates.* As a preamble, we first point out that, when building a referenced order by successive extensions from a given initial set, it is always best to choose a full candidate when one exists. Indeed, a full candidate does not increase the objective once added to the order, whereas it can only increase the number of references of its non-ordered neighbors. What is more, when more than one full candidate is available, then their relative ordering in the referenced order is irrelevant. The above two observations can be formalized with the following proposition.

Proposition 4.1. *Let σ be an incomplete referenced order such that there is a full candidate u, i.e., $u \notin \operatorname{PreIm}\sigma$ and $|R_\sigma(u)| \geq U$, then*

$$\Delta^*([\sigma|u]) = \Delta^*(\sigma). \tag{4.1}$$

Proof. Let $k = |\sigma|$ and let σ_V be an optimal completion of σ (hence $\Delta(\sigma_V) = \Delta^*(\sigma)$). We get a completion of $[\sigma|u]$ by switching u and $\sigma_V^{-1}(k+1)$ in σ_V. Denoting this completion as $\overline{\sigma}_V$, we get

$$\Delta(\overline{\sigma}_V) = \Delta(\sigma) + \delta_{\overline{\sigma}_V}(u) + \sum_{v \notin \operatorname{PreIm}(\sigma), v \neq u} \delta_{\overline{\sigma}_V}(v).$$

By assumption, u is a full candidate. Given that $\overline{\sigma}_V$ is also a completion of σ, $\delta_{\overline{\sigma}_V}(u) = 0$. Now, let $v \notin \operatorname{PreIm}(\sigma), v \neq u$. Given that the only difference between σ_v and $\overline{\sigma}_V$ is in the rank of u, which is smaller in $\overline{\sigma}_V$, we know that $|R_{\overline{\sigma}_V}(v)| \geq |R_{\sigma_V}(v)|$. As a consequence,

$$\sum_{v \notin \operatorname{PreIm}(\sigma), v \neq u} \delta_{\overline{\sigma}_V}(v) \leq \sum_{v \notin \operatorname{PreIm}(\sigma), v \neq u} \delta_{\sigma_V}(v),$$

which yields $\Delta(\overline{\sigma}_V) \leq \Delta(\sigma_V)$. Finally, $\overline{\sigma}_V$ is a completion of $[\sigma|u]$, so by definition, $\Delta^*([\sigma|u]) \leq \Delta(\overline{\sigma}_V)$. Moreover, every completion of $[\sigma|u]$ is also a completion of σ, so $\Delta^*([\sigma|u]) \geq \Delta^*(\sigma)$, hence the result. □

Let σ be an incomplete referenced order. Based on Proposition 9, we consider a particular extension of σ that sequentially selects full candidates until there is none. We will call such an extension the *propagation of full candidates in σ* and denote it as $\Pi(\sigma)$. Algorithm 3 gives the pseudo-code of function `propagate(σ)`, which executes this operation.

The recursive application of Proposition 9 immediately shows that full candidates can always be propagated without loss in the objective function. We thus get the following fundamental motivation for an enumeration scheme based on the greedy search. Indeed, it implies that suboptimal extensions of an incomplete order can only be made if there is no full candidate.

```
function propagate(σ):
    Π(σ) ← σ;
    while Full(Π(σ)) ≠ ∅ do
        v ← any element of Full(Π(σ));
        Π(σ) ← [Π(σ)|v];
    return Π(σ);
```

Algorithm 3: Propagation of an incomplete referenced order

Proposition 4.2. *Let σ be an incomplete referenced order, and $\overline{\sigma} = \Pi(\sigma)$ the propagation of full candidates in σ. Then, $\Delta^*(\overline{\sigma}) = \Delta^*(\sigma)$.*

4.4. **Enumeration of partial candidates.** Algorithm 4 describes a procedure that enumerates completions of a feasible initial set $S \in \mathcal{S}$. It builds an enumeration tree where every node corresponds to an incomplete referenced order, and stores the pending nodes in Q. The root node is given by the propagation of the set S. Then, for each partial candidate v of an incomplete referenced order σ, one new branch is created. The child node corresponding to v is given by `propagate`$([\sigma|v])$. The propagation preceding every inclusion of σ to the list of pending nodes guarantees that every pending node corresponds to an incomplete referenced order without any full candidate. In the remainder, the node corresponding to the referenced order σ will simply be called *enumeration node σ*.

```
   input  : A feasible initial set S ∈ 𝒮
   output: An optimal completion of S
1  UB ← +∞;
2  σ ← propagate(S);
3  Q ← {σ};
4  while Q ≠ ∅ do
5      σ ← an element of Q;
6      Q ← Q \ {σ};
7      if |σ| = n then
8          if Δ(σ) < UB then  σ̄ ← σ;  UB ← Δ(σ);
9          goto step 2;
10     for v ∈ Partial(σ) do
11         Π([σ|v]) ← propagate([σ|v]);
12         Q ← Q ∪ {Π([σ|v])};
13 return σ̄;
```

Algorithm 4: Enumeration algorithm for MIN REVORDER

Theorem 4.1. *Let $S \in \mathcal{S}$ be a feasible initial set. Then Algorithm 4 computes an optimal completion of σ.*

Proof. Using the notations of Algorithm 4, we show that at the start of any iteration of the **while** loop, $\min\{\mathrm{UB}, \min_{\tau \in Q}\{\Delta^*(\tau)\}\} = \Delta^*(S)$.

At the first iteration, $\sigma = \Pi(S)$, so Proposition 10 guarantees that $\Delta^*(\sigma) = \Delta^*(S)$. So assume that the property remains true at the start of a given iteration, and denote as σ the element of Q selected at step 5.
If $\Delta^*(\sigma) > \Delta^*(S)$, then $\min\{\mathrm{UB}, \min_{\tau \in Q}\{\Delta^*(\tau)\}\} = \Delta^*(S)$ after removal of σ from Q so it will still be true at the start of next iteration.
Assuming that $\Delta^*(\sigma) = \Delta^*(S)$, there is a candidate for extension, v, such that $\Delta^*([\sigma|v]) = \Delta^*(S)$. Given that there is no full candidate, there is $v \in \mathrm{Partial}(\sigma)$

such that $\Delta^*([\sigma|v]) = \Delta^*(S)$. Once again Proposition 10 yields $\Delta^*(\Pi([\sigma|v])) = \Delta^*(S)$. We conclude by observing that $\Pi([\sigma|v])$ is added to Q at some point of the **for** loop. □

5. Speeding-up the enumeration algorithm

The main benefit of the enumeration given in Algorithm 4 is that it allows to focus on the partial candidates when selecting the next vertex in the order. Here, we show that some additional properties of graph G may be used to further reduce the enumeration to a subset of the partial candidates.

5.1. **Dominance rules.**

Definition 5.1. Let σ and τ be two incomplete referenced orders. We say that σ *dominates* τ if and only if $\Delta^*(\sigma) \leq \Delta^*(\tau)$.

Let τ be a pending node and σ be either a pending node or a treated node on a different branch than τ. This definition of dominance implies that if σ dominates τ, then τ can be pruned from the tree because σ will be completed into a referenced order with smaller or equal cost. This definition cannot be used in practice though, because it requires the computation of $\Delta^*(\sigma)$ and $\Delta^*(\tau)$. We propose a practical sufficient condition below.

Proposition 5.1 (Basic dominance)**.** *Let σ and τ be two incomplete referenced orders. If $\Delta(\sigma) \leq \Delta(\tau)$ and $\mathrm{PreIm}(\tau) \subseteq \mathrm{PreIm}(\sigma)$, then σ dominates τ.*

Proof. Let τ_V be an optimal completion of τ. Starting with σ, we can construct a vertex order of G, σ_V, by adding the vertices of $V \setminus \mathrm{PreIm}(\sigma)$ after those of $\mathrm{PreIm}(\sigma)$ so that:

$$\forall u, v \in V \setminus \mathrm{PreIm}(\sigma) : \ (\tau_V(u) < \tau_V(v)) \implies (\sigma_V(u) < \sigma_V(v)).$$

Using that $\mathrm{PreIm}(\tau) \subseteq \mathrm{PreIm}(\sigma)$, the above yields

$$\forall v \in V \setminus \mathrm{PreIm}(\sigma) : \ |R_{\sigma_V}(v)| \geq |R_{\tau_V}(v)|.$$

This yields that σ_V is a referenced order and $\delta_{\sigma_V}(v) \leq \delta_{\tau_V}(v), \forall v \in V \setminus \mathrm{PreIm}(\sigma)$. Moreover, σ_V completes σ, so

$$\begin{aligned}\Delta(\sigma_V) &= \Delta(\sigma) + \sum_{v \in V \setminus \mathrm{PreIm}(\sigma)} \delta_{\sigma_V}(v) \\ &\leq \Delta(\tau) + \sum_{v \in V \setminus \mathrm{PreIm}(\tau)} \delta_{\tau_V}(v) = \Delta(\tau_V).\end{aligned}$$

□

We first examine how the above rule applies to dominance among children node of a given pending node.

Proposition 5.2. *Let σ be an incomplete order without full candidate such that there are $u, v \in \mathrm{Partial}(\sigma)$. If $u \in \mathrm{PreIm}(\Pi([\sigma|v]))$, then $\Pi([\sigma|v])$ dominates $\Pi([\sigma|u])$.*

In particular, if $\{u, v\} \in E$ and $|R_\sigma(u)| = U - 1$, $\Pi([\sigma|v])$ dominates $\Pi([\sigma|u])$.

Proof. Vertices u and v are both partial candidates, so by Proposition 10, we know that

$$\Delta(\Pi([\sigma|v])) = \Delta(\Pi([\sigma|u])) = \Delta(\sigma) + 1.$$

Moreover, if $u \in \mathrm{PreIm}(\Pi([\sigma|v]))$, then any full candidate added to the order during the propagation of $[\sigma|u]$ will also be added during that of $[\sigma|v]$, so

$$\mathrm{PreIm}(\Pi([\sigma|u])) \subseteq \mathrm{PreIm}(\Pi([\sigma|v])).$$

Therefore, $\Pi([\sigma|v])$ dominates $\Pi([\sigma|u])$ (by Proposition 13).

In particular, if $\{u, v\} \in E$ and $|R_\sigma(u)| = U - 1$, then $\left|R_{[\sigma|v]}(u)\right| = U$. Hence $u \in \mathrm{PreIm}(\Pi([\sigma|v]))$. □

We apply this rule to compare each new enumeration node with two sets of vertices. From the above property, we can see that the verification of dominance among children nodes has a rather low computational cost. Hence, when branching, we can start by comparing children nodes with each other for dominance. Then, the search for dominance can be extended as follows. Let T be the set of all treated nodes that have not been pruned yet, and denote as $T(z)$ the nodes of T with cost z. Each new node σ is compared with every vertex in $T(z)$ for $\Delta(\sigma) - \Delta_z \leq z \leq \Delta(()\,\sigma)$. We make this restriction, because the dominance criterion of Proposition 13 is transitive. Hence, larger dominance is unlikely if smaller dominance does not occur. Moreover, this restriction saves a significant amount of computational time (especially for larger instances).

Another case of dominance can be detected by comparing the neighborhoods of partial candidates.

Proposition 5.3. *Let σ be an incomplete referenced order such that there exist $u, v \in \mathrm{Partial}(\sigma)$ with $|R_\sigma(u)| \leq |R_\sigma(v)|$ and $(\mathcal{N}(u) \setminus \mathrm{PreIm}(\sigma)) = (\mathcal{N}(v) \setminus \mathrm{PreIm}(\sigma))$, then $[\sigma|u]$ dominates $[\sigma|v]$.*

Proof. Let σ_V be any completion of $[\sigma|v]$ and build $\overline{\sigma}_V$ as the completion of $[\sigma|u]$ obtained by swapping the ranks of u and v in σ_V, i.e,

$$\begin{cases} \overline{\sigma}_V(v) = \sigma_V(u) \\ \overline{\sigma}_V(u) = \sigma_V(v) \\ \overline{\sigma}_V(w) = \sigma_V(w), \forall w \notin \{u, v\} \end{cases}$$

Using that $(\mathcal{N}(u) \setminus \mathrm{PreIm}(\sigma)) = (\mathcal{N}(v) \setminus \mathrm{PreIm}(\sigma))$, we know that $|R_{\overline{\sigma}_V}(w)| = |R_{\sigma_V}(w)|$ for $w \neq u, v$. Moreover, $|R_\sigma(u)| \leq |R_\sigma(v)|$, so $|R_{\sigma_V}(u)| \leq |R_{\overline{\sigma}_V}(v)|$. Stated otherwise, u is fully-referenced in σ_V only if v is fully-referenced in $\overline{\sigma}_V$. Given that v is partially-referenced in σ_V and u is partially-referenced in $\overline{\sigma}_V$, we get that

$$\Delta^*([\sigma|v]) = \Delta(\sigma_V) \geq \Delta(\overline{\sigma}_V) \geq \Delta^*([\sigma|u]).$$

□

The above dominance rules are all used in the final version of the enumeration algorithm as described in steps 11-20 of Algorithm 5.

5.2. **Breaking symmetries.** Another perspective of improvement is in the identification of symmetric referenced orders. The basic idea is to avoid building several referenced orders where the list of partially-referenced vertices is the same. For this, we arbitrarily index the vertices of V and denote $u < v$ if the index of $u \in V$ is smaller than that of $v \in V$. We wish to break symmetry by constructing referenced orders σ such that a vertex v is partially-referenced in σ only if every other partially-referenced vertex u such that $u < v$ either

- has a smaller rank than u in σ, i.e., $\sigma(u) < \sigma(v)$, or
- has less than L references with rank smaller than $\sigma(v)$.

The second condition implies that we cannot obtain a valid referenced order by simply swapping the ranks of u and v in σ. In the context of Algorithm 4, the above condition is equivalent to requiring that, if u and v are two partial candidates for some incomplete order σ such that $u < v$, then u should be removed from the list of partial candidates when extending σ with v. Therefore, in the corresponding branch of the enumeration tree, it will be possible to extend σ with u only when

fully-referenced. In the remainder, this selection rule will be referred to as the *index priority rule*. As a first step, we show that there exists a solution to MIN REVORDER that satisfies this rule.

Proposition 5.4 (Index priority rule)**.** *Let τ be a referenced order. Then, there is a referenced order σ such that for all $v \in V$:*

- *if v is partially-referenced in σ, then it is partially-referenced in τ (i.e., $\Delta(\sigma) \leq \Delta(\tau)$), and*
- *$\forall v \in V$ such that $\delta_\sigma(u) = 1$, and $\forall u \in V$ such that $u < v$ and $\delta_\sigma(v) = 1$:*

$$\sigma(u) > \sigma(v) \implies |\{w \in \mathcal{N}(v) : \sigma(w) < \sigma(v)\}| < L.$$

Proof. For any referenced order, we have seen that there is a referenced order with smaller or equal cost that can be constructed by using the enumeration algorithm sketched in Algorithm 4. Therefore, without loss of generality, we can consider a referenced order, τ, where the vertex at a given rank is partially-referenced if and only if there is no full candidate for this rank. Let $v \in V$ be the partially-referenced vertex with smallest rank in τ, and consider $P_\sigma(v) = \{u \in V : |w \in \mathcal{N}(u) : \tau(w) < \tau(v)| \geq L\}$. Set $P_\sigma(v)$ includes all the (partial) candidates for rank $\tau(v)$. Assume that v does not satisfy the index priority rule. This means that the vertex with smaller index in $P_\sigma(v)$, u, is such that $u < v$. Then let σ be the vertex order obtained by inserting u at the rank of v in τ, i.e.,

$$\begin{cases} \sigma(w) = \tau(w), \forall w : \tau(w) < \tau(v), \text{ or } \tau(w) > \tau(u), \\ \sigma(u) = \tau(v), \\ \sigma(w) = \tau(w) + 1, \forall w : \tau(v) \leq \tau(w) < \tau(u). \end{cases}$$

Then, u is partially-referenced in both τ and σ, and every other vertex has at least as many references in both orders. As a consequence, every partially-referenced vertex of σ is also partially-referenced in τ, but the index priority rule is satisfied at least up to rank $\tau(v)$. We conclude by induction on the minimum rank of a vertex that does not meet the index priority rule. □

We ensure the index priority rule in the enumeration algorithm by introducing one new set of candidates, $\text{Fixed}(\sigma)$, for each incomplete order σ. Any vertex in $\text{Fixed}(\sigma)$ can be chosen to extend an order only if it has more than U references. In the enumeration algorithm, this means that those vertices can extend an order only during propagation. We then modify, the branching rules by excluding fixed candidates from the list of valid candidates, and when creating a node by extending some order σ with a partial candidate u, we set as fixed every other partial candidate v such that $v < u$. The corresponding modification appears at step 8 of Algorithm 5, and the identification of fixed vertices is detailed at steps 23.

5.3. **Bound pruning.** In addition to the above pruning rules, we use the traditional bound pruning of branch-and-bound algorithms. For this, we start the algorithm with the upper bound provided by running the greedy completion algorithm from every initial set of $\mathcal{S}$. The upper bound, UB, will then be updated every time an improving complete order is found during the enumeration. A lower bound $\text{LB}(\sigma)$ is then computed for each incomplete order σ in the pending nodes queue Q, and the order is pruned if $\text{LB}(\sigma) \geq \text{UB}$.

In our implementation, we have tried two different methods for computing the lower bound. One trivial lower bound at node σ is given by $\Delta(\sigma) + 1$, where we add one to the cost of σ, because the next vertex in the order is necessarily partially-referenced. We slightly improve this bound by acknowledging that a vertex with degree smaller than U will necessarily be partially-referenced. Likewise, if two

vertices with exactly U neighbors are linked by an edge, then at least one of them will be partially-referenced. The resulting lower bound is denoted as $\mathrm{LB}^{\mathrm{trivial}}(\sigma)$

In order to compute a better lower bound, we can solve the linear relaxation of any IP formulation of the problem. As already remarked above, $\mathrm{IP}^{\mathrm{CCG}}$ includes every cycle constraints, but we can start the cycle-cut generation with 2- and 3-cycles. Since this initial model is a relaxation of MIN REVORDER, so is its linear relaxation. For a given incomplete order σ, we can then compute a lower bound, denoted as $\mathrm{LB}^{\mathrm{CCG}}(\sigma)$, by fixing all the variables corresponding to $\mathrm{PreIm}(\sigma)$ and by solving the linear relaxation of the initial relaxation of $\mathrm{IP}^{\mathrm{CCG}}$.

The overall branch-and-bound algorithm is summarized in Algorithm 5. We can deduce from the above discussions that it yields the optimal completion of the input initial set S, or it proves that no completion of S can improve the input upper bound UB. In the algorithm, the method chosen for computing the lower bound of some incomplete order σ is abstracted by a call to function `lowerbound(`σ`)` that returns either $\mathrm{LB}^{\mathrm{trivial}}(\sigma)$ or $\mathrm{LB}^{\mathrm{CCG}}(\sigma)$, depending on the chosen option. This will be discussed further in Section 6.

5.4. **Preprocessing the initial set.** Before any call to Algorithm 5, the set of initial sets is preprocessed in order to delete as many of its entries as possible. For this, we propagate each set of $\mathcal{S}$ and compare them for dominance. We then run the greedy algorithm from every non-dominated initial set for an initial upper bound. While doing so, we also obtain the list of infeasible initial sets. In the end, we execute Algorithm 5 only with feasible and non-dominated initial sets.

This preprocessing step is similarly performed before the execution of CCG to reduce the number of initial sets. But, we also use it to compute valid inequalities based on the propagation of each initial set. Indeed, if we consider $S \in \mathcal{S}$, and $\Pi(S)$:=`propagate(`S`)`, we know that no vertex among those of $\mathrm{PreIm}(\Pi(S)) \setminus S$ will be partially-referenced if S is chosen as an initial set. Moreover, there will necessarily be one partially-referenced vertex among the partial candidates of $\Pi(S)$. As a consequence, we add the following two sets of valid inequalities to $\mathrm{IP}^{\mathrm{CCG}}$:

$$\delta_\sigma(v) \leq (1 - \kappa_S), \forall S \in \mathcal{S}, \forall v \in \mathrm{PreIm}\,(\Pi(S)) \setminus S; \tag{5.1}$$

$$\sum_{v \in \mathrm{Partial}(\Pi(S))} \delta_\sigma(v) \geq \kappa_S, \forall S \in \mathcal{S}. \tag{5.2}$$

6. Computational experiments

We have tested the branch-and-bound framework given in Algorithm 5 on five different families of instances (**Random**, **Synthetic**, **Protein**, **SensorNetwork** and **Interdiction**), and we compared it with the other methods discussed in this article. For reproducibility of our results and more efficiency in future research on the topic, our implementation of Algorithm 5 is publicly available on GitLab[1]. The same repository also contains the instances used in our computational experiments, together with the bash scripts that run the tests.

6.1. **Our instance sets. Random** and **Synthetic** sets of instances contain randomly generated instances, obtained as described in [13]. In particular, **Random** contains randomly generated instances with no specific patterns, while **Synthetic** contains the so-called "synthetic" instances, where, starting from a randomly generated instance, edges are removed in order to build the *sparsest* graph that still admits a referenced order. An additional $0.15n$ or $0.20n$ edges are then added as

[1] `https://gitlab.insa-rennes.fr/Jeremy.Omer/min-revorder.git`

```
input  : A feasible initial set S ∈ 𝒮 and an initial upper bound UB
output: An optimal completion of S
1  Q ← {propagate(S)};
2  while Q ≠ ∅ do
3      σ ← an element of Q;
4      Q ← Q \ {σ};
       // If the order is complete, update incumbent
5      if |σ| = n then
6          if Δ(σ) < UB then  σ̄ ← σ;  UB ← Δ(σ);
7          goto step 2;
       // Temporarily store valid extensions of σ in ℰ
8      for v ∈ Partial(σ) \ Fixed(σ) do
9          Π([σ|v]) ← propagate([σ|v]);
10         ℰ ← Π([σ|v])
       // Check dominance criteria
11     for Π([σ|u]) ∈ ℰ do
           // Dominance among children nodes
12         for Π([σ|v]) ∈ ℰ such that v ≠ u do
               // Apply Proposition 14
13             if u ∈ PreIm(Π([σ|v])) then
14                 goto step 11;                          // Prune Π([σ|u])
               // Apply Proposition 15
15             if 𝒩(u) \ PreIm(σ) = 𝒩(v) \ PreIm(σ) and |R_σ(u)| ≥ |R_σ(v)|
                then
16                 goto step 11;                          // Prune Π([σ|u])
           // Dominance with subsets of the enumeration tree
17         for z = Δ(Π([σ|u])) − Δ_z to Δ(Π([σ|u])) do
18             for τ ∈ T(z) do
19                 if PreIm(Π([σ|u])) ⊆ PreIm(τ) then
20                     goto step 11;                      // Prune Π([σ|u])
           // Bound-pruning
21         if lowerbound(Π([σ|u])) ≤ UB then
22             goto step 11;                              // Prune Π([σ|u])
           // Apply index priority rule in extensions of [σ|u]
23         Fixed(Π([σ|u])) ← Fixed(σ) ∪ {v ∈ Partial(σ) : v < u};
           // Add non-pruned orders to tree and pending nodes
24         T(Π([σ|u])) ← T(Π([σ|u])) ∪ {Π([σ|u])};
25         Q ← Q ∪ {Π([σ|u])};
26 return σ̄, UB;
```

Algorithm 5: Branch-and-bound algorithm for MIN REVORDER

noise. Notice that we use the same instances as those that appear in the experiments published in [13], which were made publicly available[2] by the authors. The details about the instances in **Random** and **Synthetic** are summarized in Table 1. In all our tables, we also include the values of L, because they are constant in all corresponding experiments. Moreover, $\mathcal{S}$ is given by the set of $(L+1)$-cliques of the graph.

[2] https://sites.google.com/site/mervebodr/

name	$\|V\|$	$\|E\|$	L	*name*	$\|V\|$	$\|E\|$	L
random25(1)	25	90	3	synthetic25(1)	25	91	3
random25(2)	25	90	3	synthetic25(2)	25	92	3
random25(3)	25	120	3	synthetic25(3)	25	90	3
random25(4)	25	120	3	synthetic25(4)	25	91	3
random30(1)	30	131	3	synthetic30(1)	30	111	3
random30(2)	30	131	3	synthetic30(2)	30	113	3
random30(3)	30	174	3	synthetic30(3)	30	110	3
random30(4)	30	174	3	synthetic30(4)	30	112	3
random35(1)	35	179	3	synthetic35(1)	35	131	3
random35(2)	35	179	3	synthetic35(2)	35	133	3
random35(3)	35	238	3	synthetic35(3)	35	130	3
random35(4)	35	238	3	synthetic35(4)	35	132	3

TABLE 6.1. Details about the instances in **Random** (left-side) and **Synthetic** (right-side) sets.

name	$\|V\|$	m/n	$\|E\|$	L	*name*	$\|V\|$	m/n	$\|E\|$	L
1m40(1)	60	3.2	202	3	1bpm(1)	60	3.2	212	3
1m40(2)	60	3.6	216	3	1bpm(2)	60	3.6	216	3
1m40(3)	60	4.0	238	3	1bpm(3)	60	4.0	239	3
1n4w(1)	60	3.2	221	3	1mqq(1)	60	3.2	203	3
1n4w(2)	60	3.6	221	3	1mqq(2)	60	3.6	216	3
1n4w(3)	60	4.0	238	3	1mqq(3)	60	4.0	240	3

TABLE 6.2. Details about the instances in **Protein**. The ratio m/n indicates the percentage of edges that are kept in the instance.

The **Protein** set contains the protein instances that were already used in previous publications on distance geometry for protein structure determination (see [8] for the details about the instance generation procedure from known protein conformations). In order to increase the difficulty in finding solutions, and as already proposed in [13] and [17], we have removed a subset of edges from the original graphs representing those protein instances. In order to select the number of edges in the graph, we first observe that the minimum number of edges that can be kept in a feasible instance is $m = L(n - \frac{L+1}{2}) \approx L \times n$. In [17], it was observed, for $U = L + 1$, that the most difficult instances to solve are those where the ratio m/n is near in value to 3.6. In our experiments, we generated instances where $m/n = 3.2, 3.6, 4.0$. Only the first 60 atoms of the original instances are taken into consideration, so that all the instances in this set have 60 vertices. Table 2 gives more details about those protein instances.

Another important and traditional application of distance geometry is the so-called Sensor Network Localization Problem [3, 1]. The **SensorNetwork** set consists of randomly generated instances that resemble sensor networks in dimension 2. An edge between two vertices is in this case included in the graph if the two corresponding sensors are able to communicate, that is, if their relative distance is smaller than a given threshold T, that represents the upper limit for the communication. In the generation of the instances, we control the shape of the "underlying" area where the sensors are supposed to be randomly placed. By doing that, we can artificially generate situations where the neighborhood of a sensor can have a full or partial intersection with the underlying 2D area. In the former case the sensor naturally has a larger range of communication possibilities with others. Table 3

name	$\|V\|$	h	w	T	$\|E\|$	L
sensor01	60	0.3	1.0	0.12	234	2
sensor02	60	0.3	1.0	0.15	306	2
sensor03	60	0.3	1.0	0.18	397	2
sensor04	60	0.5	1.0	0.20	317	2
sensor05	60	0.5	1.0	0.21	458	2
sensor06	60	0.5	1.0	0.22	397	2
sensor07	60	0.7	1.0	0.20	276	2
sensor08	60	0.7	1.0	0.22	297	2
sensor09	60	0.7	1.0	0.24	326	2
sensor10	60	1.0	1.0	0.25	263	2
sensor11	60	1.0	1.0	0.30	363	2
sensor12	60	1.0	1.0	0.35	542	2

TABLE 6.3. Details about the instances in **SensorNetwork** set.

name	$\|V\|$	p	$\|E\|$	L	*name*	$\|V\|$	p	$\|E\|$	L
interdiction010	60	0.10	585	1	interdiction055	60	0.55	108	1
interdiction020	60	0.20	309	1	interdiction060	60	0.60	102	1
interdiction030	60	0.30	108	1	interdiction065	60	0.65	94	1
interdiction040	60	0.40	157	1	interdiction070	60	0.70	86	1
interdiction045	60	0.45	123	1	interdiction080	60	0.80	67	1
interdiction050	60	0.50	112	1	interdiction090	60	0.90	66	1

TABLE 6.4. Details about the instances in our **Interdiction** set.

gives more details about these instances: we indicate with the two letters h and w the height and width, respectively, of the underlying area for the generated sensor networks. All networks are composed of 60 sensors.

The **Interdiction** set is related to instances of the interdiction problem (see Section 2.2). The instances belonging to this set have been generated by using the same procedure as Hemmati et al. [6], which had been previously proposed by Chung and Lu in [2], where we only had to make one minor modification in order to build undirected graphs. The procedure starts with a graph containing only one vertex; an expected total number of vertices is given in input, together with a probability $p \in [0, 1]$. The procedure iterates and executes one of the following two steps until the graph contains the desired number of vertices:

(1) with probability p, add one new vertex to the graph and add one edge between this new vertex and another randomly selected vertex already present in the graph;
(2) with probability $1 - p$, choose two vertices already in the graph and add an edge between them.

We generate instances with 60 vertices and use a probability p ranging from 0.1 to 0.9. Table 4 gives some additional details about these instances.

6.2. **Comparing our branch-and-bound against existing approaches.** The experimental assessment of the capability of our branch-and-bound algorithm is made through a comparison with the three IP-based approaches CCG, WITNESS and VERTEXRANK and the solution of a constraint programming model CP, which was introduced by [13]. A detailed description of the last three approaches is given in Appendix B. We chose them, because they represent the best existing solution methods for MIN REVORDER.

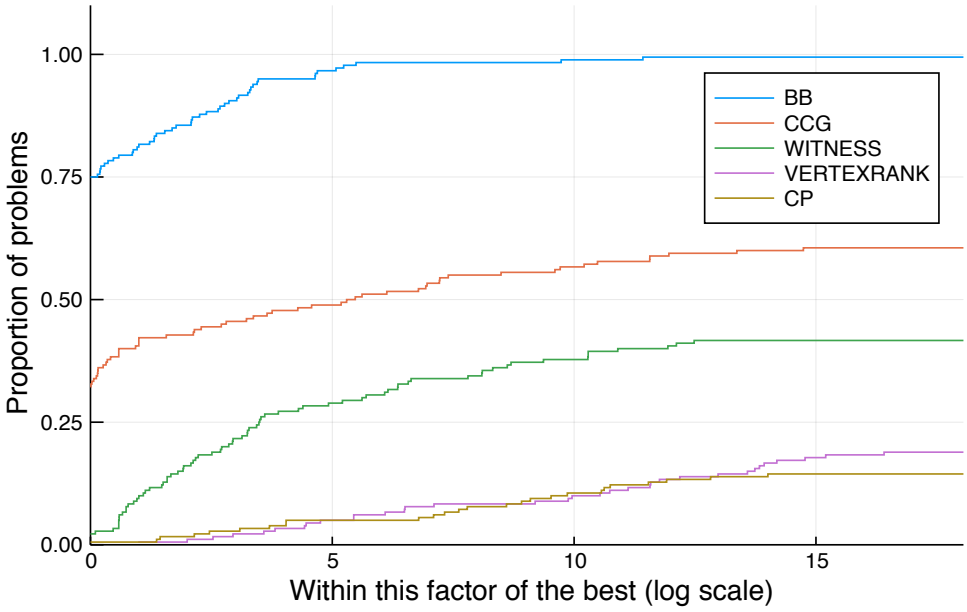


FIGURE 6.1. Comparison of our branch-and-bound algorithm to the existing methods.

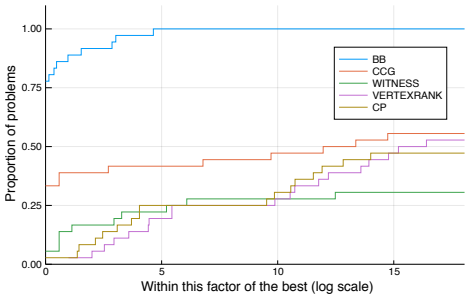
For each method, we solve all the instances described in the previous section with three different values of U chosen so that $U-L = 1, 2, 3$. As a result, this benchmark includes 180 instances. Every experiment is run on a workstation equipped with an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, 16GB RAM[3]. Every IP and constraint programming formulation is solved with CPLEX 12.9. The time limit is set to 1000 seconds, and for a fair comparison, every run is done on a single thread. In this section the reader can find the corresponding performance profiles, with our associated comments. For more details about the computational results, one can refer to Appendix B; To construct performance profiles, we apply the method[4] by Dolan and Moré initially proposed in [4].

Our implementation of the branch-and-bound algorithm corresponds to the sketch given in Algorithm 5, where `lowerbound(`σ`)` returns the trivial lower bound, $\mathrm{LB}^{\mathrm{trivial}}(\sigma)$ (see Section 5.3). The value of the parameter Δ_z is set to 1 in all the experiments. The nodes in the branch-and-bound queue are explored in a best-first fashion, where the best node is the one minimizing the ratio $\Delta(\sigma)/|\sigma|$. These two parameters have been set to these values according to preliminary tests. In the remainder, we will refer to this default implementation of our branch-and-bound as BB.

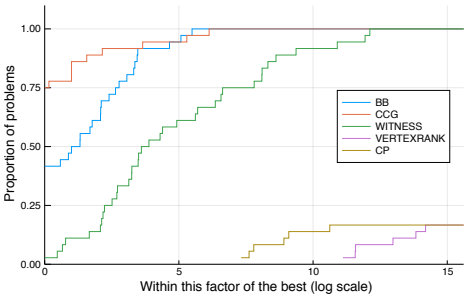
The first performance profile (see Figure 1) considers all the experiments (the logarithmic scale on the x-axis is in base 2). The profile clearly shows the superiority of the proposed method. Moreover, when comparing the other existing methods, we can remark that the results we report for WITNESS, CP and VERTEXRANK are consistent with those previously published in [13]. However, the results of CCG are better than those reported in [13]. This is because our implementation of CCG is enhanced by the initial enumeration and preprocessing of $(L+1)$-cliques, and by the addition of cutting planes (see Section 5.4).

[3]We ran experiments on a Linux kernel 4.13.16-100.fc25.x86_64 and we used GNU C++ 6.4.1 to compile our code

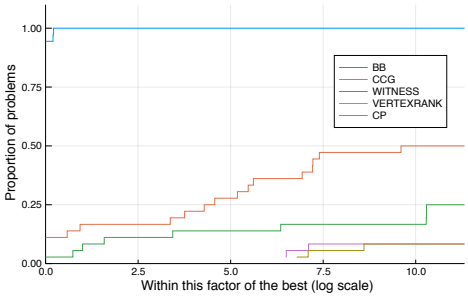
[4]We used the `Julia` package available on GitHub:
`https://github.com/JuliaSmoothOptimizers/BenchmarkProfiles.jl`



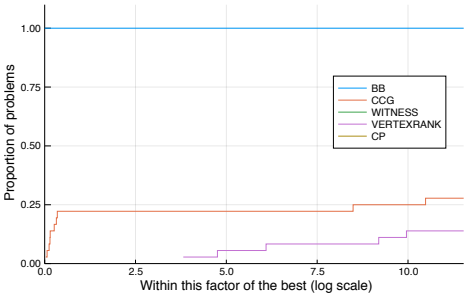
(A) Instances from the **Random** set.



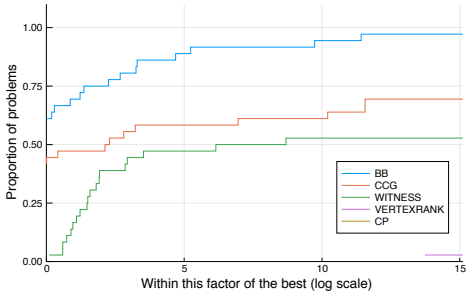
(B) Instances from the **Synthetic** set.



(C) Instances from the **Protein** set.



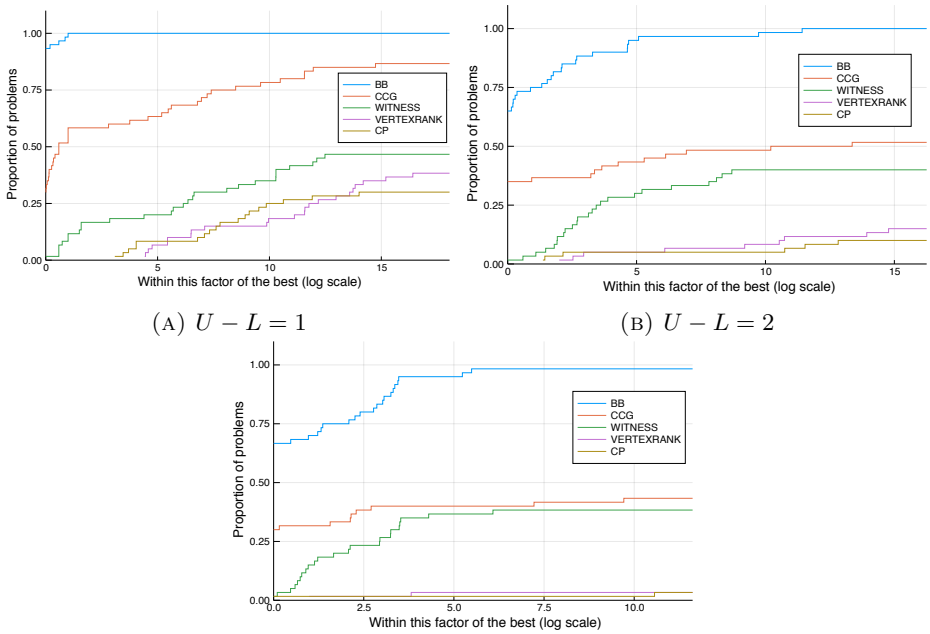
(D) Instances from the **SensorNetwork** set.



(E) Instances from the **Interdiction** set.

FIGURE 6.2. Comparison of our branch-and-bound algorithm to existing methods. Every profile is related to one family of instances: (a) **Random**, (b) **Synthetic**, (c) **Protein**, (d) **Sensor-Network**, (e) **Interdiction**.

The performance profiles in Figure 2 allow to analyse each family of instances independently. For instances of **Random**, **Protein** and **Interdiction**, the profiles seem representative of the overall results. We remark that the dominance of BB is more pronounced when the sensor network instances are concerned. Looking at the tables in Appendix B, it appears that even for BB], the solution of sensor network instances takes on average more time than for the other families of instances. Due to this additional difficulty, other methods reach the time limit. We can also remark that CCG outperforms BB on the **Synthetic** benchmark. In fact, for these particular instances, the linear relaxation of CCG provides a bound that is very close to the optimal value, leading as a consequence to good performances. Moreover, these instances include only 25 to 35 vertices which yields many computational times below one second. As a consequence, machine overhead may have an impact on this comparison.



(A) $U - L = 1$ (B) $U - L = 2$

(C) $U - L = 3$

FIGURE 6.3. Comparison to existing methods per value of $U - L$.

To establish the impact of the value of $U - L$ on the performances of the various algorithms, we provide independent performance profiles where the value of $U - L$ is fixed to 1, 2 or 3 (see Figure 3). The results confirm the intuition that the problem gets more difficult when the value of $U - L$ increases. This is expected for our branch-and-bound algorithm, because the enumeration is based on the number of partial candidates which must necessarily increase with the value of U. Since L is constant, the number of candidate vertices at a given enumeration node is the same, but the number of full ones will necessarily decrease. A similar combinatorial effect seems to affect every other approach. We also observe that WITNESS is less impacted than other approaches. This seems to indicate that the concept of "witness" yields a stronger formulation when $U - L$ increases, but additional analyses will be necessary to conclude on this issue.

6.3. **Assessment of improvements in the branch-and-bound algorithm.** In order to assess the various improvements that we have proposed for our branch-and-bound framework (see Section 5), we propose another performance study where different versions are compared.

For a reasonable computational effort, we perform this analysis on a subset of the above benchmark, where we keep only two instances per family. In doing so, we selected those that appear to be the most difficult in our previous tests[5].

The "default" version of the algorithm is the implementation used in previous sections, BB. The "no dominance" and "no symmetry break" versions correspond to the "default" version where the dominance rules or the symmetry break strategy, respectively, are not used. The "relaxation bound" version differs from the default one only in the computation of the lower bound at each enumeration node. In this

[5]Referring to tables 1-4, we keep the following ten instances: random35(1), random35(2), synthetic35(1), synthetic35(3), sensor10, sensor12, interdiction045, interdiction050, 1bpm(2) and 1bpm(3)

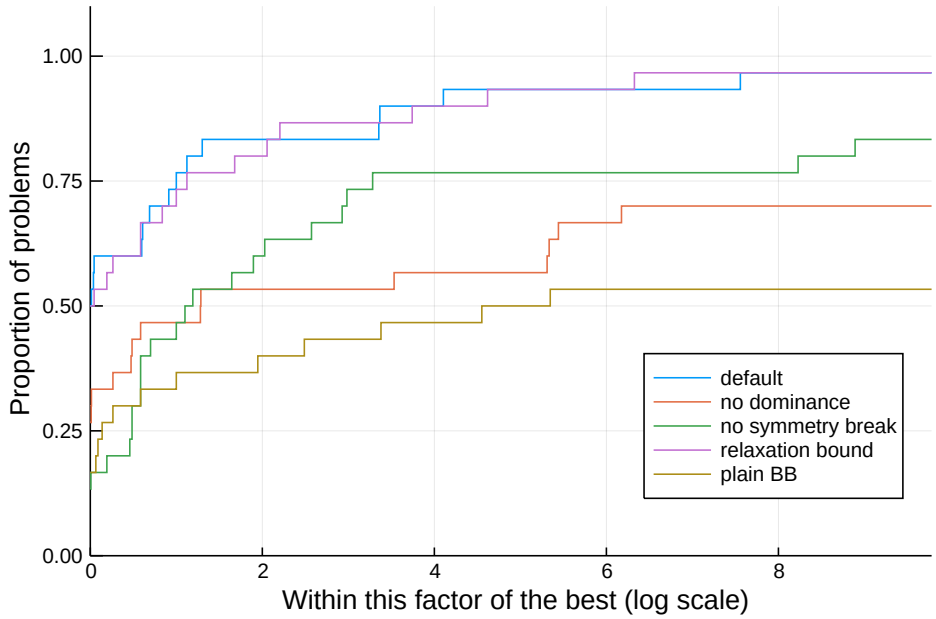
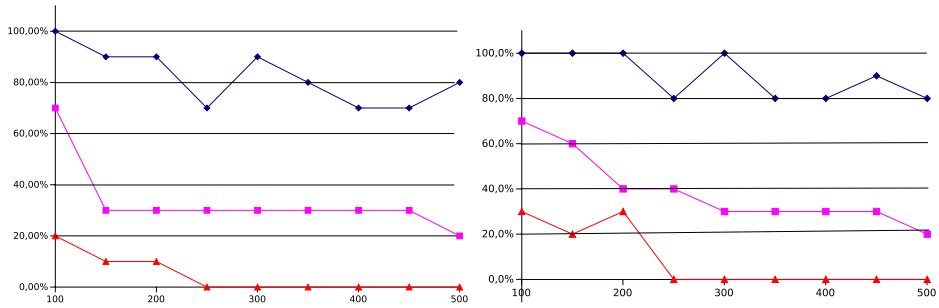


FIGURE 6.4. Performance profiles for the different versions of our branch-and-bound algorithm.

version, `lowerbound(`σ`)` returns $\mathrm{LB}^{\mathrm{CCG}}(\sigma)$ instead of $\mathrm{LB}^{\mathrm{trivial}}(\sigma)$ (see Section 5.3), meaning that the linear relaxation of an IP is solved at each enumeration node to compute a lower bound better than the trivial one. Finally, the version "plain BB" does not use any pruning technique.

Figure 4 shows the performance profile obtained when comparing these different versions against each other. The default version appears to have better performances in comparison with the others even though its profile is very similar to that observed when the "relaxation bound" version is used. With a more thorough look at results, it appears there is a clear dominance of the default version on the **Random**, **SensorNetwork** and **Interdiction** instances, whereas the "relaxation bound" version is the best performing one on **Synthetic** and **Protein**. For the latter two families of instances, we observed that $\mathrm{LB}^{\mathrm{CCG}}(\sigma)$ provides lower bounds that are in general much closer to the value of the optimal completion of σ than $\mathrm{LB}^{\mathrm{trivial}}(\sigma)$. When this is not the case, the time spent computing $\mathrm{LB}^{\mathrm{CCG}}(\sigma)$ is not compensated by the resulting reduction in the number of enumeration nodes. In contrast, the performance profiles highlight that there is a clear contribution of the pruning techniques based on dominance and symmetry. The version where none of these techniques is used indicates that CCG would be better performing if no pruning had been implemented.

6.4. **Impact of the size of the graph.** We now study the impact of the size of the graph on the performances of the branch-and-bound algorithm. For this, we run the algorithm on instances including up to 500 vertices with $U - L = 1, 2, 3$. In order to create a benchmark with representative entries among those in the instances described in Section 6.1, we keep only two instances per family and per number of vertices. The parameters we use to generate the instances are those we already identified when comparing the different versions of the branch-and-bound (which indicate the instances that are hardest to solve on average). Given that



(A) Default branch-and-bound

(B) Best branch-and-bound method among "default" and "relaxation bound" versions

FIGURE 6.5. Percentage of instances solved, per number of vertices and values of $U-L$ ($U-L=1$ in blue, $U-L=2$ in magenta and $U-L=3$ in red)

these parameters lead to infeasible or trivially solved instances for two families, we had to slightly modify some parameter values as follows.

For synthetic instances, we initially took as a reference the density ($D=0.3$) related to the most difficult instances of the previous benchmark. With increasing numbers of vertices, our preliminary tests highlighted that the resulting instances had trivial solutions that included only fully-referenced vertices. As a consequence, we reduced the density by setting it to $D=0.3\frac{\sqrt{35}}{\sqrt{n}}$, which matches $D=0.3$ when $n=35$.

In order to generate larger sensor network instances, we have selected the threshold values 0.25 and 0.35 with a square shape. We then modified the lengths of the square's sides to keep a constant density of sensors in the area.

We run BB on every large instance, and, given that last section showed that the "relaxation bound" version gets better results on two families of instances, we also run this version. In Figure 5, we report the percentage of successful runs of the branch-and-bound algorithm with respect to the instance size, for separated values of $U-L$. On the left subfigure, we focus on default parameter values, whereas the right subfigure displays the percentage of instances solved by either of the two sets of parameters. We observe that every instance can be solved for $U-L=1$ and $n=100,150,200$ and 300 with one of the two versions. Moreover, never less than 80% are solved for instances including up to 500 vertices. In comparison, existing methods could not find provable optimal solutions of more than 50% of instances including as few as 35 to 60 vertices. Given that $U-L=1$ is the value that appears in the applications related to distance geometry (see Section 2.1), these results establish a significant step towards the exact resolution of real instances where thousands of vertices need to be considered.

In contrast, as the value of $U-L$ increases, it becomes more and more difficult for the branch-and-bound algorithm to find provable optimal solutions within the required time threshold.

7. CONCLUSIONS

We have introduced a vertex ordering decision problem, REVORDER, and its optimization counterpart, MIN REVORDER. By exploiting some previous results obtained for related applications, we have proposed a brief survey on existing methods. We have then proposed new cutting planes for an existing IP formulation and developed a new branch-and-bound framework for MIN REVORDER. The computational

results highlight that the branch-and-bound clearly outperforms existing solution methods. This improvement allows the solution of instances with up to 500 vertices, even though these instances were willingly constructed to show the limits of solution methods.

MIN REVORDER is a generic problem for which some applications are already known; moreover, we believe it can serve as a basis for the (re)formulation of other problems arising in other research areas. One of the research lines that we find interesting is, for example, the one related to epidemic networks [7]. To completely fulfill this aim, there are some possible extensions that we can foresee. First of all, we have limited our current study to undirected graphs G, because it captures the main applications we took into consideration. However, the entire work may be extended to directed graphs for considering other applications where the orientation of edges is important. This would be natural if some relevant applications emerge in the area of scheduling.

Another assumption that we have considered in this work, which may be relaxed or totally removed in future works, is related to the shape of the objective function $\delta_\sigma(v)$. Again, our choice in the current work was led by the fact that the considered applications do not need more generic objective functions. However, if G is a weighted graph (with weights on the vertices and/or on its edges), then the objective function may actually take these weights into consideration, so that it does not correspond anymore with the simple count that we have used above. For instance, consider another distance geometry problem where distances are represented by real-valued intervals instead of precise distances [16]. In this context, a reference with smaller distance interval range yields a reduced set of possible realizations. To do so, the objective function needs to include a term where the lengths of the distance interval of a vertex is added every time it serves as a reference.

Appendix A. Existing integer programming and constraint programming formulations

In this section, we give more details about the approaches from the literature that are used in our computational tests. While doing this, we will simplify the presentation by assuming that the initial sets are the $(L+1)$-cliques of G, which is the case in all our test instances. The generalization to arbitrary initial sets would be possible by enumerating them as in the cycle-based extended formulation.

Please see [13] for the original descriptions of the models and other approaches to the discretization of distance geometry graphs.

A.1. **A rank-based IP formulation: vertexrank.**

The compact IP formulation described in [13] is based on a model proposed by Lavor et al. [9], where binary variables indicate which vertex is at each rank. In the following presentation of the model, $\delta^r = 1$ if and only if the vertex with rank $r \in [\![L+2, n]\!]$ is partially-referenced, $\sigma_v^r = 1$ if and only if $v \in V$ is at rank $r \in [\![1, n]\!]$ and $y_{r,v} = 1$ if v is at rank r and v is partially-referenced.

$$
\begin{aligned}
\min \quad & 1 + \sum_{r \in [\![L+2,n]\!]} \delta^r \\
\text{s. t.} \quad & \sum_{r \in [\![1,n]\!]} \sigma_v^r = 1,\ \forall v \in V && \text{(A.1)}\\
& \sum_{v \in V} \sigma_v^r = 1,\ \forall r \in [\![1,n]\!] && \text{(A.2)}\\
& \sum_{u \in \mathcal{N}(v)} \sum_{q \in [\![1,r-1]\!]} \sigma_u^q \geq r\sigma_v^r,\ \forall v \in V, r \in [\![1,L+1]\!], && \text{(A.3)}\\
& \sum_{u \in \mathcal{N}(v)} \sum_{q \in [\![1,r-1]\!]} \sigma_u^q \geq L\sigma_v^r,\ \forall v \in V, r \in [\![L+2,n]\!], && \text{(A.4)}\\
& \sum_{u \in \mathcal{N}(v)} \sum_{q \in [\![1,r-1]\!]} \sigma_u^q \geq U(1-y_{r,v}),\ \forall v \in V, \forall r \in [\![L+2,n]\!], && \text{(A.5)}\\
& \delta^r \geq y_{r,v} + \sigma_v^r - 1,\ \forall v \in V, \forall r \in [\![L+2,n]\!], && \text{(A.6)}\\
& x \in \{0,1\}^{2|E|}, \kappa \in \{0,1\}^{|\mathcal{S}|}, \delta \in \{0,1\}^{n-L-1}, y \in \{0,1\}^{n(n-L-1)}.
\end{aligned}
$$

Constraints (9) and (10) ensure that there is exactly one vertex per rank and reciprocally. Constraints (11) ensure that the $L+1$ first vertices induce a complete subgraph of G. By (12), vertex v is at rank r only if it has at least L references, and by (13), v is fully-referenced and at rank r only if it has at least U references. Finally, constraints (14) ensure that if v is at rank r and partially-referenced, then the vertex at rank r is partially-referenced.

A.2. **The witness-based decomposition: witness.** Bodur and MacNeil [13] develop a decomposition scheme based on WITNESS vertices. A witness is a reference vertex that is necessary to satisfy the reference constraints. As a consequence, partially and fully-referenced vertices have exactly L and U witnesses, respectively. Moreover, the initial vertices are not assigned a specific rank among the first $L+1$. Instead they are all witness to one another and to all their other neighbors. The decomposition yields an extended formulation including one constraint per directed cycle of G:

$$
\begin{aligned}
\min \quad & \sum_{v \in V} \delta_\sigma(v) \\
\text{s. t.} \quad & \sum_{v \in V} \kappa_v = L+1 && \text{(A.7)}\\
& \kappa_u + \kappa_v \leq 1,\ \forall u,v \in V : u \neq v, \{u,v\} \notin E && \text{(A.8)}\\
& \kappa_v \leq w_{u,v},\ \forall v \in V, u \in \mathcal{N}(v) && \text{(A.9)}\\
& \delta_\sigma(v) \leq 1 - \kappa_v,\ \forall v \in V && \text{(A.10)}\\
& \sum_{u \in \mathcal{N}(v)} w_{u,v} = L(1-\kappa_v) + (U-L)(1-\delta_\sigma(v)) + L\kappa_v,\ \forall v \in V, && \text{(A.11)}\\
& \sum_{(u,v) \in A^C} w_{u,v} \leq \left|V^C\right| - 1 + \mathbb{1}\left(\left|V^C\right| \leq L+1\right) \times \sum_{v \in V^C} \kappa_v,\ \forall C \in \mathcal{C} && \text{(A.12)}\\
& w \in \{0,1\}^{2|E|}, \kappa \in \{0,1\}^n.
\end{aligned}
$$

where $w_{u,v} = 1$ if and only if u is witness to v, $\{u, v\} \in E$. Constraints (15)–(17) guarantee that the $L + 1$ vertices belonging to the initial clique are pairwise-connected and that they are witness to all their neighbors. The valid inequalities (18) state that the first $L + 1$ vertices are not partially-referenced. Constraints (19) ensure that the vertices of the initial clique have exactly L witnesses, and that other vertices have L witnesses if they are partially-referenced and U witnesses if they are fully-referenced. Constraints (20) forbid cycles in the directed graph containing every arcs (u, v) such that $w_{u,v} = 1$. One specificity of the model is that the first $L + 1$ vertices are witness to one another. As a consequence, the cycle constraints (20) are lifted in the space (w, κ) to allow for cycles containing only vertices among the $L + 1$ first in the order. This is required only for cycles including at most $L + 1$ vertices, because MacNeil and Bodur [13] have shown that it is not necessary to forbid directed cycles that include vertices both inside and outside the first $L + 1$ vertices.

Similarly to CCG, the above extended formulation is solved with a cutting plane algorithm, where every 2 and 3-cycle is included in the initial relaxation.

A.3. **A constraint programming approach: cp.** In [13], Bodur and MacNeil develop three different constraint programming approaches, two of which performed similarly well and better than the third one in their experiments. The model we describe, CP, is among those two best approaches. In CP, decision variable v_r is equal to the vertex whose rank is r in an optimal referenced order, and δ^r is as in VERTEXRANK. The model involves the adjacency matrix of G, A, i.e., $A_{u,v} = 1$ if and only if $\{u, v\} \in E$ for all $u, v \in V$.

$$
\begin{aligned}
\text{CP} : \min \quad & 1 + \sum_{r \in [\![L+2,n]\!]} \delta^r \\
\text{s. t.} \quad & \text{AllDifferent}(v_1, \ldots, v_n) && \text{(A.13)} \\
& A_{v_q,v_r} = 1,\ \forall r \in [\![1, L]\!],\ \forall q \in [\![r + 1, L + 1]\!], && \text{(A.14)} \\
& \sum_{q \in [\![1,r-1]\!]} A_{v_q,v_r} \geq L + (1 - \delta^r)(U - L),\ \forall r \in [\![L + 2, n]\!] && \text{(A.15)} \\
& \forall r \in [\![1, n]\!] : v_r \in [\![1, n]\!],\ \delta^r \in \{0, 1\}
\end{aligned}
$$

Constraint (21) ensures that variables $v_1, \ldots, v_n$ describe a vertex ordering. It uses the AllDifferent constraint, which is classical to handle scheduling problems. Constraints (22) state that the first $L + 1$ vertices are pairwise connected, while constraints (23) ensure that the following ones have at least L references and that they are partially-referenced if they have less than U. These two constraints make use of constraint programming *element* constraints where decision variables can be used as indices.

Appendix B. Details on computational experiments

The following tables show our computational experiments aimed at performing a global comparison among all methods discussed in the article. For every instance belonging to one of the five sets described in Section 6, we run the experiments for three values of $U - L$: 1, 2 and 3. For every instance, for every value of $U - L$, and for every solver, we report in the following tables: the objective value of the best solution found ("obj"), the computational time in seconds ("time"), and the number of branch-and-bound nodes explored by the solver ("#-bbnodes"). In column "time", TL indicates that the time limit was reached. In every other case, optimality is reached, so the value indicated in column "obj" is the optimal one.

		CCG			CP			VERTEXRANK			WITNESS			BB		
name	$U-L$	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes
random25(1)	1	5	2.2	4954	5	TL	11722414	5	69.59	10766	5	114.38	187090	5	0.02	34
random25(1)	2	12	0.1	90	12	TL	15128925	12	TL	118435	12	0.97	605	12	0.29	712
random25(1)	3	15	0.07	107	15	TL	12996265	15	TL	115699	15	0.54	62	15	0.57	1299
random25(2)	1	7	39.82	122132	7	TL	12722000	7	869.97	132912	7	TL	1016500	7	0.01	23
random25(2)	2	12	0.01	0	12	TL	15175386	12	TL	144810	12	0.37	6	12	0.25	541
random25(2)	3	15	0.41	256	15	TL	17959936	15	TL	109648	15	0.9	687	15	0.8	1213
random25(3)	1	1	0.02	0	1	0.11	119	1	0.24	0	1	0.02	0	1	0.02	2
random25(3)	2	2	0.07	0	2	0.19	992	2	0.28	0	2	TL	572485	2	0.07	84
random25(3)	3	7	352.36	112430	7	TL	11257825	8	TL	184717	8	TL	450520	7	0.42	970
random25(4)	1	2	0.02	0	2	18.58	136434	2	18.83	6446	2	TL	662835	2	0.02	22
random25(4)	2	4	TL	520032	4	217.15	1776663	4	468	83361	4	TL	554300	4	0.03	94
random25(4)	3	7	TL	482102	7	TL	10132537	7	TL	119473	7	TL	297100	7	2.42	4009
random30(1)	1	3	547.46	716969	3	14.72	102393	3	93.08	10146	3	TL	494050	3	0.02	10
random30(1)	2	8	316.29	319804	8	TL	9329234	9	TL	93653	9	TL	468500	8	0.03	101
random30(1)	3	12	0.33	303	13	TL	9432416	13	TL	103657	12	22.37	12975	8	21.52	15327
random30(2)	1	4	TL	823800	4	38.56	243033	4	122.76	15374	4	TL	351000	4	0.01	5
random30(2)	2	8	TL	811019	8	TL	9195733	8	841.2	58989	9	TL	271719	8	0.03	54
random30(2)	3	12	11.93	9870	13	TL	8707327	13	TL	115341	13	TL	223800	12	1.83	2626
random30(3)	1	1	0.02	0	1	0.26	1024	1	0.43	0	1	0.03	0	1	0.02	2
random30(3)	2	2	0.07	0	2	0.31	1141	2	0.54	0	2	TL	279241	2	0.09	131
random30(3)	3	4	TL	370572	4	983.9	4239704	4	TL	101573	4	TL	503000	4	0.65	1610
random30(4)	1	1	0.03	0	1	0.17	135	1	0.44	0	1	0.02	0	1	0.02	2
random30(4)	2	3	TL	463918	3	240.62	1087651	3	238.88	55488	3	TL	273530	3	0.14	335
random30(4)	3	5	TL	359273	5	TL	4686820	5	TL	90605	6	TL	321388	5	0.39	1032
random35(1)	1	3	TL	607387	3	TL	2692073	3	TL	49737	3	TL	186538	3	0.04	57
random35(1)	2	6	TL	347979	6	TL	3697439	6	TL	63703	6	TL	380186	6	0.28	703
random35(1)	3	8	TL	289870	8	TL	4229144	8	TL	54678	8	TL	189049	12	2.4	3525
random35(2)	1	3	TL	632966	3	329.29	908567	3	753.79	35131	3	TL	184608	3	0.02	38
random35(2)	2	6	TL	444300	6	TL	3880124	6	TL	34674	6	TL	337494	6	0.19	465
random35(2)	3	10	TL	197900	10	TL	4293148	10	TL	51067	10	TL	172700	9	11.18	9865
random35(3)	1	1	0.02	0	1	0.33	222	1	0.87	0	1	0.03	0	1	0.02	2
random35(3)	2	3	TL	259000	3	890.73	2757415	3	445.38	89493	3	TL	91989	3	0.3	639
random35(3)	3	5	TL	183200	5	TL	2784221	5	TL	60678	5	TL	197700	5	8.97	9930
random35(4)	1	1	0.03	0	1	0.33	98	1	0.87	0	1	0.03	0	1	0.02	2
random35(4)	2	2	0.19	0	2	0.49	1137	2	1.1	0	2	TL	94546	2	0.21	266
random35(4)	3	3	TL	190650	3	0.54	1136	3	1.07	0	3	TL	237060	3	0.75	1596

TABLE B.1. Experiments with our **Random** set of instances.

		CCG			CP			VERTEXRANK		WITNESS			BB			
name	$U-L$	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes
synthetic25(1)	1	2	0.010	0	2	1.600	18932	2	22.200	5156	2	0.520	678	2	0.010	13
synthetic25(1)	2	11	0.130	61	11	TL	11294319	11	TL	196178	11	1.580	702	11	0.240	558
synthetic25(1)	3	15	0.180	14	15	TL	13137482	15	TL	135958	15	0.570	38	15	1.230	2327
synthetic25(2)	1	2	0.010	0	2	1.960	19601	2	30.690	8133	2	0.990	1102	2	0.020	13
synthetic25(2)	2	11	0.100	28	11	TL	13844611	11	TL	166552	11	0.570	34	11	0.250	592
synthetic25(2)	3	15	0.260	12	15	TL	9976122	15	TL	128996	15	0.360	34	15	2.590	3722
synthetic25(3)	1	2	0.020	0	2	4.820	70283	2	145.790	75206	2	0.210	14	2	0.010	4
synthetic25(3)	2	12	0.090	26	12	TL	12548277	12	TL	210283	12	0.790	34	12	0.290	598
synthetic25(3)	3	15	0.550	66	15	TL	11884224	16	TL	124126	15	0.870	725	15	1.370	2126
synthetic25(4)	1	2	0.010	0	2	15.760	189141	2	187.100	64075	2	0.490	244	2	0.010	5
synthetic25(4)	2	12	25.160	41922	12	TL	11946287	12	TL	216127	12	98.840	84086	12	0.360	752
synthetic25(4)	3	15	0.580	11	15	TL	12901645	15	TL	116698	15	0.520	50	15	2.210	2999
synthetic30(1)	1	2	0.020	0	2	4.430	25371	2	60.800	9201	2	1.640	1372	2	0.030	17
synthetic30(1)	2	14	0.130	61	14	TL	9598874	14	TL	81628	14	1.920	980	14	0.810	1464
synthetic30(1)	3	18	2.490	2780	19	TL	7978909	19	TL	153096	18	3.660	2367	18	0.840	1617
synthetic30(2)	1	2	0.020	0	2	5.470	29041	2	80.620	8470	2	19.140	19839	2	0.010	17
synthetic30(2)	2	13	0.130	74	13	TL	8408225	14	TL	80395	13	3.940	4080	13	0.560	1062
synthetic30(2)	3	18	0.540	795	18	TL	7541638	19	TL	137281	18	5.110	3360	18	4.510	5725
synthetic30(3)	1	3	0.020	0	3	TL	8350496	3	TL	77855	3	0.950	879	3	0.010	17
synthetic30(3)	2	15	5.150	7905	15	TL	8091100	15	TL	112953	15	130.760	89674	15	0.410	878
synthetic30(3)	3	19	0.100	13	19	TL	9251095	19	TL	159631	19	1.970	1090	19	1.090	2127
synthetic30(4)	1	3	0.020	0	3	TL	9331139	3	TL	95557	3	7.860	8310	3	0.020	17
synthetic30(4)	2	14	7.860	10565	15	TL	10379489	15	TL	128804	14	44.750	31158	14	0.200	474
synthetic30(4)	3	18	0.140	132	18	TL	7515367	19	TL	134799	18	1.330	510	18	1.550	2823
synthetic35(1)	1	4	0.020	0	4	TL	6441068	4	TL	65115	4	13.200	13761	4	0.020	11
synthetic35(1)	2	16	0.510	466	17	TL	6256288	18	TL	86973	16	2.300	1129	16	1.740	1863
synthetic35(1)	3	22	0.320	31	22	TL	5620016	22	TL	85581	22	1.350	88	22	14.390	11746
synthetic35(2)	1	4	0.030	0	4	TL	6812934	4	TL	76016	4	117.760	96801	4	0.030	11
synthetic35(2)	2	15	0.570	464	16	TL	6885998	16	TL	46069	15	2.670	1119	15	2.410	2590
synthetic35(2)	3	21	1.440	1640	22	TL	6340791	21	TL	81603	21	16.280	10230	21	14.790	12475
synthetic35(3)	1	4	0.020	0	4	TL	7283278	4	TL	62925	4	5.460	5213	4	0.020	11
synthetic35(3)	2	17	0.270	45	17	TL	7368929	17	TL	77026	17	1.770	148	17	6.780	4725
synthetic35(3)	3	22	1.620	1826	23	TL	5831533	23	TL	88920	22	2.780	1890	22	8.540	8026
synthetic35(4)	1	4	0.020	0	4	TL	6382280	4	TL	66996	4	88.710	82676	4	0.020	11
synthetic35(4)	2	16	0.200	38	16	TL	6939860	16	TL	32678	16	1.290	92	16	6.750	4737
synthetic35(4)	3	21	23.000	16402	22	TL	6056361	23	TL	63480	21	58.020	16785	21	5.190	5703

		CCG			CP			VERTEXRANK		WITNESS			BB			
name	$U-L$	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes
1m40(1)	1	30	6.620	4614	30	TL	1112382	30	TL	500	30	613.740	315659	30	0.490	456
1m40(1)	2	46	TL	363589	47	TL	1220052	47	TL	507	47	TL	395623	46	4.850	2850
1m40(1)	3	50	549.340	167758	51	TL	1174242	53	TL	381	51	TL	224000	50	3.680	2003
1m40(2)	1	17	9.040	4812	17	TL	967496	17	TL	556	17	474.020	178621	17	0.380	322
1m40(2)	2	38	TL	309054	38	TL	977320	39	TL	480	38	TL	437200	38	24.110	10540
1m40(2)	3	45	TL	85336	47	TL	1009019	47	TL	462	46	TL	324000	45	41.580	18744
1m40(3)	1	1	0.030	0	1	3.310	1270	1	2.700	0	1	0.050	0	1	0.030	2
1m40(3)	2	27	TL	115357	27	TL	926963	27	TL	977	27	TL	255300	26	7.340	4672
1m40(3)	3	37	TL	89700	38	TL	921894	38	TL	130	38	TL	187700	37	70.280	25527
1bpm(1)	1	25	155.530	91373	25	TL	1195798	25	TL	658	25	TL	384096	25	0.200	215
1bpm(1)	2	41	152.030	107600	42	TL	1261314	42	TL	496	41	TL	446500	41	7.790	3450
1bpm(1)	3	47	TL	320003	48	TL	1396375	49	TL	635	47	TL	284832	47	23.800	8691
1bpm(2)	1	20	55.640	30029	20	TL	1142028	20	TL	789	20	TL	383800	20	0.330	285
1bpm(2)	2	37	161.940	41913	39	TL	1171695	39	TL	524	39	TL	351300	37	15.680	8951
1bpm(2)	3	45	TL	215772	45	TL	1138511	46	TL	486	46	TL	251000	45	199.090	43899
1bpm(3)	1	1	0.030	0	1	4.100	1386	1	2.720	0	1	0.060	0	1	0.030	2
1bpm(3)	2	26	TL	97840	27	TL	871617	27	TL	403	26	TL	218242	26	4.820	2919
1bpm(3)	3	37	TL	68400	37	TL	1017428	38	TL	516	37	TL	227200	37	100.510	33645
1n4w(1)	1	13	11.760	7318	13	TL	860256	13	TL	1005	13	TL	370028	13	0.240	198
1n4w(1)	2	34	3.650	1747	34	TL	979688	35	TL	511	34	39.490	16126	34	4.240	3720
1n4w(1)	3	45	TL	111812	46	TL	876161	46	TL	405	45	TL	381300	45	4.500	4418
1n4w(2)	1	13	37.130	19013	13	TL	786149	13	TL	1003	13	TL	203857	13	0.250	198
1n4w(2)	2	34	6.220	2435	34	TL	948214	35	TL	516	34	3.260	482	34	3.750	3157
1n4w(2)	3	45	TL	105902	46	TL	1161690	46	TL	513	45	TL	296800	45	4.510	4313
1n4w(3)	1	1	0.030	0	1	7.770	2027	1	2.750	0	1	0.060	0	1	0.020	2
1n4w(3)	2	25	TL	126734	26	TL	871293	26	TL	714	25	218.230	68542	25	2.670	2206
1n4w(3)	3	37	TL	128691	37	TL	833270	37	TL	486	37	TL	200994	37	10.630	7554
1mqq(1)	1	30	22.880	30306	30	TL	880203	30	TL	601	30	789.110	519953	30	0.630	604
1mqq(1)	2	45	TL	412104	45	TL	946784	45	TL	505	45	TL	477228	45	5.020	4741
1mqq(1)	3	50	TL	255852	52	TL	1124135	53	TL	135	50	TL	248659	50	3.670	3049
1mqq(2)	1	18	10.230	7052	18	TL	993822	18	TL	1573	18	TL	330375	18	0.230	205
1mqq(2)	2	38	510.640	272831	39	TL	959719	40	TL	141	38	TL	344100	38	4.180	3583
1mqq(2)	3	47	TL	116055	48	TL	913917	48	TL	518	47	TL	269900	46	9.180	7985
1mqq(3)	1	2	0.030	0	2	TL	859283	2	TL	6444	2	TL	288934	2	0.030	3
1mqq(3)	2	26	TL	120342	27	TL	850747	27	TL	505	26	TL	225500	26	3.090	2158
1mqq(3)	3	39	TL	85635	39	TL	817882	39	TL	802	39	TL	162689	39	9.430	6071

TABLE B.3. Experiments with the instances in the **Protein** set.

		CCG			CP			VERTEXRANK		WITNESS			BB			
name	$U-L$	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes
sensor01	1	11	17.960	30019	11	TL	1595660	11	TL	985	11	TL	233900	11	0.050	9
sensor01	2	23	TL	884721	23	TL	1680326	23	TL	514	23	TL	241357	23	2.080	515
sensor01	3	31	TL	413900	31	TL	1643260	31	TL	712	31	TL	0	31	9.990	4162
sensor02	1	3	100.210	102084	3	TL	1374315	3	TL	649	3	TL	136738	3	0.070	10
sensor02	2	7	TL	302900	7	TL	1493383	7	TL	1427	7	TL	228800	7	4.440	781
sensor02	3	14	TL	208200	14	TL	929604	14	TL	954	14	TL	298658	14	31.270	7606
sensor03	1	2	0.520	0	2	TL	1685170	2	TL	3115	2	TL	66560	2	0.470	2
sensor03	2	5	TL	127210	5	TL	1405327	5	TL	1802	5	TL	67000	5	2.190	1345
sensor03	3	9	TL	153921	9	TL	1377440	9	TL	142	9	TL	131100	9	23.310	17698
sensor04	1	2	0.120	0	2	TL	1674848	2	TL	10896	2	TL	103078	2	0.100	3
sensor04	2	7	TL	233643	7	TL	1302856	7	TL	506	7	TL	0	7	7.520	1539
sensor04	3	13	TL	261500	13	TL	1197783	13	TL	1081	13	TL	243900	13	467.760	25593
sensor05	1	2	0.500	0	2	TL	1801309	2	TL	25911	2	TL	50336	2	0.490	3
sensor05	2	4	TL	123305	4	TL	1075126	4	TL	1057	4	TL	41300	4	28.720	3269
sensor05	3	6	TL	54600	6	TL	1344428	6	TL	2236	6	TL	35000	6	200.940	11334
sensor06	1	2	0.370	0	2	TL	1506982	2	TL	4617	2	TL	71725	2	0.340	4
sensor06	2	5	TL	183500	5	TL	1388480	5	TL	1020	5	TL	51800	5	13.650	2631
sensor06	3	9	TL	107400	9	TL	1470629	9	TL	1568	9	TL	0	9	124.990	14816
sensor07	1	2	0.050	0	2	TL	1439466	2	TL	28277	2	TL	262200	2	0.040	3
sensor07	2	7	TL	427100	7	TL	1345068	7	TL	599	7	TL	355500	7	2.790	712
sensor07	3	15	TL	182100	15	TL	990029	15	TL	785	15	TL	0	14	16.780	5674
sensor08	1	5	TL	613200	5	TL	1358343	5	TL	2091	5	TL	142000	5	0.070	7
sensor08	2	11	TL	258000	11	TL	1417955	12	TL	549	12	TL	195300	11	4.950	931
sensor08	3	16	TL	205300	16	TL	1099678	16	TL	523	16	TL	0	16	29.850	7710
sensor09	1	2	0.140	0	2	TL	1867949	2	TL	40553	2	TL	112377	2	0.110	4
sensor09	2	5	TL	157300	5	TL	1319484	5	TL	2364	5	TL	182200	5	6.970	800
sensor09	3	9	TL	125600	9	TL	1228634	9	TL	1405	9	TL	218800	9	24.800	4589
sensor10	1	5	TL	639869	5	TL	1564508	5	TL	4427	5	TL	258400	5	0.090	24
sensor10	2	14	TL	397400	14	TL	1172828	14	TL	673	14	TL	0	14	5.580	1637
sensor10	3	22	TL	239098	22	TL	1444724	22	TL	562	22	TL	140800	21	198.970	31357
sensor11	1	2	0.100	0	2	TL	1613310	2	89.500	143	2	TL	69847	2	0.090	2
sensor11	2	3	TL	164872	3	TL	1263191	3	661.920	1943	3	TL	0	3	1.130	909
sensor11	3	5	TL	83000	5	TL	1285511	5	TL	1779	5	TL	193605	5	1.310	1114
sensor12	1	2	2.220	0	2	TL	1983541	2	57.240	95	2	TL	38830	2	2.120	2
sensor12	2	3	TL	81500	3	TL	1472746	3	381.670	511	3	TL	29200	3	5.590	2316
sensor12	3	4	TL	44300	4	TL	1506089	4	944.530	2848	4	TL	21657	4	67.090	38957

TABLE B.4. Experiments with the instances in the **SensorNetwork** set.

		CCG			CP			VERTEXRANK		WITNESS			BB			
name	$U-L$	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes	obj	time	#bb-nodes
nterdiction010	1	5	75.100	0	5	TL	1354902	5	TL	1056	5	TL	30455	5	74.590	2
nterdiction010	2	7	TL	74387	7	TL	1730146	7	TL	525	7	TL	23451	7	75.560	582
nterdiction010	3	9	TL	31135	9	TL	1174350	9	TL	488	9	TL	18025	9	75.620	629
nterdiction020	1	8	TL	227076	8	TL	1703568	8	818.680	2320	8	TL	72282	8	0.060	2
nterdiction020	2	12	TL	155510	12	TL	1259155	12	TL	1436	12	TL	52650	12	0.410	327
nterdiction020	3	14	TL	153567	14	TL	1545547	14	TL	5717	14	TL	70703	14	4.240	6094
nterdiction030	1	10	TL	544761	10	TL	1594289	10	TL	2157	10	TL	96500	10	0.020	2
nterdiction030	2	18	TL	320820	18	TL	1461710	18	TL	5670	18	TL	127507	17	0.210	201
nterdiction030	3	23	TL	276500	23	TL	1426006	23	TL	5103	23	TL	208953	23	0.350	388
nterdiction040	1	7	0.020	0	7	TL	1504179	7	TL	1985	7	TL	297809	7	0.020	2
nterdiction040	2	19	TL	392443	20	TL	1369075	20	TL	2959	19	TL	726658	19	1.570	1426
nterdiction040	3	30	TL	409100	32	TL	1256436	33	TL	4669	30	TL	255300	30	35.530	22382
nterdiction045	1	21	TL	1356237	21	TL	1516114	21	TL	3275	21	TL	401148	21	0.010	5
nterdiction045	2	30	129.970	101067	30	TL	1135958	30	TL	5848	30	TL	797679	30	0.110	169
nterdiction045	3	38	1.790	1679	40	TL	1207098	40	TL	6239	38	1.920	462	38	67.410	46914
nterdiction050	1	18	30.300	61748	18	TL	1529937	18	TL	3563	18	TL	443300	18	0.010	2
nterdiction050	2	33	0.630	802	34	TL	1349539	34	TL	4625	33	0.950	598	33	536.030	88326
nterdiction050	3	42	0.390	58	43	TL	1125931	43	TL	5785	42	0.910	570	42	TL	131035
nterdiction055	1	24	30.270	71347	24	TL	1461811	24	TL	6008	24	TL	548960	24	0.010	2
nterdiction055	2	36	6.360	7267	38	TL	1179962	36	TL	11619	36	281.910	175861	36	0.680	1109
nterdiction055	3	43	1.660	1256	46	TL	1217963	46	TL	5783	43	2.910	3038	43	0.380	720
nterdiction060	1	25	8.710	18772	25	TL	1461033	25	TL	3089	25	TL	667066	25	0.070	55
nterdiction060	2	38	0.350	220	38	TL	1152288	38	TL	6938	38	1.320	199	38	9.040	9433
nterdiction060	3	45	2.690	2508	48	TL	1052867	47	TL	5220	45	6.330	6935	45	0.550	976
nterdiction065	1	26	0.620	1258	26	TL	1527522	26	TL	3065	26	4.500	6408	26	1.130	810
nterdiction065	2	42	0.300	200	43	TL	1204494	43	TL	3565	42	1.050	702	42	823.270	112091
nterdiction065	3	48	0.510	451	50	TL	1186105	50	TL	12003	48	0.770	206	48	4.860	6573
nterdiction070	1	35	0.770	2581	35	TL	1459210	35	TL	4358	35	7.810	17486	35	0.110	162
nterdiction070	2	46	0.300	287	48	TL	1153092	47	TL	19815	46	0.640	200	46	2.940	4948
nterdiction070	3	50	0.180	99	51	TL	1064518	51	TL	5386	50	0.350	0	50	0.860	1807
nterdiction080	1	51	0.080	27	51	TL	1322006	51	TL	13187	51	0.170	20	51	0.060	240
nterdiction080	2	55	0.050	0	55	TL	1228694	55	TL	4693	55	0.190	0	55	0.320	166
nterdiction080	3	57	0.090	12	57	TL	843813	57	TL	3528	57	0.150	0	57	0.210	33
nterdiction090	1	52	0.070	25	52	TL	1492574	52	TL	10191	52	0.210	4	52	0.080	123
nterdiction090	2	56	0.090	24	56	TL	1128489	56	TL	2802	56	0.250	4	56	0.110	93
nterdiction090	3	57	0.070	0	58	TL	1127325	58	TL	9057	57	0.130	0	57	0.180	22

References

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

[2] F.R.K. Chung and L. Lu. *Complex Graphs and Networks*, volume 107. American Mathematical Society and CBMS Regional Conference Series in Mathematics, 2006.

[3] Y. Ding, N. Krislock, J. Qian, and H. Wolkowicz. Sensor network localization, euclidean distance matrix completions, and graph realization. *Optimization and Engineering*, 11:45–66, 2010.

[4] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[5] Douglas S. Gonçalves and Antonio Mucherino. Optimal partial discretization orders for discretizable distance geometry. *International Transactions in Operational Research*, 23(5):947–967, 2016, `https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.12249`.

[6] Mehdi Hemmati, J. Cole Smith, and My T. Thai. A cutting-plane algorithm for solving a weighted influence interdiction problem. *Computational Optimization and Applications*, 57(1):71–104, 2014.

[7] M.J. Keeling and K.T.D Eames. Networks and epidemic models. *Journal of the Royal Society Interface*, 2:295–307, 2005.

[8] C. Lavor, L. Liberti, N. Maculan, and A. Mucherino. The discretizable molecular distance geometry problem. *Computational Optimization and Applications*, 52(1):115–146, 2012, `https://arxiv.org/abs/0608012`.

[9] Carlile Lavor, Jon Lee, Audrey Lee-St. John, Leo Liberti, Antonio Mucherino, and Maxim Sviridenko. Discretization orders for distance geometry problems. *Optimization Letters*, 6(4):783–796, 2012.

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

[11] L. Liberti, B. Masson, J. Lee, C. Lavor, and A. Mucherino. On the number of realizations of certain Henneberg graphs arising in protein conformation. *Discrete Applied Mathematics*, 165:213–232, 2014.

[12] Leo Liberti, Carlile Lavor, and Nelson Maculan. A branch-and-prune algorithm for the molecular distance geometry problem. *International Transactions in Operational Research*, 15(1):1–17, 2008, `https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1475-3995.2007.00622.x`.

[13] Moira MacNeil and Merve Bodur. Integer Programming, Constraint Programming, and Hybrid Decomposition Approaches to Discretizable Distance Geometry Problems. `https://arxiv.org/abs/1907.12468`, Preprint manuscript, 2019.

[14] C. Miller, A. Tucker, and R. Zemlin. Integer programming formulations and the travelling salesman problems. *Journal of the ACM*, 7:326—-329, 1960.

[15] A. Mucherino, C. Lavor, and L. Liberti. The discretizable distance geometry problem. *Optimization Letters*, 6:1671–1686, 2012.

[16] Antonio Mucherino. On the Identification of Discretization Orders for Distance Geometry with Intervals. In *Lecture Notes in Computer Science*, pages 231–238. Springer, 2013.

[17] Jérémy Omer and Douglas S. Gonçalves. An integer programming approach for the search of discretization orders in distance geometry problems. *Optimization Letters*, 2017.

[18] Jeremy Omer and Tangi Migot. Vertex ordering with optimal number of adjacent predecessors. *Discrete Mathematics and Theoretical Computer Science*, 22(1):19, 2019.

Univ Rennes, INSA Rennes, CNRS, IRMAR, UMR 6625, F-35000 Rennes, France
E-mail address: `jeremy.omer@insa-rennes.fr`

IRISA, University of Rennes 1, Rennes, France
E-mail address: `antonio.mucherino@irisa.fr`