Constructing Maps Using the Span and Inclusion Relations

Dan Fasulo* Tao Jiang† Richard M. Karp‡ Nitin Sharma§

Abstract

Many computational problems in DNA mapping and sequencing involve determining the relative positions of DNA fragments derived from a target genome region. In the past, many such problems were approached by modeling the fragments as intervals on the real number line and computing the pairwise overlap relation among some or all pairs of intervals. The relative positions of the intervals could then be determined by an error-tolerant interval realization algorithm.

In this paper, we present a new model for the problem in which the fragments are once again intervals, but instead of pairwise overlap we compute two alternate relations which we call the *inclusion* and *span* relations. We show that in the ideal case in which both relations are known exactly, they lead to an efficient algorithm for interval realization. We also demonstrate how our methods apply to a subproblem in multiple complete digest mapping, allowing us to order the endpoints of clones accurately even in the presence of false or misleading data. Our solution to this problem is interesting because it demonstrates how the span and inclusion relations make error detection easier, and it also gives the program the freedom to temporarily discard potentially erroneous clone fingerprints so that the remainder may be correctly ordered.

*Supported in part by NSF Grant DBI-9601046 and the Human Genome Center at the University of Washington. Address: Department of Computer Science and Engineering, University of Washington, Seattle, Washington 98195-2350. E-mail: dfasulo@cs.washington.edu.

†Supported in part by NSERC Research Grant OGP0046613 and Canadian Genome Analysis and Technology Grant GO-12278. Address: Department of Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, Canada. E-mail: jiang@maccs.mcmaster.ca.

‡Supported in part by DOE Grant DE-FG03-94ER61913/A000 and NSF Grant DBI-9601046. Address: Department of Computer Science and Engineering, University of Washington, Seattle, Washington 98195-2350. E-mail: karp@cs.washington.edu.

§Supported in part by NSF Grant DBI-9601046. Address: Department of Computer Science and Engineering, University of Washington, Seattle, Washington 98195-2350. E-mail: nitin@cs.washington.edu.

1 Introduction

The central computational problems in DNA sequencing and physical mapping involve determining the relative positions of fragments of a target DNA molecule. In physical mapping the fragments are typically clones; in sequence assembly, they are short fragments that have been directly sequenced. Typical approaches to such problems use a pairwise statistic such as overlap probability for a given pair of fragments, and then extrapolate from this information to generate an ordering for all of the fragments. This paper presents a new approach to these problems based on two relations among fragments: the *inclusion* relation, which holds whenever one fragment contains another, and the *span* relation, which holds whenever the union of two fragments includes a third, but neither of the two fragments covers the third by itself. These relations are as easy to estimate as the overlap relation for most mapping problems, but seem to provide more information on the relative placement of the intervals, thus making error detection and correction easier.

The rest of the paper is organized as follows. We begin by considering an abstract problem in which the fragments are viewed as intervals on the real line, and the inclusion and span relations are assumed to be known exactly. This problem is closely related to the well-known interval realization problem, in which the interleaving of a set of intervals is determined from a list of the overlapping pairs. Next, we present an efficient algorithm to compute the interleaving of intervals in absence of any errors in the inclusion and span relations. We then go on to describe our experience in applying the approach to Multiple Complete Digest (MCD) physical mapping, a more realistic case in which errors occur in determining the inclusion and span relations among clones from clone fingerprint data. We describe a heuristic for determining the order of a set of non-nested clones which remains accurate even in the case of erroneous and misleading data.

2 The Interleaving Problem

An inclusion relation $I(A, B)$ holds between intervals A and B on the real line if interval A includes interval B. The span relation $S(A, B, C)$ holds if interval B is included in the union of intervals A and C, but neither A nor C by itself includes B. Given span and inclusion relations for intervals, the interleaving problem is to find if there exists a realization of the intervals which satisfies the relations and, if so, to find such a realization.

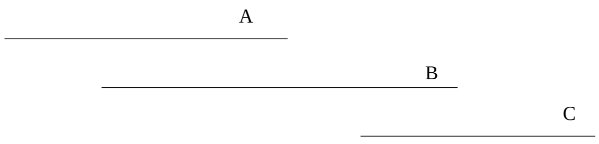


Figure 1: An example of an interleaving.

The *interleaving* of a set of intervals is defined as the ordered sequence of their initiation and termination events. For example, the interleaving of three intervals A, B, and C in Figure 1 is $A\ B\ A'\ C\ B'\ C'$. Here A means the initiation of interval A, and A' means termination of A. The *flip* of an interleaving is obtained by reversing the sequence of events and then replacing each initiation event by the corresponding termination event, and each termination event by the corresponding initiation event. For example, the flip of the above interleaving is $C\ B\ C'\ A\ B'\ A'$. Notice that at best, the span and inclusion relations determine the interleaving only up to a flip.

3 The Algorithm

In any realization of the I and S relations, the following must hold:

1. If $I(A, B)$ then intervals A and B intersect.
2. If $S(A, B, C)$ then A, B, C mutually intersect.
3. If intervals A and B intersect and $I(C, A)$ holds, then C and B intersect.

All intersections that can be inferred from the above rules are called *detectable* intersections.

Lemma 1. *If a realization of I and S exists, then there exists a realization in which the only intersections are the detectable intersections.*

Proof:
Given a realization with some undetectable intersections, we construct a realization which does not have the undetectable intersections, but still satisfies the I and S relation.

At each step of the construction, a pair (A, C) of intervals is considered such that A has an undetectable intersection with C, but no other interval contained in A has an undetectable intersection with C. The interval A is then shortened just enough so that it no longer intersects C.

The construction process continues until all undetectable intersections have been removed. It can be shown that the span and inclusion relations are preserved at each step. □

Hereafter we may assume that in the interleaving to be constructed, two intervals intersect if and only if their intersection is detectable. We also assume that the inclusion and span relations have a realization, and our task is to construct one. Detecting that no realization exists is an easy byproduct of this construction process.

Let us say that an interval is *maximal* within a set of intervals V if no other interval in V includes it. If all the intervals in V are maximal then it is particularly easy to determine whether a realization exists and, if so, construct one. Moreover, the realization is canonical in the following sense. Call a subset C of V *isolated* if there are no detectable intersections between intervals in C and intervals in $V - C$. Define a *contig* as a minimal nonempty isolated set. Then V is the disjoint union of its contigs, and the interleaving for V is unique up to changing the order of the contigs and flipping the interleavings within some of the contigs.

Let D be the set of intervals over which the relations S and I are defined. The main steps of the algorithm are as follows:

1. Partition D into disjoint sets $D_1, D_2, \ldots, D_t$ such that:
   - D_1 is the set of intervals that are maximal within D;
   - For $i = 2, 3, \ldots, t$, D_i is the set of intervals that are maximal in $D - \cup_{j=1}^{i-1} D_j$
2. For $i = 1, 2, \ldots, t$ partition D_i into contigs and, within each contig, determine the interleaving up to a flip.
3. For $i = 1, 2, \ldots, t$ construct E_i, a realizable interleaving of $D_1 \cup D_2 \cdots \cup D_i$.

The crux of the algorithm is the construction of E_i, given E_{i-1} and the interleaving for each contig within D_i. The construction depends on the fact that all the intervals in D_i are maximal in D_i, and each interval in D_i is included in some interval in D_{i-1}. The construction of E_i has two parts:

- Determine the ordering of the contigs within D_i and orient each contig; to *orient* a contig is to choose one of its two possible interleavings.
- Construct the interleaving E_i by merging the oriented contigs of D_i into E_{i-1}.

3.1 Ordering and Orienting Contigs

3.1.1 Ordering the Contigs Within D_i

We begin with some definitions. Intervals A and B *partially overlap* if they detectably intersect, but neither includes the other. Intervals A and B are *orderable* if neither contains the other. Since all intervals in D_i are maximal in D_i, all pairs of intervals in D_i are orderable. Given a realization and an orderable pair $\{A, B\}$ of intervals, we say that $A \prec B$ if the left endpoint of A is to the left of the left endpoint of B in the realization.

We determine the position of each contig C in D_i relative to the endpoints of the intervals occuring in E_{i-1}. There are two cases:

Partial Overlap Case Contig C contains an interval X that partially overlaps some interval A in E_{i-1}. Interval X also must be included in some interval B within D_{i-1}. The pair $\{A, B\}$ of intervals is orderable. If $A \prec B$ then the right endpoint of A lies within X, and hence within contig C. If $B \prec A$ then the left endpoint of A lies within X, and hence within C.

Inclusion Case No interval within contig C partially overlaps any interval within E_{i-1}.In this case C is contained in the intersection of all those intervals within E_{i-1} that contain all the intervals in C. C therefore lies between the rightmost left endpoint of such a containing interval and the leftmost right endpoint of such a containing interval.

The information obtained in the partial overlap case and the inclusion case is sufficient to determine the ordering of the contigs within D_i, except for the case of two contigs that are included in the same intervals within E_{i-1} and contain no intervals that partially overlap intervals within E_{i-1}. The ordering of two such contigs is arbitrary. Below is a more algorithmic description:

1. We find an interval X in contig C_j which *partially* overlaps with some interval A considered in the preceding phases.
2. We then determine if the interval X overlaps with the left endpoint or the right endpoint of A.
3. We associate event A with C_j if X overlaps with the left endpoint of A. Otherwise we associate A' with C_j. If there is *no* such X in step 2, the entire contig C_j is contained in some interval. Pick an interval X in C_j and find an interval A with *rightmost* initiation that contains X. Associate event A with contig C_j.

   This associated event, A or A', defines the relative position of contig C_j in the final map. Note that all the associated events must be in E_{i-1} and must have been ordered before.
4. We then output the contigs in the order their associated events appear in the interleaving E_{i-1}.

3.1.2 Orienting the Contigs within D_i

Let $\{A, B\}$ and $\{A', B'\}$ be two orderable pairs of intervals. The ordered pair (A, B) is called a *distinguishing pair* for the ordered pair (A', B') if one of the following conditions holds:

1. A includes A', B includes B' and B does not include A';
2. A includes A', B includes B' and A does not include B';
3. A overlaps A', B overlaps B' and A does not overlap B';
4. A overlaps A', B overlaps B' and B does not overlap A'.

Lemma 2. *If (A, B) is a distinguishing pair for (A', B') then, in any realizable interleaving, $A' \prec B'$ if and only if $A \prec B$.*

We next describe how to determine which of the two possible orientations of a contig C within D_i is consistent with the interleaving E_{i-1}. If no interval in C partially overlaps an interval in E_{i-1} then either orientation is consistent. Otherwise, it suffices to determine the ordering of any two intervals in C. Let $X \in C$ be an interval in C that partially overlaps an interval $A \in E_{i-1}$. Let B be an interval in D_{i-1} that contains X and let Y be an interval in C adjacent to X (if there is no interval adjacent to X in C then C must have only one interval and hence has only one possible orientation). Consider the possible interactions of A and Y:

1. $I(A, Y)$: In this case, (A, B) is a distinguishing pair for (Y, X), since $\{A, B\}$ is orderable, A includes Y, $Y \subseteq A$, B includes X and A does not include X. Thus $Y \prec X$ if and only if $A \prec B$. The ordering between Y and X is determined, and this determines the orientation of C. This case is depicted in Figure 2.

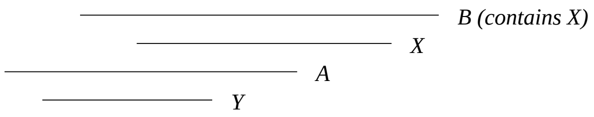


Figure 2: Case 1: If Y is contained in A, (A, B) is a distinguishing pair for (Y, X)

2. Y partially overlaps with A: In this case all pairs of intervals from $\{A, X, Y\}$ partially overlap, so there must be a span relation among the three. Therefore the ordering of A and B determines the ordering of A and X, and hence the ordering of X and Y. This case is depicted in Figure 3.

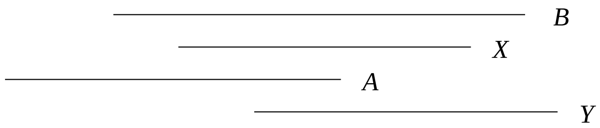


Figure 3: Case 2: If Y partially overlaps A, then A, X, and Y must be in a span relation.

3. Y and A do not overlap: Let K be an interval in D_{i-1} that contains Y. Then (A, K) is a distinguishing pair for (X, Y), as A and K are orderable, A and X overlap, K and Y overlap, and A and Y do not. This case is depicted in Figure 4.

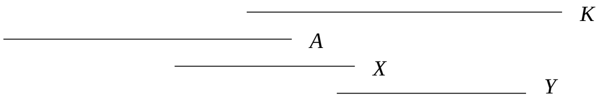


Figure 4: Case 3: If Y does not overlap A, (A, K) is a distinguishing pair for (X, Y).

So in all the cases, the ordering of some pair of intervals in contig C, and hence the orientation of contig C, is determined.

Once the ordering and orientation of components is determined, the interleaving for the set D_i is just the concatenation of the interleavings for the contigs in the order and orientation determined above. Let us call this interleaving F_i.

3.2 Merging the Interleavings

As the final step, we merge the interleaving E_{i-1} with F_i, the interleaving for D_i. Recall that, for any interval A, the symbol A denotes its initiation event and A' denotes its termination event.

Merging two interleavings is similar to merging two ordered sequences. We repeatedly compare the heads of the sequences, and output the event which should occur first in the merged interleaving. At a general step there are the following cases to consider for comparing the heads. Assume that the symbol at the head of the remaining part of E_{i-1} is either A or A', and the symbol at the head of the remaining part of F_i is either B or B'.

1. Comparing A and B: There are two sub-cases:
   (a) A includes B : Event A occurs before event B.
   (b) A does not include B: Since interval B was not maximal at phase $i-1$, there exists an interval X in D_{i-1} that includes B. If X occurs before A in E_{i-1}, B must occur before A. This is because if A occurred before B, it would lead to a contradiction for the following reason. Since A does not include B, B' must occur after A'. However, X includes B so X' occurs after B' and hence after A'. Also, since X precedes A in E_{i-1}, it follows that X includes A, contradicting the fact that A lies in some D_j, $j \leq i-1$, and X is contained in D_i. Similarly, if X does not occur before A, A must occur before B.
2. Comparing A and B': If intervals A and B intersect, A must occur first. Otherwise B' must occur first.
3. Comparing A' and B: If intervals A and B intersect, B must occur first. Otherwise A' must occur first.
4. Comparing A' and B': Intervals A and B must overlap. If A includes B, B' occurs first. Otherwise, if A occurred before B, A' must occur before B' else B' must occur before A'.

3.3 Overall Run-time Analysis

We derive an upper bound on the running time of the algorithm. Let n be the number of intervals and m, the number of pairs of intervals that detectably intersect. The main steps of the algorithm are computing detectable intersections, partitioning the set D into sets D_i, constructing, ordering and orienting contigs, and merging the interleavings. We analyze each of these steps below.

3.3.1 Computing Detectable Intersections

This step involves repeatedly applying rule 3 described in section 3. Define the *level* of an interval A belonging to the set D_i to be i, and the level of an intersecting pair of intervals (A, B) to be sum of the levels of A and B. Note that the level of the new pair of intersecting intervals obtained by the application of rule 3 is always *lower*. We apply rule 3 to the intersecting pairs of intervals in decreasing order of their levels. Using a priority queue with at most $2n$ priority levels, all detectable intersections can be computed in $O(n+m)$ time.

3.3.2 Ordering and Orienting Contigs

The sets D_i are constructed by repeatedly extracting out the intervals that are not included by any other interval. This operation can be implemented in $O(m + n \log n)$ time using a heap of intervals with the key for an interval being the number of intervals that include it.

Ordering and orienting contigs requires finding for a given interval an interval X belonging to the preceding phases that partially overlaps with it, and an interval A belonging to the immediately preceding phase that includes it. We can pre-compute the lists for each interval that have the above information in $O(n+m)$ time. Note that the lists for each interval are traversed only once during the entire algorithm. So, overall this step takes $O(m + n \log n)$ time.

3.3.3 Merging Interleavings

For each i, merging interleaving E_{i-1} with interleaving F_i takes time $O(n)$. Since the number of merges is $O(n)$, the time for all merges is $O(n^2)$. Merging could be done more efficiently by maintaining the interleaving (an ordered list of events) as a balanced binary search tree and inserting each event in F_i in this sequence in $O(\log n)$ time. So, over all phases, merging takes $O(n \log n)$ time.

Therefore, the entire algorithm runs in $O(m + n \log n)$ time.

3.4 An Alternative Approach

We sketch a second quadratic-time algorithm for the Interleaving Problem. It consists of the following steps:

1. Determine the detectable intersections.
2. Construct a graph G in which the vertices are the intervals and the edges are the pairs of intervals that detectably intersect;
3. Determine the maximal cliques of G; because G is a chordal graph, this can be done in linear time by a standard algorithm.
4. For each maximal clique, determine the interleaving of the intervals within the clique. This interleaving is unique up to a flip, and can be constructed in linear time.
5. Construct a PQ-tree representing all those orderings of the intervals with the property that the intervals within each maximal clique occur consecutively. This can be done in linear time.
6. Modify the PQ-tree to exclude those orderings of the cliques that are inconsistent with the inclusion and span relations.
7. Determine a mutually consistent set of orientations of the interleavings associated with the maximal cliques. This can be done in linear time using an algorithm for 2-CNF Satisfiability.
8. Merge the interleavings for the individual cliques to determine a valid interleaving of all the intervals.

4 Interleaving Clones for MCD Mapping

In this section, we show how to apply and extend the theoretical results of the prior sections to the *Multiple Complete Digest* (MCD) mapping problem. MCD mapping is a type of physical mapping in which a clone library is selected which covers the target region with high redundancy. The clones are then completely digested with several restriction enzymes, and the lengths of the resulting fragments are measured using gel electrophoresis. The goal is to generate a map showing the relative locations of all of the restriction sites on the original DNA spanned by the library. Such a map is a useful in selecting clones for sequencing and verifying sequencing results [11]. In previous work, we described how to generate an MCD map given both the clone fragment data and an interleaving of the clone endpoints[2, 6]. In this section, we show how to obtain such an interleaving accurately, even in the face of data errors such as measurement error, deletions, in which a portion of the clone has been removed, and coligations, in which some amount of DNA from the host has been appended to the clone.

We begin with a formal statement of the problem and an outline of prior approaches, followed by an outline of our approach. Suppose that we are given an input set of clone *fingerprints* $C_1, C_2, \ldots, C_N$ such that each C_n is of the form $\{F_{i1}, F_{i2}, \ldots, F_{iM}\}$, with each F_{im} being a multiset $\{f_1, f_2, \ldots, f_I\}$ representing the fragment sizes measured from the digestion of the clone by enzyme m. We can think of the target DNA region defining a coordinate system in which points are measured in base pairs from the end, and the clones as being intervals in this system.

Several software systems have been constructed to solve this problem, including DNAM by Gillet *et al.* [4], and FPC by Soderlund *et al.* [9], and FPC's predecessor, CONTIG9 [10]. These past approaches have mainly been based on overlap statistic methods, in which the pairwise overlap probablility is computed for some or all pairs of clones, and then some heuristic is used to construct an ordering. There have also been many methods published for estimating overlap probability. In [3], Gillet *et al.* describe an overlap statistic called the *measured overlap statistic* which is related to overlap probability. Balding and Torney [1] have developed a very good mathematical estimate of overlap probability, but it is too expensive to compute in practice. Nelson and Speed [8] have published an approach which is based on calculating the likelihood of a certain vector representing an alignment between each pair of clones, but the method assumes that the size of each fragment on a clone is independent of the sizes of the others. This assumption is not true for many types of clones, which occur only within a small range of lengths. Finally, Mumey has developed a program which estimates overlap probabilities and uses search heuristics to generate an ordering for sets of maximal clones [7].

The main weakness to these prior approaches is that the overlap relation provides less information on ordering than our span and inclusion relations. A common assumption is that the higher the estimated degree of overlap in two clones, the nearer they are in the order. However, it is easy to construct examples in which a clone is overlapped less by its immediate predecessor than by several of its successors, thus making it seem farther away from its actual predecessor than it really is.

Another weakness of prior MCD mapping programs is that they try to incorporate all of the input clones into the final map. This sounds reasonable, but does not work in practice; much of the human intervention which is required by current MCD mapping software involves identifying and removing clones which don't fit easily into the map, either because of data errors or because of ambiguity in the data. In order to make mapping software more automatic, it must be able to take similar steps on its own.

Our solution to the MCD mapping problem attempts to address these problems. In our formulation of the problem, however, we are primarily interested in creating an interleaving of the maximal clones; the subclones take a secondary role. Hence, we simplify our task to that of creating an ordering of the maximal clones by right endpoint, using the following steps:

- Compute an approximation of the span and inclusion relations, and eliminate all non-maximal clones.
- For each clone, use the relations to select a small set of ranked potential neighbors in the interleaving.
- Group the clones into lists which contain an "obvious" ordering based on the neighbor sets.
- Use heuristics to arrange all of the sublists into a single ordering.

We describe each of these steps in more detail in the following subsections, and finish with an extended example of the algorithm in operation.

4.1 Computing the Span and Inclusion Relations

We say that two fragment measurements f_1 and f_2 are *compatible* if and only if f_1 and f_2 could both be observed measurements of the same original fragment under an empirically-determined model of measurement error. Compatibility forms the basis for attempting to determine the span and inclusion relations. We say that a clone C_j is a subclone of C_i, i.e., $I(C_i, C_j)$, if and only if every fragment in F_{jm} can be matched with a distinct compatible fragment in F_{im}. Similarly, we say that C_i and C_j span C_k, i.e., $S(C_i, C_k, C_j)$ if all of the fragments in F_{km} can be matched with distinct compatible fragments in $F_{im} \cup^* F_{jm}$, where $\cup^*$ denotes multiset union, and none of the clones C_i, C_j, and C_k includes another.

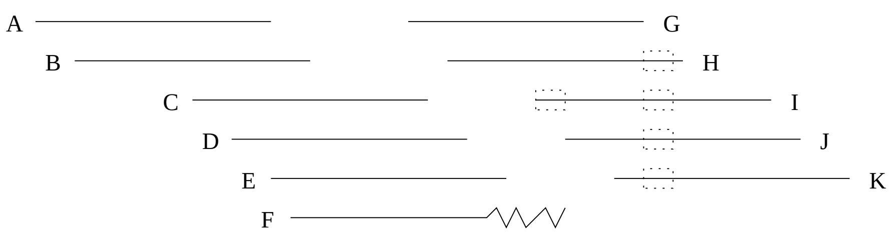
Clearly, this method of producing the span and inclusion relations is only an approximation; however, empirical evidence (see Section 4.6) shows that it is quite accurate. Nevertheless, the presence of false positives and negatives as well as the presence of defective clones and measurement errors prevents us from applying the algorithms of the previous sections naively.

4.2 Determining the Neighbors of a Clone

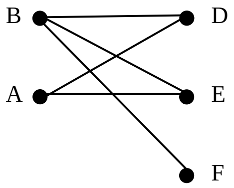
Assume that we have now computed the span and inclusion relations, and eliminated all subclones from the data. In addition, two clones C_i and C_j may seem to be *identical* if $I(C_i, C_j)$ and $I(C_j, C_i)$; for each set of identical clones we select one representative and discard the rest. The remaining clones constitute the set of maximal clones.

For a moment we consider the ideal case under which the span relation is known perfectly for each clone. Then, suppose we have a clone C for which we wish to determine the neighbors in the clone ordering. Recall that $S(A, C, B)$ implies that A and B neighbor C on opposite sides. Hence, we could determine two sets, $\mathcal{A} = \{A_1, A_2, \ldots, A_I\}$ and $\mathcal{B} = \{B_1, B_2, \ldots, B_J\}$ such that for all A_i, there exists a B_j such that $S(A_i, C, B_j)$, and analogously, for all B_j, there exists an A_i such that $S(A_i, C, B_j)$. We could then construct a bipartite graph G_C in which $\mathcal{A} \cup \mathcal{B}$ form the set of vertices and there is an edge from A_i to B_j if and only if $S(A_i, C, B_j)$. Without loss of generality, we say that the set of clones $\mathcal{A}$ precedes C and the set $\mathcal{B}$ follows it. Then, the clone A_i immediately preceding C in the ordering would have three properties:

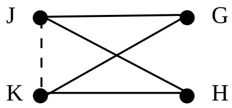
- For all $A_j \in \mathcal{A}$, $i \neq j$, $S(A_j, A_i, C)$.
- The number of neighbors of A_i in G_C would be greater than or equal to the number of neighbors of any other member of $\mathcal{A}$.
- The number of compatible fragments of A_i with those of C would be greater than or equal to the number of neighbors of any other member of $\mathcal{A}$.



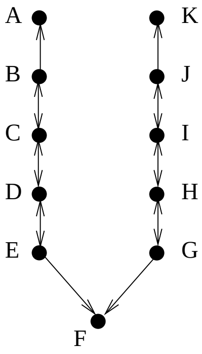
(A) A sample input data set. The jagged line at the right end of clone F represents a coligation, in which host DNA has attached itself to the clone. The dotted box at the left end of I is a fragment which is identical to another fragment in the center of I, and hence also to the copies of that fragment on H, J, and K.



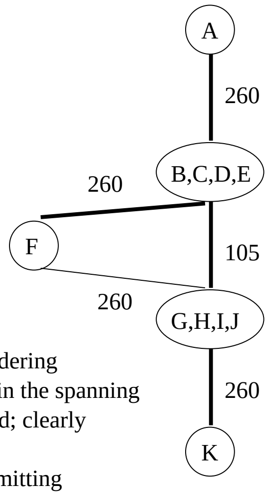
(B) The bipartite neighbor graph for C. Here, B is clearly one neighbor; the other is D because D is spanned by E and C.



(C) The bipartite neighbor graph for I. Note that because of the fragment on J which is compatible with the left end of I, it appears that J and K span I, hence causing an "illegal" edge in the graph represented with a dotted line.



(D) A graph showing the nearest neighbors for each clone. There is an edge from X to Y if Y is a top-ranked neighbor of X. Hence, the strongly-ordered components are {A}, {B,C,D,E}, {F}, {G,H,I,J}, and {K}.



(E) The component-ordering graph. The edges in the spanning tree are emphasized; clearly the longest path is ABCDEGHIJK, omitting the damaged clone F.

Each oval actually represents a pair of vertices connected by an edge of weight one. Edges touching the top of the oval attach to one vertex, and those touching the bottom attach to the other.

Figure 5: An example of the interleaving algorithm's operation.

Analogous statements can also be made for the clone immediately following C in $\mathcal{B}$.

Even in the case of erroneous relations, the preceding observations give us strong clues as to the identities of the clones neighboring C in the ordering. Thus, to determine the neighbors of C, we begin by creating a neighbor graph G_C with an edge from clone A to B if and only if $S(A, C, B)$. Ideally, this graph should be bipartite, but false positives in the span relation may introduce "illegal" edges. Hence, we use the following simple randomized routine to assign sides to the vertices of G_C in a way which attempts to minimize the number of illegal edges:

1. Randomly assign an initial side to each vertex.
2. For each vertex v with more neighbors on the same side than the opposite side, switch the side of v.
3. For each vertex v with the same number of neighbors on each side, switch the side of v with probability 0.5.
4. Goto step 2 and repeat until the process converges, so that the only changes are being made in step 3.

Once we have created G_C, we can use the previous observations to order the sets $\mathcal{A}$ and $\mathcal{B}$. We order the clones lexicographically in decreasing order using the the following values for each A_i (or, analogously, B_i): the number of $A_j \in \mathcal{A}$ $(i \neq j)$ such that $S(A_j, A_i, C)$, the number of legal neighbors of A_i in G_C, and the number of compatible fragments between A_i and C. We call the first clone in each ordered list a *top-ranked neighbor* of C.

4.3 Generating Strongly-ordered Components

In general, the quality of the neighbor selections from the previous step are very good; most of the errors occur either from false positives in the span relation or in the neighborhoods of "damaged" clone fingerprints. Suppose that each clone C now has two lists of potential neighbors, A_{C1}, A_{C2}, $\ldots$, A_{Ci} and B_{C1}, B_{C2}, $\ldots$, B_{Cj}, sorted as described in the previous subsection. We say that C and D are *nearest neighbors* C is a top-ranked neighbor of D and D is a top-ranked neighbor of C. It is easy to generate lists of clones such that each clone in the list is preceded and followed by its nearest neighbors; the lists terminate at clones which have one or fewer nearest neighbors under the above definition. We call such lists *strongly-ordered components*. Because the conditions for being a nearest neighbor are so conservative, we assume that the strongly-ordered components can be incorporated as-is into the clone ordering; the remaining work then is to figure out the ordering of the components.

4.4 Ordering the Components

Let L_1, L_2, $\ldots$, L_k be the (possibly trivial) strongly-ordered components determined above. Without loss of generality, we refer to each component as having a "left" end and and "right" end. The two leftmost and rightmost clones of some component are referred to as *end* clones, and come in two varieties: *terminal* clones which are the first and last clones in a component, and *near-terminal* clones which are the neighbors of the terminal clones in the component. Note that in the case of a component of size less than 4, a single vertex can be a terminal clone with respect to one end of the component and near-terminal with respect to the other, or one clone can be terminal or near-terminal at both ends.

We now create a new graph, called the component-ordering graph. For each L_i the graph has two vertices, L_i^ℓ and L_i^r, representing the left and right ends respectively. There is an edge between L_i^ℓ and L_i^r of weight 1. We then use the neighbor lists of the end clones of each component to create additional edges between components. The method of determining the edge weights is based on two definitions. Suppose that we are considering adding an edge between a component with end clone C and distinct component with D, an end clone in one of C's neighbor lists. We call C and D a *confirmed* pair if C is also in a neighbor list of D. We call C and D a *bridging* pair if joining the components by having D follow C causes C and/or D to be a bridge in the map, i.e., a clone which is not spanned by its neighbors in the ordering. Given these definitions, we can describe our algorithm for generating edges between components. For every pair of end clones C and D in componentes L_i and L_j respectively, we add an edge between the appropriate vertices of L_i and L_j if either C appears in the neighbor list of D or vice-versa. We then give the edge a heuristic weight based on the following criteria, with lower weights being considered better:

1. Whether or not C and D form a confirmed pair, and the rank of each in the other's neighbor list.
2. Whether or not joining L_i to L_j by making C and D adjacent in the ordering would cause C or D to be a bridging clone.
3. Whether C and D are terminal clones, or only near-terminal clones.

From the above construction, it is obvious that a path through the graph which alternates between the edges between L_i^ℓ and L_i^r and edges between components forms an ordering of the clones contained in all of the components it passes through. Furthermore, the lower the weight of the path, the higher the expected quality of the ordering. Hence, it may seem that we are interested in finding a minimum-weight hamiltonian path for the graph, assuming that the edge between L_i^ℓ and L_i^r is of negligible weight. There are two caveats, however. Recall that the data may contain damaged clones which contain deletions, coligations, fragment multiplicity miscounts, or other problems. We wish our algorithm to omit these from the ordering rather than be forced to include them. There are two ways that this can occur. First, selecting an edge which was created based on near-terminal clones causes the corresponding terminal clones to be omitted when the components are joined in the ordering. Second, we do not generate a hamiltonian path for the graph; instead, we use Held-Karp iteration[5] to generate a spanning tree which has the form of a long path of low weight, together with some short branches. The short branches correspond to components which will be discarded from the ordering. A simple traversal then serves to collect and order the clones along the longest path in the spanning tree.

4.5 An Example

Figure 5 outlines the steps needed to order the set of clones given in part A of the figure. It is assumed that the clones detectably overlap as shown in the figure, and that the relations among the clones are computed accurately except for the errors involving clones F, I, J, and K. The ligated host DNA prevents any clones from spanning F, while a repeated

Relation	False Pos.	False Neg.	Total Errors	Total Tests	Accuracy
Inclusion	427	0	427	31,610	98.6%
Span	5,547	6,003	11,550	617,720	98.1%

Figure 6: Accuracy of Span and Subclone Relations on 17 Simulated Data Sets.

fragment on clone I causes our algorithm to believe that J and K span I.

Figures 5(B) and 5(C) give examples of the bipartite neighbor graphs constructed for each clone. The graph for clone C contains no illegal edges, and allows us to easily identify the neighbors of C. Clone B is one top-ranked neighbor because it has more neighbors in the graph than C, and clone D is the other because both E and F span it with C. The graph for clone I contains an illegal edge, but the other edges allow our algorithm to determine that J lies on the same side of I as K, since placing it on the opposite side would cause more illegal edges.

Figure 5(D) depicts the top-ranked neighbor relations among the clones; for example the arrow from C to B represents the fact that B is a top-ranked neighbor of C, based on C's bipartite neighbor graphs. Double-arrowed edges thus represent nearest neighbor relations, and chains of vertices connected by double-arrowed edges represent the strongly-ordered components in the data. By examining this graph, the ends of the contig A and K can clearly be seen. It also superficially appears that F is a bridging clone between E and G, since they each consider F a neighbor, but F is not spanned by any pair of clones (and hence has no candidate neighbors from its own perspective).

Finally, Figure 5(E) shows the component-ordering graph generated by our heuristics, and the result of applying the incomplete Held-Karp iteration to it. For simplicity, each component is represented in the figure as a single node, although in the actual graph each is represented by two vertices connected with an edge as described previously. The example clearly shows why it is better to allow branching than to force a hamiltonian path; dropping the edge of weight 105 in favor of the edge of weight 260 causes a longer but obviously less desirable path. On the other hand, however, often local rearrangements to form a longer path are desirable when the change in the total weight of the selected edges is smaller. Once the final spanning tree has been generated, we generate a contig by finding the longest path in terms of number of clones in the graph. If there are sufficiently many clones left over, we can mark the visited nodes as deleted and repeat the process to collect additional contigs.

4.6 Experimental Results

We developed and tested our algorithm on MCD data provided by the Genome Center at the University of Washington. The data is part of the Center's large-scale mapping and sequencing effort on Human chromosome 7. However, the results in this paper are based upon simulated data, for which it is easier to gather exact performance measures. The simulated data was created using an accurate model of the data presented by the Genome Center, and the subjective quality of the results appears to be about the same.

To generate our data, we used a Poisson process to place the restriction sites for three different six-cutting enzymes along the target "DNA." Then, some number of clones of

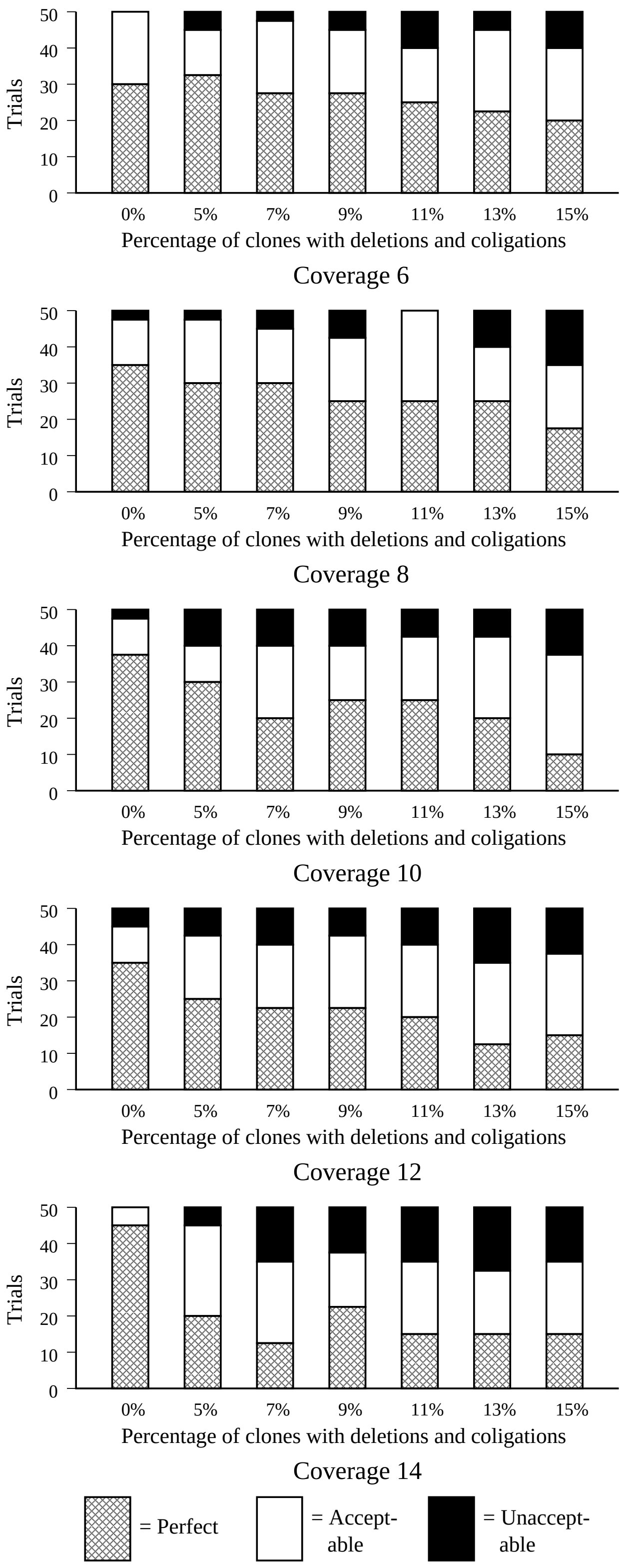


Figure 7: The number of perfect and acceptable solutions for given coverages and rates of error. The diamond-filled bar represents the number of perfect solutions in 50 trials, and the unfilled bar represents the number of acceptable solutions.

Coverage	% of damaged clones						
	0%	5%	7%	9%	11%	13%	15%
6	N/A	84	66	69	74	72	70
8	N/A	72	74	75	76	65	72
10	N/A	78	76	76	76	75	69
12	N/A	85	80	70	77	70	71
14	N/A	71	77	77	72	71	77

Figure 8: The percentage of damaged maximal clones successfully identified and removed by the algorithm.

Coverage	% of damaged clones						
	0%	5%	7%	9%	11%	13%	15%
6	11	12	11	11	12	12	14
8	9	8	9	9	9	9	9
10	9	8	8	9	8	7	8
12	9	9	9	8	9	9	8
14	8	9	9	7	9	9	7

Figure 9: The percentage of undamaged maximal clones inappropriately removed by the algorithm.

length 40,000 bases — approximately the length of the cosmid clones used by the Center — were distributed at random over the region. We then generated the fingerprints for all of the clones, and subjected the measurements in each to a random amount of measurement error, within the bounds typically found in the Center's data. Finally, a certain percentage of the clones were selected to be given data errors of two forms: coligations and deletions. The latter removes some portion of the end of the clone and the restriction sites it contains; the former removes some portion of the clone and replaces it with another random portion of simulated DNA representing contamination by the host organism. We call clones which have had such errors introduced *damaged*.

Figure 6 shows the accuracy of the computed relations on 17 sets of simulated data. For the purposes of these results, the damaged clones were removed, since they no longer constituted intervals in the given area. Each of the 17 data sets consisted of a region of about 180 kilobases to 250 kilobases and 35 to 55 clones, numbers typical of the Genome Center data.

To demonstrate the performance of our system, we then ran our algorithm on a total of 1,750 simulated data sets. Each consisted of a genome of 250,000 bp., with an average coverage of between 6 and 14 times with 40,000 bp. clones. We also varied the percentage of contaminated clones in the set between 0 and 15%. Figure 7 shows the results of these tests. For each coverage value, there is a graph showing the number of correct trials out of 50 for 7 different percentages of damaged clones. We distinguish between two types of correctness: a trial is said to be *perfect* if all of the clones in all of the resulting contigs are in exactly the order that they appear in the correct solution, up to a flip. Now, suppose that the clones are represented by integers, and their correct left-to-right ordering is 1, 2, 3, ... If the sequence of clones produced by our algorithm is perfect, it should contain no local maxima or minima; it should be monotonically increasing or decreasing. We say a solution is *acceptable* if it contains two or fewer local maxima/minima. Having a pair of such extremal points typically corresponds to one region of "reversed" clones. By definition, any perfect solution is also acceptable. Note that neither of these definitions penalizes solutions which contain an incorrect number of contigs, but in practice this is not a serious problem.

Figures 8 and 9 display two other relevant performance statistics. The first is the percentage of the damaged maximal clones which are successfully identified and discarded, and the second is the percentage of valid maximal clones which were mistakenly removed from the solution. As mentioned in the next section, we hope to reduce the latter by attempting to insert some removed clones back into the ordering.

A couple of observations may be made about these results. One would expect the frequency of perfect solutions to increase with the coverage, but surprisingly this is only the case in error-free data. For higher coverages, the frequency of perfect solutions drops more rapidly as the error rate increases than for lower ones. This suggests that the algorithm is sensitive to the density of damaged clones; the greater the number of damaged clones in a region, the worse the result, even though the number of undamaged clones is also proportionally higher. Second, although increasing the coverage generally increases the number of perfect answers in the error-free case, it has little effect on the number of acceptable answers. Thus, increasing the coverage does not significantly improve the quality of the ordering results, although of course it is important to use enough clones to ensure complete, redundant coverage.

5 Future Work

The theoretical work on the abstract interval interleaving problem is largely complete, although we are developing an alternate, equally-efficient approach based on PQ-trees which will describe all interleavings consistent with the given relations rather than simply choosing one.

However, we still have a considerable amount of work to perform on the MCD mapping problem. In the short term, we hope to improve the performance of our current algorithm by reintroducing some discarded clones after generating an initial ordering. This will hopefully reduce the number of unnecessarily discarded clones. We also hope to reduce another common type of error, in which extraneous clones are placed at the end of contigs because the lower degree of coverage towards the ends provides less verification for each placement.

In the longer term, we would like to extend our algorithm to robustly handle the addition of subclones to the interleaving. We also wish to interface our current clone ordering software with our previous work on fragment identification in order to provide a complete MCD mapping system.

6 Conclusion

In this paper, we have presented a new paradigm for solving a common type of problem in physical mapping and sequencing based on the span and inclusion relations, rather than the typical pairwise overlap relation. Just as with the overlap relation, our span and inclusion relations lead to an efficient solution for the abstract interval interleaving problem. They are also similarly easy to approximate from data for many real mapping problems. However, our relations provide more ordering information than simple overlap, thus making the task of detecting and correcting errors in the approximated relations easier.

We have demonstrated the utility of our approach on a problem in multiple complete-digest mapping. Developed and tested with real MCD data from the University of Washington Genome Center, our approach is accurate not only in the presence of measurement error, but also in the face of data contamination such as deletions, coligations, and gel proofreading errors. It is also the first system to automate the process of detecting and eliminating false or misleading data. This step is crucial to the construction of the maps at the UW Genome Center, but is currently performed manually.

References

[1] D. Balding and D. Torney. Statistical analysis of DNA fingerprint data for ordered clone physical mapping of human chromosomes. *Bulletin of Mathematical Biology*, 53(6), 1991, pp. 620–629.

[2] D. Fasulo, T. Jiang, R. Karp, R. Settergren, and E. Thayer. An algorithmic approach to multiple complete digest mapping. *Proceedings of the First Annual International Conference on Computational Molecular Biology* (1997), pp. 118–127.

[3] W. Gillet and J. Daues. DNA mapping algorithms: Synchronized double digest mapping. Technical report, Washington University, 1993.

[4] W. Gillet, L. Hanks, G. Wong, J. Yu, R. Lim, and M. Olson. Assembly of high-resolution restriction maps based on multiple complete digests of a redundant set of overlapping clones. *Genomics*, 33, 389–408.

[5] M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming* 1(1), 1971, pp. 6–25.

[6] T. Jiang and R. Karp. Mapping clones with a given ordering or interleaving. *Algorithmica*, in press.

[7] B. Mumey. *Some Computational Problems in Genomic Mapping.* Ph.D. thesis, University of Washington, 1997.

[8] D. O. Nelson and T. P. Speed. Statistical issues in constructing high resolution physical maps. *Statistical Science*, 9(3), 1994, pp. 334–354.

[9] C. Soderlund, I. Longden, and R. Mott. FPC: A system for building contigs from restriction fingerprinted clones. *CABIOS* (13)5, 1997, pp. 523–535.

[10] J. Sulston, F. Mallet, R. Staden, R. Durbin, T. Horsnell, and A. Coulson. Software for genome mapping by fingerprinting techniques. *Comput. Applic. Biosci.*, 4, 125–132.

[11] G. Wong, J. Yu, E. Thayer, and M. Olson. Multiple-complete-digest Restriction Fragment Mapping: Generating Sequence-ready maps for large-scale DNA sequencing. *Proc. of the National Academy of Sci. USA* (1997), pp. 5225–5230.