Point Placement in an Inexact Model with Applications

Kishore Kumar V. Kannan, Pijus K. Sarker, Amangeldy Turdaliev, and Asish Mukhopadhyay(✉)

School of Computer Science, University of Windsor, Windsor, ON N9B 3P4, Canada
{varadhak,sarkerp,turdalia,asishm}@uwindsor.ca

Abstract. The point placement problem is to determine the locations of n distinct points on a line uniquely (up to translation and reflection) by making the fewest possible pairwise distance queries of an adversary. A number of deterministic and randomized algorithms are available when distances are known exactly. In this paper, we discuss the problem in an inexact model. This is when distances returned by the adversary are not exact; instead, only upper and lower bounds on the distances are provided. We propose an algorithm called DGPL for this problem that is based on a distance geometry approach that Havel [8] used to solve the molecular conformation problem. Our algorithm does not address the problems of query choices and their minimization; these remain open. We have used our DGPL algorithm for the probe location problem in DNA mapping, where upper and lower bounds on distance between some pairs of probes are known. Experiments show the superior performance of our algorithm compared to that of an algorithm by Mumey [9] for the same problem.

Keywords: Probe location problem · Distance geometry · Point placement · Bound smoothing · Embed algorithm · Eigenvalue decomposition

1 Introduction

Retrieving the coordinates of n points $P = \{p_0, p_2, \ldots, p_{n-1}\}$ from their mutual distances is a problem that is interesting both from a theoretical as well as a practical point of view. Young and Householder [10] showed that a necessary and sufficient condition is that the matrix $B = [b_{ij}] = [(d_{i0}^2 + d_{j0}^2 - d_{ij}^2)/2]$ is positive semi-definite, where d_{ij} is the Euclidean distance between p_i and p_j. The rank of B also gives the minimum dimension of the Euclidean space in which the point set P can be embedded.

In another variation of the problem, given a set of mutual distances and an embedding dimension k, can P be embedded in Euclidean space E^k to realize the given distances [11]? This problem is strongly NP-complete even when $k = 1$ and the distances are restricted to the set $\{1, 2, 3, 4\}$. It is, however, possible to solve this problem for special classes of graphs and the embedding dimension is 1 [12]. The edges of such graphs join pairs of points for which the distances are known.

We, and other researchers, extensively studied **the point placement problem** [2,3,6,13,14], which is to determine an unique (up to translation and reflection) one-dimensional embedding of n distinct points by making the fewest possible pairwise distance queries of an adversary. The queries, spread over one or more rounds, are modeled as a graph whose vertices represent the points and there is an edge connecting two vertices, if the distance between the corresponding points is queried. The goal is to keep the number of queries linear, with a constant factor as small as possible. This requires meeting a large number of constraints on the pairwise distances, making the resulting algorithms very intricate [15].

The simplest of all, the 3-cycle algorithm, has the following query graph (Fig. 1).

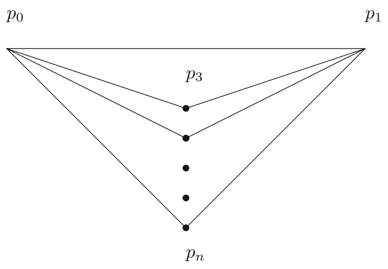

Fig. 1. Query graph using triangles

If G = *(V,E)* is a query graph, an assignment l of lengths to the edges of G is said to be valid if there is a placement of the vertices V on a line such that the distances between adjacent vertices are consistent with l. Here in this problem, the distance between a pair of vertices returned by the adversary are exact. The algorithm designer tries to construct a graph over fixed number of rounds to minimize the number of edge queries and also make sure that there is a unique placement of the vertices. The construction of such a graph is the heart of different algorithms for this problem.

In this paper, we study a one-round version of the point placement problem in the adversarial model when the query distances returned by the adversary are not necessarily exact. A formal definition of the problem goes as follows.

Problem Statement: Let $P = \{p_0, p_1, ..., p_{n-1}\}$ be a set of n distinct points on a line. For a pair of points p_i and p_j, the query distance $D(p_i, p_j)$ lies in an interval $[l_{ij}, u_{ij}]$, where $l_{ij} \leq u_{ij}$ denote the lower and upper bound respectively. The objective is to find a set of locations of the points in P that are consistent with the given distance bounds.

The distance bounds are collectively represented by an upper distance matrix, U, for the upper bounds and a lower distance matrix, L, for the lower bounds.

The corresponding entries in the upper and lower bound matrices for pairs of points for which $l_{ij} = u_{ij}$, that is, the pairwise distances are exactly known, are identical. When the distance between a pair of points is unknown, the distance interval bound is set to $[-\infty, \infty]$.

For example with three points ($n = 3$), a possible pair of upper and lower bound distance matrices are shown below:

$$U(p_0, p_1, p_2) = \begin{pmatrix} 0 & 60 & \infty \\ 60 & 0 & 3 \\ \infty & 3 & 0 \end{pmatrix}$$

$$L(p_0, p_1, p_2) = \begin{pmatrix} 0 & 60 & -\infty \\ 60 & 0 & 3 \\ -\infty & 3 & 0 \end{pmatrix}$$

Thus, $l_{01} = u_{01} = 60$ for the pair of points p_0 and p_1, whereas $l_{02} = -\infty$ and $u_{02} = \infty$ for the pair p_0 and p_2.

Motivation: This study is motivated by a problem from computational biology, where probe locations are to be mapped on a chromosome, given distance estimates between probe pairs that are obtained from FISH experiments [16–18]. This is known in the literature as the probe location problem.

Overview of Contents: The rest of the paper is organized thus. In the next section, we show how the point placement problem in the inexact model can be formulated in the distance geometry framework. Based on this formulation, we propose an algorithm, called DGPL (short for Distance Geometry based Probe Location). We follow up with an analysis of some experimental results (1-dimensional layouts) obtained from an implementation of our DGPL algorithm. In the third section, we briefly review an algorithm by Mumey [9] for the probe location problem. In the next section, experimental results are given, comparing the performance of the DGPL algorithm with that of Mumey's. In the final section, we conclude with some observations and directions for further research.

2 The Distance Geometry Approach

The fundamental problem of distance geometry is this: "Given a set of m distance constraints in the form of lower and upper bounds on a (sparse) set of pairwise distance measures, and chirality constraints on quadruples of points, find all possible embeddings of the points in a suitable k-dimensional Euclidean space" [1,19].

Crippen and Havel's EMBED algorithm for the molecular conformation problem is based on a solution to the above fundamental problem. Let d_{ij} denote the distance between a pair of points p_i and p_j; then each of the m constraints above specifies an upper and lower bound on some d_{ij}. For our 1-dimensional point placement problem, chirality constraints do not come into the picture. Determining a feasible set of locations of the set of points in P is equivalent to finding the coordinates of these locations, with reference to some origin of coordinates. Thus the approach used for the EMBED algorithm can be directly adapted to the solution of our problem. Due to the non-availability of an implementation of the EMBED algorithm, we wrote our own.

Below, we describe the DGPL algorithm, on which our implementation is based. Before this, the so-called bound-smoothing algorithm used in the EMBED algorithm deserves some mention. The main underlying idea is to refine distance bounds into distance limits. Thus if $[l'_{ij}, u'_{ij}]$ are the distance limits corresponding to distance bounds $[l_{ij}, u_{ij}]$, then $[l'_{ij}, u'_{ij}] \subset [l_{ij}, u_{ij}]$. The distance limits are defined as follows.

Let l_{ij} and u_{ij} be the upper and lower distance bounds on d_{ij}. Then

$$l'_{ij} = \inf_e\{d_{ij} | l_{ij} \leq d_{ij} \leq u_{ij}\}$$

and

$$u'_{ij} = \sup_e\{d_{ij} | l_{ij} \leq d_{ij} \leq u_{ij}\},$$

where the *inf* and *sup* are taken over all possible embeddings $e : P \times P \to R$ of the points in P on a line.

When this is done, assuming that the triangle inequality holds for the triplet of distances among any three points p_i, p_j, p_k, then these distance limits are called triangle limits.

In [20], a characterization of the triangle limits was established. It was also shown how this could be exploited to compute the triangle limits, using a modified version of Floyd's shortest path algorithm [15].

2.1 Algorithm DGPL

An input to the DGPL program for a set of n points in its most general form consists of the following: (a) exact distances between some pairs of points; (b) finite upper and lower bounds for some other pairs; (c) for the rest of the pairs, the distance bounds are not specified. However, as we have conceived of DGPL as a solution to the point placement problem in the inexact model, the input is assumed to be adversarial and is simulated by generating an initial adversarial layout of the points. This layout is used by the algorithm to generate input data of types (a) and (c); however, DGPL will also work if some of the input data are of type (b).

The DGPL algorithm works in three main phases, as below.

Phase 1: [Synthetic Data Generation] Based on the user-input for n, the algorithm simulates an adversary to create a layout of n points; further, based on the user-input for the number of pairs for which mutual distances are not known, the algorithm assigns the distance bounds $[-\infty, \infty]$ for as many pairs and uses the layout created to assign suitable distance bounds for the rest of the pairs; the output from this stage are the lower and upper bound distance matrices, L and U.

Phase 2: [Point location or Coordinate generation] A set of coordinates of the n points, consistent with the bounds in the matrices L and U, are computed, following the approach in [8] for the molecular conformation problem.

Phase 3: [Layout generation] Finally, a layout of the embedding is generated, which allows us to verify how good the computed layout is vis-a-vis the adversarial layout.

A more detailed description of the algorithm is given below.

Algorithm 1. DGPL

```
Data: 1. The size $n$ of $P$.
      2. The number of pairs for which $l = -\infty$ and $u = \infty$.
      3. Embedding dimension
Result: Locations of the $n$ points in $P$, consistent with the distance bounds
(1) Create a random valid layout of the points $p_i$ in $P = \{p_0, p_1, ...., p_{n-1}\}$.
(2) Create distance bound matrices $L$ and $U$.
    (2.1) Assign $-\infty$ and $\infty$ respectively to the corresponding values in the $L$ and
          $U$ matrices for as many pairs as the the number unknown distances
          (user-specified).
    (2.2) Assign the distances determined from the layout of Step 1
          to the remaining entries of both $L$ and $U$
(3) Apply a modified version of Floyd's shortest path algorithm [8] to compute
    triangle limit matrices, $LL$ and $UL$, from the distance bound matrices, $L$ and $U$.
(4) If all triangle limit intervals have been collapsed to a single value (which is now
    the distance between the corresponding pair of points) go to Step 5, else collapse
    any remaining pair of triangle limit interval to a randomly chosen value in this
    interval and go to Step 3 (this step is called metrization)
(5) Compute matrix $B = [b_{ij}] = [(d_{i0}^2 + d_{j0}^2 - d_{ij}^2)/2]$, where $d_{ij}$ is the distance
    between points $p_i$ and $p_j$, $1 \leq i, j \leq n - 1$ [10].
(6) Compute the eigenvalue decomposition of the matrix $B$; the product of the
    largest eigenvalue with its corresponding normalized eigenvector gives the
    one-dimensional coordinates of all the points.
(7) The computed coordinates are plotted on a line, placing $p_0$ at the origin of
    coordinates, and are compared with the adversarial layout for the extent of
    agreement.
```

In the next section, we discuss the simulation of DGPL for a small value of n.

2.2 Simulating DGPL

1. Let the input to the program be as follows:
Number of points: 4
Number of unknown distances: 1
Embedding dimension: 1.

2. The following random one-dimensional layout of the four points is created:

$$p_0 = 0, p_1 = 59, p_2 = 48, p_3 = 74.$$

3. The input lower and upper bound distance matrices, L and U, are set to:

$$L = \begin{pmatrix} 0 & 59 & 48 & 74 \\ 59 & 0 & 11 & -\infty \\ 48 & 11 & 0 & 26 \\ 74 & -\infty & 26 & 0 \end{pmatrix}, U = \begin{pmatrix} 0 & 59 & 48 & 74 \\ 59 & 0 & 11 & \infty \\ 48 & 11 & 0 & 26 \\ 74 & \infty & 26 & 0 \end{pmatrix},$$

where the distance d_{13} between the points p_1 and p_3 is contained in the interval $[-\infty, \infty]$.

4. The lower and upper triangle limit matrices, LL and UL, obtained by bound smoothing from the lower and upper distance bound matrices are:

$$LL = \begin{pmatrix} 0 & 59 & 48 & 74 \\ 59 & 0 & 11 & 15 \\ 48 & 11 & 0 & 26 \\ 74 & 15 & 26 & 0 \end{pmatrix}, UL = \begin{pmatrix} 0 & 59 & 48 & 74 \\ 59 & 0 & 11 & 37 \\ 48 & 11 & 0 & 26 \\ 74 & 37 & 26 & 0 \end{pmatrix}.$$

5. The distance limits are then metrized, as discussed in Step 4 of the DGPL algorithm, to fixed distances as:

$$D = \begin{pmatrix} 0 & 59 & 48 & 74 \\ 59 & 0 & 11 & 35 \\ 48 & 11 & 0 & 26 \\ 74 & 35 & 26 & 0 \end{pmatrix}.$$

6. Relative to the point p_0 as the origin of coordinates, the matrix B computes to:

$$B = \begin{pmatrix} 3481 & 2832 & 3866 \\ 2832 & 2304 & 3552 \\ 3866 & 3552 & 5476 \end{pmatrix}.$$

7. From the eigenvalue decomposition of the matrix B, we obtain the following coordinate matrix as the product of the square root of largest eigenvalue with its corresponding normalized eigenvector:

$$X = \begin{pmatrix} 58.677 \\ 48.828 \\ 73.671 \end{pmatrix}.$$

The 1-dimensional embedding of the computed coordinates, after rounding to integer values, is shown in Fig. 2; the agreement with the adversarial layout is seen to be near-perfect.

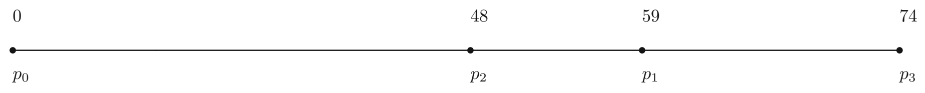

Fig. 2. Final embedding of the four input points

3 Mumey's Approach to the Probe Location Problem

Mumey [9] considered the problem of mapping probes along a chromosome based on separation or distance intervals between probe pairs, estimated from *fluorescence in-situ hybridization (FISH)* experiments. He named this as the probe location problem. The problem is challenging as the distance intervals are known only with some confidence level, some may be error-prone and these need to be identified to find a consistent map. Previous algorithmic approaches based on seriation [16], simulated annealing [17] and branch and bound algorithm [18], relying as these did on exhaustive search, could handle inputs of only up to 20 or fewer probes efficiently. However, Mumey's algorithm claims to be able solve the problem for up to 100 probes in a matter of few minutes. We briefly review Mumey's algorithm next.

From the above discussion it is clear that Mumey's problem can also be conveniently cast in the framework of distance geometry. Indeed, we have done this and the results are discussed in the next section.

3.1 Overview of Mumey's Algorithm

Let $P = \{p_0, p_1, ..., p_{n-1}\}$ be the list of probes in a chromosome, it being given that the distance interval between a pair of probes p_i and p_j lie in an interval $[l_{ij},\ u_{ij}]$, where l_{ij} and u_{ij} are respectively the lower and upper bounds on the distance. The probe location problem is to identify a feasible set of probe locations $\{x_0, x_1, ..., x_{n-1}\}$ from the given distance intervals such that $|x_i - x_j| \in [l_{ij}, u_{ij}]$.

The distance bounds on probe pairs leads to a special kind of linear program, which is a system of difference constraints. A directed graph derived from these constraints is input to Bellman-Ford's single source shortest path algorithm. The shortest paths from the specially-added source vertex to all the destination vertices is a solution to the system of difference constraints [15].

For the probe location problem, the directed graph in question is called an edge orientation graph, whose construction is described next.

Edge Orientation Graph. The first step in the construction of an edge orientation graph is to set the orientation of each edge by choosing one placement. If x_i and x_j are the positions of probes i and j then one of the following is true:

$$x_j - x_i \in [l_{ij}, u_{ij}] \quad (1)$$

$$x_j - x_i \in [l_{ij}, u_{ij}] \quad (2)$$

If (1) holds then x_i is to the left of x_j whereas if (2) holds, then x_j is to the left of x_i. Assume that (1) holds. We can express this equivalently as.

$$l_{ij} \leq x_j - x_i \leq u_{ij}$$

or as:

$$x_j - x_i \leq u_{ij} \tag{3}$$
$$x_i - x_j \leq -l_{ij} \tag{4}$$

Corresponding to the two inequalities above, we have two edges in the edge orientation graph, one going from x_i to x_j, with weight u_{ij} and the other from x_j to x_i with weight $-l_{ij}$.

Similarly, if (2) holds, we can express this equivalently as:

$$l_{ij} \leq x_i - x_j \leq u_{ij}$$

or as:

$$x_i - x_j \leq u_{ij} \tag{5}$$
$$x_j - x_i \leq -l_{ij} \tag{6}$$

with two directed edges in the edge orientation graph of weights u_{ij} and $-l_{ij}$ respectively.

When $l_{ij} = u_{ij}$ for a pair of probes p_i and p_j, there is exactly one edge connecting the corresponding nodes in the edge orientation graph, its orientation determined by the relative linear order of the probes.

Finding Feasible Probe Positions. Once all the edge weights are fixed, a special source vertex is added whose distance to all other vertices is initialized to 0 and Bellman-Ford's algorithm is run on this modified edge orientation graph. If there is no negative cycle in the graph, Bellman-Ford's algorithm outputs a set of feasible solutions $(x_0, x_1, ..., x_{n-1})$. Otherwise, the algorithm resets the edge weights by changing the relative order of a probe pair and re-runs the Bellman-Ford algorithm. These two steps are repeated till a feasible solution is found.

4 Experimental Results

We implemented both the DGPL algorithm and Mumey's, discussed in the previous sections in Python 2.7. The programs were run on a computer with the following configuration: Intel(R) Xeon(R) CPU, X7460@2.66GHz OS: Ubuntu 12.04.5, Architecture:i686. We used the mathematical package numpy.linAlg to calculate eigenvalue decompositions and also for solving linear equations; the package matplotlib.pyplot was used to plot an embedding of final coordinates obtained from the programs in a space of specified dimensionality.

The chart below compares the running time of Mumey's algorithm with that of DGPL. Each of these algorithms were run on point sets of different sizes, up to 101 points. We also recorded the effect on both algorithms of the number of pairs of points (probes) for which the distances are not known or are unspecified. In these cases, we set $l_{ij} = -\infty$ and $u_{ij} = \infty$ (Table 1).

Table 1. Performance comparison of Mumey's and DGPL algorithm

No. of points	No. of unknown distances	Mumey's approach running time (hrs:mins:secs)	DGPL algorithm running time (hrs:mins:secs)
3	1	0:00:00.000184	0:00:00.001514
10	2	0:00:00.001339	0:00:00.006938
10	5	0:00:00.024560	0:00:00.006816
10	8	0:00:00.060520	0:00:00.017163
20	2	0:00:00.001369	0:00:00.007464
20	5	0:00:00.001336	0:00:00.007743
20	10	0:00:01.164363	0:00:00.007436
40	5	0:00:00.947250	0:00:00.328563
40	8	0:00:07.369925	0:00:00.315001
40	10	0:00:30.857658	0:00:00.312674
80	5	0:00:10.609233	0:00:02.503798
80	10	0:06:15.443501	0:00:02.496285
80	15	5:00:00.000000+	0:00:02.687672
101	5	0:00:14.256343	0:00:05.020695
101	10	0:10:32.299084	0:00:05.282747
101	15	5:00:00.000000+	0:00:05.192594

Interestingly, the chart (Fig. 3) shows that Mumey's approach takes a longer time when the number of unknown distances increases. Each of these algorithms were run on point sets of different sizes, up to 100 points.

Clearly, the DGPL algorithm is consistently faster; as we can see from the graph, irrespective of the number of unknown distances in the fixed number of points, DGPL's run-time is linear in the number of points. However, this is not true of Mumey's approach. This can be explained by the fact that when a negative cycle is detected Bellmann-Ford's algorithm has to be run again. This is more likely to happen as the number of unknown distances increase. Thus the running time increases rapidly. Furthermore, after each detection of a negative cycle, a new distance matrix will have to be plugged into the Bellmann-Ford algorithm to find the feasible solutions. In addition, the cost of keeping track of the distances chosen for unknown distance pairs increases with the number of such pairs.

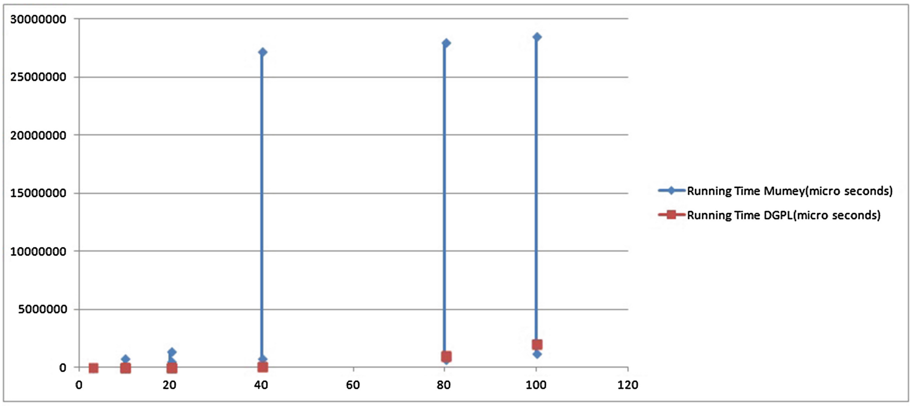

Fig. 3. Time complexity graph Mumey's vs DGPL algorithm - Increasing number of unknown distances between fixed number of points

5 Three Dimensional Embedding

The steps applied to generate the coordinates in three-dimensions using the DGPL program are slightly different from the generation of the coordinates in one-dimension. The input to DGPL are a set of upper and lower distance intervals for each pair of points, with three as the embedding dimension. Steps 1 to 5 are the same in this case also. Finally, the eigenvalue decomposition of the

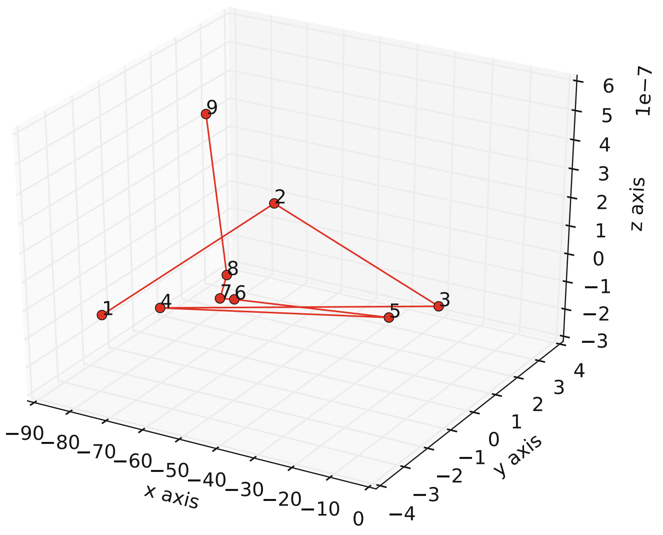

Fig. 4. A sample three-dimensional embedding of 9 points generated by the DGPL program

B matrix is obtained and the product of the three largest eigenvalues with their corresponding normalized eigenvectors yield the three-dimensional coordinates of all the points. A sample plot of a three-dimensional embedding generated by DGPL is shown in Fig. 4.

6 Conclusions

Summary of work done: The probe-location problem is an important one in Computational Genomics. In this paper we have proposed an efficient algorithm for its solution. The novelty of our contribution lies in the use of the distance geometry framework. The core of this work took the form of experiments. A representative set of results from these experiments have been presented in an earlier section. Both Mumey's algorithm (see Sect. 3) and ours were tested with different sets of points, ranging in size from 3 to 100, each with a different set of unknown distances. As the results show, the run-time of Mumey's algorithm is highly sensitive to the number of unknown distances, while that of DGPL is almost impervious to this.

Some interesting conclusions can be drawn from the experiments of the previous section. For example, with 15 unknown distances for a set 80 points, the DGPL algorithm shows a vastly superior performance as compared to that of Mumey, since the latter has to keep track of the distances chosen between each pair of unknown distances. The results have been shown in a graph with the coordinates being plotted in one-dimension. This final graph can also be used as an aid to validate the correctness of the placement of points by verifying all coordinate values against the initial layout generated by the program.

Future directions: Further work can be done on several fronts. Except for the point placement problem in one dimension, the challenging problem of finding lower bounds on the number of pairwise distances needed to embed a set of points in three and higher dimensions has not been addressed in the surveyed literature.

In our earlier approaches to the point placement problem, when the basic point-placement graph (such as the 5-cycle, 5:5 jewel etc.) was not rigid, we formulated rigidity conditions that were met over two or more rounds of queries. Now, if the distances returned by the adversary are not exact, there arises the challenging problem of satisfying the rigidity conditions, using exact lengths as well as the distance bounds returned by the adversary, over multiple query rounds.

Another important and useful line of work would be to test the DGPL program on NMR (Nuclear Magnetic Resonance) data for macromolecules, particularly protein molecules. The NMR technology is used to obtain pairwise distance data of the atoms of molecules that are not available in crystalline form. The process of obtaining coordinates of the atoms from the distance data is well-known as the molecular conformation problem. This would be a good test of the robustness of our implementation of the DGPL algorithm.

References

1. Blumenthal, L.M.: Theory and Applications of Distance Geometry. Chelsea, New York (1970)
2. Chin, F.Y.L., Leung, H.C.M., Sung, W.K., Yiu, S.M.: The point placement problem on a line – improved bounds for pairwise distance queries. In: Giancarlo, R., Hannenhalli, S. (eds.) WABI 2007. LNCS (LNBI), vol. 4645, pp. 372–382. Springer, Heidelberg (2007)
3. Damaschke, P.: Point placement on the line by distance data. Discrete Appl. Math. **127**(1), 53–62 (2003)
4. Eckart, C., Young, G.: The approximation of one matrix by another of lower rank. Psychometrika **1**(3), 211–218 (1936)
5. Malliavin, T.E., Mucherino, A., Nilges, M.: Distance geometry in structural biology: new perspectives. In: Distance Geometry, pp. 329–350. Springer, New York (2013)
6. Mukhopadhyay, A., Sarker, P.K., Kannan, K.K.V.: Randomized versus deterministic point placement algorithms: an experimental study. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9156, pp. 185–196. Springer, Heidelberg (2015)
7. Roy, K., Panigrahi, S.C., Mukhopadhyay, A.: Multiple alignment of structures using center of proteins. In: Harrison, R., Li, Y., Măndoiu, I. (eds.) ISBRA 2015. LNCS, vol. 9096, pp. 284–296. Springer, Heidelberg (2015)
8. Havel, T.F.: Distance geometry: theory, algorithms, and chemical applications. Encycl. Comput. Chem. **120**, 723–742 (1998)
9. Mumey, B.: Probe location in the presence of errors: a problem from DNA mapping. Discrete Appl. Math. **104**(1), 187–201 (2000)
10. Young, G., Householder, A.S.: Discussion of a set of points in terms of their mutual distances. Psychometrika **3**(1), 19–22 (1938)
11. Saxe, J.B.: Embeddability of weighted graphs in k-space is strongly NP-hard. In: 17th Allerton Conference on Communication, Control and Computing, pp. 480–489 (1979)
12. Mukhopadhyay, A., Rao, S.V., Pardeshi, S., Gundlapalli, S.: Linear layouts of weakly triangulated graphs. In: Pal, S.P., Sadakane, K. (eds.) WALCOM 2014. LNCS, vol. 8344, pp. 322–336. Springer, Heidelberg (2014)
13. Alam, M.S., Mukhopadhyay, A.: More on generalized jewels and the point placement problem. J. Graph Algorithms Appl. **18**(1), 133–173 (2014)
14. Alam, M.S., Mukhopadhyay, A.: Three paths to point placement. In: Ganguly, S., Krishnamurti, R. (eds.) CALDAM 2015. LNCS, vol. 8959, pp. 33–44. Springer, Heidelberg (2015)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company (1989)
16. Buetow, K.H., Chakravarti, A.: Multipoint gene mapping using seriation. i. general methods. Am. J. Hum. Genet. **41**(2), 180 (1987)
17. Pinkerton, B.: Results of a simulated annealing algorithm for fish mapping. Communicated by Dr. Larry Ruzzo, University of Washington (1993)

18. Redstone, J., Ruzzo, W.L.: Algorithms for a simple point placement problem. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 32–43. Springer, Heidelberg (2000)
19. Crippen, G.M., Havel, T.F.: Distance Geometry and Molecular Conformation, vol. 74. Research Studies Press Somerset, England (1988)
20. Dress, A.W.M., Havel, T.F.: Shortest-path problems and molecular conformation. Discrete Appl. Math. **19**(1–3), 129–144 (1988)