Randomized Versus Deterministic Point Placement Algorithms: An Experimental Study

Asish Mukhopadhyay(✉), Pijus Kumar Sarker, and Kishore Kumar Varadharajan Kannan

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

Abstract. The point location problem is to determine the position of n distinct points on a line, up to translation and reflection by the fewest possible pairwise (adversarial) distance queries. In this paper we report on an experimental study of a number of deterministic point placement algorithms and an incremental randomized algorithm, with the goal of obtaining a greater insight into the behavior of these algorithms, particularly of the randomized one

Keywords: Computational geometry · Point-placement · Turnpike problem · Experimental algorithms

1 Introduction

The Point Placement Problem: Let $P = \{p_1, p_2, ..., p_n\}$ be a set of n distinct points on a line L. The point location problem is to determine the locations of the points uniquely (up to translation and reflection) by making the fewest possible pairwise distance queries of an adversary. The queries can be made in one or more rounds and are modeled as a graph whose nodes represent the points and there is an edge connecting two points if the distance between the corresponding points is being queried. The distances between the pairs of points returned by the adversary are exact.

A special version of this problem is when a query graph is presented with assigned edge lengths and all possible placements of its vertices are to be determined. In [1], this problem was solved for weakly triangulated graphs.

A classical version of this problem is the construction of the coordinates of a set of n points, given exact distances between all pairs of points (see [2], [3]). Algorithms exist that not only determine the coordinates but also the minimum dimension in which the points can be embedded (see [4]).

Motivation: The motivation for studying this problem stems from the fact that it arises in diverse areas of research, to wit computational biology, learning theory, computational geometry, etc.

A. Mukhopadhyay—Research supported by an NSERC Discovery Grant.

In learning theory [5] this problem is one of learning a set of points on a line non-adaptively, when learning has to proceed based on a fixed set of given distances, or adaptively when learning proceeds in rounds, with the edges queried in one round depending on those queried in the previous rounds.

The version of this problem studied in Computational Geometry is known as the turnpike problem. The description is as follows. On an expressway stretching from town A to town B there are several gas exits; the distances between all pairs of exits are known. The problem is to determine the geometric locations of these exits. This problem was first studied by Skiena *et al.* [6] who proposed a practical heuristic for the reconstruction. A polynomial time algorithm was given by Daurat *et al.* [7].

In computational biology, it appears in the guise of the restriction site mapping problem. Biologists discovered that certain restriction enzymes cleave a DNA sequence at specific sites known as restriction sites. For example, it was discovered by Smith and Wilcox [8] that the restriction enzyme Hind II cleaves DNA sequences at the restriction sites GTGCAC or GTTAAC. In lab experiments, by means of fluorescent in situ hybridization (FISH experiments) biologists are able to measure the lengths of such cleaved DNA strings. Given the distances (measured by the number of intervening nucleotides) between all pairs of restriction sites, the task is to determine the exact locations of the restriction sites. The point location problem also has close ties with the probe location problem in computational biology (see [9])

The turnpike problem and the restriction mapping problem are identical, except for the unit of distance involved; in both of these we seek to fit a set of points to a given set of inter-point distances. As is well-known, the solution may not be unique and the running time is polynomial in the number of points. While the point placement problem, prima facie, bears a resemblance to these two problems it is different in its formulation - we are allowed to make pairwise distance queries among a distinct set of labeled points. It turns out that it is possible to determine a unique placement of the points up to translation and reflection in time that is linear in the number of points.

Overview of Contents: In the next section we briefly review some of the well-known deterministic algorithms and the only known incremental randomized algorithm. In the following section we report on the experimental results obtained by careful implementations of several deterministic algorithms and the incremental randomized algorithm. This is followed by a detailed discussion of the results and we conclude in the next section.

2 Overview of Some Current Point Placement Algorithms

Several algorithms are extant that work in one or more rounds. The current state of the art is summarized in Table 1.

Comment: The $9n/8$ lower bound on 2-round algorithms was proved in [10], improving the lower bound of $30n/29$ by Damaschke [5] and the subsequent

Table 1. *The current state of the art*

Algorithm	Rounds	Query Complexity		Time Complexity
		Upper Bound	Lower Bound	
3-cycle	1	$2n-3$	$4n/3$	$O(n)$
4-cycle	2	$3n/2$	$9n/8$	$O(n)$
5-cycle	2	$4n/3+O(\sqrt{n})$	$9n/8$	$O(n)$
5:5 jewel	2	$10n/7+O(1)$	$9n/8$	$O(n)$
6:6 jewel	2	$4n/3+O(1)$	$9n/8$	$O(n)$
3-path	2	$9n/7$	$9n/8$	$O(n)$
randomized	2	$n+O(n/\log n)$	?	$O(n^2/\log n)$

improvement to $17n/16$ by [11] and the further improvement to $12n/11$ by [12]. As for the lower bound on 1-round algorithms, the following result was proved in [5].

Theorem 1. *[5] The density of any line rigid graph is* $4/3$ *with the exception of the jewel,* $K_{2,3}, K_3$ *and* K_4^- *(shown in Fig 1).*

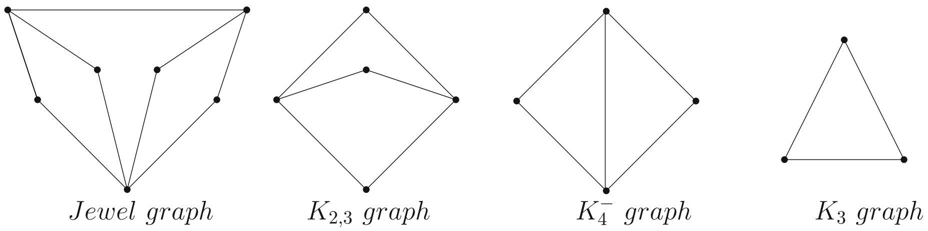

Fig. 1. Graphs quoted in Theorem1.

The density, multiplied by n, gives the lower bound of $4n/3$.

The simplest of all, the 3-cycle 1-round algorithm, has the query graph shown in Fig. 2:

The query complexity of this algorithm is $2n-3$ self-evident as this is the number of edges in the graph. The 4-cycle 2-round algorithm is typical of the other 2-round algorithms listed in Table 1 and thus merits a brief description.

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 nodes V on a line such that the distances between adjacent nodes are consistent with l. We express this by the notation (G,l). By definition (G,l) is said to be line rigid if there is a unique placement up to translation and reflection, while G is said to be line rigid if (G,l) is line rigid for every valid l. A 3-cycle (or triangle) graph is line rigid,

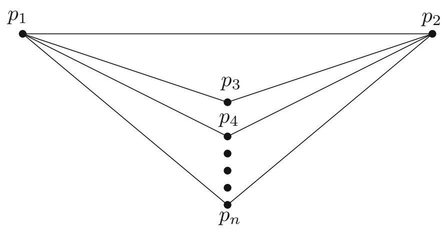

Fig. 2. Query graph using triangles.

which is why the 3-cycle algorithm needs only one round to fix the placement of all the points. A 4-cycle (or quadrilateral) is not line rigid, as there exists an assignment of lengths that makes it a parallelogram whose vertices have two different placements as in Fig. 3.

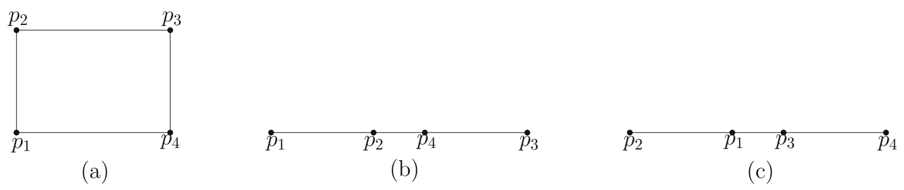

Fig. 3. Two different placements of a parallelogram *abcd*

4-cycle Algorithm

For this algorithm, the query graph presented to the adversary in the first round has the structure shown in Fig. 4.

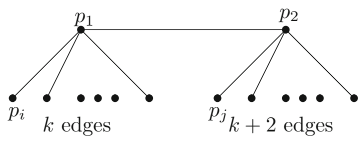

Fig. 4. Query graph for first round in a 2-round algorithm using quadrilaterals.

Making use of the following simple but useful observation,

Observation 1. *At most two points can be at the same distance from a given point p on a line L,* in the second round we query edges connecting pairs of leaves, one from the group of size k and the other from the group of size $k+2$, making quadrilaterals that are not parallelograms (the rigidity condition $|p_1p_i| \neq |p_2p_j|$ ensures that the quadrilateral $p_1p_ip_jp_2$ is not a parallelogram).

5-cycle Algorithm

In the 5-cycle algorithm [11], the query graph submitted to the adversary in the first round is shown in Fig. 5.

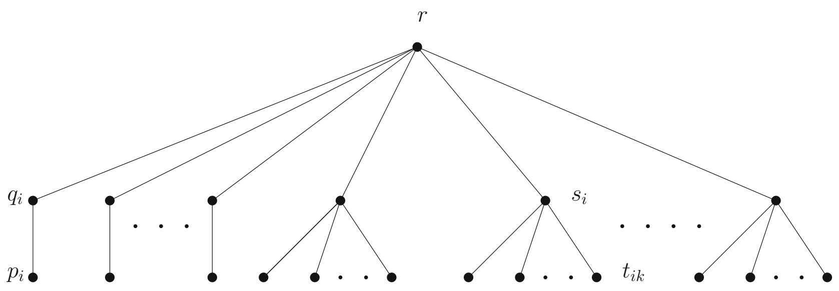

Fig. 5. *Query graph for the 5-cycle algorithm*

Each five cycle is completed by selecting edges to ensure that the following rigidity conditions are satisfied. For more details on this algorithm see [11].

1. $|p_iq_i| \neq |rs_j|$
2. $|p_iq_i| \neq |s_jt_{jk}|$
3. $|p_iq_i| \neq ||rs_j| \pm |s_jt_{jk}||$
4. $|s_jt_{jk}| \neq |q_ir|$
5. $|s_jt_{jk}| \neq ||p_iq_i| \pm |q_ir||$

3-path Algorithm

In the 3-path algorithm [13], the query graph submitted to the adversary in the first round is shown in Fig. 6.

In the second round, the algorithm select edges suitably to satisfy the following rigidity conditions.

1. $|p_1p_2| \notin \{|r_1s|, |r_2s|, ||r_1s| \pm |r_2s||\}$,
2. $|p_2p_3| \notin \{|r_2s|, |r_3s|, ||r_2s| \pm |r_3s||\}$,
3. $|p_3p_1| \notin \{|r_3s|, |r_1s|, ||r_3s| \pm |r_1s||\}$,
4. $|p_1q_1| \notin \{|r_1s|, |r_2s|, ||r_1s| \pm |r_2s||, ||p_1p_2| \pm |r_1s||, ||p_1p_2| \pm |r_2s||, ||p_1p_3| \pm |r_1s||, ||p_1p_3| \pm |r_3s||, ||p_1p_2| \pm |r_1s| \pm |r_2s||, ||p_1p_3| \pm |r_1s| \pm |r_3s||\}$,
5. $|p_2q_2| \notin \{|r_1s|, |r_2s|, |p_1q_1|, ||r_1s| \pm |r_2s||, ||p_1p_2| \pm |r_1s||, ||p_1p_2| \pm |r_2s||, ||p_2p_3| \pm |r_2s||, ||p_2p_3| \pm |r_3s||, ||p_1q_1| \pm |r_1s||, ||p_1q_1| \pm |r_2s||, ||p_1p_2| \pm |r_1s| \pm |r_2s||, ||p_2p_3| \pm |r_2s| \pm |r_3s||, ||p_1q_1| \pm |r_1s| \pm |r_2s||, ||p_1q_1| \pm |p_1p_2| \pm |r_1s||, ||p_1q_1| \pm |p_1p_2| \pm |r_2s||, ||p_1q_1| \pm |p_1p_2| \pm |r_1s| \pm |r_2s||\}$,

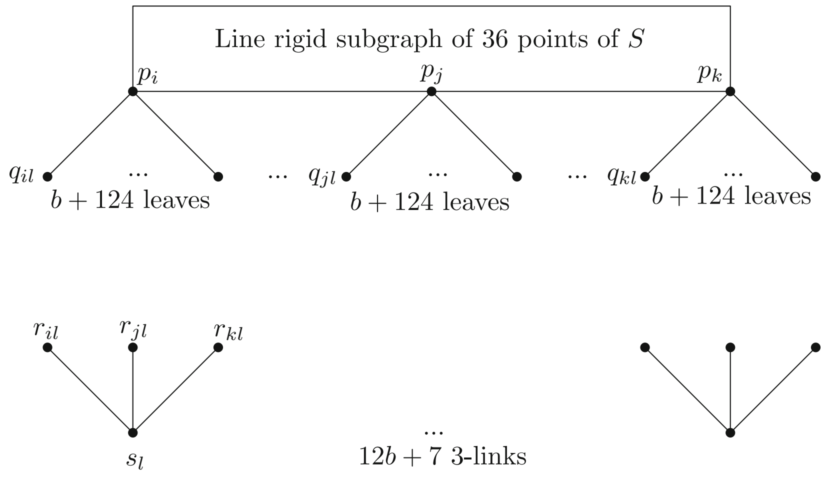

Fig. 6. *Query graph for the 3-path algorithm*

6. $|p_3q_3| \notin \{|r_1s|, |r_2s|, |r_3s|, |p_1q_1|, |p_2q_2|, ||r_2s| \pm |r_3s||, ||r_3s| \pm |r_1s||, ||p_1p_3| \pm |r_3s||, ||p_2p_3| \pm |r_3s||, ||p_1q_1| \pm |r_1s||, ||p_1q_1| \pm |r_3s||, ||p_2q_2| \pm |r_2s||, ||p_2q_2| \pm |r_3s||, ||p_1p_3| \pm |r_1s| \pm |r_3s||, ||p_2p_3| \pm |r_2s| \pm |r_3s||, ||p_1q_1| \pm |r_1s| \pm |r_3s||, ||p_2q_2| \pm |r_2s| \pm |r_3s||, ||p_1q_1| \pm |p_1p_3| \pm |r_3s||, ||p_2q_2| \pm |p_2p_3| \pm |r_3s||, ||p_1q_1| \pm |p_1p_3| \pm |r_1s| \pm |r_2s||, ||p_2q_2| \pm |p_2p_3| \pm |r_2s| \pm |r_3s||\}$.

on each 3-path component shown in Fig. 7. For more details on this algorithm see [13].

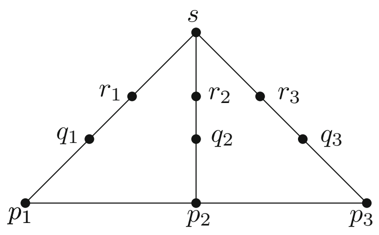

Fig. 7. *A 3-path component*

Randomized Algorithm

Damaschke [14] proposed an incremental randomized algorithm (for an introduction to randomized algorithms see [15]) that expands a set L of points whose positions have been fixed. The set L is initialized by picking an arbitrary point p_0 from S and setting it as the origin of the line on which the points lie. Relative to p_0 a random path $P = p_0p_1p_2...$ is incrementally constructed by choosing a point p_i at random from the set $S - L$, and measuring the distance $d(p_i, p_{i+1})$ for each $i = 0, 1, 2, ...$ Simultaneously, the algorithm maintains all possible signed sums $\pm d(p_0p_1) \pm d(p_1p_2) \pm \cdots \pm d(p_i, p_{i+1}) \cdots$, until for some p_{k+1} the signed sums are no longer all distinct.

If a signed sum that repeats is the actual distance of p_{k+1} from p_0, then the placement of p_k relative to p_{k+1} becomes ambiguous. We stop at this point, query the distance $d(p_0, p_k)$ and use the signed sum equal to this distance to fix the placements on L of all the points on the path from p_1 to p_k (in Damaschke's description the position of p_k is fixed relative to two points in L and the signed sum corresponding to this position is chosen to fix the placements of the other points on the path constructed thus far).

Resetting p_k as the new p_0 and p_{k+1} as the new p_1, the algorithm repeats until $L = S$.

Damaschke proved the following result.

Theorem 1. *[14] The above randomized algorithm for the point location problem has, for any instance, performance ratio 1 + O(1/ log n) with high probability.*

The term performance ratio is the number of distance queries divided by the number of points.

It is straightforward to turn this into a 2-round algorithm. Fix the placement of 2 points p_0 and p_1 and choose a random path $P = p_1p_2 \ldots p_n$ on all the remaining points to be placed and submit this query graph to the adversary. As before, we compute signed sums, stopping when two signed sums are equal when we have reached the point p_{k+1} on P. We resolve the ambiguity in the placement of p_{k+1} by adding edges from p_{k+1} to p_0 and p_1, whose lengths we will query in the second round. Continue as in the incremental algorithm from p_{k+1} on.

3 Experimental Results

We implemented all the four deterministic algorithms and the 2-round version of the incremental randomized algorithm, discussed in the previous section. The control parameters used for comparing their performances are: query complexity and time complexity. The results of the experiments for the deterministic algorithms are shown in the graphs below. In our experiments, we simulated an adversary by creating a linear layout and checking the placements of the points by the algorithms against this. This also solved the problem of ensuring a valid assignment of lengths to the queried edges. We will have more to say about this in the next section.

Predictably enough, the above chart shows that the behavior of the algorithms with respect to query complexity is consistent with the upper bounds for these algorithms shown in Table 1. Each of these algorithms were run on points sets of different sizes, up to 50000 points.

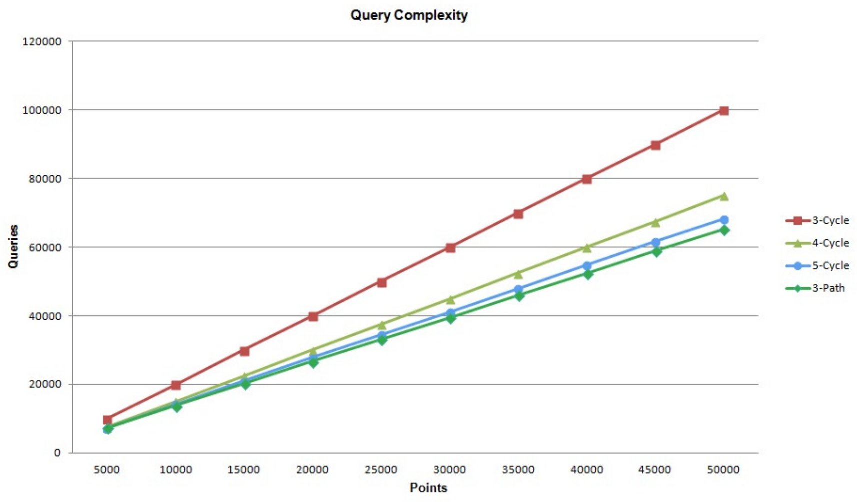

Fig. 8. *Query Complexity Graph*

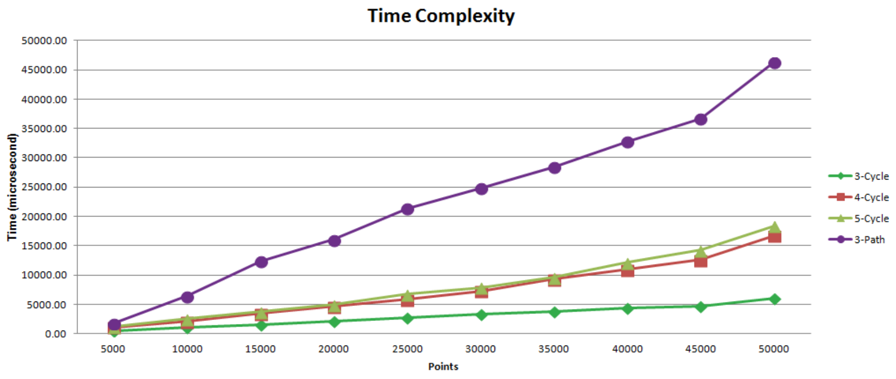

Fig. 9. *Time Complexity Graph*

Clearly, 3-cycle is consistently the fastest; but despite its complex structure the 3-path algorithm does well as compared to the 4-cycle and the 5-cycle algorithms. We have not included the performance of the randomized algorithm in the above graphs as it is incredibly slow and we ran it for point sets of size up to 16,000. Table 2 below shows its performance details.

4 Discussion

The behavior of the deterministic algorithms with respect to time complexity is opposite to their behavior with respect to query complexity. The growth-rate of

Table 2. Performance of 2-round randomized algorithm

Number of points	Number of Distance Queries	Running time (hrs:mins:secs)
2000	2382	0:10:41
4000	4712	0:57:32
6000	7048	2:25:38
8000	9348	5:27:53
10000	11668	8:38:34
12000	13999	13:25:24
14000	16282	18:34:58
16000	18625	23:19:40

the running time versus the size of the input point-set is also near-linear. Both results are as expected.

As reported, in none of the deterministic algorithms it was explicitly stated how to obtain an actual layout from the rigid graph constructed on the input point set. In our implementations we devised a signed-sum technique to generate a layout.

The assumption that an assignment of lengths is valid is a strong one and, as mentioned earlier, we circumvented this problem by creating a layout and reporting queried lengths based on this. The correctness of the placements of the points by an algorithm is verified by checking that it generates a layout identical to the one used to report queried lengths.

An algorithmic approach to the solution of this problem is based on constructing the Cayley-Menger matrix out of the squared distances of a query graph.

For a query graph with n vertices, the pre-distance matrix $D = [D_{ij}]$ is a symmetric matrix such that $D_{ij} = d_{ij}^2$, where d_{ij} is the distance between the vertices (points) i and j of the query graph. The Cayley-Menger matrix, $C = [C_{ij}]$ is a symmetric $(n+1) \times (n+1)$ matrix such $C_{0i} = C_{i0} = 1$ for $0 < i \leq n$, $C_{00} = 0$ and $C_{ij} = D_{ij}$ for $1 \leq i, j \leq n$ [16], [2].

The vertices of the query graph has a valid linear placement provided the rank of the matrix B is at most 3 (this is a special case of the result that there exists a d-dimensional embedding of the query graph if the rank of B is at most $d+2$; our claim follows by setting $d = 1$) [2].

It's interesting to check this out for the query graph in Fig. 10 on 3 points. The Cayley-Menger matrix B for the above query graph is:

$$B = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & x^2 \\ 1 & 1 & 0 & 4 \\ 1 & x^2 & 4 & 0 \end{bmatrix},$$

where $x = d_{13}$, the unknown distance between the points p_1 and p_3.

By the above result, the 4×4 minor, det(B) $= 0$. This leads to the equation

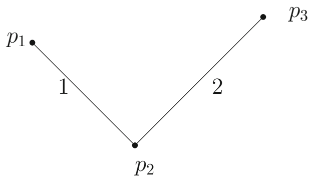

Fig. 10. *A query graph on 3 vertices*

$$x^4 - 10x^2 + 9 = 0$$

which has two solutions $x = 3$ and $x = 1$, corresponding to the two possible placements (embeddings) of the points p_1, p_2 and p_3. Assuming p_2 is placed to the right of p_1, in one of these placements p_3 is to the right of both p_1 and p_2; in the other, to the left of them both.

4.1 Deterministic versus Randomized

Table 2 lends credence to the claim by Damaschke [14] that the number of distance queries of the incremental randomized algorithm is bounded above by $O(n(1 + 1/\log n))$ in the worst case. Unfortunately, it is too slow to be run with very large inputs.

We suspect that the number of times signed sums become equal is intimately connected with the distribution of the points that we generate by pretending to be the adversary. To test this we generated the layout by picking a point at random in a fixed size interval, and picking the next random point in the same fixed-size interval whose left end point is the last point selected. In our experiments we varied this fixed interval from 5 units to 500000 units and reported the number of times we got equal signed sums for points sets of sizes varying from 20 to 1000. Interestingly enough, as can be seen from Table 3 below that the numbers decrease as the interval-size increases.

Table 3. Performance of incremental randomized algorithm for nearly uniform distributions

	Range									
# of points	1-5	1-10	1-20	1-50	1-100	$1-10^3$	$1-10^4$	$1-5*10^4$	$1-10^5$	$1-5*10^5$
20	7	7	6	5	4	3	3	2	2	1
50	16	13	11	10	9	7	6	6	5	4
100	25	23	20	19	15	11	9	8	8	7
200	45	39	35	33	29	22	18	17	16	
400	78	70	61	56	49	41	39	34		
1000	167	149	140	123	111	94	82	76		

The incremental randomized algorithm is often held up as an example of simplicity in comparison to deterministic algorithms, like the 3-path one, for example. The above experiments paint a completely different picture. From a practical point of view, it is completely ineffective as it is essentially a brute-force algorithm. The 3-path algorithm, on the other hand, scores high on both parameters - low query complexity and low time complexity.

5 Conclusions

All algorithms have been implemented in C on a computer with the following configuration: Intel(R) Xeon(R) CPU, X7460 @ 2.66GHz OS: Ubuntu 12.04.5, Architecture: i686.

Further work can be done on several fronts. Particularly worthwhile is to conduct further experiments into the behavior of the randomized algorithm, specifically the influence of floating point arithmetic on keeping signed sums unequal. On the theoretical side, it might be interesting to come up with a completely different randomized algorithm - one that does not depend on maintaining an exponential number of signed sums.

References

1. 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)
2. Young, G., Householder, A.S.: Discussion of a set of points in terms of their mutual distances. Psychometrika **3**(1), 19–22 (1938)
3. Blumenthal, L.M.: Theory and applications of distance geometry, 2nd edn. Chelsea, New York (1970)
4. Crippen, G., Havel, T.: Distance Geometry and Molecular Conformation. John Wiley and Sons (1988)
5. Damaschke, P.: Point placement on the line by distance data. Discrete Applied Mathematics **127**(1), 53–62 (2003)
6. Skiena, S.S., Smith, W.D., Lemke, P.: Reconstructing sets from interpoint distances (extended abstract). In: Proceedings of the Sixth Annual Symposium on Computational Geometry, SCG 1990, pp. 332–339. ACM, New York (1990)
7. Daurat, A., Gérard, Y., Nivat, M.: The chords' problem. Theor. Comput. Sci. **282**(2), 319–336 (2002)
8. Smith, H., Wilcox, K.: A restriction enzyme from hemophilus influenzae. i. purification and general properties. Journal of Molecular Biology **51**, 379–391 (1970)
9. 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)
10. 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)

11. 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)
12. Alam, M.S., Mukhopadhyay, A.: More on generalized jewels and the point placement problem. J. Graph Algorithms Appl. **18**(1), 133–173 (2014)
13. Alam, M.S., Mukhopadhyay, A.: Improved upper and lower bounds for the point placement problem. CoRR abs/1210.3833 (2012)
14. Damaschke, P.: Randomized vs. deterministic distance query strategies for point location on the line. Discrete Applied Mathematics **154**(3), 478–484 (2006)
15. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
16. Emiris, I.Z., Psarros, I.D.: Counting euclidean embeddings of rigid graphs. CoRR abs/1402.1484 (2014)