Point Placement Algorithms: An Experimental Study

Asish Mukhopadhyay *asishm@uwindsor.ca*
School of Computer Science
University of Windsor
Windsor, ON N9B 3P4
Canada

Pijus K. Sarker *sarkerp@uwindsor.ca*
ValidateIt Technologies Inc.
478 Queen Street East, Suite# 400, Toronto, ON M5A 1T7
Canada

Kishore Kumar V. Kannan *varadhak@uwindsor.ca*
IBM Canada Ltd.
8200 Warden Avenue
Markham ON L6G 1C7
Canada

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 practical utility of these algorithms, particularly of the randomized one.

Keywords: Computational Geometry, Point-placement, Turnpike Problem, Experimental Algorithms.

1. INTRODUCTION

The point placement problem: Let $P = \{p_1, p_2, \ldots, p_n\}$ 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. It is assumed that the distances returned by the adversary are exact and valid. The assumption of validity means that there exists a placement consistent with these distances. The queries are presented to the adversary as a graph, called a point placement graph (*ppg*) or query graph, whose edges connect pairs of points whose distances are being queried. Depending upon the adversary's answers, the *ppg* is adaptively modified by inserting additional edges and querying these. This process is repeated over the successive rounds. A *ppg G* is said to be line rigid if (or simply rigid) if its vertices have a unique placement on *L* for a valid assignment of lengths to its edges.

A non-adversarial version of the above problem is this: a query graph is presented with assigned edge lengths and all possible linear placements of its vertices are to be determined. In [1] this problem was solved in polynomial time for weakly triangulated graphs. However, Saxe [2] showed that the linear embedding problem for an arbitrary graph whose edge-lengths are positive integers is strongly NP-complete.

An offline 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 [3], [4]). Algorithms exist that not only determine the coordinates but also the minimum dimension in which the points can be embedded (see [6]). A more challenging version of this problem is when all the entries of the distance matrix are not known and necessary and sufficient conditions are sought under which the unknown distance entries can be determined. This is the well-known and extensively studied distance matrix completion problem [5], [6], [7].

Motivation: The point-placement problem appears in different guises in several diverse areas of research, to wit computational biology, learning theory, computational geometry, etc.

In learning theory [8] 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.* [9] who proposed a practical heuristic for the reconstruction. A polynomial time algorithm was given by Daurat *et al.* [10].

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 [11] 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 [12]).

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.

This paper has several objectives. The point placement algorithms that we have designed rely critically on the correct and exhaustive enumeration of a large number of rigidity conditions (the algorithm based on the 6:6 jewel, for example). One of the goals of our implementations was to obtain certificates of their correctness. While theoretical comparisons of the query complexities are possible up to the O-notation, the experiments were designed to obtain exact numbers for very large inputs. More importantly, we wanted to verify our intuition about the relative time complexities of these algorithms. While the randomized algorithm is theoretically very attractive, we were interested in checking how it behaves in practice.

Prior Work: Early research on this problem was reported in [12] [13]. In [8] it was shown that the jewel (see Fig. 1) and $K_{2,3}$ are both rigid, as also how to build large rigid graphs of density 8/5 (this is an asymptotic measure of the number of edges per vertex as the number of vertices go to infinity) out of the jewel. In a subsequent paper, Damaschke [14] proposed a randomized 2-round strategy that makes $(1+o(1))n$ distance queries with high probability and also showed that this is not possible with 2-round deterministic strategies. The computational complexity of this algorithm is exponential, making it a completely impractical point placement algorithm (our implementation confirms this, too). Chin et al. [15] improved many of the results of [8]. Their principal contributions are a 3-round construction of rigid graphs of density 5/4 from 6-cycles and a lower bound of $17n/16$ for any 2-round algorithm. They also introduced the following concept of a *layer graph*, useful for finding conditions that make a *ppg* rigid.

Definition 1 *We first choose two orthogonal directions* **x** *and* **y** *(actually, any 2 non-parallel directions will do). A graph G admits a layer graph drawing if the following 4 properties are satisfied:*

P1 Each edge e of G is parallel to one of the two orthogonal directions **x** *and* **y**.

P2 The length of an edge e is the distance between the corresponding points on L.

P3 Not all edges are along the same direction (thus a layer graph has a two-dimensional extent).

P4 When the layer graph is folded onto a line, by a rotation either to the left or to the right about an edge of the layer graph lying on this line, no two vertices coincide.

Chin *et al.* [15] proved the following result.

Theorem 1 *A ppg G is rigid iff it cannot be drawn as a layer graph.*

In [16] we proposed a 2-round algorithm that makes $4n/3+O(1)$ distance queries to construct rigid *ppg* on n points using a 6:6 jewel as the basic component.

Overview of contents: In the next section we provide a tabular summary of the state-of-the-art of deterministic point placement algorithms and the only known incremental randomized algorithm. In section three, we review several 1-round algorithms. This is followed by a discussion of 2-rounds and 3-rounds algorithm in the next two sections respectively. In section six, 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. STATE-OF-THE-ART FOR POINT PLACEMENT ALGORITHMS

As mentioned in the introduction, the edge queries of a point placement algorithm can be spread over several rounds. Several algorithms are extant that work in one or more rounds. The current state of the art is summarized in Table 1.

Algorithm	Rounds	Query Complexity		Time Complexity
		Upper Bound	Lower Bound	
3-cycle	1	$2n-3$	$4n/3$	$O(n)$
$K_{2,3}$	1	$5n/3-c$	$4n/3$	$O(n)$
4:4 jewel	1	$8n/5-c$	$4n/3$	$O(n)$
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	$n+O(n/\log n)$	*unavailable*	$O(n^2/\log n)$

TABLE 1: The current state of the art.

Definition 2 *The density of a graph G with m edges and n vertices is defined to be m/n.*
Thus the number of edges in a query graph is its density times the number of vertices, *n*.

Comment: The 9*n*/8 lower bound on 2-round algorithms was proved [17], improving the lower bound of 30*n*/29 by Damaschke [8] and the subsequent improvement to 17*n*/16 by [15] and the further improvement to 12*n*/11 by [16]. As for the lower bound on 1-round algorithms, the following result was proved in [8].

Theorem 2 [8] *The density of any line rigid graph is at least 4/3, the only exceptions being the jewel,* $K_{2,3}$*,*K_3 *and* K_4^- *(shown in Fig 1).*

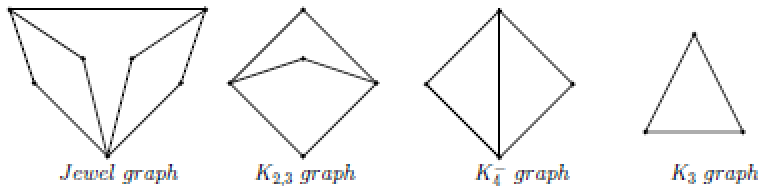

FIGURE 1: Graphs quoted in Theorem 1.

The density, multiplied by the number of vertices *n*, gives the lower bound of 4*n*/3.

3. 1-ROUND ALGORITHMS

1-round algorithms are based on gluing together multiple copies of a simple line rigid graph. We have performed experiments with the graphs $K_{2,3}$, the 4:4 jewel or simply the jewel graph and K_3 , the 3-cycle graph.

The simplest of all, the 3-cycle 1-round algorithm, has the query graph shown in Fig. 2. It is obtained by gluing together multiple copies of a line rigid triangle.

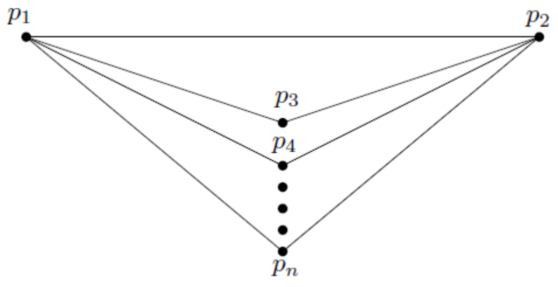

FIGURE 2: Query graph using triangles.

The query complexity of this algorithm is 2*n*–3, as this is the number of edges in the graph.

Using $K_{2,3}$ as the basic line rigid graph, we attach multiple copies of this graph from a common strut (edge). This requires querying 5 edges for each 3 newly-added points. Thus if *n*=3*k*+2, for some *k*≥1, then the number of edges queried is 5*k*+1 or 5*n*/3–10/3. When *n*=3*k* or 3*k*+1, we make triangles with the strut and each of the remaining points. In all cases, the number of edges queried is at most 5*n*/3–*c*, where *c* is at most 4.

Using the jewel (leftmost graph in Fig. 1) as the basic line rigid graph, we attach multiple copies of this graph from a common strut. This requires querying 6 edges for each 5 newly-added points. Thus if *n*=5*k*+2, for some *k*≥1, then the number of edges queried is 8*k*+1 or 8*n*/5–11/5. When *n*=5*k*+*d* where *d*=0,1,3,4, we make triangles (or perhaps a $K_{2,3}$ if there are enough points) with the strut and each of the remaining points. In all cases, the number of edges queried is at most 8*n*/5–*c*, where *c* is a small constant.

4. 2-ROUND ALGORITHMS

We have experimented with three different types of graphs as basic building blocks: the 4-cycle, 5-cycle, 3-path; in addition, we have also implemented a randomized algorithm due to [16].

The 4-cycle 2-round algorithm is typical of the other 2-round algorithms listed in Table 1 and thus merits an extended description.

4-cycle Algorithm

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, 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. This has two implications: the first is that we have to find suitable rigidity conditions by making sure that no drawing layer graph drawing is possible; the second is that the queries will have to be spread over several rounds.

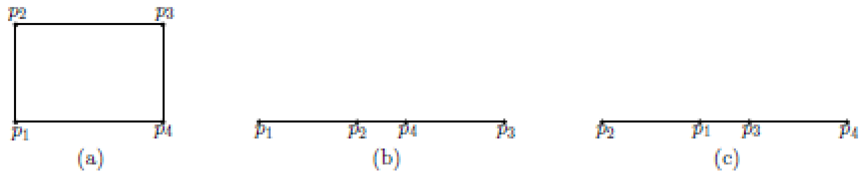

FIGURE 3: Two different placements of a rectangle $p_1p_2p_3p_4$.

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

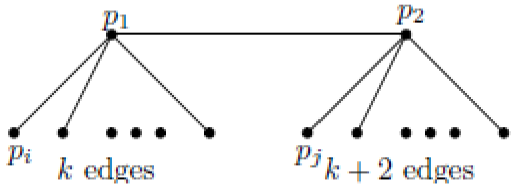

FIGURE 4: Query graph for first round in a 2-round algorithm using quadrilaterals.

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 [15], the query graph submitted to the adversary in the first round is shown in Fig. 5.

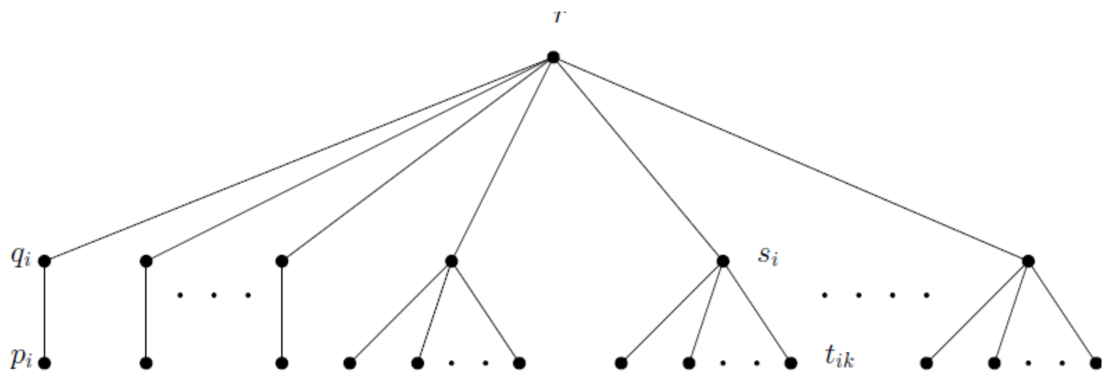

FIGURE 5: Query graph for the 5-cycle algorithm.

Each five cycle is completed by selecting edges to ensure that the following rigidity conditions are satisfied. For more details on this algorithm see [17].

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 [17], the query graph submitted to the adversary in the first round is shown in Fig. 6.

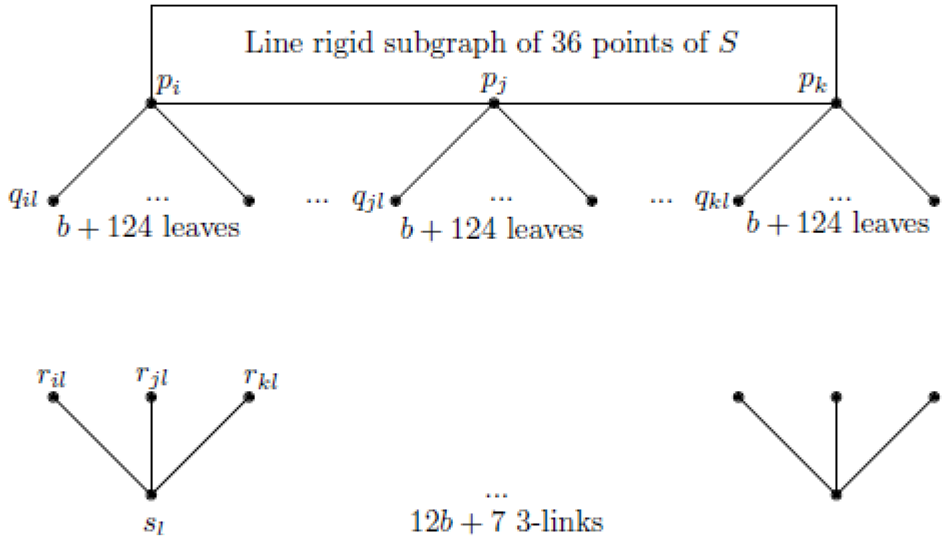

FIGURE 6: Query graph for the 3-path algorithm**.**

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. $|p_2q_2| \notin \{|r_1s|, |r_2s|, |p_1q_1|, ||r_1s| \pm |r_2s||, ||p_1p_2| \pm |r_1s||, ||p_1p_2| \pm |r_2s||, ||p_2p_3| \pm |r_2s||, ||p_2p_3| \pm |r_3s||, ||p_1q_1| \pm |r_1s||, ||p_1q_1| \pm |r_2s||, ||p_1p_2| \pm |r_1s| \pm |r_2s||, ||p_2p_3| \pm |r_2s| \pm |r_3s||, ||p_1q_1| \pm |r_1s| \pm |r_2s||, ||p_1q_1| \pm |p_1p_2| \pm |r_1s||, ||p_1q_1| \pm |p_1p_2| \pm |r_2s||, ||p_1q_1| \pm |p_1p_2| \pm |r_1s| \pm |r_2s||$
$\}$,
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 [17].

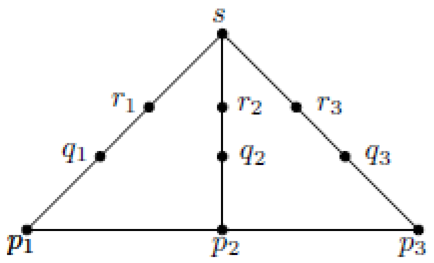

FIGURE 7: A 3-path component.

Randomized Algorithm

Damaschke [14] proposed an incremental randomized algorithm (for an introduction to randomized algorithms see [18]) 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 [14] proved the following result.

Theorem 3 [14] *The above randomized algorithm for the point location problem has, for any instance, performance ratio 1 + O(1/ log n) with high probability.*

Performance ratio is defined as 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.

5. 3-ROUND ALGORITHM

The 3-round algorithm we have implemented is due to Chin et al. [15] and uses the 6-cycle graph as the basic building block. The query graph submitted to the adversary in the first round is shown in Fig. 8.

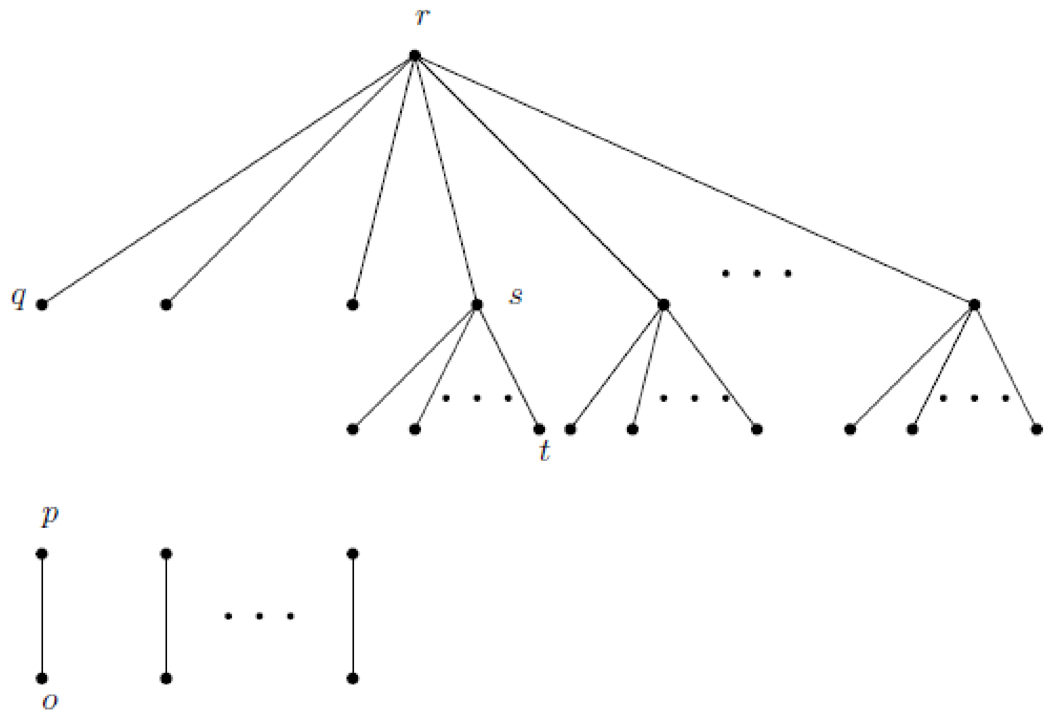

FIGURE 8: Query graph for the 6-cycle algorithm.

Each six cycle is completed over two additional rounds by selecting edges to ensure that the following rigidity conditions are satisfied. For more details on this algorithm see [DBLP\s\do5(c)onf\s\do5(w)abi\s\do5(C)hinLSY07].

1. $|op|\notin\{|qr|,|rs|,|st|,||qr|\pm|rs||,||rs|\pm|st||,||qr|\pm|st||,||qr|\pm|rs|\pm|st||\}$
2. $|pq|\notin\{|rs|,|st|,|rs|\pm|st||\}$
3. $|qr|\notin\{||st|||op|\pm|s_jt_{jk}||\}$
4. $|rs|\neq||op|\pm|pq||$
5. $|st|\notin\{||op|\pm|pq||,||pq|\pm|qr||,||op|\pm|qr||,||op|\pm|pq|\pm|qr||\}$
6. $||op|\pm|pq||\neq||rs|\pm|st||$

6. EXPERIMENTAL RESULTS

We implemented seven deterministic algorithms (three 1-round, three 2-rounds, one 3-rounds) 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.

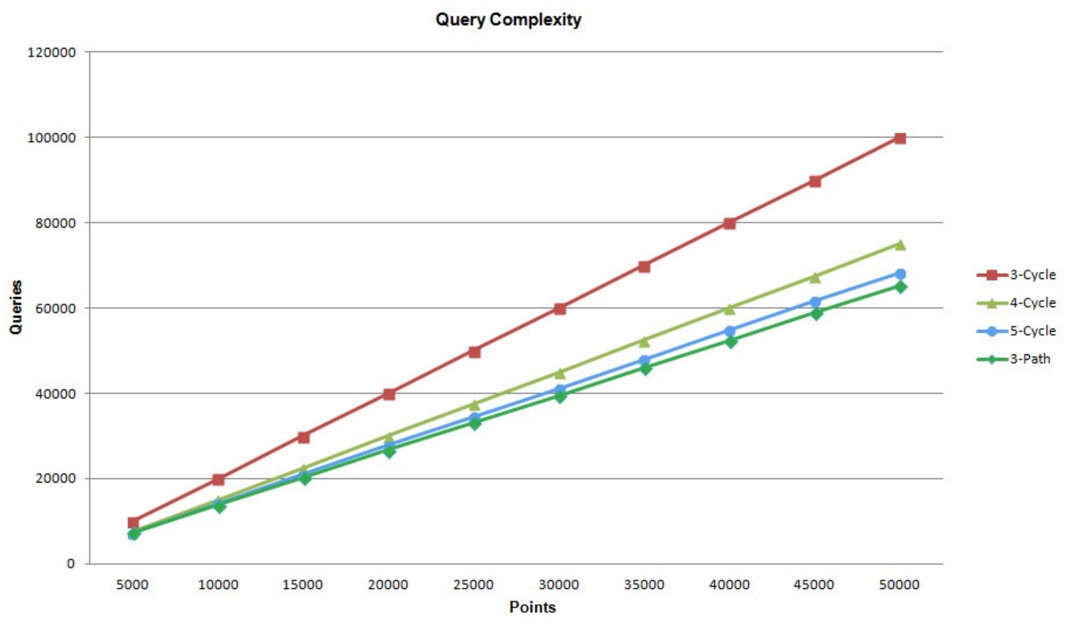

FIGURE 9: Query Complexity Graphs.

Predictably enough, the above chart of Fig. 9 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.

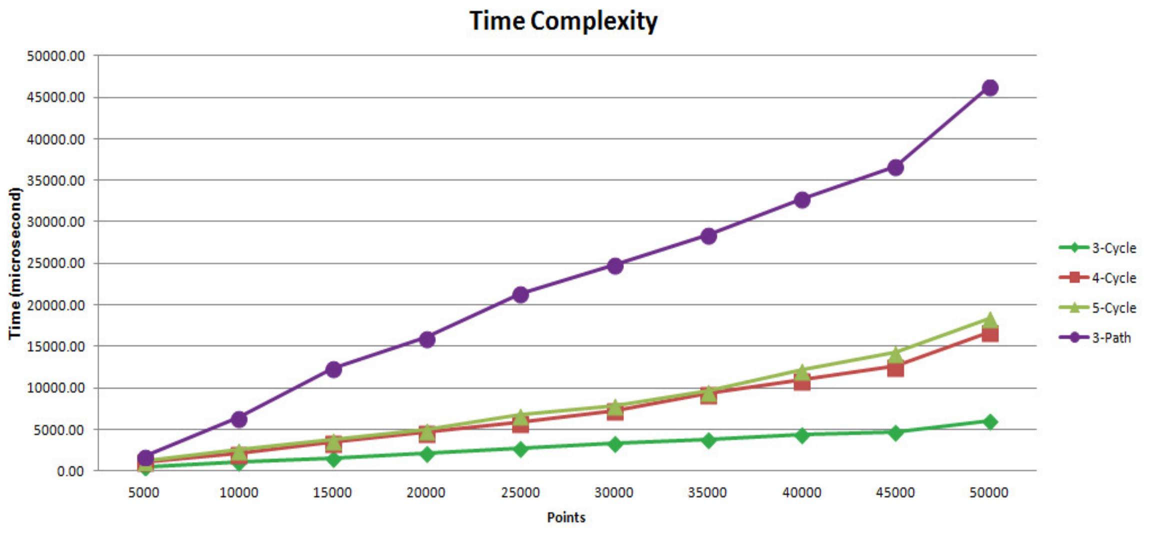

FIGURE 10: Time Complexity Graphs.

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.

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

TABLE 2: Performance of the 2-rounds randomized algorithm.

7. 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 the running time versus the size of the input point-set is also near-linear. Both results are as expected. It is also seen (Fig. 10) that increasing the number of rounds has a noticeable impact on the running time for large input size.

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. It needs be pointed out that we have relied on exact integer arithmetic to test the rigidity conditions in the 2-rounds deterministic algorithms. Floating-point arithmetic can be handled provided exact arithmetic packages are used. Point location in an inexact model is an interesting topic for further research.

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 $1\leq i,j\leq n$, [19].

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 $d=1$, see [20]) .

It's interesting to check this out for the query graph in Fig. 11 on 3 points.

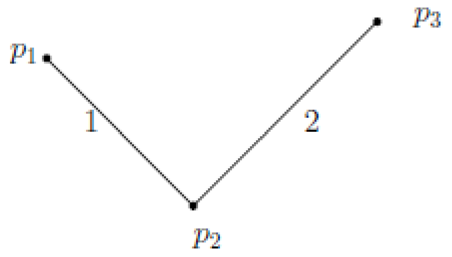

FIGURE 11: A query graph on 3 vertices.

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

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

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.

7.1 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.

	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		

TABLE 3: Performance of incremental randomized algorithm for nearly uniform distributions.

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 deterministic algorithms, on the other hand, score high on both parameters - low query complexity and low time complexity.

8. CONCLUSIONS

In this paper we have reported the first-ever attempt to make an experimental study of algorithms for the point placement problem. The results substantiate the correctness of the intricate theoretical analysis, reported in [16], [17], [22] and [23].

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.

9. REFERENCES

[1] A. Mukhopadhyay, S. Rao, S. Pardeshi and S. Gundlapalli. "Linear Layouts of weakly triangulated graphs". *Workshop on Algorithms and Computation*, Chennai, 2014.

[2] J. Saxe. "Embeddability of weighted graphs in k-space is strongly NP-hard". *17th Allerton Conference on Communication, Control and Computing*, 1979.

[3] G. Young and A. Householder. "Disussion of a set of points in terms of their mutula distances". *Pychometrika,* vol. 3, no. 1, pp. 19-22, 1938.

[4] L. Blumenthal. *Thory and applications of distance geometry, 2nd edition.* New York: Chelsea, 1970.

[5] A. Y. Alfakih, A. Khandani and H. Wolkowicz. "Solving euclidean matrix distance completion problems using semi-definite programming". *Comput. Optim. Applications,* vol. 12, no. 1-3, pp. 13-30, 1999.

[6] M. Laurent. "Matrix completion problems". *Encyclopedia of Optimization, Second Edition*, 2009, pp. 1967-1975.

[7] M. Bakonyi and C. Johnson. "The euclidean matrix distance completion problem". *SIAM J. Matrix Anal. Appl.,* vol. 16, no. 2, pp. 646-654, 1995.

[8] P. Damaschke. "Point placement on the line by distance data". *Discrete Applied Mathematics,* vol. 127, no. 1, p. 53–62, 2003.

[9] S. Skiena, W. Smith and P. Lemke. "Reconstructing sets from interpoint distances". In *Sixth Annual ACM Symposium on Computational Geometry*, New York, 1990.

[10] A. Daurat, Y. Gérard and M. Nivat. "The chords' problem". *Theor. Comput. Sci.,* vol. 282, no. 2, p. 319–336, 2002.

[11] H. Smith and K. Wilcox. "A restriction enzyme from hemophilus influenzae. i. purification and general properties". *Journal of Molecular Biology,* vol. 51, p. 379–391, 1970.

[12] J. Redstone and W. W. L. Ruzzo. "Algorithms for a simple point placement problem". In *CIAC'00:Proceedings of the 4th Italian Conference on Algorithms and Complexity*, London, UK, 2000.

[13] B. Mumey. "Probe location in the presence of errors: a problem from DNA mapping". *Discrete Applied Mathematics,* vol. 104, no. 1-3, pp. 187-201, 2000.

[14] P. Damaschke. "Randomized vs. deterministic distance query strategies for point location on the line". *Discrete Applied Mathematics,* vol. 154, no. 3, pp. 478-484, 2006.

[15] F. Chin, H. Leung, W.-K. Sung and S.-M. Yiu. "The point placement problem on a line - improved bounds for pairwise distance queries". In *Proceedings of the Workshop on Algorithms in Bioinformatics*, 2007.

[16] M. Alam and A. A. Mukhopadhyay. "More on generalized jewels and the point placement problem". *J. Graph Algorithms Appl.,* vol. 18, no. 1, pp. 133-173, 2014.

[17] M. Alam and A. Mukhopadhyay. "Three paths to point placement". In *Proceedings of the Conference on Algorithms and Discrete Applied Mathematics*, 2015.

[18] R. Motawani and P. Raghavan. *Randomized Algorithms*. New York, NY: Cambridge University Press, 1995.

[19] G. Crippen and T. Havel. *Distance Geometry and Molecular Conformation*. John Wiley and Sons, 1988.

[20] I. Emiris and I. Psarros. "Counting Euclidean Embeddings of line rigid graphs". 2014.

[21] A. Mukhopadhyay, P. Sarker and K. Kannan. "Randomized versus deterministic point placement algorithms: An experimental study". In *ICCSA*, Banff, 2015.

[22] M. Alam and A. A. Mukhopadhyay. "A new algorithm and improved lower bound for point placement on a line in two rounds". In *Proceedings of the 22nd Canadian Conference on Computational Geometry*, 2010.

[23] M. Alam and A. Mukhopadhyay. "Improved upper and lower bounds for the point placement problem". 2012.