Algorithms for Ordering DNA Probes on Chromosomes

Joshua Redstone
Walter L. Ruzzo

Technical Report UW-CSE-98-12-04
December, 1998

Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195

Algorithms for Ordering DNA Probes on Chromosomes

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

December 31, 1998

Abstract

Accurate estimates of the ordering and positioning of DNA markers (probes) on a chromosome are valuable tools used, for example, to help researchers isolate genetic factors in diseases. One such mapping technique, called fluorescent *in situ* hybridization (FISH), obtains approximate pairwise distance measurements between probes on a chromosome. We have developed two algorithms for computing least squares estimates of the ordering and positions of the probes: a branch and bound algorithm and a local search algorithm motivated by gradient descent. Simulations demonstrate the effectiveness of the branch and bound pruning heuristic and show that the local search algorithm is usually fast and accurate. The branch and bound algorithm is able to solve to optimality problems of 18 probes 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 18 probes in about a minute. We also investigate (via simulation) the accuracy with which maps can be constructed from FISH data.

1 Introduction

The problem of mapping genetic information has been the subject of extensive research since experimenters started breeding fruit flies for physical characteristics. The goal is to determine information about the sequence of DNA in chromosomes. This information is valuable in areas such as determining genetic causes or predispositions to diseases. For a particular disease, DNA structure information can be used to isolate defective parts in a chromosome, hopefully allowing researchers to develop techniques to treat or prevent the disease. The chromosome is a long chain molecule curving in three dimensions inside a cell. 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. Some examples are in [Bishop 94].

Van den Engh and Trask of University of Washington's Molecular Biotechnology department have developed a technique for obtaining estimates of relative DNA sequence positions called fluorescent *in situ* hybridization (FISH)[van den Engh 92]. In this technique, probes are hybridized (attached) to different sites in the chromosome, and colored. A measurement between a pair of probes is made by measuring the distance between the probes projected onto a plane. If the experiment is repeated in many cells, the genomic distance can be estimated. In [van den Engh 92], they propose that the distance between the two probes follows a random walk model, and they provide a distribution of the probability of measuring a given physical distance, R, between two probes given the actual number of genomic links (the distance between the two probes on a chromosome in DNA base pairs), n, separating the two probes:

$$P(R) = \left(\frac{\pi R}{2nL^2}\right) e^{-\pi R^2/(4nL^2)} \tag{1}$$

In Equation 1, L is a constant representing the average projected length of a single genomic link. The measurements between multiple pairs of probes on a chromosome can be summarized as a matrix. In general, this matrix will be incomplete since not all pairwise measurements are made. The problem is thus:

Problem: Given an incomplete matrix of pairwise measurements between probes, determine the best estimate as to the ordering and position of the probes on a line.

This problem is known to be NP-hard.

A solution should provide the correct ordering and reasonable position information. In this domain, "reasonable" might mean accurate to within 10% according to some measure of similarity. One factor limiting the accuracy of any solution is the high variance of Equation 1. Higher order structural elements in the chromosome can also add noise to Equation 1. The effect of such structural elements is not known. Accuracy may also be limited by the difficulty of incorporating other physical or genetic mapping information about ordering and placement into this framework. Examples of other information include a technique to determine which probe of two probes is closer to a particular end of the chromosome (resolution is about five times worse than FISH), and a technique that places probes into bands (subdivisions of a chromosome visible under a microscope; resolution is about 60 times worse than FISH). Given these sources of inaccuracy, any solution to this problem should include a list of the top orderings deemed most likely.

1.1 Previous Work

Brian Pinkerton previously investigated solving this problem using the seriation algorithm of [Buetow 87], 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 gene mapping problems, but they are based on distance estimates from techniques other than FISH. The other algorithms are tailored to the particular statistical properties of the distance measurements.

One example is the distance geometry algorithm of [Newell 92], based on recombination frequency data. Another example is [Boehnke 91], which investigated branch and bound, simulated annealing, and maximum likelihood algorithms based on data from radiation hybrid mapping.

1.2 Work Presented in This Paper

In this work, we chose a sum of squares cost function to evaluate different probe orderings and positions. From this model, we develop and examine the effectiveness of a branch and bound and a local search technique. The branch and bound algorithm searches through a tree of all possible probe orderings. For each probe ordering, the optimal (in the least-squares sense) positions of the probes are determined. We develop various pruning heuristics, extending the work of Brian Pinkerton by implementing a more complicated pruning heuristic. We also develop a local search algorithm motivated by gradient descent, and demonstrate that it outperforms the branch and bound algorithms in execution time, and is competitive with the branch and bound algorithm in quality of solution produced over the data sets we examine. Exact solutions via branch and bound are feasible up to about 18 probes. Local search can usually give optimal solutions within about a minute for problem sizes up to 18 probes, which was the maximize size we could verify with branch and bound. On a problem size of 50, it produced an answer in about 15 minutes, but at that size it is computationally infeasible to verify the optimality of the solution with the branch and bound algorithm.

In the first section of the paper (Section 2) we describe the computer science aspects of the problem. Section 3 develops a cost function to evaluate solutions. Sections 4 and 5 outline the branch and bound algorithm, and in Section 6 we describe the local search algorithm. We then present the results of simulations of the two algorithms in Section 7. Finally, we conclude and offer suggestions for future work.

2 Description of the Problem Domain

2.1 Initial Data Processing

The data returned by the FISH process is in the form of a physical distance between two probes, measured in micrometers. This physical distance is the result of measuring the distance between the two probes under a microscope. We are searching for the genomic distance between the two probes which measures the number of base pairs separating them on the chromosome. Given the physical pairwise distance measurements, the first step is to convert these measurements into estimates of the genomic distance between probes. The genomic distances, rather than the physical distance will be used when computing orderings and positions of probes. [van den Engh 92] provides an equation relating the expected value of the measurements between a pair of probes, $\langle R \rangle$, to the genomic distance, n, between the probes:

$$\langle R \rangle = \int_0^\infty A\, P(A) dA = L\sqrt{n}$$

where P is probability distribution of Equation 1. Thus, an approximation of the genomic distance between the probes is:

$$n = \overline{R}^2 / L^2 \tag{2}$$

where $\overline{R}$ is the average of the measurements between a pair of probes. It is not clear from the distribution of Equation 1 whether, given a fixed number of measurements between a pair of probes, the n returned by Equation 2 is the best estimator of the genomic distance between the pair. It would be interesting to investigate other estimators of n that may be more accurate. The maximum likelihood estimator is one possibility. An investigation of the maximum likelihood estimators is beyond the scope of this paper.

Equation 2 is used to convert all physical distance measurements into estimates of genomic distance. For the rest of this paper, the "distance" between two probes will refer to the genomic distance between the probes.

2.2 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 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.

Figure 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.

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.

Figure 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 4 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

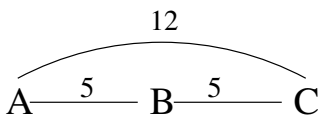

Figure 3: There is no way to linearly place these probes on a line and respect all the measurements.

case, the distances between the probes in any linear placement will unavoidably be different from the measured distances.

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

If we construct a cost function to evaluate the "goodness" of a solution then we can solve the problem by finding the answer that has the best cost. This cost function will be a function of an estimate of the positions of the probes. These estimates will be generated by the search algorithms. One intuitive cost measure to use is the sum of squares of the difference between the measured distance between two probes and the distance between the probes in the estimated linear placement of the probes. Let N be the number of probes, x_i be the position of probe i, and d_{ij} be the measured distance between probe i and probe j ($d_{ij} = d_{ji}$). We can write the sum of squares of differences (errors) as

$$Cost(x_1, \ldots, x_N) = \sum_{\substack{i<j \\ d_{ij} \text{ measured}}} (\mid x_i - x_j \mid - d_{ij})^2 \tag{3}$$

This formulation of a cost function is appealing because it trades off the different constraints (the pairwise measurements between probes). Another advantage of this approach is that we can develop a branch and bound pruning heuristic based on explicitly solving for the minimum of this cost function. It also is a straightforward basis from which to build a gradient descent based local search algorithm.

3.1 Is a Squared Error Cost Function Appropriate?

One important question to ask is if this cost function accurately reflects good solutions, i.e., does a low cost correspond to a correct probe ordering with reasonable positions. The procedure of minimizing this cost function is an instance of Multiple Linear Regression using Least Squares. We will use this model to help assess the appropriateness of the cost function formulated in Equation 3. The standard formulation of the Multiple Linear Regression model can be found in [Myers 1986] or any introductory statistics text, and is:

$$y_p = \beta_0 + \beta_1 z_{1p} + \beta_2 z_{2p} + \cdots + \beta_k z_{kp} + \varepsilon_p \qquad (p = 1, 2, \ldots, m;\ m \geq k+1) \tag{4}$$

where m is the number of observations and k is the number of estimators. This equation is used to model a series of observations of a value y as a linear combination of input parameters. In Equation 4, y_p is termed the measured response variable and is the result of an observation. The z_{jp}'s are the regressor variables and the parameters of an observation. If we consider each observation as an experiment, then the y_p is the measured result of the experiment, and the z_{jp} are the control variables of the experiment.

The β_k are estimators that represent our theory as to how the parameters of an experiment are related to the measured result. These β_k are optimized (by the least-squares method) to best fit the observations. Since this model may not model the observations perfectly, ε_p is included as the model error. ε_p is the difference between the result of the experiment predicted by the model and the measured result.

In our application, an experiment is a measuring of the distance between a pair of probes. So p will correspond to a specific pair of probes being measured. y_p will be the distance measured between the pair of probes. In our case, the input parameters of the experiment, z_{jp}, are values specifying which probes are measured. The β_k will be estimators of the coordinates of each probe.

Let us consider a concrete example of this model for an experiment measuring the distance between two probes, A and B. We will specialize Equation 4 for this experiment. Since p is an index of the measured values in the experiment, we will replace p with AB in the equation modeling this experiment. y_p, the measured result, will be denoted d_{AB}, the measured distance between probes A and B. β_A and β_B are estimators of the positions of probes A and B, and we will denote them x_A and x_B. The z_{jp} are values specifying which probes were measured to produce the value d_{AB}. Since the measurement between probes A and B does not involve any other probes, all z_{jp} except for the z_{jp} of the terms for x_A and x_B will be 0. We will write the z_{jp} of these two non-zero terms as $z_{A_{AB}}$ and $z_{B_{AB}}$. Lastly, the model error, ε_p, will be written as ε_{AB} for this experiment. Thus, the model formulation for this observation becomes:

$$d_{AB} = z_{A_{AB}} x_A + z_{B_{AB}} x_B + \varepsilon_{AB}$$

We now set the value of the z's so that the sum of the x_A and x_B terms becomes the distance between the estimators of the positions of probes A and B. The ε_{AB} thus becomes the error due to sampling from the distribution of Equation 1. To substitute values for the z's, we must assume an ordering of the two probes, A and B. We will suffer the same restriction of having to specify the order of the probes when solving the problem explicitly in the cost function computation in the branch and bound algorithm (see Section 3.4). Let us assume that

probe A is to the right of probe B. In this case the model of the observation of the distance between probes A and B becomes:

$$d_{AB} = x_A - x_B + \varepsilon_{AB}$$

There will be an equation similar to this for each pair of probes measured.

The objective of regression in our application is to find x's that minimize ε_{ij}, the difference between the measured and estimated distances between probes. Least squares is one method used to solve this problem.

The estimators provided by a linear regression model using least squares have nice properties like being maximum-likelihood estimators and having minimum variance over all unbiased estimators provided a few assumptions hold:

1. The mean of the model error ε_{ij} for the measurement between probes i and j is 0.
2. The ε_{ij} are uncorrelated.
3. The variance of ε_{ij} is constant (homogeneous variances).
4. The ε_{ij} are distributed normally.

In our domain, the distribution of ε_{ij} is determined by Equation 1. Unfortunately, this means the sum of squares formulation violates the last two assumptions. A check in a statistics text such as [Mood 63] will reveal the probability distribution of Equation 1 to be the Weibull distribution, violating assumption 4. However, since the estimated distance between two probes is based on an average of 100 to 200 measurements, the distribution of ε_{ij} approaches the normal distribution, which is closer to satisfying assumption 4.

The variance of the Weibull distribution is not constant, violating assumption 3. Since each measurement is an independent sampling of the Weibull distribution, the ε_{ij}'s are uncorrelated, supporting assumption 2. Finally, to judge the validity of assumption 1, we note that our distance measurements are computed from Equation 2. That equation estimates the genomic distance by approximating the expected value of the measured physical distance by averaging the measurements made between two probes. ε_{ij} is the deviation from this average, and so the mean of ε_{ij} is 0, supporting assumption 1.

The fact that the Weibull distribution violates the assumptions above does not mean that it is inappropriate to use a sum of squares cost function, but it does alert us to weaknesses of the model. In the Section 3.2 we discuss how to improve the cost function to compensate for the violation of the homogeneous variance assumption.

3.2 Improving the Sum of Squares Cost Function by Including Weights

One improvement to the cost function is to modify it to compensate for the violation of the constant variance assumption of the multiple linear regression model of Section 3.1. The variance of the distribution function of Equation 1 is:

$$\sigma^2 = \left(\frac{(4-\pi)L^2}{\pi} \right) n$$

We can see that the variance of measurements between a pair of probes is directly proportional to the genomic distance, n, separating the probes. This means that the terms in the cost function for probes separated by long distances will have a higher variance. If we are optimizing using least squares then, intuitively, the proportional variance means that we will be putting too much weight on terms for measurements between probes that are far apart. Luckily, it is not difficult to compensate for inhomogeneous variances in a multiple linear regression model. If we were performing a standard least squares optimization in which the only assumption of the standard multiple linear regression model that we were violating was the homogeneous variance assumption, then the correct way to modify the sum of squares would be to weight each term by $w = 1/\sigma^2$. This would preserve the property that the estimators are maximum likelihood estimators and have minimum variance over all unbiased estimators. Weighting each term in this way intuitively compensates for the extra noise in terms for measurements between probes that are far apart.

Unfortunately, since the model errors are not normally distributed, we can not make such guarantees. The strongest claim we can make is that if we compensate for the variances by weighting each term with $w = 1/\sigma^2$, the estimators will have minimum variance over all *linear* unbiased estimators.

The distribution of Equation 1 has a high variance and a long tail. Even compensating for the inhomogeneous variances will not prevent inaccuracies due to the non-normal shape of the distribution. However, compensating for the inhomogeneous variances does improve the accuracy of the estimators as is demonstrated in Section 7, Results. Inserting the weights into the cost function of Equation 3 produces:

$$Cost(x_1, \ldots, x_N) = \sum_{\substack{i<j \\ d_{ij} \text{ measured}}} w_{ij}(\mid x_i - x_j \mid - d_{ij})^2 \quad \text{where} \quad w_{ij} \propto \frac{1}{d_{ij}} \tag{5}$$

Equation 5 is the cost function used in all the experiments performed in this work. Having examined the weaknesses of the sum of squares cost function, we now examine the benefits.

3.3 Benefits of a Sum of Squares Cost Function

Computationally, the least squares approach is very attractive because of its simplicity. Another nice property is that the cost of an optimal solution for a reduced set of constraints (measurements) is a lower bound on the cost of the optimal solution of the same problem with additional constraints. Adding an additional constraint to a problem simply adds another term to the summation of Equation 3. Since each term in the summation is nonnegative, the addition of another term cannot reduce the sum of terms corresponding to the original set of constraints.

3.4 Finding Least Squares Solutions

Given the sum of squares formulation, one approach is to solve it explicitly. We can take the partial derivative of the sum of squares with respect to each of the x_i's, set them equal to 0, and solve. In order to take the derivative of Equation 3, we need to eliminate the absolute value signs. The only way to do this is to assume an ordering of the probes. Without loss of

generality, assume $x_1 < x_2 < \cdots < x_N$. Then for a given probe k:

$$\frac{\partial}{\partial x_k}\left(\sum_{\substack{i<j \\ d_{ij} \text{ measured}}} w_{ij}(\mid x_i - x_j \mid - d_{ij})^2\right) = \sum_{\substack{1\le i\le k-1 \\ d_{ik} \text{ measured}}} 2w_{ik}(x_k - x_i - d_{ik}) - \sum_{\substack{k+1\le i\le N \\ d_{ki} \text{ measured}}} 2w_{ki}(x_i - x_k - d_{ki}) \quad (6)$$

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

$$x_k \sum_{\substack{1\le i\le N \\ d_{ik} \text{ measured}}} (w_{ik}) + \sum_{\substack{1\le i\le N \\ d_{ik} \text{ measured}}} (-w_{ik}x_i) = \sum_{\substack{1\le i\le k-1 \\ d_{ik} \text{ measured}}} (w_{ik}d_{ik}) - \sum_{\substack{k+1\le i\le N \\ d_{ik} \text{ measured}}} (w_{ki}d_{ki}) \quad (7)$$

We can transform Equation 7 to the form

$$\mathbf{M}\mathbf{x} = \mathbf{r} \quad (8)$$

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

$$M_{ij} = \begin{cases} -w_{ij} & i \neq j,\ d_{ij} \text{ measured}, \\ \sum_{\substack{1\le j\le N,\ j\neq i \\ d_{ij} \text{ measured}}} (w_{ij}) & i = j, \\ 0 & \text{otherwise}, \end{cases} \quad (9)$$

and r_k is a vector formed from the right hand side of Equation 7. Thus, in matrix form, Equation 7 can be written as:

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

where M_{kk}, the summation term in Equation 9, represents the sum of the weights of the measurements from probe k to other probes. If we solve for $\mathbf{x}$ we can find the optimal positioning (in the least-squares sense) of the probes. We compute the cost function by computing the sum of squares (Equation 3) using the optimal positions found. Note that there is no guarantee in the solution of $\mathbf{M}\mathbf{x} = \mathbf{r}$ that the resulting ordering of the probes will respect the ordering used to construct the linear system. This does not affect the correctness of the algorithm, as is discussed in Section 5.1. Thus, the problem has been reduced to that of computing the matrix solution over all probe orderings and choosing the ordering with the lowest cost among those whose solution respects the ordering. (Quadratic Programming is a technique that will find an optimal solution with respect to an ordering, and is described in Section 5.2.)

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 [Press 92]), this means that we

can find a solution in $O(N^2)$ time per ordering once we perform an initial $O(N^3)$ operation on $\mathbf{M}$. We can take advantage of this time savings in our local search algorithm and at leaf nodes in the branch and bound algorithm, but we must perform the full $O(N^3)$ LU decomposition at every interior node of branch and bound since the $\mathbf{M}$ matrix varies with the number of measurements considered, which is different for each node. Additional time savings may be possible by caching the different LU decompositions of $\mathbf{M}$ at nodes above the leaf nodes in the tree. For example, in a problem of N probes, there will be N different $\mathbf{M}$ matrices at the level above the leaf nodes in the search tree. The construction of the search tree in branch and bound is discussed in Section 4.

4 The Branch and Bound Algorithm

To construct a branch and bound search, we need to choose a tree representation. Since computing the matrix solution of Section 3.4 requires an ordering of the probes, a search tree over probe orderings is one natural approach. In this case, the leaves will be complete orderings and the interior nodes will be a partially specified ordering. 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.

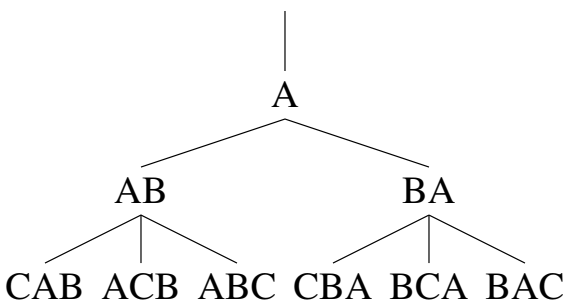

Figure 4: At a node, the children are orderings in which an additional probe is placed in all possible positions with respect to the ordered probes.

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 this work, we choose the second approach. The primary reason is that we are able to devise a better pruning heuristic if we know that the unordered probes are to the right of the ordered probes at interior nodes in the tree. This allows us to include constraints between ordered probes and unordered probes in the cost function.

The first approach is also worth consideration, although it isn't investigated in this work. The first approach has a lower branching factor at the top of the tree. (In general, the branching factor is $l + 1$ at level l in the tree where level 1 is the root as in Figure 4.) A further possible advantage of the first approach is that we can specify the order in which the probes are ordered during the search. On the other hand, since the unordered probes may be placed at any location

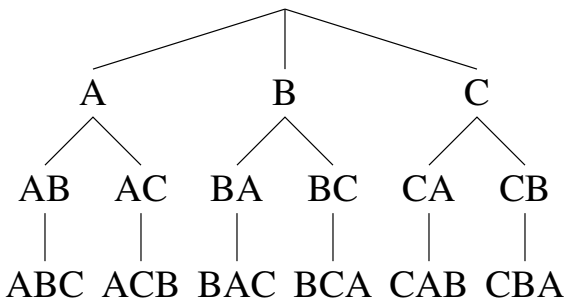

Figure 5: At a node, the children are orderings in which each of the unordered probes has been placed to the right of the rightmost ordered probe.

relative to the ordered probes, we are unable to include constraints between the ordered probes and the unordered probes in the cost function. It may or may not be the case that the penalty of the weaker cost function of the first approach is offset by the benefit of a lower branching factor higher in the search tree. As a first approximation, we can perform some simple measurements on the search tree generated by each approach to try to ascertain which might allow a better search. One measure we can look at is to fix the number of nodes visited by a breath first search in the tree from each approach and try to estimate how well we are able to prune. One coarse measure of pruning effectiveness at a node might be the number of constraints used in computing the cost of a node, based on the idea that a greater number of constraints would lead to a cost that is a tighter lower bound. If we assume that all constraints contribute equally to the cost function, then calculations on the search tree from a problem of size 12 indicate that the second approach averages more constraints per node at the higher levels in the tree. This would suggest that we might be more effective pruning a tree generated by the second approach. However, one could imagine that constraints from ordered probes to unordered probes in the second approach might not contribute as much to the lower bound since the unordered probes are relatively unconstrained. A more thorough investigation to compare the two approaches seems warranted.

We now turn to details of the branch and bound algorithm.

5 Executing Branch and Bound

In this section we will describe details of the branch and bound algorithm including the structure of nodes in the search tree and argue the correctness of the pruning heuristic.

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 3.4. 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. We would like to find a function that provides the greatest lower bound to improve pruning. 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

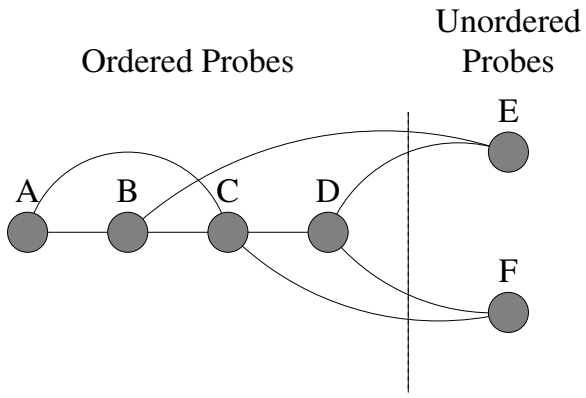

Figure 6: An Interior Node

two probes. Probes A, B, C, and D have been ordered (in that order). Probes E 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 3.4) 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 8 for measurements that we are not considering are 0, and there is no contribution from them in the $\mathbf{r}$ vector.

The cost of an interior node computed in this way will be a lower bound on the cost of all nodes in the subtree. Consider the hypothetical case of a node Q in the subtree of a node P, where Q has a lower cost than P. Since Q is a descendant of P, we know it has a superset of the constraints (pairwise measurements) of node P. Since each constraint yields a non-negative term in the cost function, if Q has a lower cost than P then the cost of the set of constraints that Q shares with P is lower than the cost of P. This would mean that the cost computed at node P was not the minimum for the set of constraints at P, which is impossible since computing the matrix solution returns the minimum cost for an ordering.

Note that since we are changing the set of constraints (measurements) that we are solving for in the subtree of node P (by adding constraints from additional probes that become ordered in the subtree), the $\mathbf{M}$ matrix at each node is different which means that we must perform the $O(N^3)$ LU decomposition to solve the matrix at each interior node. However, the cost function described here is a lower bound, and establishes the basic branch and bound algorithm. In general, this lower bound holds for the solution to the sum of squares for a set of constraints $\mathcal{S}$ with respect to the solution for any set of constraints $\mathcal{T}$ where $\mathcal{T}$ is a superset of $\mathcal{S}$. We can use this fact to improve the cost function computed at a node.

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). This negative impact on pruning performance is compounded by the fact that the upper nodes in the tree have the highest branching factor. The cost function described evaluates to zero for the first and second level in the tree (when only one or two probes are ordered) because we are minimizing a cost function con-

sisting only of terms involving probes that are ordered. However, consider the measurement in Figure 6 between probe D, that is ordered, and probe E, that is unordered. Even though the position of E is undetermined, due to the structure of the tree, we know that E will be to the right of D. This allows us to remove the absolute value sign in the sum of squares term of Equation 5 for this measurement and include the term 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 allows us to compute a non-zero cost function for nodes at the second level in the tree (when two probes are ordered). With this improvement, the only constraints we are not considering at a node are those between unordered probes. The cost function described here is the cost function we use in the simulations reported in Section 7, Results.

We now address concerns of correctness about the branch and bound algorithm.

5.1 Details of Performing Least Squares in the Search Tree

There are four details to address to obtain a working branch and bound algorithm. The first completes the specification of the algorithm, and the next three address correctness issues.

First, we need to modify the construction of the $\mathbf{M}$ of Equation 8. As it stands, the linear system $\mathbf{Mx} = \mathbf{r}$ of Section 3.4 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 replacing the top row of the $\mathbf{M}$ matrix with $(1\ 0\ 0\ \ldots\ 0)$ and setting the first element of $\mathbf{r}$ to 0, which has the effect of arbitrarily placing probe 1 at location $x = 0$.

To understand why there is a degree of freedom in the matrix $\mathbf{M}$, consider a graph in which the nodes are the probes and an edge (i, j) exists if there is a measurement between probes i and j, and further assume there is only one connected component in the graph. The sum of the rows of the $\mathbf{M}$ matrix constructed from these measurements is $\mathbf{0}$. To see this, consider the sum of a column j of $\mathbf{M}$. Probe j is in the connected component, so entry j in column j will be the sum of the weights of measurements from probe j to all other probes. Furthermore, entry M_{ij} of $\mathbf{M}$ for $i \neq j$ is $-w_{ij}$ if there is a measurement of weight w_{ij} between probes i and j. Thus, the sum of column j is 0, and the rows of $\mathbf{M}$ are linearly dependent. Thus, we are free to replace one row with the corresponding row from the identity matrix, which removes the degree of freedom in the system.

In general, there will be a degree of freedom in the system $\mathbf{Mx} = \mathbf{r}$ for each connected component in the graph. The argument above can be extended to the case in which there are multiple connected components in the graph. Nodes in the search tree with multiple connected components can occur commonly in the interior of the tree where there are only a limited number of constraints (i.e., only between probes ordered at a node). In this work, for all connected components of the graph associated with $\mathbf{M}$, we apply the procedure described above, replacing the row of the lowest numbered probe in each connected component with the corresponding row from the identity matrix with the effect that the indicated probe is assigned to an arbitrary coordinate $x_i = i$. The arbitrary positioning may violate the probe ordering but it does not affect the value of the cost function. This is discussed further at the end of this section.

Next, we allay a few potential concerns about the correctness of the algorithm. The first

concern is the case in which the matrix solution of a leaf node, P, returns a positioning of probes that is inconsistent with the ordering of the probes at node P. For example, one possibility is that there is a probe for which there are no measurements between it and any other probe. It is not surprising that the matrix solution for the ordering at node P might return a solution in which this probe is placed somewhere else, resulting in a solution with a different ordering than the ordering at node P. However, the matrix solution can produce a different ordering even in the case where every probe has been measured sufficiently to eliminate such degrees of freedom (as is described in Section 2.2) in the solution. Is the cost of the solution in such cases an attainable lower bound on the cost of the ordering of P? The cost is a lower bound, but it is not attainable. The cost function minimized at node P is the sum of squares cost function with the absolute value signs removed according to the ordering of probes at P. The matrix solution minimizes the cost function over all $\mathbf{x}$, not just the region in which $\mathbf{x}$ respects the ordering at P. A minimum over all $\mathbf{x}$ is a lower bound on the minimum of the cost function over the region in which $\mathbf{x}$ respects the order at P. The effect of computing a cost function that minimizes over all $\mathbf{x}$ rather than $\mathbf{x}$ that are consistent with the ordering at P is that if we are searching for nodes that have a cost within 10% of the minimum cost, then we might erroneously report node P as having a cost within 10% of the minimum when the real minimum of any solution respecting the ordering at P is not within 10% of the minimum cost. This does not affect the correctness of the search for the global minimum since if the ordering returned by the matrix solution computed at node P disagrees with the ordering at node P, the global minimum can't be the ordering of node P.

One other potential concern is if there is a probe A for which there is only one measurement to another probe, say B. In this case, A could be placed on either side of B with the same cost. One might be concerned that this means that the matrix solution of a leaf node of an order in which A is to the left of B might be an ordering in which A is to the right of B and that the cost of the two orderings might be the same. This is impossible since the removal of the absolute value signs in the sum of squares removed the symmetry in the placement of probe A.

To understand this better, consider the case of a leaf node P with ordering O_P including adjacent probes A and B in which the matrix solution respects the ordering O_P. Now consider an ordering O_Q at node Q that is identical to O_P except that the two adjacent probes, A and B, have been flipped. The original cost function will have the same value for the orderings O_P and O_Q, but the cost function used at leaf node P won't. The only term that would change in the cost function between the two orderings is the term involving the measurement between the two probes A and B. At leaf node P we have replaced this term with a simple quadratic. Let's say that at this leaf node, the ordering specifies that A is to the right of B, ($x_A > x_B$). The cost function for the term is $(\mid x_A - x_B \mid - d_{AB})^2$, which becomes $(x_A - x_B - d_{AB})^2$ at node P. If we fix x_B and plot this term of the cost function versus x_A, we obtain the parabola on the right side of Figure 7.

While the original cost function would return equal values for probe A positioned on either side of probe B, with the absolute value removed from the term in the cost function, the minimum of the cost function at node P is fixed with A to the right of B (the parabola on the right in Figure 7). The equal cost solution with A placed to the left of B will be found when exploring orderings with A to the left of B.

The final concern is minimizing the cost function at a node in which there are two sets of probes that have no measurements between them. This is depicted in Figure 8.

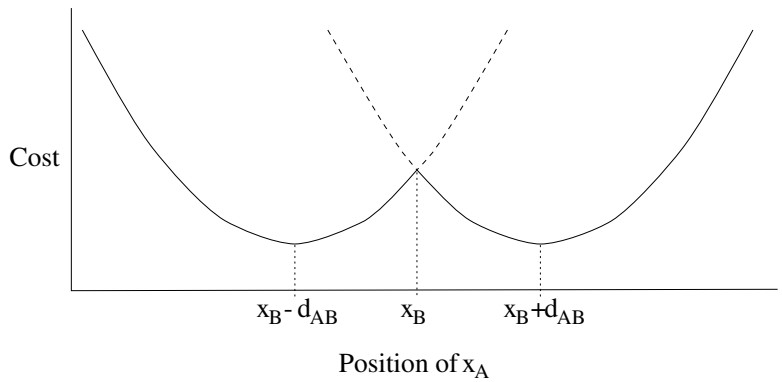

Figure 7: The solid line is $(\mid x_A - x_B \mid - d_{AB})^2$. The parabola on the right is the term used at node P, $(x_A - x_B - d_{AB})^2$.

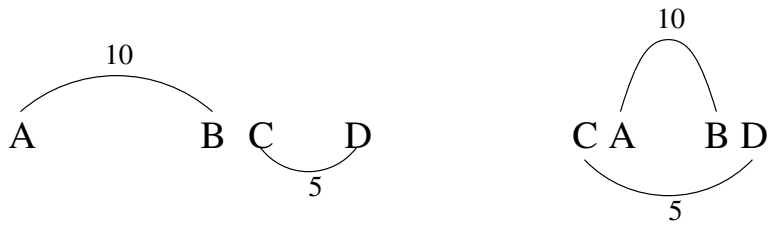

Figure 8: The correct cost for the order $CABD$ is non-zero. We can see from the drawing on the right that there is no way to respect the pairwise measurements between probes A, B and C, D in this ordering. However, the matrix solution will return a zero cost and positions similar to the drawing on the left.

At the leaf node for the ordering $CABD$, the method we use will compute a cost of 0 for the ordering. However, we can see that the cost is non-zero as there is no way to place the probes on a line respecting both the ordering and the measurements between the probes. 0 is a lower bound on the cost for the ordering, but is not an attainable cost for the ordering. As with the previous example, this does not affect the correctness of the search for the global minimum. However, since this is the only node where this ordering is considered, we will report this node as having a cost within 10% of the global minimum (since the cost function can not be less than 0), when in fact the cost of any solution respecting the ordering might have a cost greater than 10% of the minimum. In general this means that we might erroneously report some nodes as having a cost within 10% of the global minimum in the cases when the matrix solution returns a ordering that does not respect the ordering of the node at which it was evaluated.

In this section we have seen that the matrix solution is still a lower bound on the cost for a node in the case of leaf nodes. The arguments can easily be generalized for interior nodes in the search tree. Although the algorithm described does provide a lower bound, we would like a tighter lower bound. The problems described in this section have been due to the fact that the matrix solution is a minimum over all $\mathbf{x}$ when we are really searching for a minimum over

$\mathbf{x}$ that respect the ordering of the node at which it is evaluated. In the next section we describe quadratic programming, a technique that finds such minimum over restricted domains.

5.2 Improving the Least Squares Approach With Quadratic Programming

Quadratic Programming is an optimization method that minimizes a quadratic function subject to a set of linear inequalities. In our case, inequalities specify the probe ordering. The problem has been studied extensively. See, for example, [Goldfarb 93] for a brief description of quadratic programming and an algorithm to solve it.

By using quadratic programming instead of computing a matrix solution for a node P, we would obtain the lowest cost solution to the sum of squares that respects the ordering of P. This improved lower bound could potentially improve the effectiveness of the pruning. Whether this potential improvement is sufficient to offset its extra computational cost is unclear. Quadratic programming was not considered in this work due to lack of time.

6 Local Search

We now describe the local search algorithm. The idea of the local search algorithm is fairly straightforward. It is motivated by a gradient descent algorithm with random starts. The algorithm is:

1. Pick a random starting position for each of the N probes.
2. For each probe i, construct a signed integer, Δx_i:

$$\Delta x_i = \sum_{\substack{\forall j,\, x_j > x_i \\ d_{ij} \text{ measured}}} w_{ij}((x_j - x_i) - d_{ij}) - \sum_{\substack{\forall j,\, x_j < x_i \\ d_{ij} \text{ measured}}} w_{ij}((x_i - x_j) - d_{ij})$$

   The vector $\Delta\mathbf{x}$ comprised of these Δx_i's is the gradient vector derived from the cost function in Equation 5 (see Equation 6).

3. Adjust Δx_i by performing:

$$\Delta x_i = \Delta x_i \quad / \sum_{\substack{\forall j \\ d_{ij} \text{ measured}}} w_{ij}$$

   The summation term can be thought of as the number of measurements from probe i to other probes compensated by the weight of each measurement.

4. Attenuate Δx_i by dividing by a constant factor, γ. Initially, this constant is 1, and is increased to 2 if the local search has not converged after 1000 iterations.
5. Update the position of probe i by Δx_i.
6. Repeat from step 2 until $\Delta x_i \leq \varepsilon$ (to integer precision) for all i. Since Δx_i is an integer, ε can be 0, and that is the value we use in our algorithm.

One way of interpreting this algorithm is to think of the measurement between two probes as a spring separating them that has a relaxed length equal to the measured distance between the probes. Under this model, the algorithm simulates the movement of the probes attached to each other by these springs. Note that the adjustment in step 3 to the gradient vector computed in step 2 will most likely not result in a vector pointing in the direction of the gradient. The adjustment in step 3 is *ad hoc*, and while in early experiments it seemed to be beneficial, later experiments cast doubt on its utility. However, in the interests of consistency of the experimental results, it was retained in all experiments.

To compete the specification of this algorithm, we need to specify how a random start is constructed. Ideally, we would like to choose initial positions for the probes randomly over an interval that approximately matches the interval over which the probes are actually located. This interval is estimated by estimating the average distance between adjacent probes and then multiplying by the number of probes. The average distance between probes is estimated by computing the average pairwise measurement and dividing it by one plus the expected value of the number of probes separating a measurement between two probes. If we approximate the distribution of probe measurements as uniformly random over all possible pairs, then:

$$E(\text{Number of probes separating two randomly chosen probes}) = \frac{N+1}{3} - 1$$

In the algorithm, the expected distance between two adjacent probes is multiplied by $(N + 3)$, not N, to obtain the estimation of the interval. The reason is that a few problem instances arose in which using $(N + 3)$ significantly reduced the number of random starts required by local search to find the global minimum. The choice of $(N + 3)$ is *ad hoc* and seemed to improve the performance of the local search in some cases while not hurting it in general.

We have no proof of the convergence of the local search algorithm, but empirically, it does converge. One nice property of the local search algorithm is that the value it converges to is the minimum value of the cost function for some ordering (the same value as the matrix solution of that ordering). This is because the algorithm iterates until $\Delta x_i = 0$ (to integer precision) for all i. Since the adjustments to the components of the gradient vector computed in step 2 are by non-negative constant factors, all Δx_i's will be 0 only when the gradient vector is 0. This occurs when the cost function is at a minimum value for some ordering of the probes.

However, in practice, the accuracy of the solution to which it converges is limited by the convergence criterion (the test in step 6 above). To obtain convergence to a local minimum of the cost function, the Δx_i would have to be floating point precision and the algorithm would have to be run until it converged to machine precision. Our experience is that the algorithm does not obtain additional digits of precision in the answer quickly. For this reason, once the Δx_i's converge to 0 (to integer precision), the resulting ordering of the probes is used as input to compute a matrix solution to refine the precision of the result. Since computing the matrix solution finds the minimum cost of an ordering, theoretically, the local search algorithm only needs to iterate until the positions of the probes settle into an order, not until they converge. It appears that a fair number (maybe $1/2$) of the iterations of the local search occur after the probes have settled into an order. This suggests that we could speed up the local search procedure if we could stop it when we knew the order of the probes had stopped changing, and then compute the matrix solution to return the minimum cost for the ordering. Unfortunately, it is difficult to make such a prediction about the movement of the probes.

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

7 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. The branch and bound algorithm evaluates the weighted sum of squares cost function, and initializes the lower bound on the cost function to the results of a local search. It also searches for the global minimum only, rather than orderings with a cost within 10% of the global minimum. The local search algorithm performs a constant number (300) of random starts.

We describe the data over which the simulations were run and present the results of the simulations categorized by the type of simulation.

7.1 Synthetic Data Generation

All of the simulations of the algorithms were on synthetic data. We attempted to create synthetic data that produces performance similar to the performance of the algorithm on real data. To create a problem instance, we first created a set of probes and positioned them on a line. We then simulated performing measurements on various pairs of probes. In this section we describe this process. We define "Problem Size" to be the number of probes in a problem instance.

7.1.1 Generating the Probe Distribution

The first step in data generation is to place the probes on a line. We model the placement of probes by placing them randomly over an interval while limiting the minimum and maximum separation between adjacent probes. Although the abstract model of the problem presented in Section 1 regards the probes as points, they are in fact DNA segments of about 40,000 base pairs in length. Consequently, since probes can not overlap on a chromosome (otherwise they will not hybridize), the centers of adjacent probes must be separated by at least this length. In addition, it appears that in some examples of real data, a majority of the measurements between presumed adjacent probes were performed, meaning that those probes are separated by the minimum resolution of FISH, and no further apart than the maximum resolution. To better understand the utility of these observations, we perform two experiments to gauge the effect of limiting probe separation on the performance of the branch and bound algorithm. First, we fix the maximum separation and graph the branch and bound performance versus the minimum separation of the probes. Then we perform the same experiment, but without limiting the maximum separation between probes. The results are presented in Figure 9.

The units of the graph of Figure 9 are arbitrary. The important point is that the performance appears not to be sensitive to the maximum probe separation. It also does not seem to be sensitive to non-zero minimum probe separations. However, performance does seem to deteriorate when the minimum separation is 0. It's intuitively plausible that the search algorithms will have trouble handling cases of near-zero separation between probes. We choose a minimum

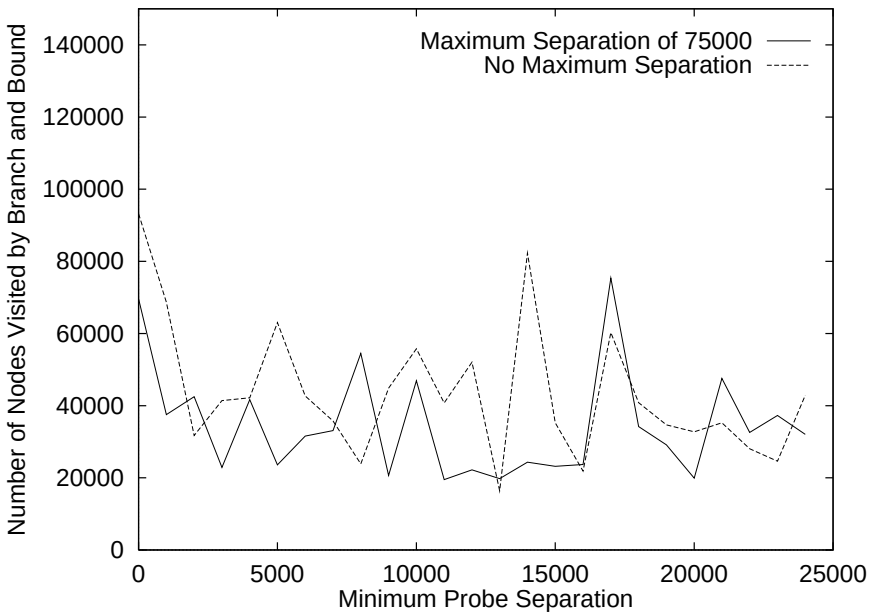

Figure 9: The two lines show the median number of nodes visited by the branch and bound algorithm as the minimum separation of the probes was changed. The median is over 20 problem instances of size 10, and the range over which probes where placed is 75000 units $\times$ the number of probes.

separation of 1250 units and arbitrarily set the maximum to 75000. The distance is chosen uniformly over this interval. Scaled to genomic distance, the minimum separation corresponds roughly to the minimum resolution of FISH.

Having established the generation of probe positions, we now discuss the simulation of measurements.

7.1.2 Generation of Measurements

When synthesizing measurements, the three parameters to specify are the number of repeated measurements performed between a specific pair of probes, the percentage of different pairs measured, and the distribution of the pairs chosen.

The first parameter considered is the effect of the number of measurements for each pair of probes on the quality of the solution returned. This simulation was run on problems of size 10 and 16. We measure all possible pairs of probes and for each pair of probes perform the same constant number of measurements. To assess the quality of the solution returned, we used the metric of the square root of the mean squared error of the probe positions as returned by the branch and bound. This metric compares the calculated position of the probes to the true positions from which the synthetic data was generated. In the graph in Figure 10, we plot the quality of solution returned for a problem size of 10. A similar graph for a problem size of 16 is shown in Figure 11.

Judging from the curves, it appears that doubling the number of measurements between

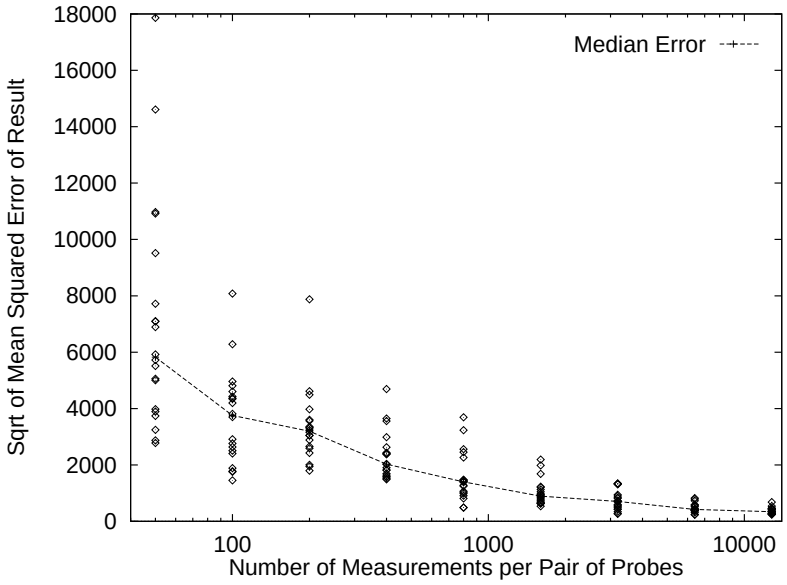

Figure 10: Quality of solution found versus number of measurements made for each probe pair. Problem size is 10.

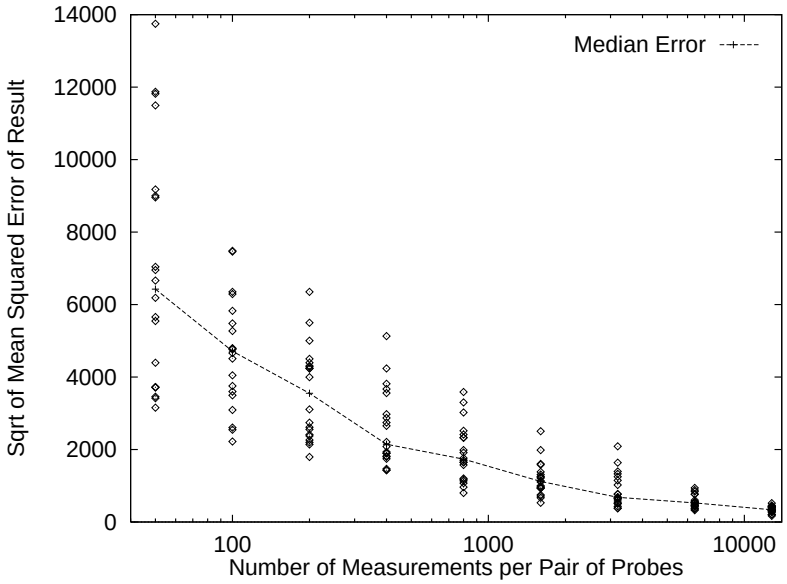

Figure 11: Quality of solution found versus number of measurements made for each probe pair. Problem size is 16.

each pair of probes reduces the error by about 25% for the data tested. Note the median RMS error is about 13% of the average spacing between adjacent probes (approximately 37,000 units) for experiments of 100 measurements between probes, and decreases as the number of measurements increases. The data presented in [van den Engh 92] is based on 100 - 200 measurements per probe pair, and so we simulate 100 measurements per pair of probes in the experiments reported below.

Another concern is the distribution of pairs of measurements to be made. [van den Engh 92] and [Trask, personal communication, 1996] present matrices of pairwise measurements where the columns (and rows) of a matrix reflect the presumed ordering of the probes. If pairs of probes chosen to be measured were selected at random, we would expect the matrix to be uniformly randomly populated. However, in practice, the experimenters seem to perform more measurements on nearby pairs than distant ones, presumably by choosing pairs to measure at least partially based on tentative ordering obtained from early measurements. This bias is evident in the matrices of data as clustering along the diagonal. To evaluate the sensitivity of the performance of the branch and bound algorithm to distribution of pairs of probes measured, we examine two methods to choose probe pairs on a few test cases.

In the first method, we attempt to duplicate the distribution of measured pairs of probes as observed in real data. To duplicate the clustering of measurements along the diagonal, we choose a pair to measure with probability inversely related to its distance from the diagonal. This probability function is constructed from the real data available. For a each problem instance, the matrix of all probes, ordered in their actual order, is constructed, and the pairs to be measured are chosen using this probability function.

In the second method, we randomly choose the pairs to measure with a fixed probability. To allow comparison with the first method, this probability is chosen such that the number of pairs of probes measured will be about the same in both methods. For each problem instance, we compare the performance of the branch and bound algorithm using data generated by the two methods.

The branch and bound algorithm visits about twice as many nodes when executed on a problem where the probes to be measured where chosen randomly (the second method). Thus, to be conservative, in the experiments performed in this work, the pairs to be measured are chosen randomly. The number of pairs chosen is determined so as to be consistent with the real data. We did not investigate the effect of the distribution of the chosen probe pairs on the quality of the solution obtained, although this is an interesting question.

Each problem instance generated is screened to remove instances in which there could be multiple global minima. For example, a set of probes with no measurements to another set of probes would result in multiple global minima. Other cases in which the problem instance is under-constrained (as is described in Section 2.2) are also filtered out. The reason for filtering out such instances is that it appears that such patterns of measurements do not appear in real problem instances such as described in [van den Engh 92] and [Trask, personal communication, 1996].

The algorithm to test if an instance has multiple global minima is conservative in that it could flag an instance as having multiple global minima when in fact it is sufficiently constrained. The algorithm constructs sets of probes for which the sub-problem consisting of only those probes has a single global minimum. Each probe is initially placed into its own set. The problem instance is declared to have a single global minimum if there is only one remaining

set when all possible merges have been performed. The conditions under which two sets are merged are as follows:

1. If both sets are singletons and there is a measurement between them.
2. If only one set is a singleton and there are at least two measurements from it to the other set.
3. If one of the sets has two probes with the property that one probe has at least one measurement to the second set and the other probe has at least two measurements to the second set.

The experiments described in this section by no means represent an exhaustive exploration of the proper way to synthesize data. As mentioned above, examination of data provide by Prof. Trask suggests that successive probes to be measured are influenced by some form of ordering analysis on data accumulated so far. Prof. Trask also mentioned that it is not known what the best method to perform real data collection is. It would be interesting to investigate ways of integrating ordering analysis like branch and bound search into data collection to improve both the efficiency of collection and the performance of the branch and bound search.

7.2 Timing Results

The first performance metric used is total CPU time taken by both the branch and bound and the local search algorithms. CPU time is an appropriate performance measure to gauge the running time of the algorithms. All experiments reporting CPU time 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 in this work. In particular, the LU decomposition routine was copied without modification from [Press 92]. The running times of these algorithms could vary by as much as a small constant factor if optimized. Since the purpose of the simulations was to examine characteristics of performance that are of a larger magnitude than a small constant factor, extensive optimization of the simulation code was deemed unnecessary.

The final performance note to consider is that CPU time is different from wall clock time by the time spent doing non-computational tasks (such as paging). Since the process size for these algorithms was around 3 MB, since the simulation code is CPU intensive, and since the unoptimized C code contributes a small constant factor to the total CPU time, the time due to non-CPU activities does not significantly affect the results shown.

We present the total time for the branch and bound algorithm using weighted least-squares in Figure 12, and the total time for local search in Figure 13. Each point in the graph is a problem instance.

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 local search is a little more difficult to assess. For a problem size of 18, the local search took about 40 seconds. It is difficult to assess whether or not the time taken is increasing exponentially. Data from much larger problem instances might help clarify. Since

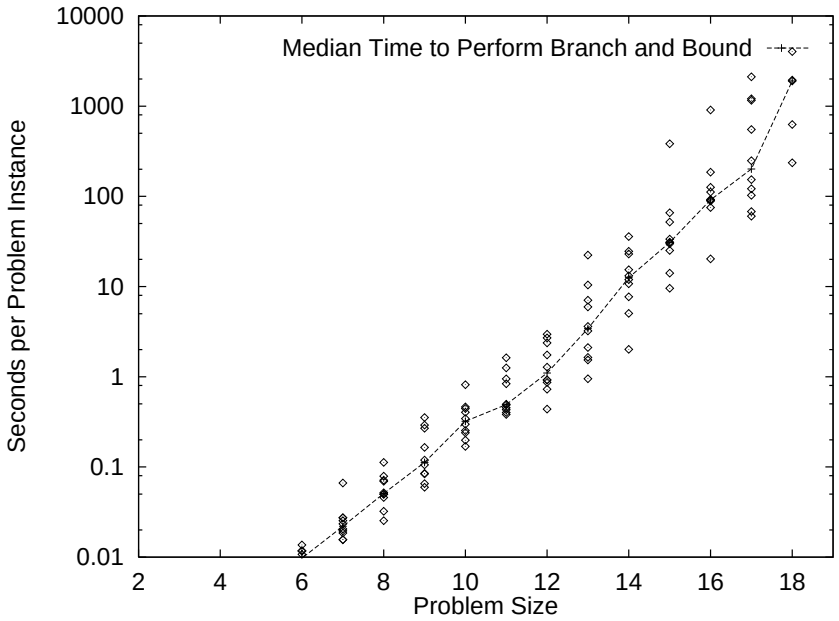

Figure 12: Time for Branch and Bound.

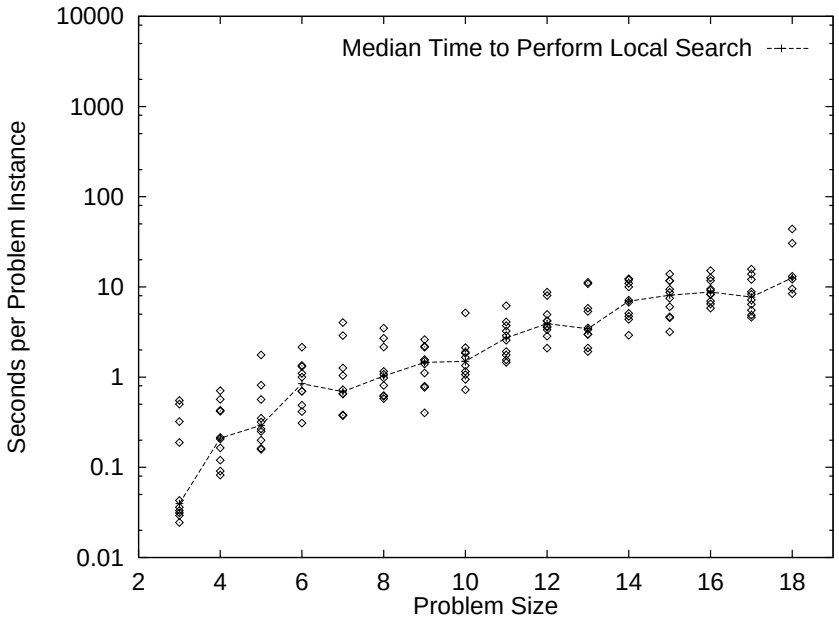

Figure 13: Time for Local Search (300 random starts).

the local search is a gradient descent algorithm and we know the function that it is minimizing, it also might be possible to analyze the running time. See also Section 7.4.

7.3 Simulations of Branch and Bound

In this section, we examine the effect on branch and bound performance of various parameters of the algorithm. We consider the effect of performing a local search to find a good lower bound on cost when starting the branch and bound search, and the performance penalty of expanding the search to find orderings of cost within 10% of the lowest cost.

First, we present data showing the effectiveness of using local search to provide a starting cost for branch and bound. Without local search, the branch and bound algorithm must start with a pessimistic assumption of a lower bound on the cost function. Since the lower bound on the cost used in pruning is taken only over leaf nodes, with a pessimistic assumption of the lowest cost, the branch and bound algorithm is not able to prune any nodes until it reaches a leaf node. Even then, the pruning will not become effective until the search reaches a leaf node with a relatively low cost. The result is that the side of the search tree where the branch and bound algorithm begins will be fairly heavily visited. The graph in Figure 14 shows the factor of reduction in the number of nodes visited by the branch and bound when the lower bound on cost is initialized to the result of a local search.

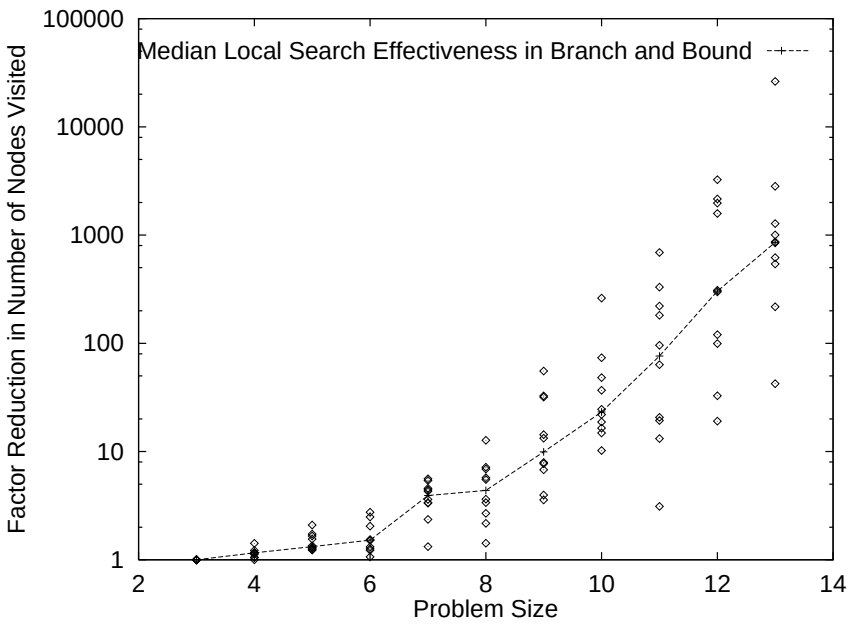

Figure 14: Factor reduction in the number of nodes visited by branch and bound when the lower bound on cost is initialized to the result of a local search (300 random starts).

Previously, we suggested searching for leaf nodes with a cost within 10% of the minimum as a technique to compensate for the inaccuracies of the least squares approach and the noisy data. To ascertain the penalty on pruning effectiveness of this alternative goal, we plotted the

factor reduction in nodes visited in a search for the global minimum only versus a search for nodes costing within 10% of the minimum. This is plotted versus problem size in Figure 15.

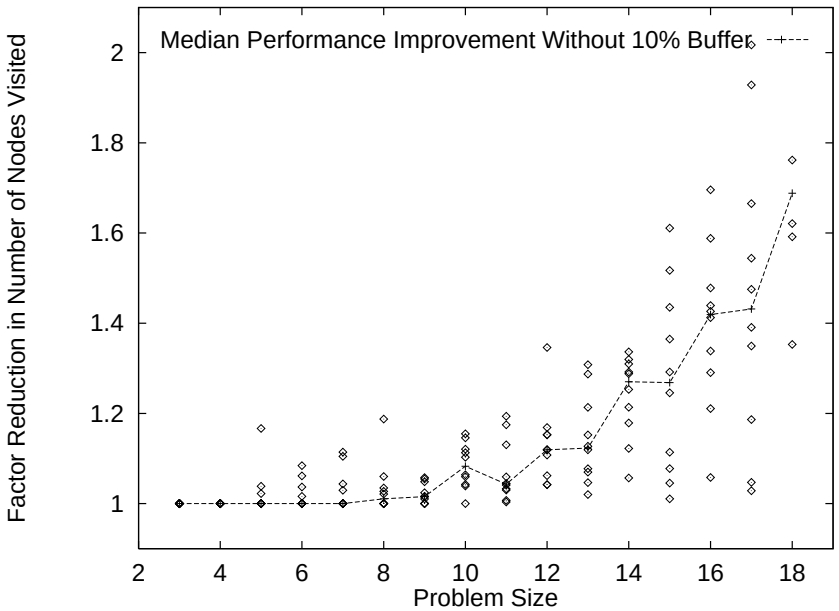

Figure 15: Factor reduction in the number of nodes visited by branch and bound in a search for the global minimum only versus a search for nodes costing within 10% of the minimum.

From the graph we can see that searching for nodes of cost within 10% of the minimum causes branch and bound to explore many more nodes in the search tree. For example, at a problem size of 18, the search found only about three or four nodes of cost within 10% of the node with the globally minimum cost. Those nodes tend to be similar to the ordering of the node with the minimum cost except for a few, mostly adjacent, pairs of probes transposed. Of course, these results are dependent on the factor 10%. If we increased that percentage, we would see a greater number of nodes returned, but also a significantly larger fraction of the tree searched. One reason that a large portion of the tree is searched may be due to the limits of the cost function calculation at interior nodes in the tree. At high levels in the search tree, it is not able to produce a tight lower bound, which makes the pruning heuristic sensitive to the 10% limit. Unfortunately, given the high variance of the probability distribution of Equation 1, even 10% might be an unreasonably strict limit (i.e., not reporting nodes that could be the correct ordering). Further investigation might suggest a more appropriate limit. Another possible approach to avoid the performance penalty of this limit would be to run the branch and bound algorithm searching only for the global minimum so as to maximize its performance, and then develop a method to tweak the ordering produced to try to generate other orderings with similar cost.

We now examine the performance of the local search algorithm.

7.4 Local Search

In all the previous experiments involving local search, the number of random starts has been fixed at 300. To gauge how many random starts were on average required for local search to find the global minimum, we performed a two stage experiment. First, branch and bound was run on a problem instance to determine the global minimum. Then, local search was run on the same instance to measure the number of random starts required for local search to find the global minimum. The results are shown in Figure 16.

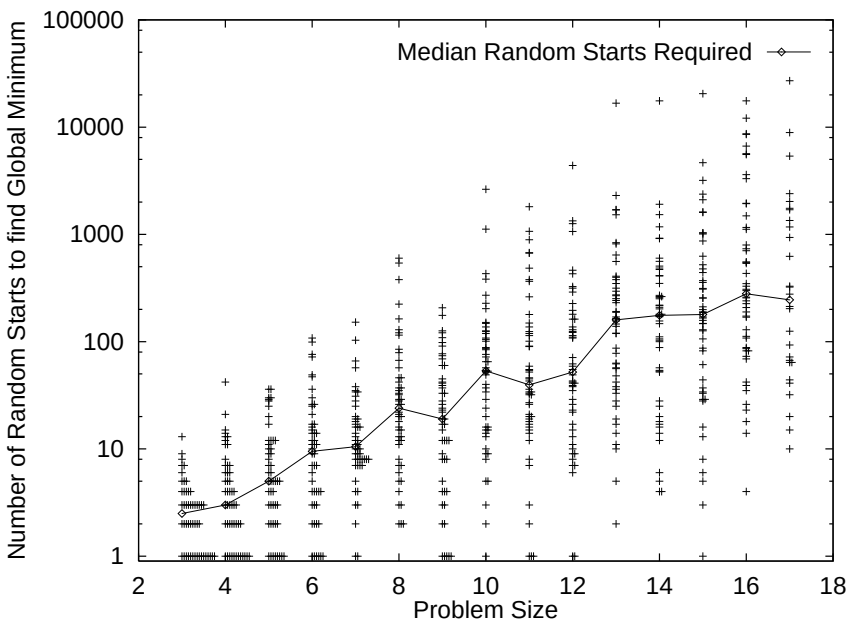

Figure 16: Number of random starts before local search found the global minimum. Wider points refer to multiple problem instances for which local search performed similarly.

Although the number of random starts has a high variance, it seems that the average number is increasing slowly, but exponentially. It is surprising that the median number of random starts for a problem size of 17 was under 300. The high variance of the results suggest that the local search algorithm is very sensitive to certain characteristics of problem instances. It would be interesting to investigate this further.

The final test of local search performed was to measure the time for a single random start. This was done by measuring the number of iterations for the algorithm in Section 6 to converge per random start. The results are shown in Figure 17. The number of iterations to converge appears to be growing polynomially in the problem size, but it is difficult to judge due to the increasing variance in the data for larger problem sizes. Remember that each iteration of the algorithm in Section 6 involves an $O(N^2)$ computation of $\Delta\mathbf{x}$.

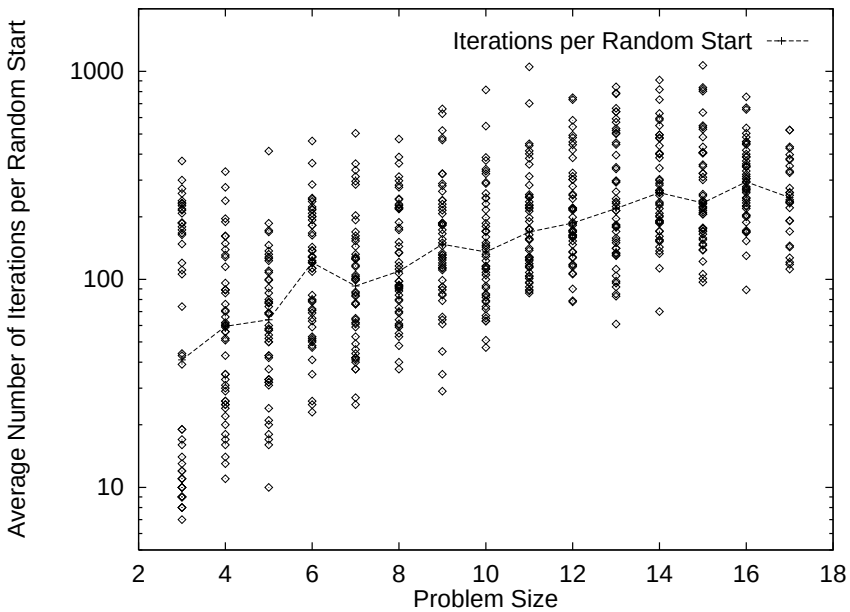

Figure 17: Number of iterations for local search to converge (per random start).

7.5 Performance Improvements in Branch and Bound Due to a Weighted Cost Function

In this section, we compare the accuracy and performance of branch and bound when the terms in the cost function are weighted to compensate for the inhomogeneous variances. The graph in Figure 18 shows the fractional reduction in the square-root of the mean squared error of probe positions found when weights, as described in Section 3.2, are applied to the terms in the cost function. Specifically, the value on the y axis is:

$$\frac{\text{RMS Error}_{\text{no weights}} - \text{RMS Error}_{\text{with weights}}}{\text{RMS Error}_{\text{no weights}}}$$

We can see that for most of the problem instances, using the weighted cost function produces significantly more accurate results. Instances in which an unweighted cost function performs better might be due to noise in the data as well as to the fact that the distributions of the measurements are violating the normality assumption of multiple linear regression.

The second experiment was to measure the impact on performance of using the weighted cost function. Figure 19 is a graph of the factor reduction in the number of nodes visited when using a weighted cost function versus the original unweighted cost function. It shows a substantial improvement in performance. Perhaps this is because the cost function with weighted terms is providing a tighter lower bound, improving the pruning on interior nodes in the tree.

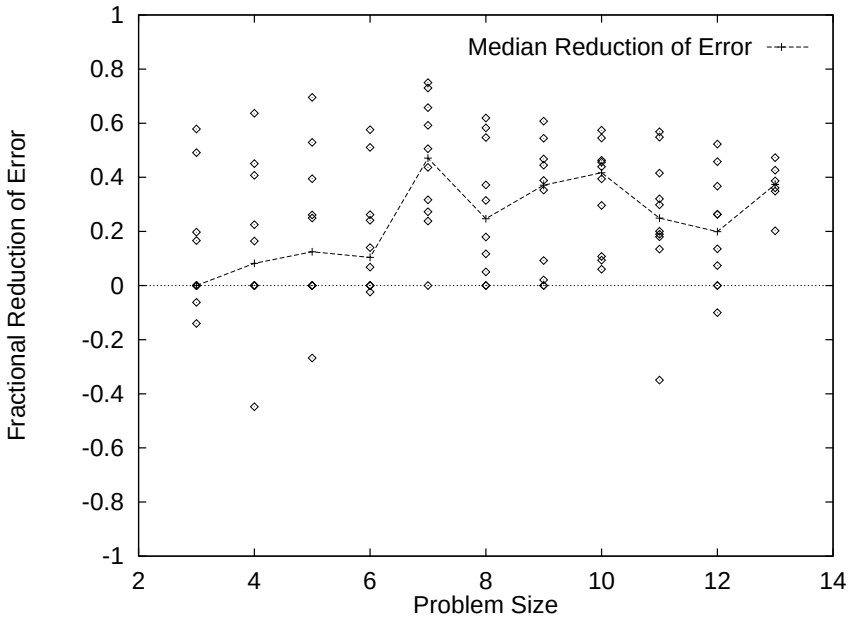

Figure 18: Fractional reduction in the square root of the mean squared error due to a weighted sum of squares cost function.

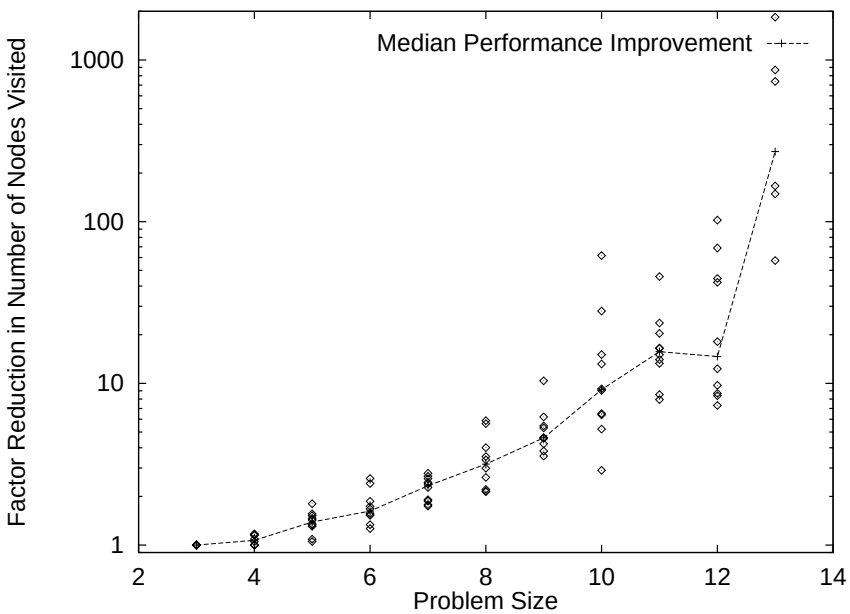

Figure 19: Factor Reduction in number of nodes visited with a weighted cost function versus an unweighted cost function in branch and bound.

8 Conclusions

In this work, we have developed two search algorithms, a branch and bound algorithm and a gradient descent based local search algorithm, both of which attempt to minimize a weighted least-squares cost function. We have examined the effectiveness of different techniques to improve the performance of the branch and bound algorithm, and characteristics of performance of the local search algorithm.

The simulations show that a weighted cost function improves the quality of the result and the performance of the branch and bound algorithm. 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 probes or less. Since it finds the global minimum, it is useful as a benchmark against which to compare other algorithms.

The local search algorithm performed surprisingly well. For problem sizes of 17 or less, the median number of random starts required was under 300. A more thorough investigation of the local search algorithm could produce an algorithm competitive with brand and bound.

9 Future Work

Here are some ideas for future work:

- Further improve the least squares model for branch and bound. There are more advanced statistical methods to compensate for a non-normal distribution. Compensation might lead to a faster algorithm with better results.
- It would be interesting to investigate improvements to the algorithm to find a tighter lower bound on the cost function for interior nodes in the search tree of the branch and bound algorithm. A tighter bound would improve pruning. One possible way to improve the cost function would be to include some contribution from measurements among unordered probes.
- More thorough exploration of the properties of the local search algorithm. As it was designed mostly to assist the branch and bound algorithm, we have not yet explored many directions to improve its performance.

  For example, it would be useful to ascertain the exact performance characteristics of the algorithm, especially the optimum value(s) of the attenuation factor. In addition to providing a starting cost for the branch and bound, it may have significant usefulness as a stand alone algorithm for larger problem sizes.
- Apply the method of maximum likelihood. There has been much work on applying maximum likelihood to genetic mapping data obtained through other techniques and probability distributions. It would be nice to see how far the maximum likelihood techniques could be developed in this domain. A maximum likelihood approach could provide answers with greater accuracy as it derives directly from the Weibull distribution. This may be more effective than a least-squares approach since the Weibull distribution has a non-normal shape.

10 Acknowledgments

We would like to thank the Statistics Consulting Group for their generous assistance.

References

[Bishop 94] Bishop, Timothy. Linkage analysis: progress and problems. *Phil. Trans. R. Soc. Lond.*, 344:337-343, 1994.

[Boehnke 91] Boehnke, Michael, Lange, Kenneth, and Cox, David. Statistical Methods for Multipoint Radiation Hybrid Mapping. *Am. J. Hum. Genet.*, 49:1174-1188, 1991.

[Buetow 87] Buetow, Kenneth H., and Chakravarti, Aravinda. Multipoint Gene Mapping Using Seriation. I. General Methods. *Am. J. Hum. Genet.*, 41:180-188, 1987.

[Goldfarb 93] Goldfarb, Donald, and Liu, Shucheng. An $O(n^3L)$ primal-dual potential reduction algorithm for solving convex quadratic programs. *Mathematical Programming*, 61:161-170, 1993.

[Lathrop 84] Lathrop, G.M., Lalouel, J.M., Julier, C., and Ott, J. Strategies for multilocus linkage analysis in humans. *Proc. Natl. Acad. Sci. USA*, 81:3443-3446, June 1984.

[Mood 63] Mood, Alexander M., Graybill, Franklin A., and Boes, Duane. *Introduction to the Theory of Statistics*. McGraw-Hill Book Company, pp. 542-543, 1963.

[Myers 1986] Myers, Raymond. *Classical and Modern Regression with Applications*. Duxbury Press, 1986.

[Newell 92] Newell, William R., Mott, Richard, Beck, S., and Lehrach, Hans. Construction of Genetic Maps Using Distance Geometry. *Genomics*, 30:59-70, 1995.

[Press 92] Press, William H., Teukolsky, Saul A., Vetterling, William T., and Flannery, Brian. *Numerical Recipes in C*. Cambridge University Press, 1992.

[Thompson 87] Thompson, E.A. Crossover Counts and Likelihood in Multipoint Linkage Analysis. *IMA Journal of Mathematics Applied in Medicine & Biology*, 4:93-108, 1987.

[van den Engh 92] van den Engh, Ger, Sachs, Rainer, and Trask, Barbara J. Estimating Genomic Distance from DNA Sequence Location in Cell Nuclei by a Random Walk Model. *Science*, 257:1410-1412, 4 September 1992.