From: ISMB-97 Proceedings. Copyright © 1997, AAAI (www.aaai.org). All rights reserved.

A Fast Heuristic Algorithm for a Probe Mapping Problem

Brendan Mumey
Department of Computer Science
University of Washington
Seattle, Washington 98195–2350
brendan@cs.washington.edu

Abstract

A new heuristic algorithm is presented for mapping *probes* to locations along the genome, given noisy pairwise distance data as input. The model considered is quite general: The input consists of a collection of probe pairs and a confidence interval for the genomic distance separating each pair. Because the distance intervals are only known with some confidence level, some may be erroneous and must be removed in order to find a consistent map. A novel randomized technique for detecting and removing bad distance intervals is described. The technique could be useful in other contexts where partially erroneous data is inconsistent with the remaining data. These algorithms were motivated by the goal of making probe maps with inter-probe distance confidence intervals estimated from *fluorescence in-situ hybridization (FISH)* experiments. Experimentation was done on synthetic data sets (with and without errors) and FISH data from a region of human chromosome 4. Problems with up to 100 probes could be solved in several minutes on a fast workstation. In addition to FISH mapping, we describe some other possible applications that fall within the problem model. These include: mapping a backbone structure in folded DNA, finding consensus maps between independent maps covering the same genomic region, and ordering clones in a clone library.

Introduction

This paper presents a new heuristic algorithm for mapping *probes* to locations along the genome given noisy pairwise distance data as input. The model considered is quite general: The input consists of a collection of probe pairs and a confidence interval for the genomic distance separating each pair. Because the distance intervals are only known with some confidence level, some may be erroneous and must be removed in order to find a consistent map. A novel randomized technique for detecting and removing bad distance intervals is described. The technique could be useful in other contexts where partially erroneous data are present. These algorithms were motivated by the goal of mapping probes with distance confidence intervals estimated from *fluorescence in-situ hybridization (FISH)* experiments[1]. As the problem is too big to solve by hand, some previous algorithmic solutions have been tried, including: a seriation algorithm (Buetow & Chakravarti 1987), a simulated annealing approach (Pinkerton 1993), and a branch and bound algorithm (Redstone 1996). None of these methods can solve a realistic problem with 20 or more probes. Our probe-location algorithm has been implemented and can solve problems with 100 probes in several minutes on a fast workstation. Experimental results have been collected for synthetic data sets (with and without errors) and FISH data from a region of human chromosome 4.

We also suggest some other applications for which the probe-location algorithm could be useful: finding and mapping a *backbone* structure within folded DNA using FISH data, finding a *consensus map* from a set of independent maps covering a genomic region, and *ordering clones* in a clone library.

The remainder of the paper is organized as follows: In the first section, the PROBE-LOCATION problem is formally defined and shown to be equivalent to a particular graph problem. Certain sparse instances of the problem are shown to be NP-complete. A heuristic algorithm to find all the feasible solutions to an instance of PROBE-LOCATION is described in the next section. Provided that the input is not from the small class of hard sparse instances, the algorithm is quite effective. The subsequent section presents a general

[1] A FISH experiment measures the physical distance (on a microscopic slide) between pairs of fluorescently marked probes hybridized to an interphase chromosome(van den Engh, Sachs, & Trask 1992; Trask *et al.* 1993). For genomic distances of up to about 1-2 Mb, DNA folding can be described by a random walk model. Statistics can be used to estimate a confidence interval for the genomic distance (in base-pairs) between two probes given a measured sample of physical distances.

Copyright © 1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

randomized technique for finding and removing erroneous distance intervals in the input. Experimental results on synthetic data sets as well as FISH distance data for human chromosome 4 are given in the next section. Other problems for which the probe-location algorithm could be useful are described in the last section. Possible generalizations to the error model are also briefly mentioned.

The Problem

In this section we define the probe-location problem formally and show that it is equivalent to finding feasible edge orientations in an *edge orientation graph.*

PROBE-LOCATION

Input: A list of probes $\{p_1, \ldots, p_n\}$ and a list of m distance intervals $\{D(i,j)\}$. Each $D(i,j)$ is an interval of the form $[l,u]$, with $0 \leq l < u$, and indicates that the separation distance between p_i and p_j falls in this interval.

Output: A description of the set of feasible probe positions $\{(x_1, \ldots, x_n)\}$ such that $|x_i - x_j| \in D(i,j)$ for all distance intervals $D(i,j)$.

If $(x_1, \ldots, x_n)$ is a solution, then for each distance interval $D(i,j)$, exactly one of the following containments must hold:

$$x_j - x_i \in D(i,j) \quad (1)$$

$$x_i - x_j \in D(i,j) \quad (2)$$

These containment choices are represented in an *edge orientation graph,* $G = (V,E)$. The vertices of the graph are the variables x_i. Each distance interval constraint $D(i,j)$ contributes a directed edge in the graph between x_i and x_j. If (1) holds, then the edge is directed from x_i to x_j and is said to be *left oriented.* If (2) holds, then the edge is directed from x_j to x_i and is said to be *right oriented.*

If the orientation of each edge is specified, then finding the feasible set of probe positions reduces to solving a linear program of a particular form. Solutions to the linear program will also be solutions to the probe location problem instance. If a solution exists, the edge orientation is said to be *feasible.* Thus, the problem is to find all the feasible edge orientations and their probe position solution sets. The probe positions solution sets can be visualized by a two-dimensional apparatus consisting of vertical rods for the probes and sliding boxes rigidly attached to horizontal rods for the distance constraints (see Figure 1). If a probe bar has an attached bead that falls within a box, then the bead is constrained to stay within this box. This fixes the relative orientation of the two probes and constrains the separation distance to fall in the interval determined by the length of the horizontal rod and the size of the box.

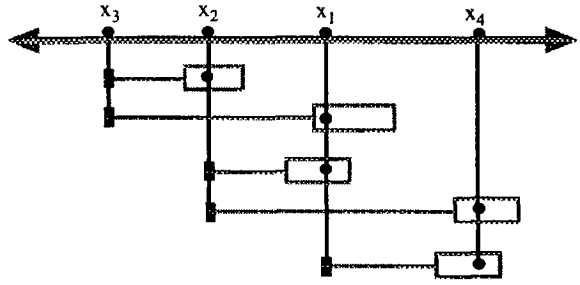

Figure 1: Representing Probe Position Solutions

We next show that a standard algorithm can be used to quickly check whether a particular edge orientation is feasible. Each directed edge in the edge orientation graph contributes two such constraints, since the containment $x_j - x_i \in [l,u]$ can be represented as

$$x_j - x_i \leq u \text{ and } x_i - x_j \leq -l.$$

A linear program with constraints entirely of this form has a useful property: It has a feasible solution if and only if there are no negative weight cycles in the edge orientation graph (Cormen, Leiserson, & Rivest 1990). The weight of a cycle is computed as follows: If an edge with associated distance interval $[l,u]$ is traversed in the forward direction, u is added to the weight; if the edge is traversed in the backward direction, l is subtracted from the weight. The Bellman-Ford algorithm (Cormen, Leiserson, & Rivest 1990) can be used to check for the existence of negative cycles in $O(nm)$ time, where n is the number of vertices and m is the number of edges. If the linear program is found to be feasible, a solution vector $(x_1, \ldots, x_n)$ is also reported.

Actually finding a feasible edge orientation can sometimes be hard as the following theorem indicates.

Theorem 1 *PROBE-LOCATION is NP-complete for edge orientation graphs consisting of one large cycle.*

Proof. The proof is via a reduction from the NP-complete problem SET-PARTITION (Garey & Johnson 1979). This is the problem of deciding whether there is a partition of a set of real numbers such that the sums of the numbers in each side of the partition are equal. Let the set of numbers of a particular instance I_S of SET-PARTITION be $\{d_1, \ldots, d_n\}$. Consider the instance I_P of PROBE-LOCATION with

probes $\{p_1, \ldots, p_n\}$ and edge set $E = \{e_1, \ldots, e_n\}$ where $e_i = (i, (i \bmod n) + 1)$ for $i = 1, \ldots, n$. Let $D(e_i) = [d_i]$. We will show there is a one-to-one correspondence between solutions of I_S and I_P. Let $(x_1, \ldots, x_n)$ be a solution I_P. We identify it with unique partition of the $\{d_i\}$ as follows: If e_i is oriented from i to $(i \bmod n) + 1$ then we put i on the left side L of the partition, otherwise if e_i is oriented from $(i \bmod n) + 1$ to i, we put i on the right side R of the partition. Notice that the sum around the cycle is:

$$\begin{aligned} 0 &= (x_1 - x_2) + \ldots + (x_{n-1} - x_n) + (x_n - x_1) \\ &= \sum_{i \in R} (x_i - x_{(i \bmod n)+1}) - \sum_{i \in L} (x_i - x_{(i \bmod n)+1}) \\ &= \sum_{i \in R} d_i - \sum_{i \in L} d_i. \end{aligned}$$

So (L, R) is a solution of I_S. Likewise any partition (L, R) which is a solution of I_S is identified with a unique edge orientation which is a solution of I_P. The reduction from I_S to I_P can clearly be done in polynomial time, so we conclude that PROBE-LOCATION is NP-complete. Δ

Fortunately, the problem becomes easier when additional distance intervals are known. Additional intervals add new edges in the edge orientation graph. New edges decompose large cycles into many smaller cycles, each of which is constrained to be non-negative. These additional constraints help to guide the search for feasible edge orientations. The next section presents an algorithm which relies on this fact to exhaustively find all the feasible edge orientations.

Finding Feasible Edge Orientations

In the previous section it was shown that PROBE-LOCATION was equivalent to finding the set of feasible edge orientations in the edge orientation graph. We present an algorithm to determine this set. The general idea is to represent edge orientations as paths in a binary tree called the *edge orientation tree*. All the nodes at a particular level in the tree are associated with one edge in $e \in E$. Left branches from nodes at this level fix e to have a left orientation. Right branches fix e to have a right orientation. Each edge in E is assigned a unique level. A depth-first exhaustive search is conducted from the root to find feasible paths. At each new node the Bellman-Ford algorithm is run to determine whether the edge orientations fixed so far are feasible. If not, the subtree rooted at this node is not considered in the search.

The performance of this approach is highly dependent on the order that edges occur in the edge orientation tree. If a poor edge ordering is used, then many dead-end branches will be explored before a feasible path is found. On the other hand, if a good ordering is used, then the search should never stray too far from a feasible path. All of the feasible paths will be quickly found. Lemma 1 shows that if an infeasible branch is taken when the opposite branch is feasible, then any infeasible set of edges will consist partially of edges above the infeasible branchpoint.

Lemma 1 *Let v be the infeasible node encountered when the algorithm first strays from a feasible path. Let the edge associated with this node be e_v and let the orientations from v to the root are fixed. Let S be any set of edges which, when considered in isolation, cannot be feasibly oriented. Then S contains at least one edge above e_v in the edge orientation tree.*

Proof. We claim that S must contain at least one edge above e_v in the edge orientation tree. If not, S consists entirely of edges e_v and below. Since v was the first node to stray from a feasible path, we know there is a feasible orientation of S following a path through the sibling of v in the edge orientation tree. (See Figure 2.) Consider the path p through v which is the mirror image of this path, i.e. starting at the parent of v, if the original path veers in one direction, then p veers in the opposite direction. We observe that the feasible orientations of set of edges considered in isolation are closed under reversal. Thus p will provide a feasible orientation of S. This contradicts the assumption that S could not feasibly be oriented given the orientations above v are fixed. Δ

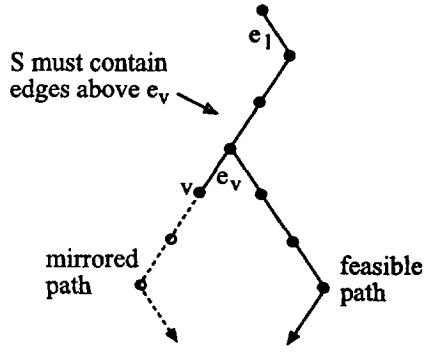

Figure 2: Taking an Infeasible Branch

A minimal infeasible set of edges is either a single infeasible cycle or the union of overlapping cycles which are simultaneously infeasible. This suggests that the edges should be ordered in such a way that cycles appear in the edge list as nearly consecutive runs. This way, when the search proceeds down a dead-end path it will be likely to encounter the remaining edges of

an infeasible set quickly and be stuck. The following heuristic attempts to order the edges so that this is the case.

The Edge Ordering Heuristic

Assume that the graph is connected. If this is not the case, then each connected component is independent and can be treated separately. The general idea is to order the edges of the subgraph induced by a set of vertices V and incrementally increase the size of V. Initially, V consists of the singleton vertex of greatest degree. The edge ordering is found by successively adding edges to the end of an ordered edge list L. The vertices are sorted in descending order of degree. A set of vertices V is maintained which contains all the vertices that are endpoints of some edge in L. Initially L is empty. A search is conducted to find the shortest path between vertices in V consisting of edges not in L. The search is done by looking for paths of length $k = 1$ which begin and end at a vertex in V. The vertices are considered in sorted order. If no path is found, k is increased by one and the path search is repeated. If a path is found, the path edges are added to L. If the path contains vertices not in V (this will happen with all paths of length greater than one), then these vertices are added to V. New paths are added in this way until any outgoing path from V consisting of edges not in L does not return to V. These edges are added to the end of L.

Detecting Errors

As mentioned in the introduction, it is possible that some of the distance intervals in the input are bad. With these bad values present, it may be impossible to find a consistent solution. This problem is ubiquitous in combinatorial algorithms which deal with real data. We outline a technique for finding a set of bad edges given an infeasible input data set. A randomized procedure is used to select a small set of inputs which, when removed, permit a solution to be found. This set is called a *bad cover*. The idea is that bad edges will have a much greater probability of being selected than good edges in a random bad cover. A random sample of bad covers can be used to estimate the underlying likelihood that each particular edge belongs to a random bad cover. The edges are sorted into list by decreasing order of this estimated likelihood. Bad edges will be biased to appear near the front of the list, while good edges will be biased towards appearing in the rear. Binary search is used to locate the minimum index in the list where discarding all the edges from the beginning of the list to that index and including the remainder yields a feasible solution.

For the probe-ordering problem, a bad cover will consist of edges from a collection of infeasible cycles. Random bad covers are constructed as follows: First a random k-coloring of the edges in the edge orientation graph is found, starting with $k = 4$. This done by randomly permuting the edges and assigning the i-th edge in the permutation the color $i \bmod k$. We consider the effect of partitioning the edges into two sets: all the edges of the first color and the edges of the remaining colors. If either of these sets is infeasible, then we recurse to look for infeasible cycles in the infeasible subsets. If not, the next color is isolated. If none of the color-isolation partitions yields infeasible subsets, then k is increased by one. An infeasible cycle of size s is guaranteed to be isolated when $k = s + 1$. If k equals the number of edges present, and none of the color-isolation partitions yield an infeasible subset, then all of the edges present must form an infeasible cycle. This cycle can then be added to the bad cover. Because a fraction $1/s$ edges are removed each time, cycles of size s will be found with at most $\log_s m$ input feasibility checks. Each feasibility check is quite fast in practice, so this approach is much faster than the naive algorithm of checking all possible s-cycles exhaustively to find an infeasible one. When the input minus the bad cover is feasible a complete bad cover has been found.

Experimental Results

In this section, we give the results of some initial experiments with an implementation[2] of the proposed probe-location algorithms. In the program, the input is first tested for feasibility. If it is found to be infeasible, the error-detection algorithm is invoked and the minimum number of bad distance intervals are removed from the input. The set of feasible solutions is reported. Testing was done with synthetic data and FISH distance data from human chromosome 4.

Synthetic Data

One interesting question is which graph structure gives the most map information for a given number of edges? In other words, if one is designing an experiment with no *a priori* knowledge of the probe locations, which pairs of probes should one select to measure and how many are enough? In the following experiment we consider four different types of graph structures built with the same number of vertices and edges:

- *random*: The endpoints of each edge are selected uniformly at random (without replacement).

[2] C++ source code is available from the author.

- *ring*: The vertices are placed in a ring, with edges between consecutive vertices. Any remaining edges are added randomly.
- *1-star*: One vertex is selected at random to be a center. Every other vertex is connected to the center via an edge. Any remaining edges are added randomly.
- *2-star*: Two vertices are selected at random to be centers. Each center is connected to every other vertex. Any remaining edges are added randomly.

To determine how well each graph type performed, a number of random instances of each type were generated for input to the program. In addition to varying the graph type, the number of edges was also varied to get an idea about the number of probe pairs needed to constrain the solution. Problem instances were generated as follows: First 40 points were sampled uniformly at random from the interval $[0, 100]$. Then edges were randomly sampled using the appropriate graph type model. For each edge distance d, the distance interval $[d/(1+s), d(1+s)]$ was used, where s is a fixed *interval tolerance*. In all of the trials, s was set to 0.1. Hence an actual distance of 10 is supplied to the program as the distance interval $[9.09, 11.0]$. Table 1 presents the results. Each input combination was run twenty times and collective statistics were tabulated. The *ambiguous probe pairs* column gives the average of the logarithm (base 2) of the number of different feasible edge orientations. The reason for taking the logarithm is that the number of feasible edge orientations should be roughly two to the number of ambiguous probe pair orientations, provided they are mostly independent. The *tree nodes* column gives the geometric mean of the number of nodes examined while searching the edge orientation tree and is roughly proportional to the running time. The number of nodes examined should be roughly proportional to the number of feasible edge orientation paths. The distribution of the number of feasible paths should be roughly the exponential of the distribution of the number of probe pair amibiguities, and so the geometric mean seems appropriate.

To test the performance of the error detection method, 10 edges were chosen at random to be bad edges. For 5 of these edges, if the true distance of the edge was d then the bad distance interval was constructed as if the distance was $d/2$. For the other 5, if the true distance of the edge was d then the distance interval was constructed as if the distance was $2d$. Table 2 presents the results averaged over twenty independent runs. The number of edges was fixed at 200. The *bad edges accepted* column gives the average number of bad edges that were not excluded from the input. The *good edges removed* column gives the average number of good edges which were removed from the input. The results do not strongly favor any one type of graph. If there are fewer edges and error-free data it appears that the *2-star* topology is the best. On the other hand, if more edges are provided and the data may contain errors the *1-star* topology appears slightly superior.

Table 1: Experimental results for error-free data sets (40 probes)

number of edges	graph type	tree nodes	ambiguous probe pairs
100	random	3458	6.1
100	ring	6349	6.0
100	1-star	1698	5.5
100	2-star	1362	4.0
200	random	3972	3.8
200	ring	3981	3.8
200	1-star	3309	4.0
200	2-star	2722	3.5

Table 2: Experimental results for data sets with errors (40 probes, 200 edges)

graph type	tree nodes	ambig. probe pairs	bad edges accepted	good edges removed
random	3384	3.9	1.0	0.3
ring	5130	4.5	0.8	0.3
1-star	2941	3.9	0.6	0.3
2-star	3633	3.9	1.1	0.4

Chromosome 4 Data

The next data set considered was based on FISH distance measurements for 10 cosmid probes on band $p16$ of chromosome 4.[3] Maps of these probes in this 4 Mb region have been published in the literature (see (van den Engh, Sachs, & Trask 1992) for a list of references). The published reference maps agree on the probe order, but the inter-probe distances reported vary between the maps by 10 to 30%. The data was collected as follows: Interphase distance measurements were made for forty-four pairs of probes. Each pair considered was measured approximately 100 times. For each pair, the mean of the squared measurements was computed (the random-walk model predicts the genomic distance to be linear in this quantity).

[3]Data set supplied by B. Trask, Dept. of Molecular Biotechnology, Univ. of Washington.

The mean values were converted to distance intervals by assuming the true distance to lie within 30% of the measured distance in either direction. Previous studies (van den Engh, Sachs, & Trask 1992; Trask *et al.* 1993; Yokota *et al.* 1995) have shown this corresponds to a high confidence interval (at least 90%) for genomic distances up to about 1.5 Mb. These distance intervals were supplied as input to the program. Only one solution was found, and it agreed with the established map in both probe order and approximate probe location. It was interesting that it was necessary to discard about 20% of the intervals in order to find a consistent solution. If the interval tolerance was lowered to 25% a few more intervals were discarded but the same general solution was found. Likewise, if the tolerance was raised to 35%, the same solution was found, with a few less interval discarded. The discarded edges were mostly too short to fit into the solution map, which predicted their separation to be larger. One explanation would be that the chromosome is folding back on itself more than a random walk would predict. Thus the genomic separation distance between some pairs of probes are underestimated by the random-walk model. We discuss this more in the future work section.

Future Work

Three other problems for which the probe-location algorithm could be useful are mentioned. The issue of generalizing the pairwise distance error model is also discussed.

Mapping DNA Backbone Structure

It has been proposed (Sachs *et al.* 1995) that nucleic DNA folds along a protein backbone structure which itself behaves like a random walk (Yokota *et al.* 1995). Along the backbone, loops of size 1-2 Mb occur periodically (see Figure 3). The physical distance separating a pair of probes is described by a random walk of length d, where d is the genomic distance separating the probes, where possible shortcuts along the backbone are taken. Thus for pairs of probes which are both close to the backbone (the black probes in the figure), the distance separating them along the backbone can be estimated from FISH distance measurements. For these probes it should be possible to build a consistent linear map using the probe-location algorithm. In general it will not be possible to consistently place the other probes which are further away from the backbone (white in the figure) into this map. These inconsistencies should be detectable by the error-detection algorithm. If this is the case, then a map of the backbone structure will remain when these inconsistencies are removed.

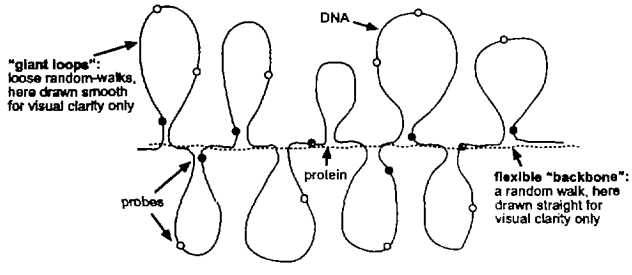

Figure 3: DNA folding along a random walk backbone

Consensus Maps

Another application is to the problem of building *consensus maps*. If several maps have been made for the same region of the genome, it would nice to know if they are mutually consistent, and if so what total constraints on the positions of the markers are imposed. Each map provides distance constraints for some set of location markers. Typically these distance constraints can be written in the form of confidence intervals. For example, distances can be estimated in restriction fragment maps by summing up the lengths of restriction fragments known to fall between a pair of markers. For STS content maps made with uniformly sized clones, distances can be estimated by the number of clones separating two markers. The constraints that each map imposes could be considered together to find a consensus map. If no solution exists, the error-detection algorithm could be used to locate the inconsistencies for further checking.

Ordering Clone Libraries

The probe-location algorithm could be useful in assisting the mapping of clones libraries, provided the map-making process provides distance information between pairs of clones. In constructing restriction fragment maps, contigs of clones that overlap substantially are first found. Distances between pairs of clones in each contig can be estimated by summing up the lengths of fragments postulated to lie between the clones. If multiple restriction enzymes are used, the contigs found in each enzyme digestion will, in general, be different. The probe-location algorithm could find clone orderings simultaneously compatible with pairwise distance data estimated from each enzyme digestion.

Generalizing the Distance Error Model

It would also be interesting to consider generalizations to the distance error model. A *biased confidence interval model* would possibly be relevant to mapping a genomic region that behaved mostly like a random walk, but occasionally looped back upon itself. In this

model, the probability that the confidence interval underestimates the true genomic distance is greater than the probability that it overestimates. In other words, it is more likely that the true distance falls beyond an inconsistent distance interval, rather than falls short of the interval. These error priors could be used in deciding which edges to remove in the error-detection phase. Finally, it would be interesting to consider more general models of pairwise measurement error. If the error distribution function was known, one could search for a maximum likelihood probe positioning. This would invariably be more computationally intensive, and might be intractable for realistic problem sizes.

References

Buetow, K. H., and Chakravarti, A. 1987. Multipoint gene mapping using seriation. I. General methods. *Am. J. Hum. Genetics* 41:180–188.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms.* Cambridge: MIT Press.

Garey, M., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness.* San Francisco: W. Freeman and Co.

Pinkerton, B. 1993. Results of a simulated annealing algorithm for FISH mapping. Communicated by Dr. Larry Ruzzo, Univ. of Washington.

Redstone, J. 1996. Algorithms for ordering DNA probes on chromosomes. Ph.D. Qualifying Exam paper. Dept. of Computer Science, Univ. of Washington.

Sachs, R. K.; van den Engh, G.; Trask, B.; Yokota, H.; and Hearst, J. E. 1995. A random-walk / giant-loop model for interphase chromosomes. *Proc. Natl. Acad. Sci. USA. Biochemistry* 92:2710–2714.

Trask, B. J.; Allen, S.; Massa, H.; Fertitta, A.; Sachs, R.; van den Engh, G.; and Wu, M. 1993. Studies of metaphase and interphase chromosomes using fluorescence in situ hybridization. *Cold Spring Harbor Symposia on Quantitative Biology* 58:767–775.

van den Engh, G.; Sachs, R. K.; and Trask, B. J. 1992. Estimating genomic distance by a random walk model. *Science* 257:1410–1412.

Yokota, H.; van den Engh, G.; Hearst, J. E.; Sachs, R. K.; and Trask, B. J. 1995. Evidence for the organization of chromatin in megabase pair-size loops arranged along a random walk path in the human g0/g1 interphase nucleus. *The Journal of Cell Biology* 130(6):1239–1249.