ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

Parallel Implementation of Successive Sparse SDP Relaxations for Large-Scale Euclidean Distance Geometry Problems

Sunyoung Kim, Masakazu Kojima
and Makoto Yamashita

November 2012, B–470

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-470 Parallel Implementation of Successive Sparse SDP Relaxations for Large-Scale Euclidean Distance Geometry Problems

Sunyoung Kim*, Masakazu Kojima†, Makoto Yamashita‡

November 2012

Abstract. The Euclidean distance geometry problem (EDGP) includes locating sensors in a sensor network and constructing a molecular configuration using given distances in the two or three-dimensional Euclidean space. When the locations of some nodes, called anchors, are given, the problem can be dealt with many existing methods. An anchor-free problem in the three-dimensional space, however, is a more challenging problem and can be handled with only a few methods. We propose an efficient and robust numerical method for large-scale EDGPs with exact and corrupted distances including anchor-free three-dimensional problems. The method is based on successive application of the sparse version of full semidefinite programming relaxation (SFSDP) proposed by Kim, Kojima, Waki and Yamashita, and can be executed in parallel. Numerical results on large-scale anchor-free three-dimensional problems with more than 10000 nodes demonstrate that the proposed method performs better than the direct application of SFSDP and the divide and conquer method of Leung and Toh in terms of efficiency and/or effectiveness measured in the root mean squared distance.

Key words. Euclidean distance geometry problem, molecular conformation, the successive sparse SDP relaxations, parallel implementation.

AMS Classification. 90C22, 90C52, 65Y05

* Department of Mathematics, Ewha W. University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. The research was supported by NRF 2012-R1A1A2-038982 and NRF 2010-000-8784.
skim@ewha.ac.kr

† Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan, and Research and Development Initiative & JST CREST, Chuo University, 1-13-27, Kasuga, Bunkyo-ku, Tokyo 112-8551 Japan. This research was supported by Grant-in-Aid for Scientific Research (B) 22310089 and the Japan Science and Technology Agency (JST), the Core Research of Evolutionary Science and Technology (CREST) research project.
kojima@is.titech.ac.jp

‡ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan.
makoto@is.titech.ac.jp

1 Introduction

The Euclidean distance geometry problem (EDGP) of finding the locations of nodes with a given incomplete distance data is considered with focus on anchor-free 3-dimensional problems. The EDGP has been one of the problems studied widely in many areas of applications such as sensor network [2], data visualization [4, 8], multivariate analysis problems in statistics [21], robotics [29] and machine learning [34]. See also [24] and the references therein. In particular, the 3-dimensional structure of a molecule in the protein folding framework with sparse distance data between atoms has been studied by many researchers [1, 5, 7, 12, 13, 14, 15, 22, 25, 26, 36]. Molecular conformation is indeed an anchor-free 3-dimensional EDGP. In this paper, no additional information except the given distances is used to find the configuration of molecules. Thus, we choose to differentiate molecular conformation and anchor-free 3-dimensional EDGPs. When describing the EDGP, nodes indicate sensors (or anchors) in the sensor network localization or atoms in the configuration of molecules.

The EDGP is described in terms of an undirected graph $G(N_*, E_*)$ with a node set N_* and an edge set $E_* \subset \{(p,q) : p,\ q \in N_*,\ p \neq q\}$ in this paper. We assume that the nodes are placed in the ℓ-dimensional Euclidean space $\mathbb{R}^\ell$. In practice, $\ell = 2$ and $\ell = 3$ are important. For each edge $(p,q) \in E_*$, a (corrupted) Euclidean distance d_{pq} is given, and (p,q) is identified with (q,p) so that $d_{pq} = d_{pq}$. The nodes are classified into two sets, the set $A_* \subset N_*$ of anchors, whose locations in $\mathbb{R}^\ell$ are known, and the set $N_* \backslash A_*$ of sensors, whose locations in $\mathbb{R}^\ell$ are unknown. We denote the known location of each anchor $r \in A_*$ by $\boldsymbol{a}_r^*$. The EDGP is to find unknown locations $\boldsymbol{x}_p$ $(p \in N_* \backslash A_*)$ that (approximately) satisfy the system of distance equations

$$\|\boldsymbol{x}_p - \boldsymbol{x}_q\| = d_{pq}\ (p,q) \in E_* \text{ with } \boldsymbol{x}_r = \boldsymbol{a}_r^*\ (r \in A_*).$$

When the anchor set A_* is empty as in molecular conformation problems, the locations of nodes are not uniquely determined. This is because the above distance equations with $A_* = \emptyset$ is invariant under any parallel transition, rotation and reflection. A precise formulation of the EDGP is given in Section 2.

Among the methods proposed for the EDGP, the methods based on semidefinite programming (SDP) relaxation [1, 2, 3, 16, 18, 22, 33, 35] have shown to be very successful. But large-scale EDGPs, especially anchor-free 3-dimensional EDGPs, still remain a very challenging problem, as SDP solvers [6, 32, 30, 31] are very memory-intensive. There have been attempts to solve large-scale EDGPs by exploiting a structured sparsity that exist implicitly in their full semidefinite programming (FSDP) relaxations in [2], called a sparse version of full semidefinite programming relaxation (SFSDP) [16, 18]. When SFSDP was tested for randomly generated EDGPs in [18], the largest 3-dimensional problem included 5000 sensors and 500 anchors on a single machine with 4GB memory. SFSDP works very efficiently and effectively if the degree of each node $p \in N_*$ (the number of edges incident to each node $p \in N_*$) is large (*e.g.*, greater than 50) and a sufficient number of anchors (*e.g.*, 10% of sensors) exist. The tested EDGPs in [16, 18] satisfy these conditions, and it was successful for SFSDP to extract a subgraph with the node set N_* that satisfies a structural sparsity characterized as a sparse chordal extension. The obtained locations of nodes were as accurate as the ones from solving the given

FSDP itself. See Section 3.3 of [16] for more details. It is, however, our experience that a uniformly sparse and large-scale EDGP was difficult to solve by SFSDP.

We say that EDGP is uniformly sparse if the degrees of all nodes $p \in N_*$ of the associated graph $G(N_*, E_*)$ are uniformly bounded by a small number (*e.g.*, less than 40). Handling anchor-free problems by SFSDP is more difficult, as we will see in Section 5.1. We mention that molecular conformation problems in practice are usually uniformly sparse; for example, the degrees of all nodes are bounded by 32 and the average degree of all nodes is bounded 14 in the test problems in Section 5.2.

Although FSDP and SFSDP can not handle uniformly sparse EDGPs with increasingly large size, they can be used to solve small-sized subproblems of the EDGP. In the distributed SDP approach proposed in [1, 3, 22], the graph $G(N_*, E_*)$ was first subdivided into smaller subgraphs using cluster methods — the divide phase. Then, FSDP combined with the gradient method was used to locate the nodes in each subgraph. Finally, a stitching algorithm was used to relocate all the nodes so that they could fit into the entire system of distance equations — the conquer phase. In the recent work in [22], Leung and Toh proposed an SDP-based divide-and-conquer algorithm, called DISCO, for molecular conformation. They demonstrated that DISCO could reconstruct the conformation of large molecules with corrupted distances less than radius 6Å with 20%-30% noise. Test results were shown for 11 molecules using corrupted distances with 30% noise and the radius 6Å, and 20% noise and the radius less than 6Å.

The main purpose of this paper is to propose a new method, called the successive sparse SDP relaxation, for uniformly sparse and large-scale EDGPs. This method utilizes the efficient property of SFSDP for solving an EDGP with the underlying graph satisfying a structured sparsity characterized as a sparse chordal extension. Let us sketch the basic idea of the method for an anchor-free EDGP with a graph $G(N_*, E_*)$. The method successively applies SFSDP to increasingly large subproblems. See Figure 1. Initially an $(\ell+1)$-clique (of $\ell+1$ nodes) of the graph $G(N_*, E_*)$ is chosen and their locations are fixed as an initial (temporary) set of anchors A_0, using the system of distance equations for the initial set of anchors. Initialize $t = 0$. The following four steps are repeated. (i) Take a neighborhood N_t of the anchor set to construct a subproblem. We use $\mathrm{P}(N_t, A_t)$ to denote this subproblem. Nodes that are not fixed in the neighborhood are called sensors. (ii) Apply SFSDP to $\mathrm{P}(N_t, A_t)$ to roughly estimate the locations of the sensor nodes, (iii) Refine the locations of both sensors and anchors by a gradient method. (iv) Based on information available from (ii) and (iii), fix sensors to expand the anchor set A_t to A_{t+1}. (iv) Increase t by 1, and go back to (i). We repeat these steps until the neighborhood N_t reaches N_* and no more sensors can be fixed.

As the iteration of the successive sparse SDP relaxation method proceeds, the anchor set A_t gradually expands from the core of the subgraph associated with the subproblem $\mathrm{P}(N_t.A_t)$, and sensors $p \in N_t \backslash A_t$ are located on its thin boundary. This feature of the subproblem $\mathrm{P}(N_t, A_t)$ considerably enhances the structural sparsity characterized as a sparse chordal extension and makes the application of SFSDP to $\mathrm{P}(N_t, A_t)$ quite efficient. Another important feature of the proposed method is that parallel implementation is straightforward, thus, the method can be run in parallel to improve the accuracy of estimated locations of the nodes. Many candidates are available for an initial $(\ell+1)$-clique to start the proposed method. Therefore, using multiple $(\ell+1)$-cliques, an EDGP

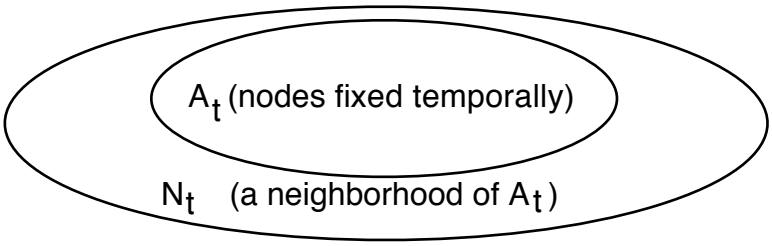

Figure 1: Subproblems to which SFSDP is successively applied. Both A_t and N_t expand until N_t reaches N_*.

can be solved by the method in parallel from those cliques independently. As a result, multiple estimated locations of the nodes are obtained, and it is possible to compound them for a more accurate estimate.

A parallel implementation of the successive sparse SDP relaxation method for EDGPs, which is called passSDP, has been conducted using Matlab. In comparison to SFSDP, passSDP successfully solves randomly-generated 3-dimensional anchor-free EDGPs with 32000 nodes that could not be handled by SFSDP, as shown in Section 5. For the EDGPs tested in [22] from the protein data bank, passSDP provides approximate locations with comparable accuracy to DISCO except for the largest problem 1YGH with 13488 nodes. The error computed by the root mean square distance attained by passSDP is less than half of the one attained by DISCO for 1YGH.

In Section 2, we formulate the EDGP after presenting notation and symbols used throughout the paper. In addition, we introduce a family of subproblems of the EDGP. SFSDP and the gradient method are applied to these subproblems at each iteration of the successive sparse SDP relaxation. The successive sparse SDP relaxation method starting from a single $(\ell + 1)$-clique is outlined in Section 3.1, and its parallel implementation, passSDP in Section 3.2. Section 4 contains technical details of our methods. In Section 5, we present numerical results on passSDP applied to 3-dimensional randomly-generated EGDPs and EGDPs from the protein data bank in comparison to SFSDP and DISCO. In Section 6, concluding remarks are given.

2 Preliminaries

2.1 Notation and symboles

We use the following notation and symbols throughout the paper.

N_* = a finite set of nonnegative integers = the node set of the EDGP,
ℓ = 2 or 3, the dimension of the space where the nodes are placed,
ρ = an inter-nodes distance range or a radio range, $\rho > 0$,
E^ρ = the set of pairs of nodes p and q $(p \neq q)$ with distance not greater than ρ, where $(p, q) \in E^\rho$ is identified with $(q, p) \in E^\rho$,
d_{pq} = an estimated distance between nodes p and q, $(p, q) \in E^\rho$,

where $d_{pq} = d_{qp}$ is assumed,

A_* = the set of given anchors $\subset N_*$,

E_* = a subset of $\subset E^\rho$; E_* is considered because only partial distances not greater than ρ are usually available,

$G(N, E)$ = the undirected graph with a node set $N \subset N_*$ and an edge set $E \subset E_*$.

We assume that the location of each node $r \in A_*$ is, denoted by $\boldsymbol{a}_r^*$, is given. The unknown location of each node $p \in N_* \backslash A_*$ is denoted by $\boldsymbol{x}_p$. For simplicity, $(\boldsymbol{x}_p : N)$ indicates the locations $\boldsymbol{x}_p$ $(p \in N)$ for $N \subset N_*$, and $(d_{pq} : E)$ the set of distances d_{pq} $((p, q) \in E)$ for $E \subset E_*$. Each node $r \in A_*$ is called an anchor and each node $p \in N_* \backslash A_*$ a sensor. $\#A$ stands for the cardinality of the set A.

2.2 A Euclidean distance geometry problem and a family of its subproblems

We formulate the EDGP as

$$\text{minimize} \quad \sum_{(p,q)\in E_*} \left| d_{pq}^2 - \|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2 \right| \quad \text{subject to} \quad \boldsymbol{x}_r = \boldsymbol{a}_r^* \ (r \in A_*). \tag{1}$$

For the description of the successive SDP relaxation method for solving (1), we consider a family of subproblems of (1). Let

$$E(N, A) \ = \ \{(p, q) \in E_* : (p, q) \in (N \times N)\,,\ (p, q) \notin (A \times A)\}\,,$$

for every pair of a nonempty $N \subset N_*$ and $A \subset N$. For such a pair (N, A), consider

$$\begin{aligned} \mathrm{P}(N, A)\text{: minimize} \quad & \sum_{(p,q)\in E(N,A)} \left| d_{pq}^2 - \|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2 \right| \\ \text{subject to} \quad & \boldsymbol{x}_r = \widehat{\boldsymbol{a}}_r \ (r \in A). \end{aligned}$$

Here A is a set of temporary anchors and N is a *neighborhood* of A. Temporary anchors mean nodes fixed to $\widehat{\boldsymbol{a}}_r$ $(r \in A)$ temporarily at an iteration of the successive SDP relaxation method. The concept of neighborhood is crucial for the successive SDP relaxation method. Indeed, the basic idea of the successive SDP relaxation method for solving (1) lies on the expansion of a neighborhood N to the entire set of nodes N_*.

After an $(\ell+1)$-clique C of $G(N_*, E_*)$ is chosen for A_0 in the beginning of the method, one-step neighborhood is considered. The clique consists of $\ell + 1$ nodes $q_1, q_2, \ldots, q_{\ell+1}$ such that $(q_i, q_j) \in E_*$ $(1 \le i < j \le \ell+1)$. For the description of a *one-step neighborhood* N of A, we use

$$U(A) = \left(\bigcup_{p\in A} \{q \in N_* : (p, q) \in E_*\} \right) \bigcup A.$$

It is natural that a neighborhood N of A is chosen such that

$$U^m(A) \subset N \subset U^{m+1}(A) \text{ for some } m \ge 0,$$

where $U^0(A) = A$ and $U^{m+1}(A) = U(U^m(A))$ $(m = 0, 1, \dots)$.

If we take $\widehat{\boldsymbol{a}}^r = \boldsymbol{a}^*_r$ $(r \in \boldsymbol{A}_*)$, $\mathrm{P}(A_*, N_*)$ coincides with EDGP (1). When the distances d_{pq} $((p,q) \in E_*)$ are exact, it is possible to consider the following distance equations,

$$\|\boldsymbol{x}_p - \boldsymbol{x}_q\| = d_{pq} \ ((p,q) \in E(N,A)) \text{ with } \boldsymbol{x}_r = \widehat{\boldsymbol{a}}_r \ (r \in A), \tag{2}$$

instead of the minimization problem $\mathrm{P}(N, A)$. We note that solving (2) is numerically unstable compared to solving (1) because of numerical errors contained in the locations $\widehat{\boldsymbol{a}}_r$ $(r \in A)$ of the temporary anchors.

3 Successive Application of the Sparse SDP Relaxation Method for Parallel Implementation

In the proposed method, the sparse SDP relaxation method (SFSDP) [16, 18] is successively applied to solve (1), thus, it is called the successive sparse SDP relaxation method. Multiple cliques of $(\ell + 1)$ nodes extracted from a given EDGP provide multiple sets of initial anchors for the method to start. As a result, the method can be implemented in parallel. The method starting from each set of the initial anchors is outlined in Section 3.1 and its parallel implementation in Section 3.2. Their technical details are presented in Section 4. In particular, SFSDP is used in Section 3.1 and explained in Section 4.3. Both of the serial and parallel algorithms, shown in Section 3.1 and Section 3.2 respectively, include initial steps of preparing $(\ell+1)$-cliques. To emphasize the differences between the serial and parallel algorithms, the algorithms for preparing $(\ell + 1)$-cliques are presented separately in Algorithms 3.1 and 3.3.

3.1 The successive sparse SDP relaxation method starting from an $(\ell + 1)$ clique

Algorithm 3.1 (Finding an $(\ell + 1)$ clique)

Output : an $(\ell + 1)$-clique C and its location $(\widehat{\boldsymbol{a}}_r : C)$ for Algorithm 3.2.

Input : $N_*,\ E_*,\ A_*,\ (a^*_r : A_*),\ (d_{pq} : E_*)$.

Step 1: Create an $(\ell + 1)$-clique C of $G(N_*, E_*)$.

Step 2: Compute $(\widehat{\boldsymbol{a}}_r = \boldsymbol{x}_r : C)$ that satisfies the distance equations $\|\boldsymbol{x}_q - \boldsymbol{x}_r\| = d_{qr}$ $(q,\ r \in C,\ q < r)$.

We need to impose an additional condition on the choice of C to ensure that the convex hull of $(\widehat{\boldsymbol{a}}_r : C)$ forms an ℓ-dimensional simplex. This is discussed in Section 4.1.

Algorithm 3.2

Output : $(\boldsymbol{x}_p : N_*)$ if the algorithm succeeds or $\emptyset$ otherwise.

Input : $N_*,\ E_*,\ A_*,\ (\boldsymbol{a}^*_r : A_*),\ (d_{pq} : E_*),\ C,\ (\widehat{\boldsymbol{a}}_r : C)$.

Step 1: Let $A_0 = C$. Initialize the set $A_{0*} = \emptyset$ of nodes whose locations will be fixed permanently. (When the EDGP is anchor-free or $A_* = \emptyset$, Step 7 is never carried out; hence $A_{t*} = \emptyset$ $(t = 0, 1, \dots)$ throughout the iteration. Additional explanation for cases with $A_* \neq \emptyset$ is given following the description of this Algorithm.) Set the iteration counter $t = 0$.

Step 2: Choose a neighborhood N_t of A_t (see Section 4.2 for details). If $A_{t*} \neq \emptyset$, (which indicates that Step 7 has been carried out once), then let $\widehat{\boldsymbol{a}}_q = \boldsymbol{a}_r^*$ $(q \in N_t)$, $A_{t*} = N_t \bigcap A_*$ and $A_t = A_t \bigcup A_{t*}$. Otherwise let $A_{t*} = \emptyset$.

Step 3: Apply SFSDP to $\mathrm{P}(A_t, N_t)$ and roughly compute locations $(\boldsymbol{x}_p : N_t \backslash A_t)$ (see Section 4.3 for details).

Step 4: Refine the computed locations $(\boldsymbol{x}_p : N_t \backslash A_t)$ and $(\widehat{\boldsymbol{a}}_r : A_t \backslash A_{t*})$ by applying the gradient method to a problem of minimizing errors in the distance equations for those locations (see Section 4.4 for details).

Step 5: Based on the information from Steps 3 and 4, expand A_t to A_{t+1}. (see Section 4.5 for details). If it is successful, continue to Step 6. Otherwise, go to Step 8.

Step 6: Let $t = t + 1$. If $A_{t*} = \emptyset$ and $\#\left(A_t \bigcap A_*\right) \geq \ell + 1$, continue to Step 7. Otherwise, go to Step 2.

Step 7: Let $A_{t*} = A_t \bigcap A_*$. Perform a nonsingular affine transformation $\boldsymbol{T} : \mathbb{R}^\ell \to \mathbb{R}^\ell$ that minimizes

$$\sum_{p \in A_t \bigcap A_*} \left\| \boldsymbol{T}(\widehat{\boldsymbol{a}}_p) - \boldsymbol{a}_p^* \right\|^2.$$

Let $\boldsymbol{x}_p = \boldsymbol{T}(\boldsymbol{x}_p)$ $(p \in N_t)$. Go to Step 2.

Step 8: If N_t is a proper subset of N_*, then stop (unsuccessful termination). If $N_t = N_*$, then there is no need to update A_t. Terminate the iteration and output $(\boldsymbol{x}_p : N_*)$.

Assume that $A_* \neq \emptyset$. Before Step 7 is carried out, the locations $(\boldsymbol{x}_p : N_t)$ of the nodes in the neighborhood N_t are computed independently from the known locations $(a_r^* : A_*)$ of the permanent anchors. When $\#(A_t \bigcap A_*) \geq \ell + 1$ holds or the temporary anchor set A_t whose locations $(\hat{\boldsymbol{a}}_r : r \in A_t)$ have been computed includes at least $\ell + 1$ permanent anchors from A_*, a nonsingular affine transformation $\boldsymbol{T} : \mathbb{R}^\ell \to \mathbb{R}^\ell$ is applied to $(\boldsymbol{x}_p : N_t)$ at Step 7 so that $(\boldsymbol{T}(\hat{\boldsymbol{a}}_r) : A_t \bigcap A_*)$ can agree with $(\boldsymbol{a}_r^* : A_t \bigcap A_*)$ as much as possible. Then the location information $\boldsymbol{a}_r^*$ is used at Step 2 right after N_t includes a permanent anchor $r \in A_*$.

Figure 2 illustrates the expansions of the set A_t of temporary anchors and the set N_t of neighborhood nodes when Algorithm 3.2 is applied to a 3-dimensional EDGP problem 1YGP, which has 13488 sensors and no anchors. More information on this problem is given in Section 5. The initial sizes of A_t and N_t are $\ell + 1 = 4$ and 252, respectively. Notice that the sizes of these sets expand slowly in earlier iterations. This is because

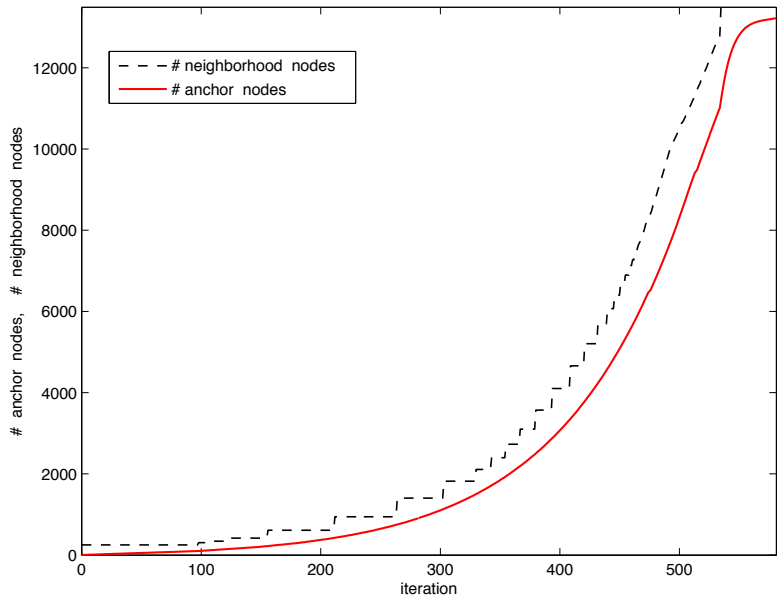

Figure 2:

fixing new nodes to expand A_t to A_{t+1} in Step 5 needs to be carried out very carefully when the number of temporarily fixed anchors is small. For example, if some nodes are mistakenly fixed far from their true locations in the earlier iterations, the misplaced locations affect the subsequent procedure of deciding temporary anchors. To minimize such misplacement, A_t is expanded by at most one node until the size of A_t reaches 100.

In the later iterations of Algorithm 3.2, however, a different strategy can be employed. If we have many nodes assigned as temporary anchors near the true locations, it is easy at Step 4 to correct the nodes that have been misplaced at Step 3. Therefore, an aggressive strategy can be applied to expand A_t to A_{t+1}. We observe in Figure 2 that the size of A_t increases rapidly after t passes 100.

The efficiency of Algorithm 3.2 is achieved by small sizes and structured sparsity of subproblems. That is, small subproblems with structured sparsity result in the SDP problems whose coefficient matrices are small and have the structured sparsity, and Schur complement matrix with the structured sparsity. At $t = 0$, SFSDP is applied to $\mathrm{P}(N_0, A_0)$ with $\#N_0 = 252$ and $\#A_0 = 4$. If the resulting SDP problem is written in the equality input format for SeDuMi, the size of the input matrix $\boldsymbol{A}$ is 1083×3965 with only 4112 nonzero elements. Moreover, the Schur complement matrix of size 1083×1083 is very sparse. The Schur complement matrix is a coefficient matrix of the linear system to be solved at each iteration of the prima-dual interior-point method. The figure on the left in Figure 3 illustrates a sparse Cholesky factorization of the Schur complement matrix computed by the Matlab functions symamd (a symmetric minimum degree ordering) and chol.

Although the size of the sparse SDP relaxation problem at Step 3 gradually increases as A_t and N_t expand, the sparsity of its coefficient matrix and the sparsity of its Schur

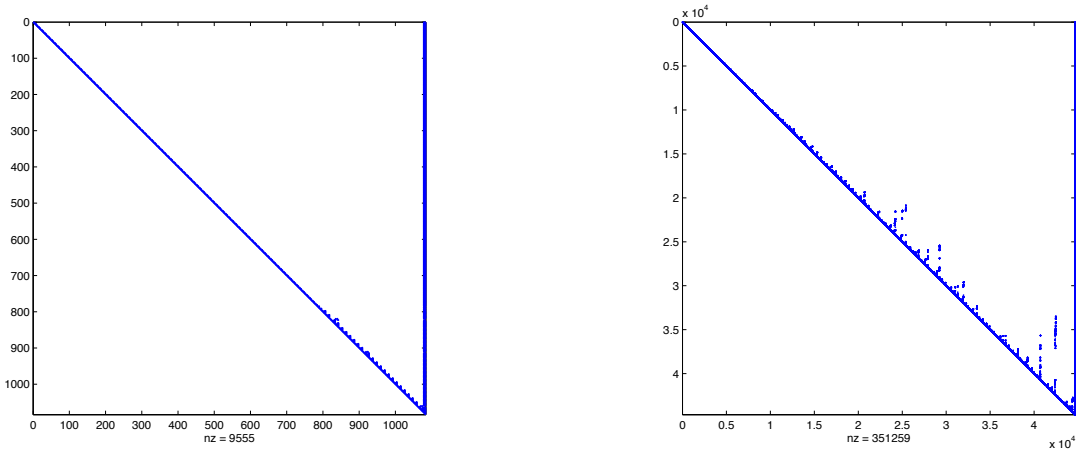

Figure 3: Sparse Cholesky factorization of the Schur complement matrix at $t = 0$ (left) and $t = 500$ (right)

complement matrix are maintained. For example, at $t = 500$, we have

$$\begin{aligned} &\#A_{500} = 8304,\ \#N_{500} = 10496, \\ &\text{the number of nonzeros of the } 44670 \times 120717 \text{ coefficient matrix } \boldsymbol{A} = 218557, \\ &\text{the number of nonzeros of the } 44670 \times 44670 \text{ Schur complement matrix } = 649534. \end{aligned} \tag{3}$$

The figure on the right in Figure 3 illustrates a sparse Cholesky factorization of the Schur complement matrix.

The sparse property mentioned above is mainly due to the construction of the problem $\mathrm{P}(N_t, A_t)$. Consider the underlying network with the node set N_t and the edge set $E(N_t, A_t)$. When almost 80% nodes of A_t are fixed as temporary anchors, they usually represent a core part of the network. Based on this observation, we categorize the edges $E(N_t, A_t)$ into two types: the first type is the edges from those core nodes to nodes residing near the boundary and the second type is the edges between nodes near the boundary. Such a graph is very sparse even when both sizes of N_t and A_t grow, since any two nodes in a core part are not connected by an edge. The sparsity of the Schur complement matrix depends mainly on the second type of edges. More precisely, the subgraph of the node set $N_t \backslash A_t$ and the edge set $\{(p, q) \in E_* : p,\ q \in N_t \backslash A_t\}$ has a sparse chordal extension, which determines the sparsity of a sparse Cholesky factorization of the Schur complement matrix. See Section 3.3 of [16].

3.2 Parallel implementation of the successive sparse SDP relaxation method

The accuracy of locations $(\boldsymbol{x}_p : N_*)$ by Algorithm 3.2 heavily depends on the choice of $(\ell + 1)$-clique C for the initial anchor set A_0 and the nodes used for updating A_t to A_{t+1} at Step 5. We discuss in Section 4.1 how an $(\ell + 1)$-clique is chosen to avoid failure at earlier iterations, and in Section 4.5 how new nodes are fixed as temporary anchors to update A_t to A_{t+1}.

Algorithm 3.2 sometimes fails to output $(\boldsymbol{x}_p : N_*)$ and stops at Step 8 before expanding the neighborhood N_t to the entire node set N_*. Or, even if the neighborhood N_t reaches the entire node set N_* and the algorithm outputs $(\boldsymbol{x}_p : N_*)$ at Step 8, the obtained locations of the nodes in $(\boldsymbol{x}_p : N_*)$ may not be within the desired accuracy. This can not be avoided even with a careful choice of initial $(\ell+1)$-clique. To overcome these difficulties, we use a collection Γ of pairs of $(\ell+1)$-cliques C and their locations $\widehat{\boldsymbol{a}}_r$ $(r \in C)$, denoted simply by $(\widehat{\boldsymbol{a}}_r : C)$. We apply Algorithm 3.2 multiple times to a given EDGP with Γ using different pairs of $(\ell+1)$-cliques and the corresponding locations. Then, a collection $\left\{(\boldsymbol{x}_p^v : N_*) : v = 1, 2, \ldots, u\right\}$ of estimated locations of the nodes are obtained.

The collection $\left\{(\boldsymbol{x}_p^v : N_*) : v = 1, 2, \ldots, u\right\}$ needs to be compounded to have a more accurate locations of the nodes without knowing their true locations. We will discuss this issue in Section 4.6. For simplicity, we denote this collection by $\Xi = \left\{(\boldsymbol{x}_p^v : N_*) : v = 1, 2, \ldots, u\right\}$. The notation δb denotes the unit size of collection of estimated locations to which the compound procedure is applied for the first time. If the compound procedure applied to some unit or multiple units of collections of estimated locations fails, then another unit of estimated locations is computed, using modified parameters to control an update from A_t to A_{t+1} from different $(\ell+1)$ cliques. Again, the compound procedure is applied to the union of the collection and the computed units of estimated locations, as described in Steps 8 and 9 of Algorithm 3.4.

To execute Algorithm 3.2 in parallel, Algorithm 3.1 is replaced by the following algorithm.

Algorithm 3.3

Output : Γ (a collection of pairs of an $(\ell+1)$-clique C and their locations $(\widehat{\boldsymbol{a}}_r : C)$).

Input : $N_*,\ E_*,\ A_*,\ (\boldsymbol{a}_r^* : A_*),\ (d_{pq} : E_*)$.

By applying Algorithm 3.1, create a collection Γ of pairs of an $(\ell+1)$-clique C and their locations $(\widehat{\boldsymbol{a}}_r : C)$.

Algorithm 3.4 (passSDP, parallel implementation of the successive sparse SDP relaxation method)

Output : $(\boldsymbol{x}_p : N_*)$ if the algorithm has succeeded, or $\emptyset$ otherwise.

Input : $N_*,\ E_*,\ A_*,\ (\boldsymbol{a}_r^* : A_*),\ (d_{pq} : E_*),\ \Gamma.\ u$: the number of available resources for parallel computing

Let Output $= \emptyset$, $\Xi = \emptyset$ and choose a unit size $\delta b \geq 4$. Let $b = \delta b$. While Output $= \emptyset$ and $\Gamma \neq \emptyset$, perform Steps 0 – 9.

Step 0: If $\#\Xi < b$ and $\Gamma \neq \emptyset$, then choose a $(\widehat{\boldsymbol{a}}_r^1 : C), (\widehat{\boldsymbol{a}}_r^2 : C), \ldots, (\widehat{\boldsymbol{a}}_r^u : C)$ from Γ, let $\Gamma = \Gamma \backslash \{(\widehat{\boldsymbol{a}}_r^v : C) : 1 \leq v \leq u\}$, and continue to Step 1. Otherwise go to Step 9.

Steps 1 – 7: Execute Steps 1 – 7 of Algorithm 3.2 on the parallel resources.

Step 8: If $N_t = N_*$, then let $\Xi = \Xi \bigcup \{(\boldsymbol{x}_p^v : N_*) : 1 \leq v \leq u\}$ and go to Step 0. Otherwise (if N_t is a proper subset of N_* and any update on N_t is impossible), choose an $(\widehat{\boldsymbol{a}}_r : C)$ from from Γ, let $\Gamma = \Gamma \backslash (\widehat{\boldsymbol{a}}_r : C)$, and go to Step 1.

Step 9: Compound the collection Ξ of estimated locations of the nodes into an accurate estimate $(\boldsymbol{x}_p : N_*)$ (see Section 4.6 for more details). If it is successful, let Output $= (\boldsymbol{x}_p : N_*)$ and terminate. If $\Gamma = \emptyset$, then let Output $= \emptyset$ and terminate. Otherwise let $b = b + \delta b$, modify some parameters for updating A_t to A_{t+1} at Step 5, and go to Step 0.

Note that Steps 1-8 can be carried out in parallel.

4 Technical Details

4.1 Initialization

We first need to choose an $(\ell+1)$-clique C that geometrically forms an ℓ-dimensional simplex. For this, we introduce the concept of admissible cliques.

Let C consist of $\ell+1$ nodes $i_1, i_2, \ldots, i_{\ell+1} \in N_*$ satisfying the following conditions (a) and (b). For simplicity, we let $C = \{1, 2, \ldots, \ell+1\}$.

Condition 4.1

(a) $(i,j) \in E_*$ $(1 \leq i < j \leq \ell+1)$.

(b) The system of quadratic equations

$$d_{ij}^2 - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = 0 \ (1 \leq i < j \leq \ell+1) \tag{4}$$

has solutions $\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots,\ \boldsymbol{x}_{\ell+1}$ that form an ℓ-dimensional simplex; $\boldsymbol{x}_2 - \boldsymbol{x}_1,\ \ldots,$ $\boldsymbol{x}_{\ell+1} - \boldsymbol{x}_1$ are linearly independent.

We call an $(\ell+1)$-clique admissible if it satisfies (a) and (b). The system of quadratic equations (4) is an underdetermined system of $\ell(\ell+1)/2$ equations in $\ell(\ell+1)$ real variables. If there exists a tuple solution $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{\ell+1})$ of (4) that satisfies the conditions (a) and (b), the following constraints are added to determine the tuple solution uniquely.

$$[\boldsymbol{x}_p]_i = 0 \ (1 \leq p \leq i \leq \ell),\ [\boldsymbol{x}_{p+1}]_p > 0 \ (p = 1, \ldots, \ell),$$

where $[\boldsymbol{x}_p]_i$ denotes the ith element of $\boldsymbol{x}_p$.

Assume that there exist many admissible cliques. One clique needs to be chosen for the initial set of temporary anchors A_0 among the admissible cliques. It is desirable to choose a clique from dense parts of the graph $G(N_*, E_*)$, considering the connectivity of the clique to other nodes of the graph. For each admissible clique C and each positive integer m, we consider the two numbers:

$$\begin{aligned} \alpha(C,m) &= \#U^m(C), \\ \beta(C,m) &= \text{the number of admissible cliques contained in } U^m(C). \end{aligned}$$

where m denotes a positive integer. If an admissible clique C results in larger values of $\alpha(C,m)$ and $\beta(C,m)$ than those of another admissible clique C', the admissible clique

C is regarded as a better candidate for the initial set of temporary anchors A_0 at Step 1 of Algorithm 3.2. We note that $\alpha(C,m)$ is simpler to compute than $\beta(C,m)$, taking less computational time. However, preliminary numerical experiments suggested that $\beta(C,m)$ works more effective in providing accurate solutions. In Section 4.6, another technique combined with $\beta(C,m)$ is discussed to improve Algorithm 3.4 (passSDP).

4.2 Neighborhood

The size of a neighborhood affects the overall performance. Suppose that $\emptyset \neq A_t \subset N_*$ at the beginning of Step 2 of an iteration $t \geq 0$. As the size of the neighborhood N_t increases, more candidates are available in N_t for updating A_t to A_{t+1} at Step 5. Thus, taking a large N_t seems to be a better choice for the overall performance, especially when $\#A_t$ is small at earlier iterations. However, solving the sparse SDP relaxation problem of $\mathrm{P}(N_t, A_t)$ takes longer computational time as $\#N_t$ increases. Considering these opposite effects by the choice of a neighborhood, the following rules are employed with the parameters m, κ_0, κ_1, and κ_2 for a neighborhood N_t.

$$\begin{aligned} N_t &= U^m(A_t) \text{ such that } \#U^{m-1}(A_t) < \#A_t + \kappa_0 \leq \#U^m(A_t) \text{ if } \#A_t \leq \kappa_1, \\ N_t &= \begin{cases} N_{t-1} & \text{if } \#N_{t-1} - \#A_t > \max\{\kappa_2, \kappa_3 \#A_t\} \\ U(A_t) & \text{otherwise,} \end{cases} \quad \text{if } \#A_t > \kappa_1 \end{aligned} \tag{5}$$

In the numerical results in Section 5, $\kappa_0 = \kappa_1 = 100,\ \kappa_2 = 200$ and $\kappa_3 = 0.1$ were used.

4.3 SDP relaxation of $\mathrm{P}(N, A)$

For simplicity, we assume that $N = \{1, 2, \ldots, n, n+1, \ldots, n_a\}$, $A = \{n+1, n+2, \ldots, n_a\}$ and set $S = N \backslash A$. Let

$$\begin{aligned} E_s &= \{(p,q) \in E_* : p \in S,\ q \in S\},\ E_a = \{(p,r) \in E_* : p \in S,\ r \in A\}, \\ E &= E(N,A) = E_s \bigcup E_a. \end{aligned}$$

Then, we can rewrite $\mathrm{P}(N,A) = \mathrm{P}(S \bigcup A, A)$ as

$$\begin{aligned} \text{minimize} \quad & \sum_{(p,q) \in E_s} (\xi^+_{pq} + \xi^-_{pq}) + \sum_{(p,r) \in E_a} (\xi^+_{pr} + \xi^-_{pr}) \\ \text{subject to} \quad & \boldsymbol{x}_p^T \boldsymbol{x}_p - 2\boldsymbol{x}_p^T \boldsymbol{x}_q + \boldsymbol{x}_q^T \boldsymbol{x}_q + \xi^+_{pq} - \xi^-_{pq} = d_{pq}\ ((p,q) \in E_s), \\ & \boldsymbol{x}_p^T \boldsymbol{x}_p - 2\widehat{\boldsymbol{a}}_r \boldsymbol{x}_p + \|\widehat{\boldsymbol{a}}_r\|^2 + \xi^+_{pr} - \xi^-_{pr} = d_{pr}\ ((p,r) \in E_a), \\ & \xi^+_{pq} \geq 0,\ \xi^-_{pq} \geq 0\ ((p,q) \in E_s),\ \xi^+_{pr} \geq 0,\ \xi^-_{pr} \geq 0\ ((p,r) \in E_a). \end{aligned}$$

Now, we introduce a matrix variable

$$\boldsymbol{Y} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n), \tag{6}$$

so that $Y_{pq} = \boldsymbol{x}_p^T \boldsymbol{x}_q$ $((p,q) \in E_s)$. Relaxing the identity (6) into the positive semidefinite condition

$$\begin{pmatrix} \boldsymbol{Y} & (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T \\ (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) & \boldsymbol{I} \end{pmatrix} \succeq \boldsymbol{O}, \tag{7}$$

which is equivalent to $\boldsymbol{Y} - (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \succeq \boldsymbol{O}$, we obtain FSDP, the full SDP relaxation [1] of P(N, A).

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{(p,q)\in E_s} (\xi^+_{pq} + \xi^-_{pq}) + \sum_{(p,r)\in E_a} (\xi^+_{pr} + \xi^-_{pr}) \\ \text{subject to} & Y_{pp} - 2Y_{pq} + Y_{qq} + \xi^+_{pq} - \xi^-_{pq} = d_{pq}\ ((p,q) \in E_s), \\ & Y_{pp} - 2\widehat{\boldsymbol{a}}_r^T \boldsymbol{x}_p + \|\widehat{\boldsymbol{a}}_r\|^2 + \xi^+_{pr} - \xi^-_{pr} = d_{pr}\ ((p,r) \in E_a), \\ & \xi^+_{pq} \geq 0,\ \xi^-_{pq} \geq 0\ ((p,q) \in E_s),\ \xi^+_{pr} \geq 0,\ \xi^-_{pr} \geq 0\ ((p,r) \in E_a), \\ & \text{the positive semidefinite condition (7).} \end{array} \tag{8}$$

Let $\boldsymbol{e}$ denote the $2(\#E_s + \#E_a)$-dimensional column vector of ones, $\boldsymbol{\xi}$ the $2(\#E_s + \#E_a)$-dimensional variable column vector of elements ξ^+_{pq}, ξ^-_{pq} $((p,q) \in E_s)$, ξ^+_{pr}, ξ^-_{pr} $((p,r) \in E_a)$, and $\begin{pmatrix} \boldsymbol{Y} & \boldsymbol{X}^T \\ \boldsymbol{X} & \boldsymbol{U} \end{pmatrix}$ the variable matrix consisting of an $n \times n$ symmetric matrix $\boldsymbol{Y} \in \mathbb{S}^n$, an $\ell \times n$ matrix $\boldsymbol{X} = (\boldsymbol{x}_1, x_2, \ldots, \boldsymbol{x}_n)$ and an $\ell \times \ell$ symmetric matrix $\boldsymbol{U}$. Then, the SDP problem (8) can be reformulated as

$$\begin{array}{ll} \text{minimize} & \boldsymbol{e}^T\boldsymbol{\xi} \\ \text{subject to} & \boldsymbol{f}_k^T\boldsymbol{\xi} + \boldsymbol{J}_k \bullet \begin{pmatrix} \boldsymbol{Y} & \boldsymbol{X}^T \\ \boldsymbol{X} & \boldsymbol{U} \end{pmatrix} = b_k\ (k = 1, 2, \ldots, m), \\ & \boldsymbol{\xi} \geq \boldsymbol{0},\ \begin{pmatrix} \boldsymbol{Y} & \boldsymbol{X}^T \\ \boldsymbol{X} & \boldsymbol{U} \end{pmatrix} \succeq \boldsymbol{O}, \end{array} \tag{9}$$

for some $2(\#E_s+\#E_a)$-dimensional column vector $\boldsymbol{f}_k$, $(n+\ell)\times(n+\ell)$ symmetric matrix $\boldsymbol{J}_k = \begin{pmatrix} \boldsymbol{F}_k & \boldsymbol{G}_k^T \\ \boldsymbol{G}_k & \boldsymbol{H}_k \end{pmatrix}$ and real number b_k $(k = 1, 2, \ldots, m)$. We note that the inequalities fixing $\boldsymbol{U}$ to the $\ell \times \ell$ identity matrix are included in the equality constraints.

The SDP (9) (equivalently (8)) satisfies a certain structural sparsity, characterized by a (symbolic) sparse Cholesky factorization of the (aggregated) sparsity pattern matrix over the coefficient matrices $\boldsymbol{J}_k$ $(k = 1, 2, \ldots, m)$. If we regard the sparsity pattern matrix as the adjacency matrix of a graph, then the sparse Cholesky factorization corresponds to a sparse Chordal extension of the graph. SFSDP [17, 18], which is used in Step 3 of Algorithms 3.2 and 3.4, effectively implements the technique [11, 16, 28] of exploiting such a structured sparsity. It converts the SDP (9) to a sparse SDP having multiple smaller-sized variable matrices so that it can be solved more efficiently. See Section 3.3 of [17] for details. We have observed numerically in the last three paragraphs of Section 3.1 that the converted SDP has a very sparse coefficient matrix $\boldsymbol{A}$ in the SeDuMi input format, and that the Schur complement matrix has a sparse Cholesky factorization. See also Figures 3.

In the remaining of this subsection, we discuss how the successive sparse SDP relaxation method enhances the structured sparsity characterized by a sparse Cholesky factorization of the sparsity pattern matrix. Let $\boldsymbol{J}_* = \begin{pmatrix} \boldsymbol{F}_* & \boldsymbol{G}_*^T \\ \boldsymbol{G}_* & \boldsymbol{H}_* \end{pmatrix}$ denote the sparsity pattern matrix over the coefficient matrices $\boldsymbol{J}_k$ $(k = 1, 2, \ldots, m)$; the (p, q)th element of $\boldsymbol{J}_*$ takes the value 1 if the corresponding element of any of the coefficient matrices J_k $(k = 1, 2, \ldots, m)$ is nonzero and the value 0 otherwise. Suppose that $1 \leq p \leq q \leq n$.

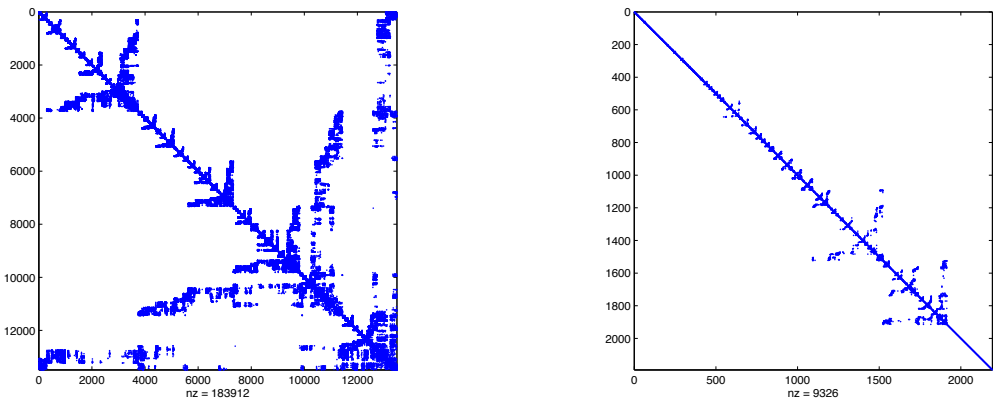

Figure 4: Sparsity pattern of the adjacency matrices of $G(N_*, E_*)$ on the left and $G(S, E_s)$ with $S = N_t \backslash A_t$ at $t = 500$ on the right.

Compare the equalities in (8) with the equality constraints of the SDP (9). We can verify that the (p, p)th element of $\boldsymbol{F}_*$ is 1 if and only if $(p, r) \in E_a$ for some $r \in A$ $(1 \leq p \leq n)$, and that (p, q)th element of $\boldsymbol{F}_*$ is 1 if and only if $(p, q) \in E_s$ $(1 \leq p < q \leq n)$. Since $\boldsymbol{F}_*$ is symmetric, its (q, p)th element is equivalent to its (p, q)th element. Therefore, the matrix $\boldsymbol{F}_*$ coincides with the (node-to-node) adjacency matrix of the graph $G(S, E_s)$ except for the diagonal elements.

At earlier iteration t (say $t \leq \kappa_1 = 100$), the set of temporary anchors A_t as well as its neighborhood N_t are small. Thus, the size of the problem $\mathrm{P}(N_t, A_t)$ is small. In this situation, exploiting the sparsity for efficiency is not an important issue. As the iteration t proceeds, A_t and N_t gradually expand, and the size of $\mathrm{P}(N_t, A_t)$ becomes larger. If two nodes in the entire graph $G(N_*, E_*)$ are connected by an edge of E_*, they should be located within the distance ρ. Hence, $G(N_*, E_*)$ itself as well as its subgraph $G(N, E)$ are expected to satisfy a structural sparsity if ρ is not too large. Recall that when $\#A_t$ is larger than $\kappa_1 = 100$, one-step neighborhood is chosen for N_t as in (5). Roughly speaking, the nodes in $S = N_t \backslash A_t$ are located on the boundary of the graph $G(N_t, E(N_t, A_t))$ and the width of the nodes on the boundary is usually thin. The interior of the graph $G(N_t, E(N_t, A_t))$ is occupied by the nodes in A_t. See Figure 1. Therefore, $G(S, E_s)$ whose adjacent matrix is $\boldsymbol{F}_*$ is more likely to satisfy a structural sparsity than $G(N_*, E_*)$.

Figure 4 illustrates the sparsity patterns of the adjacency matrices of $G(N_*, E_*)$ on the left and $G(S, E_s)$ on the right for the same numerical example used in the last three paragraphs of Section 3.1 (the 3-dimensional EDGP 1YGP with $\#N_* = 13488$ and $t = 500$). We observe that not only the size $\#S = 2192$ of S is much smaller than $\#N_* = 13488$ but also $G(S, E_s)$ is much sparser than $G(N_*, E_*)$.

The sparsity of $(\boldsymbol{G}_* \ \boldsymbol{H}_*)$ (or $(\boldsymbol{G}_* \ \boldsymbol{H}_*)^T$) should also be considered. Since the equalities fixing $\boldsymbol{U}$ to the $\ell \times \ell$ identity matrix are included in the equality constraints in (9), $\boldsymbol{H}_*$ becomes the $\ell \times \ell$ matrix of 1's. For every $p \in S$, the pth column of $\boldsymbol{G}_*$ becomes the ℓ-dimensional vector of 1's if $(p, r) \in E_a$ for some $r \in A$, and the ℓ-dimensional vector of 0's otherwise. Thus, $(\boldsymbol{G}_* \ \boldsymbol{H}_*)$ can be fully dense, but its number of rows is always ℓ

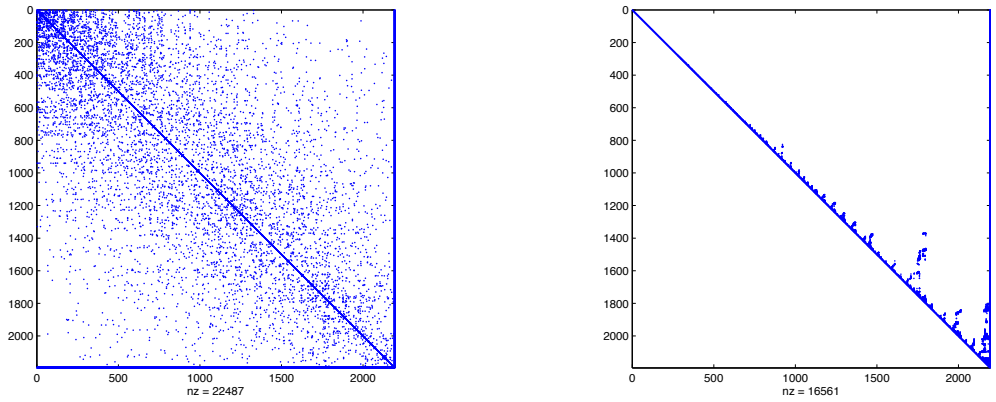

Figure 5: Sparsity pattern of $\boldsymbol{J}_*$ on the left and its sparse Cholesky factorization with symamd on the right.

throughout the iterations, and the rows are located at the lower boundary in the entire sparsity pattern matrix $\boldsymbol{J}_*$. As a result, the entire structural sparsity is not much affected by the sparsity of $(\boldsymbol{G}_* \ \boldsymbol{H}_*)$.

The figure on the left in Figure 5 illustrates the sparsity pattern matrix J_* and the figure on the right its sparse Cholesky factorization. We observe the sparse Cholesky factorization of $\boldsymbol{J}_*$ is as sparse as the adjacency matrix of $G(S, E_s)$ shown on the right figure of Figure 4.

4.4 Gradient method

Algorithm 3.2 refines an approximate solution with a gradient method. We describe how the solution is refined based on the paper [23]. Assume that a rough approximate solution $(\boldsymbol{x}_p^{\rm sdp} : p \in N_t)$ has been computed at Step 3 of Algorithm 3.2. When the SDP relaxation is applied to $\mathrm{P}(N_t, A_t)$, every node r in the set of temporary anchors has been fixed as $\widehat{\boldsymbol{a}}_r$ $(r \in A_t)$. However, there is no guarantee that all the nodes are correctly located, rather, some may still be placed far from their true locations. Those nodes need to be relocated.

Since each term $\left|d_{pq}^2 - \|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2\right|$ of the objective function of $\mathrm{P}(N_t, A_t)$ is not differentiable at $(\boldsymbol{x}_p, \boldsymbol{x}_q)$ where the distance equation $d_{pq}^2 - \|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2 = 0$ is satisfied, a different objective function is considered. More precisely, instead of the minimization problem $\mathrm{P}(N_t, A_t)$, consider

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{(p,q)\in E(N_t,A_{t*})} (d_{pq} - \|\boldsymbol{x}_p - \boldsymbol{x}_q\|)^2 \\ \text{subject to} & \boldsymbol{x}_{p^*} = \widehat{\boldsymbol{a}}_{p^*},\ \boldsymbol{x}_r = \boldsymbol{a}_r^* \ (r \in A_{t*}) \ \text{ if } A_{t*} \neq \emptyset, \end{array}$$

where p^* is a node from A_0 that is fixed throughout all the iteration. The gradient method developed by Toh is applied with the initial point $(\boldsymbol{x}_p^{\rm sdp} : p \in N_t)$ to the problem above for an approximate solution $(\boldsymbol{x}_p^{\rm grad} : p \in N_t)$.

4.5 Fixing new nodes

Let $p \in N_t \backslash A_{t*}$. We use two quantities to decide whether p can belong to A_{t+1}. The first quantity is a deviation from the error in the distance equations evaluated at $\boldsymbol{x}_p^{\rm grad}$:

$$e_{pt} \;=\; \sum_{q\,:\,q \in N_t,\,(q,p) \in E(N_t, A_t)} \left(1 - \left|d_{pq} - \|\boldsymbol{x}_p - \boldsymbol{x}_q\|\right| / \theta_e\right),$$

where $\theta_e > 0$ is a threshold parameter. For the numerical experiments in Section 5, $\theta_e = 0.1\rho$ was used. For the value of e_{pt}, we consider how many edges $(q,p) \in E(N_t, A_t)$ incident to the node $p \in N_t \backslash A_{t*}$ yield the error $|d_{pq} - \|\boldsymbol{x}_p - \boldsymbol{x}_q\||$ smaller than the threshold value $\theta_e > 0$. That is, the value becomes larger as the number of such edges increases. Conversely it becomes smaller (and even negative) as the error $|d_{pq} - \|\boldsymbol{x}_p - \boldsymbol{x}_q\||$ exceeds θ_e for more edges $(q,p) \in E(N_t, A_t)$ incident to the node p. The second quantity is the difference between the location $\boldsymbol{x}_p^{\rm sdp}$ and its refinement $\boldsymbol{x}_p^{\rm grad}$ after the gradient method. If this difference is large, the location $\boldsymbol{x}_p^{\rm sdp}$ may not be reliable. Let

$$f_{pt} \;=\; 1 - \left\|\boldsymbol{x}_p^{\rm grad} - \boldsymbol{x}_p^{\rm sdp}\right\| / \theta_d.$$

The parameter $\theta_d > 0$ serves as a threshold value for deciding whether the location $\boldsymbol{x}_p^{\rm sdp}$ is reliable. We used $\theta_d = 0.4$ in the numerical experiments. Since the values e_{ps} and f_{ps} $(s \le t-1)$ also need to be considered, we define

$$f_{ps} \;=\; e_{ps} = 0 \text{ if } p \notin N_s \;(s = 0, 1, \ldots, t-1).$$

As a measure for the reliability of $\boldsymbol{x}_p^{\rm sdp}$, let

$$h_{pt} \;=\; \sum_{s=0}^{t} \gamma^{t-s} \max\left\{0,\; (e_{ps} + \omega f_{ps})\right\} \;\; (p \in N_t \backslash A_{t*}),$$

where $\omega > 0$ serves as the weight parameter to combine e_{ps} and f_{ps}, and $\gamma \in (0,1)$ as a decaying parameter. The values of $\omega = 2$ and $\gamma = 0.3$ were used in the numerical experiments. Using h_{pt} $(p \in N_t)$ and the threshold value $\theta > 0$, we define the set of candidates $\widetilde{A}_t = \{p \in N_t \backslash A_{t*} : h_{pt} \ge \theta\}$ for the nodes of A_{t+1}. For instance, $\theta = 5$ was used in the numerical experiments.

We notice that one of the following three cases can occur:

(i) $A_t \backslash A_{t*}$ is a proper subset of $\widetilde{A}_t$.

(ii) $A_t \backslash A_{t*}$ is not a proper subset of $\widetilde{A}_t$ but $\#\widetilde{A}_t \ge \#\left(A_t \backslash A_{t*}\right) + 1$.

(iii) $\#\widetilde{A}_t \le \#\left(A_t \backslash A_{t*}\right)$.

In the case (i), A_{t+1} is the union of A_t and some nodes from $\widetilde{A}_t$ with large values h_{pt}. In the case (ii), the nodes contained in $\left(A_t \backslash A_{t*}\right) \backslash \widetilde{A}_t$ have been wrongly fixed. These

wrongly fixed nodes are replaced by nodes from $\widetilde{A}_t \backslash A_t$ with larger value h_{pt}, so that the inequality

$$\#\left(A_{t+1}\backslash A_{t*}\right) \geq \#\left(A_t\backslash A_{t*}\right) + 1. \tag{10}$$

holds.

If (iii) is true, not enough nodes in $\widetilde{A}_t\backslash A_t$ are available to replace $\left(A_t\backslash A_{t*}\right)\backslash\widetilde{A}_t$ and satisfy the inequality (10). In this case, updating A_t to A_{t+1} fails and Algorithm 3.2 stops or Algorithm 3.4 is re-executed from Step 1 with a newly chosen $(\widehat{\boldsymbol{a}}_r : C)$ from Γ.

In the case of (i) or (ii), it is important to decide how many new nodes are fixed to update A_t to A_{t+1} or how $\#\left(A_t\backslash A_{t*}\right)$ is increased. For successful implementation, the fixed nodes should be located near their unknown true locations. In fact, the increase of $\#\left(A_t\backslash A_{t*}\right)$ is handled by

$$\#\left(A_{t+1}\backslash A_{t*}\right) = \#\left(A_t\backslash A_{t*}\right) + 1$$

until $\#A_t$ gets 100. However, if this strategy is continued until the end of the iterations, it is not efficient to solve large EDGPs. For $\#A_t > 100$, we employ the following strategy to increase $\#\left(A_t\backslash A_{t*}\right)$

$$\#\left(A_{t+1}\backslash A_{t*}\right) - \#\left(A_t\backslash A_{t*}\right) = \max\left\{1,\ \min\left\{\alpha\ \#A_t, \beta\ \#\widetilde{A}_t\right\}\right\},$$

where $\alpha > 0$ and $\beta > 0$ are the parameters to control the increase of $\#\left(A_t\backslash A_{t*}\right)$. In our numerical experiments, $\alpha \in [0.005, 0.02]$ and $\beta \in [0.02, 0.1]$ were chosen, depending on the sparsity of the graph $G(N_*, E_*)$, the noise level of $(d_{pq} : E_*)$, and which of cases (i) and (ii) occurred in previous and current iterations.

4.6 Compounding the collection of estimated locations into an accurate estimate

We explain Step 9 of Algorithm 3.4 in detail. Suppose that a collection

$$\Xi = \left\{(\boldsymbol{x}_p^\nu : N_*) : v = 1, 2, \ldots, u\right\}$$

of estimated locations of the nodes $p \in N_*$ has been computed. Since $(\boldsymbol{x}_p^v : N_*)$ are obtained from an $(\widehat{\boldsymbol{a}}_r^v : C_v) \in \Gamma$, they may be located independently from their true location. Obviously, it is difficult to directly compare different $(\boldsymbol{x}_p^v : N_*)$ and $(\boldsymbol{x}_p^w : N_*)$. For the comparison, we first perform a nonsingular affine transformation $\boldsymbol{T}_{vw} : \mathbb{R}^\ell \to \mathbb{R}^\ell$ that minimizes

$$\sum_{p\in N_*} \left\|\boldsymbol{T}_{vw}(\boldsymbol{x}_p^v) - \boldsymbol{x}_p^w\right\|^2.$$

If Step 7 has been already executed, we can take the identity transformation for $\boldsymbol{T}_{vw}$. Then, we compute the relative root mean square distance (RMSD) of $(\boldsymbol{x}_p^v : N_*)$ with respect to $(\boldsymbol{x}_p^v : N_*)$ by

$$\begin{aligned} R_{vw} \ &= \ \left(\frac{1}{\#(N_*\backslash A_*)} \sum_{p\in N_*\backslash A_*} \left\|\boldsymbol{T}_{vw}(\boldsymbol{x}_p^v) - \boldsymbol{x}_p^w\right\|^2\right)^{1/2} \\ &\quad (v = 1, 2, \ldots, u,\ w = 1, 2, \ldots, u), \end{aligned}$$

where $R_{vv} = 0$ $(v = 1, 2, \ldots, u)$ is assumed.

If the true location $(\boldsymbol{a}_p^* : p \in N_+)$ is known, the quality of $(\boldsymbol{x}_p^v : N_*)$ can be evaluated by computing an RMSD of $(\boldsymbol{x}_p^v : N_*)$ with respect to the true location $(\boldsymbol{a}_p^* : p \in N_+)$

$$R_{v*} = \left(\frac{1}{\#(N_* \backslash A_*)} \sum_{p \in N_* \backslash A_*} \left\| \boldsymbol{T}_{v*}(\boldsymbol{x}_p^v) - \boldsymbol{a}_p^* \right\|^2 \right)^{1/2} \quad (v = 1, 2, \ldots, u),$$

where $\boldsymbol{T}_{v*}$ denotes a nonsingular affine transformation on $\mathbb{R}^\ell$ that minimizes

$$\sum_{p \in N_*} \left\| \boldsymbol{T}_{v*}(x_p^v) - \boldsymbol{a}_p^* \right\|^2 \ (v = 1, 2, \ldots, u).$$

Although R_{v*} is not available in practice, a large value of R_{vw} for some v and w indicates that at least one of R_{v*} and R_{w*} is large. That is, at least one of the estimated locations $(\boldsymbol{x}_p^v : N_*)$ and $(\boldsymbol{x}_p^w : N_*)$ must be inaccurate.

We construct a hypothesis: if F is a subset of $\{1, 2, \ldots, u\}$ with $\#F \geq 2$ and if all values of R_{vw} $(v,\ w \in F)$ are small, then all values of R_{v*} $(v \in F)$ are expected to be small. Based on this hypothesis, we describe an algorithm for computing F.

Algorithm 4.2

Output : $F \subset \{1, 2, \ldots, u\}$.

Input : $\Xi = \left\{ (\boldsymbol{x}_p^\nu : N_*) : v = 1, 2, \ldots, u \right\}$.

Step 0 : Choose a lower bound θ_L and an upper bound θ_U for threshold values for R_{vw} $(v = 1, 2, \ldots, u,\ w = 1, 2, \ldots, u)$ such that $\theta_L < \theta_U$, and a lower bound $f_L \geq 2$ for the size of F. Let $\theta = \theta_L$ and $F = \emptyset$.

Step 1 : If $\theta > \theta_U$, then let Output $= F$ and terminate. Find the maximal $F \subset \{1, 2, \ldots, u\}$ satisfying $R_{vw} \leq \theta$ $(v, w \in F)$. If $\#F \leq 1$, then let $F = \emptyset$.

Step 2 : If $\#F \geq f_L$, then let Output $= F$ and terminate. Otherwise, let $\theta = 2 \times \theta$ and go to Step 1.

For the numerical results in Section 5, $\theta_L = 0.5$, $\theta_U = 1.0$ and $f_L = 3$ were used. These values depend on the radio range, the noise level and the sparsity of an instance of EDGP.

Suppose that the algorithm is successfully implemented, and outputs F. We discuss a method to compound estimated locations $\boldsymbol{x}_p^v$ $(v \in F)$ into a better estimate $\boldsymbol{x}_p$ for each $p \in N_* \backslash A_*$. Let $p \in N_* \backslash A_*$ be fixed. Then, $\#F$ points $\boldsymbol{x}_p^v$ $(v \in F)$ in $\mathbb{R}^\ell$ are available for estimating the true location of the node p. One simple method is to take the mean $\left(\sum_{v \in F} \boldsymbol{x}_p^v \right) / \#F)$. This method, however, is not very attractive in the sense that outliers should be excluded and a geometrically condensed part D of the set $\left\{ \boldsymbol{x}_p^v : v \in F \right\}$ should be extracted.

For this purpose, we employ a method proposed by Nagano, Kawahara and Aihara [27] for the size-constrained densest subset problem. Given a positive integer k and a graph $G(\overline{V}, \overline{E})$ with a node set $\overline{V} = \{1, 2, \ldots, n\}$, an edge set $\overline{E}$ and edge weights $w_{ij} > 0$ ($(i,j) \in \overline{E}$), the problem is to find a node subset with size k and maximum weight, *i.e.*, a subset D of $\overline{V}$ that maximizes $\sum_{(i,j)\in D\times D} w_{ij}$ over the node subsets with size k, where w_{ij} is counted as 0 if $(i,j) \notin \overline{E}$. The method utilizes a minimum norm basis $\boldsymbol{\xi} \in \mathbb{R}^n$ with $-\xi_i \geq 0$ ($i \in \overline{V}$) based on the concept of sub-modular function [9]. For each $i \in \overline{V}$, $-\xi_i$ may be regarded as a score for contribution to a node subset with maximum weight, which we want compute; particularly, the node i is isolated if $-\xi_i = 0$. It was shown in [27] that $D = \{i \in \overline{V} : \xi_i \leq \bar{\xi}\} = \{i \in \overline{V} : -\xi_i \geq -\bar{\xi}\}$ forms a node subset with size $\#D$ and maximum weight if $-\bar{\xi} \geq 0$ and $D \neq \emptyset$.

To apply the method by [27] for choosing a condensed part D_p of the set $\{\boldsymbol{x}_p^v : v \in F\}$ for each $p \in N_* \backslash A_*$, we construct a graph $\overline{G_p}$ of the node set F and the edge set $\overline{E}_p = \{(v,w) : \|\boldsymbol{x}_p^w - \boldsymbol{x}_p^v\| \leq \delta_p\}$ with the same edge weights $w_{ij} = 1$ ($(i,j) \in \overline{E}_p$) and

$$\delta_p \;=\; \frac{\sum_{\{q:(p,q)\in E_*\}} d_{pq}}{\#\{q : (p,q) \in E_*\}}(1+\sigma)\gamma,$$

where σ indicates the noise level of d_{pq} and $\gamma > 0$ is a parameter. For the numerical experiments in Section 5, $\gamma = 1/4$ was used. Since the size k of D_p is not known in our case, we decide the size k and a node subset with size k and maximum weight simultaneously. First, we compute the minimum norm base $\boldsymbol{\xi} \in \mathbb{R}^{\#F}$ of $\overline{G_p}$ by SFO [19, 20]. Next, $-\xi_1, -\xi_2, \ldots, -\xi_{\#F} \geq 0$ is sorted in the descending order such that $-\widehat{\xi}_1 \geq -\widehat{\xi}_2 \geq \ldots \geq -\widehat{\xi}_{\#F} \geq 0$, and $\{\boldsymbol{x}_p^v : v \in F\}$ is arranged according to the descending order of $-\widehat{\xi}_i$'s as $\widehat{\boldsymbol{x}}_p^1, \widehat{\boldsymbol{x}}_p^2, \ldots, \widehat{\boldsymbol{x}}_p^{\#F}$. Then, the set D_p and its size k are decided as follows. Initially, let $D_p = \{\widehat{\boldsymbol{x}}_p^1, \widehat{\boldsymbol{x}}_p^2, \ldots, \widehat{\boldsymbol{x}}_p^{\ell+1}\}$. Then, $\widehat{\boldsymbol{x}}_p^k$ ($k = \ell+2, \ldots, \#F$) is added to D_p one by one until some $k \geq \ell+2$ satisfies $\delta_{ave}\left(\sum_{i=1}^k (-\widehat{\xi}_i)\right)/k > -\widehat{\xi}_{k+1}$ for the first time or k reaches $\#F$, where $\delta_{ave} > 0$ is a parameter. The inequality indicates that the contribution score $-\widehat{\xi}_{k+1}$ drops below the average of $-\widehat{\xi}_1, -\widehat{\xi}_2, \ldots, -\widehat{\xi}_k$ multiplied by δ_{ave}. For the numerical experiments in Section 5, $\delta_{ave} = 0.75$ was used. Consequently, $D_p = \{\widehat{\boldsymbol{x}}_p^1, \widehat{\boldsymbol{x}}_p^2, \ldots, \widehat{\boldsymbol{x}}_p^k\}$ is obtained. The compounded location $\boldsymbol{x}_p$ is computed by $\sum_{\widehat{\boldsymbol{x}}_p^v \in D_p} \widehat{\boldsymbol{x}}_p^v / \#D_p$.

The idea of using the relative RMSD for Ξ can also be applied for comparing estimated locations of any subset A of N_*. Suppose that an estimated location $(\boldsymbol{x}_p^v : p \in A_t^v)$ of a set of nodes A_t^v has been computed by Algorithm 3.4 ($v = 1, 2, \ldots, u$), where $t > 0$ denotes a fixed iteration. We want to determine whether the estimate $(\boldsymbol{x}_p^v : p \in A_t^v)$ involves a critical error that could not be corrected. If it does, it should be disregarded and the iteration be restarted from a new $(\boldsymbol{x}_r : C) \in \Gamma$. Let $A = \bigcap_{v=1}^u A_t^v$. Then Algorithm 4.2 is applied with input $\Xi = \{(\boldsymbol{x}_p^v : A) : v = 1, 2, \ldots, u\}$. If $v \notin F$, then $(\boldsymbol{x}_p^v : p \in A_t^v)$ is disregarded.

In the discussion above, it is implicitly assumed that $A = \bigcap_{v=1}^u A_t^v$ contains a sufficient number of nodes so that Algorithm 4.2 can be effectively applied with $\Xi = \{(\boldsymbol{x}_p^v : A) : v = 1, 2, \ldots, u\}$. In the implementation of Algorithm 3.4, $\beta(C, m)$ with $m = 2$ or 3 is used to choose an initial admissible $(\ell+1)$-clique C. We set the iteration t to be 100, where Algorithm 4.2 is applied. See Section 4.1. If we start each $(\boldsymbol{x}_p^v : p \in A_t^v)$ from

an admissible $(\ell+1)$-clique $C^v \subset U^m(C)$, then it can be expected that $A = \bigcap_{v=1}^{u} A_t^v$ contains a sufficient number of nodes because C^v and C^w are located nearby in the graph $G(N_*, E_*)$. In fact, we observed in the numerical experiments that $\#A \geq 80$.

4.7 Application of the linear least square method

We describe a technique that considerably improves the successive SDP relaxation method. This technique works effectively, especially when graph $G(N_*, E_*)$ associated with a given EDGP becomes dense, say, the average degree of nodes exceeds 20.

Suppose that a temporary anchor set A_t, its locations $(\widehat{\boldsymbol{a}}_t : A_t)$, and its neighborhood N_t have been determined at the end of Step 2 of Algorithm 3.2, and Step 3 is executed. Before applying the sparse SDP relaxation to $\mathrm{P}(N_t, A_t)$, we fix some additional nodes as temporary anchors.

Let $p \in N_t \backslash A_t$. We assume that the size of the edge set $E(\{p\}, A_t)$ is not less than a fixed positive integer $\delta \geq \ell + 1$. Consider the system of distance equations

$$\|\boldsymbol{x}_p - \widehat{\boldsymbol{a}}_r\|^2 = d_{pr}^2 \ ((p,r) \in E(\{p\}, A_t)$$

or equivalently

$$\boldsymbol{x}_p^T \boldsymbol{x}_p - 2\widehat{\boldsymbol{a}}_r^T \boldsymbol{x}_p + \|\widehat{\boldsymbol{a}}\|^2 = d_{pr}^2 \ ((p,r) \in E(\{p\}, A_t)).$$

Choose $q \in A_t$ from $(p,q) \in E(\{p\}, A_t)$. If $\boldsymbol{x}_p$ satisfies the system of quadratic equations above, then it satisfies the following system of linear equations

$$2(\widehat{\boldsymbol{a}}_q - \widehat{\boldsymbol{a}}_r)^T \boldsymbol{x}_p = d_{pr}^2 - d_{pq}^2 - \|\widehat{\boldsymbol{a}}_r\|^2 + \|\widehat{\boldsymbol{a}}_q\|^2 \ \ ((p,r) \in E(\{p\}, A_t \backslash \{q\})). \tag{11}$$

Since this linear system is overdetermined and inconsistent in general, a least square solution $\boldsymbol{x}^p$ is computed. We can prove that the least square solution $\boldsymbol{x}^p$ converges to the exact location of the node p if the three conditions are satisfied: (i) $\widehat{\boldsymbol{a}}_r$ converges to the true location $\boldsymbol{a}_r^*$ $(r \in A_t)$, (ii) d_{pr} converges to the exact distance between the true location $\boldsymbol{a}_p^*$, (iii) $\boldsymbol{a}_q^* - \boldsymbol{a}_r^*$ $(r \in A_t \backslash \{q\})$ are linearly independent. This procedure is much inexpensive than the SDP relaxation method applied to $\mathrm{P}(N_t, A_t)$, even if the procedure is applied to all nodes $p \in N_t \backslash A_t$ with $\#E(\{p\}, A_t) \geq \delta$.

If $\#A_t$ and the chosen δ are sufficiently large (say $\delta \geq 10$), this procedure for estimating a location of the node p is as accurate as the SDP relaxation method. But if $\#A_t$ is small or if δ is small (say $\delta \leq 7$), it is not as effective as the SDP relaxation method. To decide whether $\#A_t$ is sufficiently large, the same parameter $\kappa_1 = 100$ as the one for choosing the neighborhood N_t in (5) can be used. Now, an additional Step based on the linear least square method is presented.

Step 2a: If $\#A_t \leq \kappa_1$, go to Step 3. Let $\widehat{A}_t = \{p \in N_t \backslash A_t : \#E(\{p\}, A_t) \geq \delta\}$. For every $p \in \widehat{A}_t$, compute a least square solution $\boldsymbol{x}_p = \widehat{\boldsymbol{a}}_p$ of the linear system of equations (11). Let $A_t = A_t \bigcup \widehat{A}_t$.

This can be included between Steps 2 and 3 of Algorithm 3.2.

5 Numerical Experiments

We implemented Algorithm 3.4, which is called passSDP, in Matlab and used Matlab Parallel Computing Toolbox for parallel computation using 12 cores. SFSDP [16, 18] is called by passSDP to formulate a sparse SDP relaxation of the minimization problem $\mathrm{P}_t(A_t, N_t)$ at each iteration, and SFSDP calls SDPA [10, 37] to solve the SDP relaxation problem. All the numerical experiments were performed on Opteron 6174 (2.20GHz/12MB L3, 12 cores x 4 = 48 cores) with 256GB (16 x 16GB / 1066MHz) memory.

In Section 5.1, we present numerical results on randomly generated 3-dimensional problems and compare passSDP with SFSDP [16, 18]. See Section 6.4 of [18] for numerical results on 3-dimensional anchor-free problems of SFSDP. In Section 5.2, numerical results on 3-dimensional anchor-free EDGP in [22] are presented and passSDP is compared with SFSDP and DISCO [22].

For anchor-free problems, the absolute locations of the nodes are not uniquely determined, and only relative locations among the nodes are meaningful. Data for each test problem consist of a node set N_*, an edge set E_*, distances d_{pq} $(p,q) \in E_*$ with noise, and true locations $(\boldsymbol{x}^*_p : N_*)$. We note that $(\boldsymbol{x}^*_p : N_*)$ is an instance of the node locations $p \in N_*$ that satisfies the distance equations $\left\|\boldsymbol{x}^*_p - \boldsymbol{x}^*_q\right\| = d^*_{pq}$ $((p,q) \in E_*)$, where d^*_{pq} denotes an exact distance between the nodes p and q $((p,q) \in E_*)$. Note that $(\boldsymbol{T}(\boldsymbol{x}^*_p) : N_*)$ is a solution for any nonsingular affine transformation $\boldsymbol{T}$ on $\mathbb{R}^\ell$ which represents a reflection, a rotation and a parallel transfer. Therefore, we apply a nonsingular affine transformation $\boldsymbol{T} : \mathbb{R}^\ell \to \mathbb{R}^\ell$, provided by a Matlab program procrustes.m [1, 22], which minimizes $\sum_{p\in N_*} \|T(\boldsymbol{x}_p) - \boldsymbol{x}^*_p\|^2$, before comparing the computed location $(\boldsymbol{x}_p : N_*)$ with the true location $(\boldsymbol{x}^*_p : N_*)$ given as data. The accuracy of the computed solution is measured by the RMSD between $(\boldsymbol{T}(\boldsymbol{x}_p) : N_*)$ and $(\boldsymbol{x}^*_p : N_*)$:

$$\left(\frac{1}{\#N_*}\sum_{p\in N_*} \|\boldsymbol{T}(\boldsymbol{x}_p) - \boldsymbol{x}^*_p\|^2\right)^{1/2}. \tag{12}$$

For the randomly generated problems with anchors reported in Section 5.1, the identity transformation for $\boldsymbol{T}$ is used for the computation of the RMSD because the nonsingular affine transformation $\boldsymbol{T} : \mathbb{R}^\ell \to \mathbb{R}^\ell$ was already applied at Step 7 of Algorithm 3.2.

To describe numerical results in tables, we use the following notation.

N_*	:	the set of nodes.
A_*	:	the set of anchors.
E_*	:	the set of edges (p,q) for which estimated distances d_{pq} are given, where each $(p,q) \in E_*$ is identified with (q,p).
Ave.deg	:	the average degree of nodes $= 2\#E_*/\#N_*$.
Ξ	:	the collection of estimated locations at Step 8 of Algorithm 3.4 from which an output estimated location is compounded.
Trials	:	the number of outermost loops in Algorithm 3.4 or the number of $(\ell+1)$-cliques $(\widehat{\boldsymbol{a}}_r : C)$ chosen from Γ at Step 0 of Algorithm 3.4; Trials $\geq \#\Xi$.
Ave.Iter.	:	the average of the number of iterations for generating each estimated location $(\boldsymbol{x}_p : N_*) \in \Xi$ at Step 8 of Algorithm 3.4 over the collection Ξ.

Cpd : RMSD of the estimated location compounded from the collection Ξ at Step 8 of Algorithm 3.4 (with respect to the exact location).
Best : the smallest RMSD of the estimated locations contained in Ξ.
Wst : the largest RMSD of the estimated locations contained in Ξ.
Ave : the average of RMSDs of the estimated locations contained in Ξ.
eTime : the total execution time of passSDP, SFSDP or DISCO in seconds.

5.1 Comparison to SFSDP on randomly generated problems

For $n = 1000,\ 2000,\ 4000,\ 8000,\ 16000$, and 32000, numerical test problems are generated as follows. Let $N_* = \{1, 2, \ldots, n\}$. Distribute n nodes $\boldsymbol{x}_p^*$ $(p \in N_*)$ randomly in the unit cube $[0, 1]^3$. We take $\rho = (15/n)^{(1/3)}$ and $\rho = (10/n)^{(1/3)}$ for the inter-node distance range. This means that each cube with the size ρ and the volume ρ^3 contains 15 nodes and 10 nodes on average, respectively. Let $E^\rho = \left\{(p, q) \in N_* \times N_* : \left\|\boldsymbol{x}_p^* - \boldsymbol{x}_q^*\right\| \leq \rho\right\}$. We take three sparsity level $\eta = 1.0,\ 0.8,\ 0.6$. If $\eta = 1$, then we set $E_* = E^\rho$. Otherwise, we select a subset E_* of E^ρ such that $\#E_*/\#E^\rho \approx \eta$ satisfies. A corrupted distance d_{pq} for each $(p, q) \in E_*$ is computed such that

$$d_{pq} \quad = \quad \max\left\{0.1, (1 + \sigma\xi_{pq})\right\} \left\|\boldsymbol{x}_p^* - \boldsymbol{x}_q^*\right\|,$$

where ξ_{pq} is chosen from the standard normal distribution $\mathcal{N}(0, 1)$ and the noise level $\sigma = 0.2$ is used.

	passSDP						SFSDP	
$\#N_*$	$\#\Xi$	Ave.	Rmsd			eTime	Rmsd	eTime
($\#A_*$, Ave.deg)	(Trials)	Iter.	Cpd	Best	Wst	(sec)		(sec)
1000 (100, 46.9)	12 (12)	104.7	1.9e-2	1.9e-2	1.9e-2	126	3.8e-2	23
1000 (0, 46.9)	12 (12)	123.0	2.1e-2	2.1e-2	2.1e-2	229	3.7e-2	419
2000 (200, 50.1)	12 (12)	114.3	1.4e-2	1.4e-2	1.4e-2	170	3.6e-2	25
2000 (0, 50.1)	12 (12)	126.4	1.6e-2	1.6e-2	1.6e-2	305	2.6e-1	8558
4000 (400, 51.8)	12 (12)	120.2	1.1e-2	1.1e-2	1.1e-2	379	4.2e-2	48
4000 (0, 51.8)	12 (12)	132.2	1.3e-2	1.3e-2	1.3e-2	516	-	-
8000 (800, 54.0)	12 (12)	118.3	8.4e-3	8.4e-3	8.5e-3	712	2.3e-2	387
8000 (0, 54.0)	12 (12)	141.3	9.9e-3	9.9e-3	1.0e-2	930	-	-
16000 (1600, 55.8)	12 (12)	118.8	6.5e-3	6.5e-3	6.6e-3	2306	1.6e-2	6263
16000 (0, 55.8)	12 (12)	142.9	7.4e-3	7.4e-3	7.4e-3	2570	-	-
32000 (3200, 57.4)	12 (12)	120.6	5.0e-3	5.0e-3	5.0e-3	8037	-	-
32000 (0, 57.4)	12 (12)	152.3	5.7e-3	5.7e-3	7.61e-3	9342	-	-

Table 1: Radio range $\rho = (15/\#N_*)^{(1/3)}$. Sparsity level = 1.0. With Setp 2a (see Section 4.7).

In Table 1, the performance of passSDP is compared with SFSDP for the test problems with highest density among all the test problems in Section 5. The sparsity of test problem is measured by Ave.deg $= 2\#E_*/\#N_*$. Ave.deg is ranges from 50 to 60 in

Table 1. In the other tables, it is less than 40. SFSDP is expected to work efficiently on dense problems by extracting a subgraph of $G(N_*, E_*)$, which has a sparse chordal extension. The efficiency of SFSDP improves as the number of anchors increases. See Section 3.3 of [16]. Indeed, we observe in Table 1 that SFSDP solved the problems with 1000, 2000, 4000, 8000 nodes and 10 % anchors very efficiently, and its execution time is shorter than that of passSDP. As the number of nodes increases more that 8000, however, passSDP solved the test problems more efficiently, even the problems with 10 % anchors.

Table 1 shows that the locations computed by passSDP are more accurate than those by SFSDP. While executing passSDP for the numerical results in Table 1, there was no failure at Step 5 of Algorithm 3.4 because $\#\Xi$ = Trials in every problem. Notice that the values of Cpd Rmsd, Best Rmsd and Wst Rmsd coincide with each other or very similar. We confirm the stability of Algorithm 3.4 for solving the test problems.

	passSDP						SFSDP	
$\#N_*$	$\#\Xi$	Ave.	RMSD			eTime	RMSD	eTime
($\#A_*$, Ave.deg)	(Trials)	Iter.	Cpd	Best	Wst	(sec)		(sec)
1000 (100, 32.5)	12 (12)	122.7	2.3e-2	2.3e-2	2.3e-2	150	5.0e-2	36
1000 (0, 32.5)	12 (12)	129.1	2.8e-2	2.7e-2	6.2e-2	154	2.4e-1	1366
2000 (200, 34.3)	12 (12)	116.8	1.7e-2	1.7e-2	1.8e-2	160	3.7e-2	149
2000 (0, 34.3)	12 (12)	135.7	1.9e-2	1.9e-2	2.2e-2	254	2.4e-2	12462
4000 (400, 35.4)	12 (12)	130.0	1.3e-2	1.3e-2	1.3e-2	324	2.6e-2	1259
4000 (0, 35.4)	12 (12)	147.4	1.5e-2	1.4e-2	6.6e-2	549	-	-
8000 (800, 36.7)	12 (12)	136.0	9.5e-3	9.5e-3	9.5e-3	802	-	-
8000 (0, 36.7)	12 (12)	162.2	1.1e-2	1.1e-2	5.0e-2	1331	-	-
16000 (1600, 37.7)	12 (12)	135.5	7.3e-3	7.3e-3	7.3e-3	2027	-	-
16000 (0, 37.7)	12 (12)	172.0	8.6e-3	8.5e-3	3.1e-1	4208	-	-
32000 (3200, 38.7)	12 (12)	132.2	5.7e-3	5.7e-3	5.9e-3	8277	-	-
32000 (0, 38.7)	12 (12)	181.9	6.6e-3	6.6e-3	2.5e-2	11939	-	-

Table 2: Radio range $\rho = (10/\#N_*)^{(1/3)}$. Sparsity level $\eta = 1.0$. Step 2a was executed.

For Table 2, we used a smaller radio range $\rho = (10/\#N_*)^{(1/3)}$ than the one for Table 1 to see how SFSDP and passSDP perform for problems with sparser graph $G(N_*, E_*)$. We observe from Table 2 that passSDP remains to work efficiently and effectively on the problems, and Ave. Iter., RMSD and eTime increased slightly. However, eTime by SFSDP increased considerably.

For the numerical experiments reported in Table 2, we incorporated Step 2a in Section 4.7. Table 3 shows numerical results on the same set of test problems as in Table 2 solved by passSDP without Step 2a. Comparing Tables 2 and 3, we confirm that Step 2a enhances the numerical efficiency of passSDP and that slightly accurate RMSDs were obtained by eliminating Step 2a.

In Tables 4 and 5, test problems with increased sparsity were solved by passSDP with Step 2a. All the problems were solved successfully. Ave. Iter., RMSD and eTime increased slightly, except for the largest problem with 32000 nodes and no anchors. We

	passSDP						
$\#N_*$	$\#\Xi$	Ave.	Rmsd				eTime
($\#A_*$, Ave.deg)	(Trials)	Iter.	Cpd	Best	Wst	Ave	(sec)
1000 (100, 32.5)	12 (12)	137.5	2.25e-2	2.25e-2	3.11e-2	2.32e-2	337
1000 (0, 32.5)	12 (12)	147.8	2.66e-2	2.66e-2	3.72e-2	2.81e-2	385
2000 (200, 34.2)	12 (12)	141.2	1.63e-2	1.62e-2	1.71e-2	1.65e-2	543
2000 (0, 34.2)	12 (12)	162.0	1.92e-2	1.88e-2	2.05e-2	1.94e-2	745
4000 (400, 35.4)	12 (12)	160.4	1.25e-2	1.25e-2	1.27e-2	1.26e-2	1140
4000 (0, 35.4)	12 (12)	177.5	1.45e-2	1.45e-2	2.17e-2	1.51e-2	1570
8000 (800, 36.7)	12 (12)	174.2	9.37e-3	9.36e-3	9.39e-3	9.37e-3	2727
8000 (0, 36.7)	12 (12)	199.6	1.17e-2	1.13e-2	4.14e-2	2.02e-2	3920
16000 (1600, 37.7)	12 (12)	173.9	7.34e-3	7.33e-3	7.45e-3	7.38e-3	6888
16000 (0, 37.7)	12 (12)	213.2	8.53e-3	8.53e-3	9.08e-3	8.64e-3	12792
32000 (3200, 38.7)	12 (12)	185.4	5.68e-3	5.67e-3	5.94e-3	5.71e-3	34992
32000 (0, 38.7)	12 (12)	259.0	6.62e-3	6.62e-3	7.50e-2	1.46e-2	102356

Table 3: Radio range $\rho = (10/\#N_*)^{(1/3)}$. Sparsity level = 1.0. Without Step 2a.

	passSDP						
$\#N_*$	$\#\Xi$	Ave.	Rmsd				eTime
($\#A_*$, Ave.deg)	(Trials)	Iter.	Cpd	Best	Wst	Ave	(sec)
8000 (800, 29.3)	12 (12)	150.8	1.08e-2	1.08e-2	1.34e-2	1.17e-2	821
8000 (0, 29.3)	12 (12)	188.8	1.52e-2	1.49e-2	2.42e-1	3.83e-2	1663
16000 (1600, 30.2)	12 (12)	169.4	8.40e-3	8.40e-3	8.58e-3	8.47e-3	2700
16000 (0, 30.2)	12 (12)	231.8	1.04e-2	1.04e-2	3.37e-1	6.17e-2	7763
32000 (3200, 31.0)	12 (12)	187.0	6.53e-3	6.36e-3	7.11e-3	6.69e-3	11189
32000 (0, 31.0)	12 (12)	291.2	7.97e-3	7.73e-3	3.01e-1	8.15e-2	31037

Table 4: Radio range $\rho = (10/\#N_*)^{(1/3)}$. Sparsitty level $\eta = 0.8$. Step 2a was executed.

	passSDP						
$\#N_*$	$\#\Xi$	Ave.	RMSD				eTime
($\#A_*$, Ave.deg)	(Trials)	Iter.	Cpd	Best	Wst	Ave	(sec)
8000 (800, 22.0)	12 (12)	161.4	1.31e-2	1.31e-2	1.35e-2	1.32e-2	856
8000 (0, 22.0)	12 (12)	214.0	1.63e-2	1.59e-2	4.52e-1	9.65e-2	3679
16000 (1600, 22.6)	12 (12)	171.2	1.00e-2	1.00e-2	1.03e-2	1.01e-2	2361
16000 (0, 22.6)	12 (12)	226.8	1.22e-2	1.22e-2	1.87e-2	1.40e-2	8022
32000 (3200, 23.2)	12 (12)	180.3	7.77e-3	7.76e-3	7.79e-3	7.78e-3	8824
32000 (0, 23.2)	12 (12)	321.7	1.36e-2	1.24e-2	2.34e-1	5.26e-2	46093

Table 5: Radio range $\rho = (10/\#N_*)^{(1/3)}$. Sparsitty level $\eta = 0.6$. Step 2a was executed.

tried to solve test problems with the sparsity level $\eta = 0.4$, but no satisfactory numerical results were obtained.

5.2 Euclidean distance geometry problems from molecular confirmation

The test problems in this section are from [22]; they consists of 13 molecular confirmation problems from the Protein Data Bank (PDB). We generated the input data including a node set N_*, an edge set E_*, exact node locations $(\boldsymbol{x}_p^* : N_*)$, and distances d_{pq} with noise $((p,q) \in E_*)$ by DISCO [22]. We can choose one of the two noise models, normal and uniform noise models with the noise level parameter ν, and the sparsity level that approximately determines $\#E_*/\#E^\rho$, where $\rho = 6\dot{\text{A}}$ is fixed, in DISCO [22]. In the numerical experiments, the normal noise model was tested with the noise level $\nu = 0.0,\ 0.1$ and 0.2, and the 30% sparsity level such that $\#E_*/\#E^\rho \approx 0.3$. See Section 5.2 of [22] for more details on the test problems.

The performance of SFSDP [16, 18] is compared in Table 6 with passSDP for smaller size problems with the number of nodes up to 1534. The parameter pars.minDegree controls the degrees of the nodes in the subgraph chosen for constructing a sparse SDP relaxation in SFSDP. See Section 4.1 of [16]. The default value for pars.minDegree is 5 $(= \ell + 2)$. Table 6 includes the results on SFSDP(dense), the dense SDP relaxation [2], for the 3-dimensional problem. We see from Table 6 that SFSDP(dense) worked better than SFSDP with minDeg = 5 and SFSDP with minDeg = 7 both in the quality of estimated locations and execution time. These problems are too sparse for SFSDP to extract a sparse subproblem. The RMSDs of passSDP are comparable to the ones of SFSDP(dense) in all the problems except 1GM2. The execution time of SFSDP(dense) is much shorter than that of passSDP for the first three problems with 166, 402 and 558 nodes, and comparable for the next two problems with 814 and 1003 nodes. For the largest problem 1F39 with 1534 nodes in Table 6, passSDP is much faster.

Tables 7, 8 and 9 display the numerical results by passSDP and DISCO [22] on the test problems with the noise levels $\nu = 0.0,\ 0.1,\ 0.2$, respectively. The RMSDs obtained by passSDP are comparable to the ones by DISCO except the largest problem 1YGP with 13488 nodes. For this problem, passSDP provided the estimated locations with much higher accuracy than DISCO in all tables. We also observe that passSDP is more than 10 times slower than DISCO for the smaller sized problems, and passSDP takes about 3 to 5 times longer than DISCO for the large-sized problems.

Ave. deg. of the test problems in this subsection ranges from 10 to 14, while Ave. deg. of the test problems in Section 5.1 is from 20 to 60. Therefore, the problems in Section 5.2 are much sparser and more difficult to solve than the ones in Section 5.1. In fact, passSDP encountered failures at Step 5 when it tried to solve the test problems 1RGS with noise level 0.0, 0.1 and 0.2, and RMSD Wst and RMSD Ave became larger than 2 in some of the test problems in Tables 7, 8 and 9. We note that the method described in Section 4.6 for compounding a more accurate estimation of location from Ξ worked successfully to produce an estimated location with RMSD Cpd comparable to RMSD Best.

Problem		SFSDP		SFSDP		SFSDP(dense)		passSDP	
$\#N_*$		minDeg = 5		minDeg = 7		RMSD	eTime	RMSD	eTime
(A.deg)	ν	RMSD	eTime	RMSD	eTime		(sec)	Cpd	(sec)
1GM2	0.0	0.36	9	0.71	39	5.0e-4	5	0.05	327
166	0.1	2.98	8	0.74	29	0.39	23	0.36	343
(13.5)	0.2	3.02	10	0.81	36	0.81	15	0.87	352
1PTQ	0.0	4.24	235	1.55	1029	0.57	95	0.75	936
402	0.1	7.13	246	1.11	877	0.70	90	0.84	895
(10.8)	0.2	3.06	295	1.41	1033	1.00	81	1.19	1133
1HOE	0.0	1.75	659	1.75	3014	0.23	205	0.51	1143
558	0.1	1.66	776	1.74	3095	0.48	202	0.67	802
(11.0)	0.2	1.17	831	1.13	2497	0.84	191	0.95	1152
1PHT	0.0	10.0	2245	-	-	0.87	2380	0.82	1789
814	0.1	9.88	2455	-	-	0.92	2916	0.90	3597
(12.8)	0.2	9.87	2310	-	-	1.31	2906	0.93	802
1AX8	0.0	12.5	9464	-	-	0.52	2846	0.74	1147
1003	0.1	-	-	-	-	0.78	3580	1.18	3116
(11.2)	0.2	-	-	-	-	1.28	2215	1.14	4408
1F39	0.0	-	-	-	-	0.63	11460	0.49	1635
1534	0.1	-	-	-	-	0.81	7987	0.80	1459
(11.1)	0.2	-	-	-	-	1.16	10414	1.25	2173

Table 6: Noise level = 0.0.

	passSDP							DISCO	
Problem	$\#\Xi$	Ave.	RMSD				eTime	RMSD	eTime
($\#N_*$, Ave.deg)	(Trials)	Iter.	Cpd	Best	Wst	Ave	(sec)		(sec)
1GM2 (166,13.5)	12 (12)	112.8	0.05	0.04	0.06	0.05	327	0.10	23
1PTQ (402,10.8)	12 (12)	177.1	0.75	0.69	0.77	0.75	936	0.34	34
1HOE (558,11.0)	12 (12)	191.6	0.51	0.36	0.59	0.50	1143	0.12	89
1PHT (814,12.8)	12 (12)	204.8	0.82	0.82	0.98	0.86	1789	0.92	110
1AX8 (1003,11.2)	12 (12)	225.8	0.74	0.73	1.11	0.90	1147	0.76	152
1F39 (1534,11.1)	12 (12)	250.2	0.49	0.47	1.31	0.59	1634	0.56	404
1RGS (2015,11.2)	9 (12)	290.8	0.59	0.60	7.04	2.00	2634	0.54	385
1KDH (2923,11.8)	12 (12)	300.5	0.64	0.60	11.01	1.58	2677	0.56	638
1BPM (3672,12.3)	12 (12)	323.3	0.35	0.35	4.89	0.78	2881	0.40	858
1TOA (4292,12.1)	12 (12)	377.6	0.49	0.45	15.37	3.88	6160	0.46	1665
1MQQ (5681,12.7)	12 (12)	362.3	0.24	0.25	0.30	0.27	4045	0.35	1660
1I7W (8629,12.3)	12 (12)	493.3	0.46	0.41	30.79	6.15	8976	0.70	3402
1YGP (13488,12.6)	12 (12)	520.0	0.32	0.33	7.11	0.91	24299	1.93	7411

Table 7: Noise level = 0.0.

Problem ($\#N_*$, Ave.deg)	passSDP $\#\Xi$ (Trials)	Ave. Iter.	RMSD Cpd	RMSD Best	RMSD Wst	RMSD Ave	eTime (sec)	DISCO RMSD	DISCO eTime (sec)
1GM2 (166,13.5)	12 (12)	112.0	0.36	0.36	0.43	0.37	343	0.36	20
1PTQ (402,10.8)	12 (12)	172.1	0.84	0.78	0.88	0.83	895	0.52	31
1HOE (558,11.0)	12 (12)	179.0	0.67	0.66	0.77	0.71	802	0.42	81
1PHT (814,12.8)	12 (12)	198.3	0.90	0.84	1.52	1.09	3597	0.82	98
1AX8 (1003,11.2)	12 (12)	211.2	0.93	0.93	2.20	1.47	802	0.76	153
1F39 (1534,11.1)	12 (12)	236.8	0.80	0.78	0.87	0.81	1459	0.77	388
1RGS (2015,11.2)	7 (12)	258.3	0.80	0.81	8.30	3.09	945	0.73	392
1KDH (2923,11.8)	12 (12)	278.7	0.85	0.79	15.04	3.45	2377	0.71	642
1BPM (3672,12.3)	12 (12)	290.7	0.56	0.53	0.95	0.64	1622	0.54	1031
1TOA (4292,12.1)	12 (12)	300.2	0.65	0.63	18.39	2.38	2519	0.60	1595
1MQQ (5681,12.7)	12 (12)	316.1	0.45	0.45	0.79	0.51	3185	0.47	1656
1I7W (8629,12.3)	12 (12)	345.7	0.73	0.73	18.90	3.78	4683	0.90	3558
1YGP (13488,12.6)	12 (12)	374.5	0.50	0.50	1.40	0.81	7727	1.54	8048

Table 8: Noise level = 0.1.

Problem ($\#N_*$, Ave.deg)	passSDP $\#\Xi$ (Trials)	Ave. Iter.	RMSD Cpd	RMSD Best	RMSD Wst	RMSD Ave	eTime (sec)	DISCO RMSD	DISCO eTime (sec)
1GM2 (166,13.5)	12 (12)	137.2	0.87	0.86	1.10	0.93	352	0.92	20
1PTQ (402,10.8)	12 (12)	223.5	1.19	1.19	1.30	1.22	1133	1.01	35
1HOE (558,11.0)	10 (12)	240.9	0.95	0.94	1.13	1.00	1152	0.79	77
1PHT (814,12.8)	12 (12)	263.9	1.18	1.23	2.16	1.46	3116	1.18	95
1AX8 (1003,11.2)	19 (24)	414.9	1.08	1.14	5.03	2.02	4408	1.30	150
1F39 (1534,11.1)	12 (12)	331.4	1.25	1.27	1.81	1.43	2173	1.17	375
1RGS (2015,11.2)	6 (24)	368.7	1.16	1.20	9.61	4.35	2540	1.09	352
1KDH (2923,11.8)	12 (12)	402.6	1.36	1.18	1.41	1.32	3844	1.03	633
1BPM (3672,12.3)	24 (24)	669.5	1.01	1.01	1.83	1.25	12883	0.94	828
1TOA (4292,12.1)	12 (12)	447.1	1.04	1.04	1.33	1.12	5123	0.88	1621
1MQQ (5681,12.7)	12 (12)	477.3	0.82	0.81	1.27	0.96	7615	0.77	1606
1I7W (8629,12.3)	12 (12)	529.6	1.16	1.19	17.34	2.67	11240	1.39	3289
1YGP (13488,12.6)	12 (12)	583.7	0.86	0.86	1.10	0.94	21803	1.78	7466

Table 9: Noise level = 0.2.

6 Concluding Remarks

We have proposed the numerical method for solving EDGPs including anchor-free 3-dimensional problems. The proposed method can be executed in parallel, using the multiple cliques as initial sets of anchors. As a result, the multiple sets of estimated locations of nodes are obtained and their accuracy is improved by the compounding procedure. Numerical results in Section 5 show that the proposed method outperforms SFSDP for uniformly sparse EDGPs, and produces comparable results to DISCO for molecular conformation test problems. For the largest problem 1YGP, the results by proposed method are more accurate.

The results obtained by the proposed method for the the molecular conformation problem with 20 % sparsity are not very accurate. Our future goal is to improve the algorithm so that the accuracy of the estimated locations of the problems with 20 % sparsity can be within the desired accuracy. In addition, it would be interesting to combine the divide and conquer method with the proposed method to tackle more difficult and large-scale problems.

Acknowledgments

The authors would like to thank Professor Kim-Chuan Toh for Matlab programs procruste.m and refinepositions.m.

References

[1] P. Biswas, K.C. Toh, and Y. Ye. A distributed SDP approach for large scale noisy anchor-free graph realization with applications to molecular conformation, *SIAM J. Sci. Comput.*, **30** (2008) 1251–1277.

[2] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization, in *Proceedings of the third international symposium on information processing in sensor networks*, ACM press, New York (2004) 46–54.

[3] P. Biswas and Y. Ye. A distributed method for solving semidefinite programs arising from Ad Hoc Wireless Sensor Network Localization, in *Multiscale Optimization Methods and Applications*, Springer, New York (2006) 69–84.

[4] I. Borg and P. Groenen, *Modern Multidimensional Scaling*, second ed. Springer, New York (2010).

[5] G.M. Crippen and T.F. Havel. *Distance Geometry and molecular Conformation*, Wiley, New York (1988).

[6] CSDP Homepage, https://projects.coin-or.org/Csdp/

[7] Q. Dong and Z. Wu. A geometric build-up algorithm for solving the molecular distance geometry problems with sparse distance data, *J. Global Optim.*, **26** (2003) 321–333.

[8] B. Everitt and S. Rabe-Hesketh. *The Analysis of Proximity Data*, Arnold, London (1997).

[9] S. Fujishige. *Submodular Functions and Optimization, 2nd ed*, Elsevier, New York (2005).

[10] K. Fujisawa, M. Fukuda, K. Kobayashi, M. Kojima, K. Nakata, M. Nakata and M. Yamashita. SDPA (SemiDefinite Programming Algorithm) User's Manual — Version 7.0.5, Research Report B-448, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo (2008).

[11] M. Fukuda, M. Kojima, K. Murota and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: General framework, *SIAM J. Optim.*, **11** (2000) 647–674.

[12] W. Glunt, T. Hayden, S. Hong, and J. Wells. An alternating projection algorithm for computing the nearest euclidean distance matrix, *SIAM J. Math. Anal. and Appl.*, **11** (1990) 589–600.

[13] I. G. Grooms, R. M. Lewis, and M. W. Trosset. Molecular embedding via a second-order dissimilarity parameterized approach, *SIAM J. Sci. Comput.*, **31** (2009) 2733–2756.

[14] T. F. Havel, I. D. Kuntz, and G. M. Crippen. The combinatorial distance geometry approach to the calculation of molecular conformation, *J. of Theor. Biol.*, **104** (1983) 359–381.

[15] T. F. Havel. A evaluation of computational strategies for use in the determination of protein structure from distance constraints obtained by nuclear magnetic resonance, *Progress is Biophysics and Molecular Bio.*, **56** (1991) 43–78.

[16] S. Kim, M. Kojima and H. Waki. Exploiting sparsity in SDP relaxation for sensor network localization, *SIAM J. Optim.*, **20** (2009) 192–215.

[17] S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. Exploiting Sparsity in Linear and Nonlinear Matrix Inequalities via Positive Semidefinite Matrix Completion, *Math. Program.*, **129** (2011) 33–68.

[18] S. Kim, M. Kojima, H. Waki, and M. Yamashita. SFSDP: a Sparse Version of Full SemiDefinite Programming Relaxataion for Sensor Network Localization Problems, *ACM Trans. Math. Softw.*, **38** (2012) 4.

[19] A. Krause. SFO: A Toolbox for Submodular Function Optimization, *J. Mach. Learn. Res.* , **3** (2010) 1141–1144.

[20] A. Krause, and C. Guestrin Near-optimal observation selection using submodular functions, in *AAAI'07 Proceedings of the 22nd national conference on Artificial intelligence*, **2** (2007)1650–1654.

[21] S. Lele. Euclidean distance matrix analysis (EDMA): Estimation of mean form and mean form difference, *Math. Geol.*, bf 25 (1993) 573–602.

[22] N.-H. Z. Leung and K.-C. Toh. An SDP-based divide-and-conquer algorithm for large scale noisy anchor-free graph realization, *SIAM J. Sci. Comput.*, **31** (2009) 4351–4372.

[23] T.-C. Lian, T.-C. Wang, and Y. Ye. A gradient search method to round the semidefinite programming relaxation solution for ad hoc wireless sensor network localization, Technical report, Dept. of Management Science and Engineering, Stanford University (2004).

[24] L. Liberti, C. Lavor, N. Maculan and A. Mucherino. Euclidean Distance Geometry and Applications, arXiv:1205.0349 [q-bio.QM], May 3, 2012.

[25] J. J. Moré, Z. Wu. Global continuation for distance geometry problems, *SIAM J. Optim.*, **7** (1997) 814–836.

[26] J. J. Moré, Z. Wu. Distance geometry optimization for protein structures, *J. Global Optim.*, **15** (1999) 219–234.

[27] K. Nagano, Y. Kawahara and K. Aihara (2011) "Size-constrained submodular minimization through minimum norm base," *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, 977–984.

[28] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima and K. Murota. Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results, *Math. Program.*, **95** (2003) 303–327.

[29] J. Nielsen and B. Roth. On the kinematic analysis of robotic mechanisms, *Internat. J. Robotics Res.*, **18** (1999) 1147–1160.

[30] SDPA Homepage, http://sdpa.indsys.chuo-u.ac.jp/sdpa/.

[31] SDPT3 homepage, http://www.math.nus.edu.sg/∼mattohkc/sdpt3.html

[32] SeDuMi Homepage, http://sedumi.mcmaster.ca.

[33] A. M. So and Y. Ye. Theory of semidefinite programming for sensor network localization, *Math. Program.*, **109** (2007) 367–384.

[34] K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction, in *Proceedings of the Twenty-First International Conference on Machine Learning (ICML ' 04)*, ACM Press, New York (2004) 839–846.

[35] Z. Wang, S. Zheng, S. Boyd, and Y. Ye. Further relaxations of the SDP approach to sensor network localization, *SIAM J. Optim.*, **19** (2008) 655–673.

[36] D. Wu and Z. Wu. An updated geometric build-up algorithm for solving the molecular distance geometry problems with sparse distance data, *J. Global Optim.*, **37** (2007) 661–673.

[37] M. Yamashita, K. Fujisawa, M.Fukuda, K. Kobayashi, K. Nakata and M. Nakata. Latest development in the SDPA family for solving large-scale SDPs. In: M. F. Anjos, J. B. Lasserre (eds.) *Handbook on Semidefinite, Conic and Polynomial Optimization*, Springer, New York (2012) 687–713.