Algorithms for a Simple Point Placement Problem

Joshua Redstone and Walter L. Ruzzo

Department of Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
{redstone,ruzzo}@cs.washington.edu

Abstract. We consider algorithms for a simple one-dimensional point placement problem: given N points on a line, and noisy measurements of the distances between many pairs of them, estimate the relative positions of the points. Problems of this flavor arise in a variety of contexts. The particular motivating example that inspired this work comes from molecular biology; the points are markers on a chromosome and the goal is to map their positions. The problem is NP-hard under reasonable assumptions. We present two algorithms for computing least squares estimates of the ordering and positions of the markers: a branch and bound algorithm and a highly effective heuristic search algorithm. The branch and bound algorithm is able to solve to optimality problems of 18 markers in about an hour, visiting about 10^6 nodes out of a search space of 10^{16} nodes. The local search algorithm usually was able to find the global minimum of problems of similar size in about one second, and should comfortably handle much larger problem instances.

1 Introduction

The problem of mapping genetic information has been the subject of extensive research since experimenters started breeding fruit flies for physical characteristics. Due to the small scale of chromosomes, it has been difficult to obtain accurate information on their structure. Many techniques relying on statistical inference of indirect data have been applied to deduce this information; see [1] for some examples.

More recently, researchers have developed many techniques for estimating of relative positions various genetic features by more direct physical means. We are interested in one called fluorescent *in situ* hybridization (FISH). In this technique, pairs of fluorescently labeled probes are hybridized (attached) to specific sites on a chromosome. The 2-d projection of the distance between the probes is measured under a microscope. Despite the highly folded state of DNA *in vivo* and the resulting high variance of individual measurements, [10] shows that the genomic distance can be estimated if the experiment is repeated in many cells.

Not surprisingly, if more pairs of probes are measured, and the measurement between each pair is repeated many times, the accuracy of the answer increases. Unfortunately, so does the cost. Hence, the resulting computational problem is the following:

Problem: Given N probes on a line, and an incomplete set of noisy pairwise measurements between probes, determine the best estimate of the ordering and position of the probes.

If the measurements were complete and accurate, the problem would be easy—the farthest pair obviously are the extreme ends, and the intervening points can be placed by sorting their distances to the extremes. However, with partial, noisy data, the problem is known to be NP-hard. (See [6, 5] for a particularly simple proof.)

1.1 Previous Work

Brian Pinkerton previously investigated solving this problem using the seriation algorithm of [3], and a branch and bound algorithm (personal communication, 6/96). The seriation algorithm, which is a local search algorithm, was only moderately effective. The branch and bound algorithm, using a simple bounding function, was able to solve problems involving up to about 16 probes.

There has been extensive work on other algorithms to solve DNA mapping problems, but they are based on distance estimates from techniques other than FISH, and are tailored to the particular statistical properties of the distance measurements. Two among many examples are the distance geometry algorithm of [7], based on recombination frequency data, and [2], which investigated branch and bound, simulated annealing, and maximum likelihood algorithms based on data from radiation hybrid mapping.

1.2 Outline

We present two algorithms for finding least-squares solutions to the probe placement problem. One is a branch and bound algorithm that can find provably optimal solutions to problems of moderate, but practically useful, size. The second is a heuristic search algorithm, fundamentally a "hill-climbing" or greedy algorithm, that is orders of magnitude faster than the branch and bound algorithm, and although it is incapable of giving certifiably optimal solutions, it appears to be highly effective on this data.

In the next section we sketch some of the more difficult aspects of the problem. Section 3 develops a cost function to evaluate solutions. Section 4 describes the heuristic search algorithm. Section 5 outlines the branch and bound algorithm. We then present the results of simulations of the two algorithms in Section 6.

2 Introduction to the Solution Space

Before explaining the development of the algorithms, it is helpful to gain some intuition about the solution space. Given that the data is both noisy and incomplete, the problem can be under-constrained and/or over-constrained. In this domain, a "constraint" refers to a measurement between two probes (since it constrains the placement of the probes).

An under-constrained problem instance is one in which a probe might not have enough measurements to other probes to uniquely determine its position. In the example of four probes in Figure 1, probe B has only one measurement to probe A, and so a

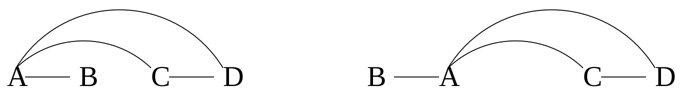

Fig. 1. An example of an under-constrained ordering (Probe B can be placed on either side of probe A). A line between two probes indicates a measurement between the probes.

location on either side of probe A is consistent with the data. It is also important to note that in all solutions, left/right orientation is arbitrary as is the absolute probe position.

In a more extreme example, a set of probes could have no measurements to another set. In Figure 2, probes A and B have no measurements to probes C and D, and placement anywhere relative to probes C and D is consistent with the data.

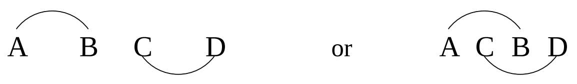

Fig. 2. Another example of an under-constrained ordering

In the examples of Figures 1 and 2, not only are the positions not uniquely determined, but different orderings are possible. When developing search algorithms, we have to be careful to recognize and treat such cases correctly. It appears that in real data such as from [Trask, personal communication, 1996], there are no degrees of freedom in the relative positioning of probes due to the careful choice of pairs of probes to measure. However, under-constrained instances do arise in the branch and bound algorithm described in Section 5 and in any algorithm that solves the problem by examining instances with a reduced set of constraints.

Due to the noise in the data, parts of a problem instance will be over-constrained. For example, as shown in Figure 3, if we examine three probes with pairwise measurements between them and there isn't an ordering such that the sum of two pairwise measurements equals the third pairwise measurement, there will be no way to place the three probes on a line. In this case, the distances between the probes in any linear placement will unavoidably be different from the measured distances.

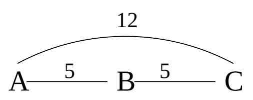

Fig. 3. There is no way to linearly place these probes on a line and respect all the measurements.

Given the existence of over and under-constrained problems, it is necessary to develop a method of evaluating how well a solution conforms to the data. This is covered in Section 3. Once we define how to evaluate a solution, we will develop algorithms to search for the best solution.

3 How to Evaluate a Probe Placement

We construct a cost function to evaluate the "goodness" of a solution, then solve the problem by finding the answer that has the least cost. Let N be the number of probes, x_i be the assigned position of probe i, d_{ij} be the measured distance between probe i and probe j ($d_{ij} = d_{ji}$), and let $w_{ij} = w_{ji}$ be a nonnegative weight associated with the measured distance between probes i and j. We define the *cost* of this placement to be the weighted sum of squares of the differences between the measured distance between two probes and the distance between the probes in the given linear placement of the probes:

$$Cost(x_1, \ldots, x_N) = \sum_{\substack{i<j \\ d_{ij} \text{ measured}}} w_{ij}(\mid x_i - x_j \mid - d_{ij})^2. \tag{1}$$

Many subsequent formulae will be simplified by assuming $w_{ij} = 0$ if $i = j$ or if the distance d_{ij} has not been measured. For example, we could have omitted the qualifier "d_{ij} measured" from Equation 1 under this assumption.

Intuitively, the weight w_{ij} reflects the relative confidence we have in measurement d_{ij}. For example, if the measurement errors were independent normal random variables, then we should choose the weight w_{ij} to be proportional to $1/\sigma_{ij}^2$, where σ_{ij}^2 is the variance of d_{ij}. Least squares solutions under these assumptions have several desirable properties, like being unbiased maximum likelihood estimators. Even though the error distribution in our motivating problem violated these assumptions, choosing weights inversely proportional to the variances substantially improved the solution quality (and speed) of our algorithms; see [9].

3.1 Finding Least Squares Solutions for a Fixed Ordering

One standard approach to solving a least squares problem is to take the partial derivatives of the cost with respect to each of the x_i's, set them equal to 0, and solve. Unfortunately, our cost function is not differentiable due to the absolute value terms. However, for a given *fixed* ordering of the probes we can bypass this difficulty, allowing us to find the placement which minimizes cost for the given ordering. Without loss of generality, assume

$$x_1 < x_2 < \cdots < x_N. \tag{2}$$

Then for a given probe k:

$$\frac{\partial}{\partial x_k}\left(\sum_{i<j} w_{ij}(\mid x_i - x_j \mid - d_{ij})^2\right) = \sum_{1\le i\le k-1} 2w_{ik}(x_k - x_i - d_{ik}) - \sum_{k+1\le i\le N} 2w_{ki}(x_i - x_k - d_{ki}). \quad (3)$$

Separating the terms and setting equal to 0, we get for $\frac{\partial}{\partial x_k}$:

$$x_k \sum_{1\le i\le N}(w_{ik}) + \sum_{1\le i\le N}(-w_{ik}x_i) = \sum_{1\le i\le k-1}(w_{ik}d_{ik}) - \sum_{k+1\le i\le N}(w_{ki}d_{ki}). \quad (4)$$

Equation 4 is of the form

$$\mathbf{Mx} = \mathbf{r} \quad (5)$$

where $\mathbf{x}$ is the vector of x_i's, $\mathbf{M}$ is the matrix defined as:

$$M_{ij} = \begin{cases} -w_{ij} & i \neq j, \\ \sum_{1\le p\le N}(w_{ip}) & i = j, \end{cases} \quad (6)$$

and $\mathbf{r}$ is the vector whose kth component r_k is given by the right hand side of Equation 4. Thus, in matrix form, Equation 4 can be written as:

$$\begin{pmatrix} \cdots \\ -w_{k1} \ \ldots M_{kk} \ldots \ -w_{kN} \\ \cdots \\ \cdots \\ \cdots \end{pmatrix} \begin{pmatrix} \cdots \\ x_k \\ \cdots \\ \cdots \\ \cdots \end{pmatrix} = \begin{pmatrix} \cdots \\ \sum_{1\le i\le k-1}(w_{ik}d_{ik}) - \sum_{k+1\le i\le N}(w_{ki}d_{ki}) \\ \cdots \\ \cdots \\ \cdots \end{pmatrix}$$

where M_{kk}, the summation term in Equation 6, is the sum of the weights of the measurements from probe k to other probes.

A critical point is that there is no guarantee that the ordering of the probes in the solution of $\mathbf{Mx} = \mathbf{r}$ will respect the ordering (2) used to construct this linear system. However, the solution to this linear system provides useful information in either case.

– If the solution *does* respect the ordering, then it provides the optimal (in the least-squares sense) positioning of the probes with respect to the given ordering, and is a local minimum of the cost function.
– If the solution does *not* respect the ordering, then it gives a lower bound on the cost of the best placement with that ordering. This is true since solution to $\mathbf{Mx} = \mathbf{r}$ gives the minimum of $\sum_{i<j} w_{ij}(x_j - x_i - d_{ij})^2$ over all $\mathbf{x}$, which is certainly no greater than the minimum over the region $\{\mathbf{x} \mid x_1 < x_2 < \cdots < x_N\}$. Furthermore, in this case the given ordering is not the optimal one, since the solution to $\mathbf{Mx} = \mathbf{r}$ gives

an ordering having a lower cost. This holds since for each pair $i < j$ for which $x_i > x_j$, we have

$$(\mid x_i - x_j \mid - d_{ij})^2 = (x_i - x_j - d_{ij})^2 < (x_j - x_i - d_{ij})^2.$$

In other words, at the point $\mathbf{x}$ solving $\mathbf{Mx} = \mathbf{r}$, each term in the true cost function is less that or equal to the corresponding term in the restricted cost function built assuming the ordering $x_1 < x_2 < \cdots < x_N$, and so that ordering cannot be optimal.

These are the key observations on which our algorithms are built. The problem has been reduced from a continuous optimization problem to a discrete one—that of computing the matrix solution over all probe orderings and choosing the ordering with the lowest cost. Our branch and bound algorithm searches over all possible probe orderings, using an extension of the method above to bound the cost of large sets of possible orderings, provably finding the one(s) of minimum cost. The branch and bound algorithm is described more fully in Section 5. Our heuristic search algorithm is even simpler. Starting from many random orderings, it merely iterates the process described above until it reaches a local minimum. Empirically, this is highly effective at finding the global minimum quickly. This is described more fully in the next section.

4 Heuristic Search

As outlined in the previous section, solution to the linear system constructed for any fixed order π of the probes either gives the optimal placement for probes in that order, which is a local minimum of the cost function, or gives a placement with another ordering π' at which the cost function is lower than it is at any placement respecting π. Our heuristic search algorithm is simply "iterated linear solve":

1. choose a random ordering π;
2. set up the linear system corresponding to that ordering;
3. solve it;
4. if the resulting order π' is equal to π, record that as a potential minimum;
5. if $\pi' \neq \pi$, replace π by π' and return to step 2.

Finally, we repeat this entire process for many random initial orderings, and report the lowest cost solution found. In different tests, we either did a fixed number of random starts, usually 300, or repeated until the known optimal solution was found.

One nice feature of the matrix formulation is that $\mathbf{M}$ is independent of the ordering of the probes. When solving this system by LU decomposition (as in [8]), this means that once we perform an initial $O(N^3)$ operation on $\mathbf{M}$, we can find a solution in $O(N^2)$ time per ordering, the time required to generate the (order-dependent) vector $\mathbf{r}$ and backsolve.

5 Branch and Bound

Our branch and bound algorithm constructs a search tree over probe orderings. The leaves will be complete orderings and the interior nodes will be partially specified orderings. There are two basic approaches to structuring the search tree. In the first approach, shown in Figure 4, the children of a node P in the tree will be the ordering of probes at node P augmented with a new probe in all possible positions among the probes ordered at P.

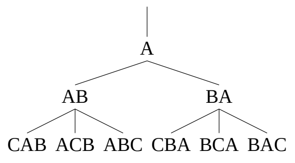

Fig. 4. At a node, the children are orderings in which an additional probe is placed in all possible positions with respect to the ordered probes.

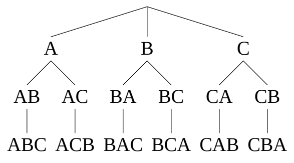

Fig. 5. At a node, the children are orderings in which each of the unordered probes has been placed to the right of the rightmost ordered probe.

For the second approach, in Figure 5, the ordering of a child of an interior node P will be the ordering of P augmented by a probe placed adjacent to the rightmost ordered probe in P.

In either approach, as is typical in branch and bound algorithms, little of the ordering is specified at higher levels of the search tree, hence the bounds computed there will be weak and pruning will be rare. Given this, the first approach has the advantage that the branching factor is much lower near the root of the tree compared to the second approach, e.g. 3 versus $N-3$ on the third level. On the other hand, the second approach has the advantage that more information is known about the partially specified ordering at an interior node P, namely that all unordered probes lie to the right of the rightmost specified probe in every node of the subtree rooted at P. We can exploit this to give a strengthened bound at internal nodes compared to approach one. In our experiments [11], approach two outperformed approach one by nearly a factor of two both in run time and in number of tree nodes visited. Throughout the remainder of this paper, we will only consider approach two.

Our branch and bound algorithm searches through nodes in a tree, pruning a node if its cost is greater than the lowest cost found in a leaf node so far. At a leaf node in the tree, we compute the cost of the ordering as described in Section 3.1. At an interior node, the cost function must be a lower bound on the cost of all nodes in the subtree to allow us to possibly prune the subtree. In this section, we describe a simple cost function based on least squares.

Consider an interior node such as that in Figure 6. In this picture of an interior node, the circles represent probes, and the edges represent the existence of a measurement between two probes. Probes A, B, C, and D have been ordered (in that order). Probes E

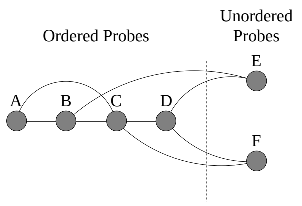

Fig. 6. An Interior Node

and F are unordered with respect to each other, but both will appear to the right of probe D. One way of computing a lower bound on the cost for this node is to consider only the measurements between the ordered probes. In this case, we compute the cost function by computing the matrix solution (as described in Section 3.1) using a matrix built from only measurements between the ordered probes. This is done by simply pretending the other measurements do not exist, i.e., the terms in $\mathbf{M}$ of Equation 5 for measurements that we are not considering are 0, and there is no contribution from them in the $\mathbf{r}$ vector.

We note that the cost function described here is ineffective at high levels in the tree (where nodes will reflect probe orderings with few constraints). In particular, the cost function described evaluates to zero for the first and second level in the tree (when only one or two probes are ordered). However, consider the measurements in Figure 6 between ordered probes C, D, and unordered probe F. Even though the position of F is undetermined with respect to E, we know that F will be to the right of D. This allows us to remove the absolute value sign in the sum of squares terms of Equation 1 for the measurements between F and C, D and include these terms in the cost function computation. Thus, for an interior node, as well as considering all edges between ordered nodes, we can consider edges between ordered nodes and unordered nodes when constructing the cost function for the node. This improvement potentially allows us to compute a non-zero cost function for nodes as high as the second level in the tree (when only two probes are ordered). With this improvement, the only constraints we are not considering at a node are those between unordered probes. The bound function described here is the one we use in the simulations reported in Section 6, Results.

The cost of an interior node P computed in this way will be a lower bound on the cost of all nodes in the subtree rooted at P, since nodes in the subtree impose additional constraints on the ordering, never remove constraints, and each additional constraint adds additional non-negative terms to the cost function.

An additional issue which has a strong effect on the performance of our branch and bound algorithm is initialization of the bound. Starting the algorithm with a conservative default value for the bound (like $+\infty$) results in very poor pruning until a reasonably good solution is encountered. Instead we first run the local search algorithm from a few random starting orderings. Empirically, this will quickly locate a good solution, facilitating good pruning from the beginning. In our experiments, branch and bound removes 100-1000 times as many nodes as a result [9].

There is one remaining detail to be specified—we need to modify the construction of the $\mathbf{M}$ of Equation 5. As it stands, the linear system $\mathbf{M}\mathbf{x} = \mathbf{r}$ of Section 3.1 is under-constrained (the rank of the null-space of $\mathbf{M}$ is non-zero). Because the system is constructed from relative orderings between the probes, there is one degree of freedom: the absolute position of the probes. This is remedied by modifying the system to arbitrarily place probe 1 at location $x_1 = 0$. There may be additional degrees of freedom in the solutions. In particular, at high levels in the tree the small set of ordered probes may be partitioned into several disconnected components whose relative positions are unconstrained. These situations are handled similarly; see [9] for details.

Finally, we remark that the cost computed by the techniques outlined above is a lower bound, but not necessarily an attainable bound, on the cost of any ordering consistent with that specified at a search tree node. In particular, in the case where the solution to the linear system $\mathbf{Mx} = \mathbf{r}$ exhibits a different ordering than the one from which the system was constructed, we know that the bound is not attainable by the desired ordering. It is still valid to use this bound to prune the search tree, since we know the bound is attainable by some (other) ordering. However, pruning could be improved if a higher lower bound could be computed in these cases. One possible approach to doing so would be to use quadratic programming—minimization of the quadratic objective function in Equation 1 subject to the linear constraints in Equation 2 is a convex quadratic optimization problem, for which polynomial time algorithms are known; see, for example, [4]. However, it is not clear whether the increased pruning efficiency would offset the extra computational cost of using the more elaborate quadratic programming algorithm. Preliminary experiments have been inconclusive [11]

We now present the results of experiments performed on the heuristic search and branch and bound algorithms.

6 Results

We ran multiple simulations to assess the performance of the two algorithms and also to gauge the sensitivity of the algorithms to different parameters. We summarize the main results here; see [9, 11] for further details.

The experiments described below were all run on synthetic data generated in accord with the motivating problem presented in Section 1. Probes were placed uniformly at random, except that adjacent probes were separated by minimum distance of approximately 3% of the average spacing. Approximately 50% of the probe pairs were “measured,” were measurement consisted of drawing a random sample from a certain distribution whose mean was the actual distance between the probes. Data sets having more than one connected component or certain other anomalies were filtered out. The results do not seem to be overly sensitive to any of these parameters.

As a measure of the quality of the solution found by the algorithms, we used RMS error—the square root of the mean squared difference between the true and calculated positions of the probes. While this quantity varied from run to run, the median value was 10%–15% of the average interprobe distance, which is reasonably good considering the variance of the “measurements.”

We present the total time for the branch and bound algorithm using weighted least-squares in Figure 7, and the total time for heuristic search in Figure 8. Each point in the graph is a problem instance.

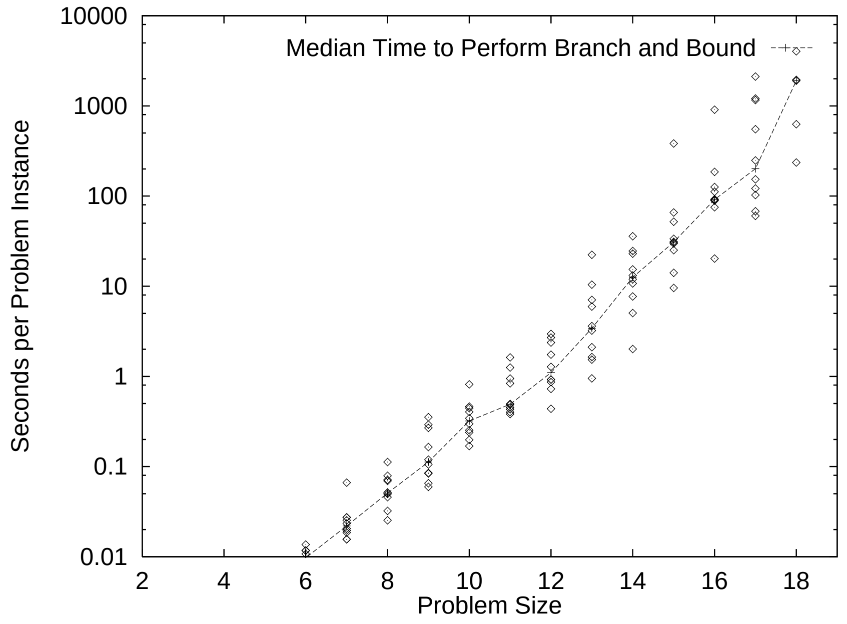

Fig. 7. Time for Branch and Bound (Seconds).

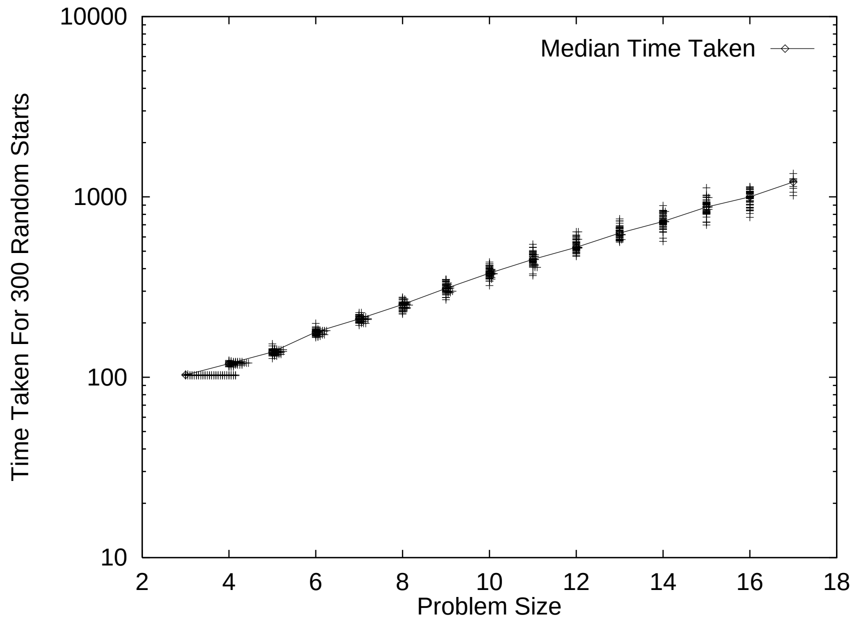

Fig. 8. Time for Heuristic Search. Trials showing identical times spread horizontally for clarity. (300 random starts; Milliseconds).

We can see that the running time of the branch and bound algorithm is exponential, as is expected, with time increasing roughly as 2.8^N. Note that at the far right of the graph, the most time taken to solve a problem of 18 probes was about 70 minutes. Since the number of nodes in a search tree of a problem that size is around 10^{16}, we can see that the pruning heuristic is quite effective; in fact it visited on the order of 10^6 nodes.

The performance of heuristic search is in some ways more difficult to assess. For a problem size of 18, the 300 random starts of heuristic search took about 1 second. The surprisingly stable growth rate also appears to be exponential, but grows much more slowly, roughly as 1.2^N. At this rate, problems of size 30 would be solvable in a few minutes and problems of size 50 in under an hour. However, note that 300 random starts is a very arbitrary choice. In most trials ($> 90\%$), the method finds the globally optimal solution within 10 random starts. In a few "hard core" cases, however, it can take several thousand starts to find the global. Unfortunately, of course, using heuristic search alone, one cannot tell when the globally optimal solution has been reached. (We compared to the provably optimal results from branch and bound.) Nevertheless, the method seems to be a powerful one and worth further study.

Timing experiments where performed on a 100 MHz DEC AlphaStation 200 4/100 with 96MB of memory. The C code was not optimized beyond the optimizations described here (and in [9, 11]). In particular, the LU decomposition routine was copied without modification from [8]. Since the process size for these algorithms was around 3 MB, and since the simulation code is CPU intensive, the time due to non-CPU activities (such as paging) does not significantly affect the results shown.

7 Conclusions

We have presented two search algorithms, a branch and bound algorithm and a heuristic local search algorithm, both of which attempt to minimize a weighted least-squares cost function to solve a one dimensional point placement problem.

Due to the exponential nature of the branch and bound algorithm, it is unlikely that it will scale to larger problem sizes. However, it does provide good performance on problems of 18-20 probes, large enough to be of practical use. Since it finds the global minimum, it is also useful as a benchmark against which to compare other algorithms.

The local search algorithm performed surprisingly well, finding optimal solutions in seconds and appears capable of handling much larger problem instances.

8 Acknowledgments

We would like to thank Brian Pinkerton, Barb Trask, Ger van den Engh, Harry Yeung, and the Statistics Consulting Group for their thoughtful comments and generous assistance.

References

1. Timothy Bishop. Linkage analysis: Progress and problems. *Phil. Trans. R. Soc. Lond.*, 344:337–343, 1994.

2. Michael Boehnke, Kenneth Lange, and David Cox. Statistical methods for multipoint radiation hybrid mapping. *Am. J. Hum. Genet.*, 49:1174–1188, 1991.
3. Kenneth H. Buetow and Aravinda Chakravarti. Multipoint gene mapping using seriation. I. General methods. *Am. J. Hum. Genet.*, 41:180–188, 1987.
4. Donald Goldfarb and Shucheng Liu. An $O(n^3 L)$ primal-dual potential reduction algorithm for solving convex quadratic programs. *Mathematical Programming*, 61:161–170, 1993.
5. Brendan Marshall Mumey. A fast heuristic algorithm for a probe mapping problem. In *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pages 191–197, 1997.
6. Brendan Marshall Mumey. *Some Computational Problems from Genomic Mapping*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1997.
7. William R. Newell, Richard Mott, S. Beck, and Hans Lehrach. Construction of genetic maps using distance geometry. *Genomics*, 30:59–70, 1995.
8. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.
9. Joshua Redstone and Walter L. Ruzzo. Algorithms for ordering DNA probes on chromosomes. Technical Report UW-CSE-98-12-04, Department of Computer Science and Engineering, University of Washington, December 1998.
10. Ger van den Engh, Ranier Sachs, and Barbara J. Trask. Estimating genomic distance from DNA sequence location in cell nuclei by a random walk model. *Science*, 257:1410–1412, 4 September 1992.
11. Harry Yeung and Walter L. Ruzzo. Algorithms for determining DNA sequence on chromosomes. Unpublished, March 1997.