Report Number: WUCS-91-34

1991-10-01

DNA Mapping Algorithms: Clone Sequencing

Judith H. Lewis and Will Gillett

The DNA restriction mapping problem can be abstracted to the Shortest Common Matching String problem by viewing the restriction fragment lengths as symbols and the clones as bags. The Shortest Common Matching String problem can be decomposed into the Bag Sequencing problem and the Symbol Sequencing problem. All three of these problems have bene shown to be NP-hard. Rhee has proposed a family of greedy algorithms to compute polynomial time approximations to the Bag Sequencing problem, which produced surprisingly good performance results on abstracted data. In the test data generated by Rhee for pragmatic analysis, the symbols in bags... **Read complete abstract on page 2.**

DNA Mapping Algorithms: Clone Sequencing

Judith H. Lewis and Will Gillett

Complete Abstract:

The DNA restriction mapping problem can be abstracted to the Shortest Common Matching String problem by viewing the restriction fragment lengths as symbols and the clones as bags. The Shortest Common Matching String problem can be decomposed into the Bag Sequencing problem and the Symbol Sequencing problem. All three of these problems have bene shown to be NP-hard. Rhee has proposed a family of greedy algorithms to compute polynomial time approximations to the Bag Sequencing problem, which produced surprisingly good performance results on abstracted data. In the test data generated by Rhee for pragmatic analysis, the symbols in bags represented instances of fragments whose length equivalency had been (magically) determined, a priori; conceptually, fragments of identical length were identified as being equivalent. Also, the distribution of the abstract symbols was assumed to be a uniform distribution, instead of a geometric distribution, as exhibited by fragment length data. This report presents modifications of Rhee's work, adapted to (non-abstracted) fragment length data, and the performance of the modified algorithms when applied to simulated data and laboratory data. The modifications reflect the actual length properties of the restriction fragments, identifying two instance of fragments as being similar if their measured lengths differ by no more than some predefined error measurement bound (often taken to be about 3%). This relation is not an equivalence relation, since it is not transitive. The modified algorithms were applied to fragments length data containing normal random (measurement) error containing the classical geometric distribution. The performance that these modified algorithms achieved when applied against such fragment length data was again surprisingly good.

DNA Mapping Algorithms: Clone Sequencing

Judith H. Lewis and Will Gillett

WUCS-91-34

October 1991

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

This work was supported by the James S. McDonnell Foundation under Grant 87-24 and NIH under Grant 1 R01 HG00180.

ABSTRACT

The DNA restriction mapping problem can be abstracted to the Shortest Common Matching String problem by viewing the restriction fragment lengths as **symbols** and the clones as **bags**. The Shortest Common Mathcing String problem can be decomposed into the Bag Sequencing problem and the Symbol Sequencing problem. All three of these problems have been shown to be NP-hard. Rhee has proposed a family of greedy algorithms to compute polynomial time approximations to the Bag Sequencing problem, which produced surprisingly good performance results on abstracted data. In the test data generated by Rhee for pragmatic analysis, the symbols in bags represented instances of fragments whose length equivalency had been (magically) determined, a priori; conceptually, fragments of identical length were assigned the same symbol. This implies a transitive relationship between *all* instances of fragments identified as being equivalent. Also, the distribution of the abstract symbols was assumed to be a uniform distribution, instead of a geometric distribution, as exhibited by fragment length data.

This report presents modifications of Rhee's work, adapted to (non-abstracted) fragment length data, and the performance of the modified algorithms when applied to simulated data and laboratory data. The modifications reflect the actual length properties of the restriction fragments, identifying two instance of fragments as being *similar* if their measured lengths differ by no more than some predefined error measurement bound (often taken to be about 3%). This relation is not an equivalence relation, since it is not transitive. The modified algorithms were applied to fragments length data containing normal random (measurement) error containing the classical geometric distribution. The performance that these modified algorithms achieved when applied against such fragment length data was again surprisingly good.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. Introduction

1.1. An Overview of DNA Mapping

DNA is the genetic material that supplies the blueprint for an organism's development. A DNA molecule is composed of **nucleotides**, with each nucleotide consisting of a sugar, a phosphate, and a "base". There are four bases: A (Adenine), T (Thymine), C (Cytosine), and G (Guanine). A nucleotide is distinguished by the base it contains. Sugar-phosphate bonds link the nucleotides into strands, and a base on one strand can "pair" with a base on another strand. However, only certain base pairings are allowed: A bonds with T, and C bonds with G. Thus, A and T are known as **complementary bases**, as are C and G. A DNA molecule is made of two complementary DNA nucleotide strands bound together by this base pairing, the base sequence on one strand determining the complementary sequence on the other strand.

DNA restriction mapping deals with determining the *positions* of specific sites of interest along a given DNA strand, or **genome**. The sites of interest are called **restriction sites**, and consist of a specific subsequence of DNA, often six nucleotides long. These restriction sites are recognized by specific enzymes, known as **restriction enzymes**; a restriction enzyme cleaves (or cuts) DNA that it encounters at exactly these restriction sites. Thus, given sufficient time, a restriction enzyme reacting with a strand of DNA will completely digest it, producing fragments of DNA whose lengths are exactly the distance between two successive restriction sites along the original DNA. The process of **electrophoresis** can be used to measure the approximate lengths of these fragments, which are known as **restriction fragments**. If it were possible to (a) identify each restriction fragment present in the genome, (b) determine the length of each restriction fragment, and (c) determine the order of the restriction fragments in the genome, then it would be possible to construct the map of the restriction sites.

The mechanism for obtaining this information is somewhat indirect. Ordering of the restriction fragments is achieved by fracturing multiple copies of the original DNA at random positions to produce randomly overlapping strands of DNA, known as **clones**. Each clone is then completely digested by the restriction enzyme (of interest), and electrophoresis is used to determine the lengths of the restriction fragments within it. This list of restriction fragment lengths is known as the **fingerprint** of the clone. Overlap between the clones is inferred based on the similarity of the fingerprints of the restriction fragment lengths, and the order of the clones is inferred based on multiple clone overlap. As overlap between the clones is inferred due to a significant number of restriction fragments of similar (within measurement error bounds) lengths, the exact order of the restriction fragments within each clone may remain unknown; only the relative (partial) order of large groups of fragments may be inferrable. As more clones are found to overlap a specific region of the original genome, the random positions of the clone ends are used to refine the original partial order (of the restriction fragments) by reducing the size of the groups for which the fragment order is unknown.

This process of DNA restriction mapping is analogous to solving a large jigsaw puzzle. However, the uncertainty of where a clone should be placed can be significant, due to measurement error (produced during electrophoresis), experimental error (produced during cloning or digestion with the restriction enzyme), and certain biological properties of the DNA being mapped (e.g., two fragments of the same length do not necessarily contain the same sequence of nucleotides). When putting together a jigsaw puzzle, the pieces of the puzzle have several cues (shape, color, pattern on the surface) which can be used to guide their ultimate positioning in the final solution. In DNA restriction mapping, the clones have no shape or color, but the fingerprint information can be viewed as a "pattern" to be matched against potentially overlapping clones. The objective is to find a consistent positioning of clones with respect to one another in which fragments in different clones can be identified with one another, while all fragments of each clone remain contiguous (i.e., no "gaps" or unpaired fragments are present internally). There may be multiple "solutions" to this restriction map puzzle, and the one (or ones) which is most compact is preferred.

1.2. Formalization of the DNA Mapping Problem

The DNA mapping problem can be abstracted to the Shortest Common Matching String (SCMS) problem, as defined by Turner. [8] Within this abstraction the clones are abstracted to **bags**, and the lengths of the restriction fragments in the clone are abstracted to **symbols**. A **bag** $b = <a_1, a_2, \cdots, a_h>$ is a multi-set in which a symbol ($a_i \in \Sigma$, where Σ is some finite alphabet) can occur more than once. This mathematical formalism is appropriate because a clone (i.e., a bag) can contain more than one fragment (i.e., a symbol) of the same apparent length.

Let $\#(x,b)$ denote the number of occurrences of the symbol x in b. Then given two bags, b_1 and b_2, over an alphabet Σ:

(a) $b_1 = b_2$ iff $(\forall y \in \Sigma)(\#(y,b_1) = \#(y,b_2))$

(b) $b_1 \subseteq b_2$ (b_1 is a **subbag** of b_2) iff $(\forall y \in \Sigma)(\#(y,b_1) \leq \#(y,b_2))$

(c) $b_1 \subset b_2$ iff $b_1 \subseteq b_2$ and $b_1 \neq b_2$

(d) $b_3 = b_1 \cup b_2$ iff $(\forall y \in \Sigma)(\#(y,b_3) = \#(y,b_1)+\#(y,b_2))$

(e) $b_3 = b_1 \cap b_2$ iff $(\forall y \in \Sigma)(\#(y,b_3) = \min(\#(y,b_1),\#(y,b_2)))$

A collection of bags, B, is said to be **subbag free** if no bag in B is a subbag of another. If $s = a_1 \cdots a_h$ is a string of symbols, then $<s>$ denotes the bag $<a_1, a_2, \cdots, a_h>$, which can be thought of as representing all the strings obtained by permuting the symbols in s. A bag b and a string s are said to **match** if s contains a substring s' such that $<s'> = b$. The SCMS problem can now be defined as follows:

Given a set of bags $B = \{b_1, \cdots, b_n\}$, find a minimum length string that matches every bag in B.

Turner [8] proved that the SCMS problem is NP-hard.

1.3. The Bag Sequencing Problem

In attempting to simplify this problem, Rhee [6] divided it into two subproblems, the **bag sequencing** problem, and the **symbol sequencing** problem. The bag sequencing problem can be stated as follows:

Given a subbag free collection of bags, B, find the sequence (defined by the order in which they match the solution) of bags as they occur in some solution to the SCMS problem given B.

The symbol sequencing problem is unimportant to this work and will not be discussed further. Rhee [6] proved that each of these subproblems is NP-complete.

Since the bag sequencing problem is NP-complete, Rhee proposed a family of greedy algorithms intended heuristically to produce an approximation to the solution of the bag sequencing problem in polynomial time. Rhee applied this family of algorithms to simulated data, based on his abstraction, i.e., bags of symbols. Surprisingly good performance was obtained when this family of algorithms was applied to his abstract simulated data.

In Rhee's abstraction, each instance of a fragments was associated with a distinct symbol. Two instances of fragments associated with the same symbol were assumed to be "identical", this fact having been magically determined, a priori. In fact, in Rhee's work, fragment length data were never produced; only "abstracted" symbolic data were produced assuming a uniform distribution. This implies an equivalence relation between instances of fragments associated with he symbol. Unfortunately, this property is not true of nonabstracted fragment length data. Although Rhee's algorithms had good performance on his abstracted data, it was unclear how they could be extended to accommodate a nontransitive fragment length similarity relation.

The reason for questioning the adaptability of Rhee's algorithms to fragment length data is that the relation which identifies whether or not two fragments are of the same apparent length is not transitive. For example, given three fragments, A of length 1000, B of length 1020, and C of length 1040, A and B are of the same apparent length (since they are within 3% of one another), and B and C are of the same apparent length, but A and C are not of the same apparent length (since they are not within 3% of one another). Since Rhee's abstraction to symbolic data produces an unrealistic transitive relation upon which his algorithms performed well, the question remained about how well fragment length modifications of them would perform in the context of a non-transitive relation.

Other related work has been done by Rhee [5] and Turner. [7]

1.4. Objectives

The objective of using a family of greedy algorithms for clone sequencing is to produce **ordered contigs**, as a preprocessing step to the formal mapping process. In this context, a contig is set of clones which have a high likelihood of coming from a contiguous region of the underlying genome; an ordered contig is one in which the clones have been sequenced with respect to their occurrence along the genome. Much of the formal mapping processing is performed within the confines of a contig, searching for the next clone to be fused into the map. It is beneficial to have relatively small contigs, in order to reduce the search for the next appropriate clone. Having the clones ordered in the sequence in which they appear in the underlying genome supplies a trivial (initial) search mechanism for selecting the next appropriate clone to include in the map.

This work reports on (a) the modification of Rhee's family of greedy algorithms to work with fragment length data instead of abstracted data, and (b) the performance of the modified algorithms when applied to simulated fragment length data and actual laboratory fragment length data.

Section 2 presents an overview of the general concept of greedy algorithms and sets forth a specific paradigm for using greedy algorithms in clone sequencing. Section 3 makes this paradigm more concrete by describing the details of a family of four greedy algorithms for clone sequencing. Section 4 investigates the performance of these algorithms over a number of data sets using a number of different parameter settings.

2. Greedy Algorithms

2.1. The Skeleton of Greedy Algorithms

Horowitz and Sahni [3] describe a greedy algorithm as one which works in stages, considering one input at a time. At each stage, a decision is made regarding whether a particular input can be included in a feasible solution. The inputs are considered in an order that is determined by some selection procedure. If including the next input in the solution at that point will result in an infeasible solution, then the input is not added to the solution. In most greedy applications, the solution is some subset of the inputs which satisfy some specific constraint. Any subset that satisfies the constraint is said to be a **feasible** solution. If the feasible solution maximizes or minimizes some objective function, then the solution is said to be **optimal**, with respect to that objective function.

Figure 1 is the Horowitz and Sahni abstraction of a greedy algorithm. **SELECT** selects an input from A, removing it from A. **FEASIBLE** determines if the input selected can be included in a feasible solution. It returns a Boolean value through the function name which represents this feasibility determination. **UNION** combines the input with the solution. [3]

2.2. Formalisms

In order to formalize the extensions of Rhee's greedy algorithms to fragment length data, it is useful to introduce a new mathematical construct similar to, but slightly different than, a bag. This will be referred to as a **fuzzybag**, because the objects contained within can be identified with each other in a "fuzzy" manner. Informally, two objects (numbers representing the lengths of corresponding fragments) in fuzzybags should be allowed to be identified with one another if they are "similar" to one another. Usually two fragments are identified as **similar** if their measured lengths are within 3% of one another. However, this concept will be abstracted to a Boolean function similar($frag_1, frag_2$).

2.2.1. The Fuzzybag

A **fuzzybag** is a multiset of positive real numbers. The operations that can be performed on fuzzybags are the same as those that can be performed on bags, with a few modifications to reflect the fuzzy

```
procedure GREEDY(A,n)
    { A(1:n) contains the n inputs }
    solution = empty_set
    for i = 1 to n do
        x = SELECT(A)
        if FEASIBLE(solution,x) then
            solution = UNION(solution,x)
        endif
    endfor
    return(solution)
end GREEDY
```

Figure 1: General Greedy Algorithm

nature of how objects are identified with one another. Consider two fuzzybags, fb_1 and fb_2, where

$$fb_1 = <a_1, a_2, \cdots, a_n>$$

and

$$fb_2 = <b_1, b_2, \cdots, b_m>.$$

The cardinality of a bag is the number of occurrences of instances of objects in the bag, i.e., multiple occurrences of an object are counted multiple times. Here, the cardinality of fb_1 is n (i.e., $|fb_1| = n$) and the cardinality of fb_2 is m. The union of two fuzzybags is easily defined as:

$$fb_1 \bigcup fb_2 = <a_1, a_2, \cdots a_n, b_1, b_2, \cdots b_m>$$

This implies that $|fb_1 \bigcup fb_2| = |fb_1| + |fb_2|$. However, in order to define intersection, some mathematical scaffolding will have to be put in place.

Let a **similarity function** between two fuzzybags be a function which maps objects in one fuzzybag only to similar objects in the other fuzzybag. In other words, if sf is a similarity function, then $(frag_1, frag_2) \in sf$ implies similar$(frag_1, frag_2)$. Consider all one-to-one partial similarity functions from fb_1 to fb_2, and denote this set of functions by APSF(fb_1, fb_2). In other words, if $f_{part} \in$ APSF(fb_1, fb_2), then (a) f_{part} is a partial function from fb_1 to fb_2 (i.e., all objects in fb_1 may not be mapped into objects in fb_2), (b) f_{part} is one-to-one (i.e., no two objects in fb_1 map onto the same object in fb_2), and (c) if $(frag_1, frag_2) \in f_{part}$, then similar$(frag_1, frag_2)$ is TRUE.

Let maxfuncs be a function whose single argument is a set of finite functions. Let maxfuncs select from its argument the subset of all functions which have maximum cardinality. In other words,

$$\text{maxfuncs}(A) = \{f_{part} \in A \mid (\forall g \in A)(|f_{part}| \geq |g|)\}$$

Let ndpick be a function whose single argument is a set of objects. Let ndpick nondeterministically pick a specific member from its argument to be returned as its value. In other words, ndpick$(\{o_1, o_2, \cdots, o_k\}) = o_i$, for some $1 \leq i \leq k$, the specific i being nondeterministically chosen.

Let ndselect be a function whose single argument is a pair of objects. Let ndselect nondeterministically select either the first component or the second component from the pair to be returned as its value. In other words, ndselect$((o_1, o_2)) = o_i$, for some $1 \leq i \leq 2$, the specific i being nondeterministically chosen.

Let ndproject be a function whose single argument is a set of pairs of objects (representing a function). Let ndproject(set_of_pairs) be the fuzzybag produced by nondeterministically selecting either the first or second component from each of the pairs in the set, the resulting fuzzybag being the aggregate of all such components selected. This is more formally defined by the two mutually recursive functions specified below.

$$\text{ndproject}(func) \equiv \begin{cases} <> & \text{if } |func| = 0 \\ \text{glue}(func, \text{ndpick}(func)) & \text{otherwise} \end{cases}$$

$$\text{glue}(func, pair) \equiv \text{ndproject}(func - \{pair\}) \cup <\text{ndselect}(pair)>$$

The intersection operation now can be defined as:

$$fb_1 \cap fb_2 = \text{ndproject}(\text{ndpick}(\text{maxfuncs}(APSF(fb_1, fb_2))))$$

This definition of intersection is intended to supply a nondeterministic abstraction of selecting a set containing the maximum number of fragments in the two fuzzybags which could be identified with one another given the similarity criteria embodied in similar. Note that although this definition of intersection is nondeterministic, the cardinality of the fuzzybag chosen is constant over all nondeterministic choices.

Three other important definitions are given below.

$$fb_1 = fb_2 \text{ iff } |fb_1 \cap fb_2| = |fb_1| = |fb_2|$$

$$fb_1 \subseteq fb_2 \text{ iff } |fb_1 \cap fb_2| = |fb_1|$$

$$fb_1 \subset fb_2 \text{ iff } |fb_1 \cap fb_2| = |fb_1| \text{ and } fb_1 \neq fb_2$$

Note that these definitions rely directly on the concept of the intersection of two fuzzybags. Also note that none of =, ⊆, and ⊂ are transitive.

2.2.2. Adjacency in Nondirectional Sequences

In this work the term **relationship** refers to one element of the set of 2-tuples which comprise a relation. The term **relation** refers to the set of relationships of which it is composed. For instance, given the relation R = {(a,b), (b,c), (c,a)}, there are three component relationships, (a,b), (b,c), and (c,a).

The relation of interest here is that of two clones being adjacent to one another when considered in the *nondirectional unique sequence paradigm.* The nondirectional unique sequence paradigm requires that no object occurs in a sequence more than once and identifies two sequences, s_1 and s_2, as being identical iff (a) $s_1 = s_2$, or (b) $s_1 = s_2{}^r$ (where s^r s the reverse of s). For instance, the two sequences ADBFE and EFBDA are identified as equivalent in this paradigm.

Given a nondirectional sequence S of length n, S can be completely characterized by specifying n-1 adjacency relationships between the objects in S. For example, the nondirectional sequence ADBFE is characterized by the four adjacency relationships A **is_adjacent_to** D, D **is_adjacent_to** B, B **is_adjacent_to** F, and F **is_adjacent_to** E. Note that in this paradigm, the **is_adjacent_to** relation is symmetric.

In general, given n distinct objects, there are exactly $\frac{n(n-1)}{2}$ possible adjacency relationships between them. Given that a specific nondirectional sequence of length n is to be constructed, the specification of S can be considered equivalent to the selection of n-1 adjacency relationships which characterize the sequence being constructed. Note that not all subsets of size n-1 (selected from the set of $\frac{n(n-1)}{2}$ possible adjacency relationships) correspond to a nondirectional sequence, since in a sequence any specific object can be adjacent to no more than two other objects.

2.3. A Paradigm for Clone Sequencing

In the case of clone sequencing (i.e., fuzzybag sequencing), the objective is to find the sequence of clones as they appear in some solution to the SCMS problem. Since clone sequencing can be characterized by selecting an appropriate set of adjacency relationships, a greedy algorithm will be used to interrogate all the possible adjacency relationships between the clones to determine a clone sequence having a high likelihood of being "close to" a solution to the abstract SCMS problem. In other words, given a set of n clones, there are $\frac{n(n-1)}{2}$ possible adjacency relationships; each must be checked, in turn, to determine if it could be part of an already partially completed feasible clone sequence.

In moving from the abstract SCMS problem to the pragmatic problem of actually doing clone sequencing, certain observations about the underlying biology and actual probability distribution need to be recognized. For instance, if two clones have *some* apparent overlap, but not *significant* overlap (say two or less fragments), the likelihood that they actually overlap may not be high at all. To include a clone in a clone sequence on the basis of such unconvincing data tends to produce questionable results. Thus, there is a minimum required apparent overlap criteria applied before a clone can be placed in a clone sequence. This often is taken to be the requirement that three or more fragments must appear to overlap before insertion is allowed.

This requirement may cause "gaps" to occur in the middle of a clone sequence producing **islands** of clones which are unconnected to other islands. In fact, an island may consist of a single clone if it has no significant overlap with any other clones. Within the greedy algorithm paradigm, this means that given n clones to be placed in sequence, fewer than n-1 adjacency relationships may actually be chosen. In fact, one adjacency relationship will be absent for each unattached island present.

Figure 2 is a pseudocode abstraction of the C function **greedy_loop**, which represents the fragment length modification of Rhee's instantiation of the greedy algorithm found in Figure 1. All of the greedy algorithms described in this work are accomplished by invoking **greedy_loop** with the appropriate arguments.

There are several input parameters to **greedy_loop**. `Select`, `Feasible`, and `Union` are C functions which correspond to the abstract functions **SELECT**, **FEASIBLE**, and **UNION** of Figure 1. `specified_overlap` specifies the minimum number of fragments which must appear to overlap between two clones before the two clones should be considered as candidates in the resulting feasible adjacency relation. In order to understand the remaining two input parameters to **greedy_loop**, `S` and `L`, certain aspects of how the algorithm works must be explained.

As a preprocessing step, all possible adjacency relationships between clones are examined. If there are n clones, then there are $\frac{n(n-1)}{2}$ such possible adjacency relationships. For each individual adjacency relationship, the apparent overlap between the clones (i.e., the number of fragments which appear to be

```
greedy_loop(S,L,Select,Feasible,Union,specified_overlap)
       LIST           S;
       LIST           L;
       FUNCTION       Select;
       FUNCTION       Feasible;
       FUNCTION       Union;
       int            specified_overlap;
{
       .
       declarations
       .

    count = 0;
    triplet_count = cardinality(L);

    while (count <= triplet_count) {

        triplet = Select(L);
        (fbag1, fbag2, overlap) = unpack(triplet);

        if (overlap >= specified_overlap) {
            if (Feasible(S,fbag1,fbag2)) Union(S,fbag1,fbag2);
        };

        count = count + 1;
    };
}
```

Figure 2: The **greedy_loop** routine

identical, within the measurement error window) is computed. A *conceptual* triplet is formed, composed of the two fuzzybags (i.e., clones) and the apparent overlap. (In fact, a fourth component is included to support weighted variations of the algorithms.) These triplets are sorted in decreasing likelihood of overlap; the likelihood of overlap is assumed to be a good approximation to the likelihood that they are adjacent in the clone sequence. The input parameter `L` is the sorted list of these triplets.

The parameter `S` is a list used as both an input parameter and an output parameter. Upon input, each element of the list is a singleton list (actually a reversible list, known as an **rlist**) containing the fuzzybag of fragment lengths associated with a specific clone. At the beginning of the algorithm, as two fuzzybags are declared (by the algorithm) to be adjacent, the corresponding elements are extracted from `S`, and a list containing the two fuzzybags (just declared to be adjacent) is inserted into `S`. Each list in `S` is referred to as an **island**; a list containing only one fuzzybag is a **trivial island**. As the algorithm progresses, islands are **fused** to create larger islands. This may occur in several different ways: (a) a fuzzybag may be added to the end of already existing non-trivial island, (b) a new non-trivial island may be created, or (c) an adjacency relationship may cause two already existing non-trivial islands to be fused into one. The fusing of islands can occur only if the adjacency relationship being considered deals with clones which are at the *ends* of the islands being fused. Any adjacency relationship dealing with clones occurring in the internal region of an island will be declared infeasible. Upon completion of the algorithm, whatever set of islands remaining in `S` represents the resulting output clone sequence. Note that this may represent a number of disjoint clones sequences, not just one clone sequence.

In order to achieve an efficient underlying implementation for manipulating clone sequences, it was necessary to implement a new list Abstract Data Type (ADT) [2] for which concatenation and reversal could be performed in constant time. A new ADT, **rlist** was implemented to achieve this execution performance.

Returning to the pseudocode shown in Figure 2, the first two statements in the `while` loop extracts a triplet from the list `L` and "unpacks" the data present. These data represent the two clones that might be adjacent to one another and their apparent overlap. If the overlap between the fuzzybags is significant, the feasibility of their adjacency is assessed, given the current state of the partially constructed islands. If this adjacency relationship is feasible (determined by `Feasible`), given the current set of islands present in `S`, then the corresponding islands are fused (by `Union`).

3. A Family of Fuzzybag Sequencing Algorithms

The important aspects of four of the basic bag sequencing algorithms developed by Rhee [6] (and modified for fragment length data) are presented in this section. In conformance with his dissertation, they are referred to here as **MGREEDY_B**, **MGREEDY_BS**, **MGREEDY_BW**, and **MGREEDY_BSW**. Prior to executing any of these algorithms, a preprocessing step is applied which extracts subbags from the list of fuzzybags (i.e., clones) input. The computation is then done on the subbag free collection thus extracted, by invoking **greedy_loop** with the appropriate parameters.

Referring to Figure 2, the `Select` function is identical for all of the algorithms; it simply extracts (i.e., retrieves and removes) the first element of the list `L`. The `Feasible` and `Union` functions will be slightly different for each of the four algorithms.

3.1. MGREEDY_B

The **MGREEDY_B** algorithm declares the adjacency relationship between two fuzzybags to be feasible if the following three constraints are satisfied:

(a) the fuzzybags have at least the minimum required overlap,

(b) the fuzzybags are not in the same fuzzybag sequence (i.e., island), and

(c) the fuzzybags are at the ends of their respective fuzzybag sequences.

Thus, the `Feasible` function for this algorithm checks the three conditions given above; it returns TRUE if all the conditions are satisfied and FALSE otherwise. The `Union` function manipulates the elements of `S` to extract the two islands in which the two fuzzybags currently reside, fuse the two islands, and insert the newly created island into `S`.

3.2. MGREEDY_BS

In the **MGREEDY_BS** algorithm, a new constraint, the **subset-consecutiveness constraint**, is added to those of the **MGREEDY_B** constraints. This constraint requires that for any three consecutive fuzzybags in a fuzzybag sequence, in addition to the facts that the first must have significant overlap with the second and the second must have significant overlap with the third, there can be no apparent "gap" between the first and the third. This constraint constitutes an interlocking requirement between the fuzzybags, i.e., a fuzzybag must have significant overlap with it neighbor *and* "touch" its neighbor's neighbor (either by overlapping with it or at minimum abutting with it).

This algorithm also allows for the possibility of swapping the order of the two fuzzybags at the appropriate end of either fuzzybag sequence. This allows for a small amount of "fine tuning" at the ends of the fuzzybag sequences, allowing for "local" errors at the extremes of the sequence.

The **MGREEDY_BS** algorithm declares the adjacency relationship between two bags to be feasible if the following constraints are satisfied:

(a) the fuzzybags have at least the minimum required overlap,

(b) the fuzzybags are not in the same fuzzybag sequence (i.e., island),

(c) the fuzzybags are at the ends of their respective bag sequences,

(d) for the new fused fuzzybag sequence that would be created, $[fb_{i_1}, fb_{i_2}, \cdots, fb_{i_k}]$, the **subset-consecutiveness constraint** must hold, i.e., $(\forall 2 \le j \le k-1)(fb_{i_j} \subseteq fb_{i_{j-1}} \cup fb_{i_{j+1}})$, and

(e) for the new fused fuzzybag sequence that would be created, $[fb_{i_1}, fb_{i_2}, \cdots, fb_{i_k}]$, the **minimum overlap constraint** must hold, i.e., $(\forall 1 \le j \le k-1)(\omega(fb_{i_j}, fb_{i_{j+1}}) \ge minimum\_required\_overlap)$, where $\omega(fb_n, fb_m) = |fb_n \cap fb_m|$.

The first three constraints above are exactly those of the **MGREEDY_B** algorithm. The fourth formalizes the subset-consecutiveness constraint, and the fifth is a restatement of the minimum overlap constraint (which, on the surface, may seem to be vacuous). These last two constraints are expressed as constraints on the entirety of the newly created fuzzybag sequence because the ability to swap the last two fuzzybags at the end of constituent fuzzybag sequences may destroy any constraints which might have been true prior to the fuzzybag swapping.

In order to help the reader understand the complexity of this algorithm, two detailed examples will be presented. In these examples, the minimum required overlap between fuzzybags is assumed to be 3.

The first example is illustrated in Figure 3. In this example, the adjacency relationship between fbag0 and fbag1 is being tested for feasibility. Fbag0 = <375, 400, 470, 580, 650>, and it is in an island containing only itself. Fbag1 = <400, 470, 580, 650>, and it is in an island containing one other fuzzybag, fbag2 = <400, 580, 650, 800>. This configuration is shown in Figure 3(a). $\omega(fbag\,0, fbag\,1) = 4 \ge 3$;

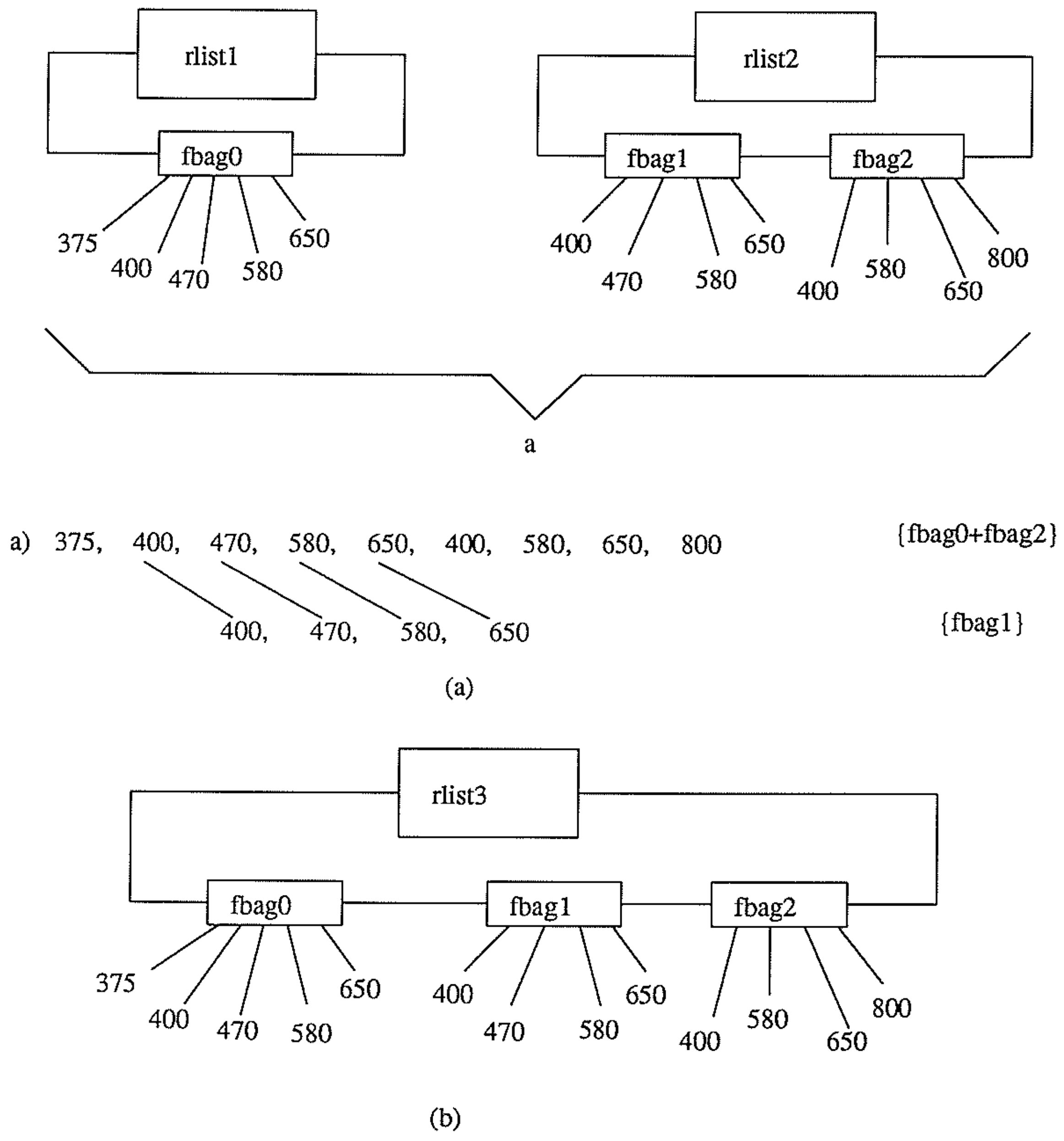

Figure 3: Subset-consecutiveness constraint applied with singleton fuzzybag

therefore, constraint (a) is satisfied. Fbag0 and fbag1 are not in the same fuzzybag sequence; therefore, constraint (b) is satisfied. Both fuzzybags are at the ends of their respective fuzzybag sequence; therefore, constraint (c) is satisfied. In order to check constraint (d), it must be determined whether or not fbag1 is a subbag of fbag0 unioned with fbag2. This analysis is shown in Figure 3(a) and verifies that constraint (d) is satisfied. (The labeled bracket under lists indicates the fuzzybags being considered, and the analysis being performed. See Figure 7 for multiple analyses.) Constraint (e) is satisfied because constraint (a) is satisfied and no swapping at the ends of either fuzzybag sequence was performed. The resulting fused fuzzybag sequence is shown in Figure 3(b).

As a second more complex example, consider how the algorithm would address the fbag3 **is_adjacent_to** fbag4 relationship, given that S contains the two islands shown in Figure 4. Conditions (a), (b), and (c) are clearly satisfied for this configuration. Conditions (d) and (e) must be checked, given the possibility of swapping the fuzzybags at the appropriate end of the two islands. The first check of condition (d) would be performed, given that no fuzzybag swap has occurred. In this case, the algorithm would check two subconditions, both of which must be true: fbag3 $\subseteq$ fbag2 $\cup$ fbag4, and fbag4 $\subseteq$ fbag3 $\cup$ fbag5. The first of these two subconditions is shown in Figure 5 to be FALSE.

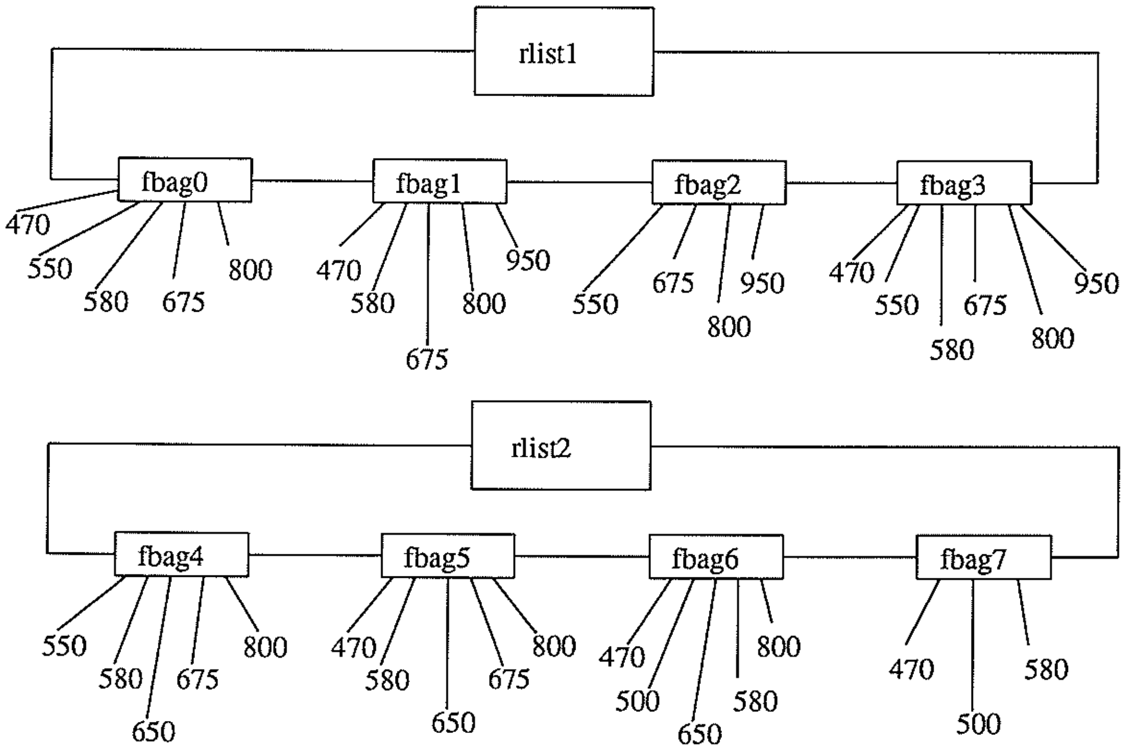

Figure 4: Two initial fuzzybag sequences

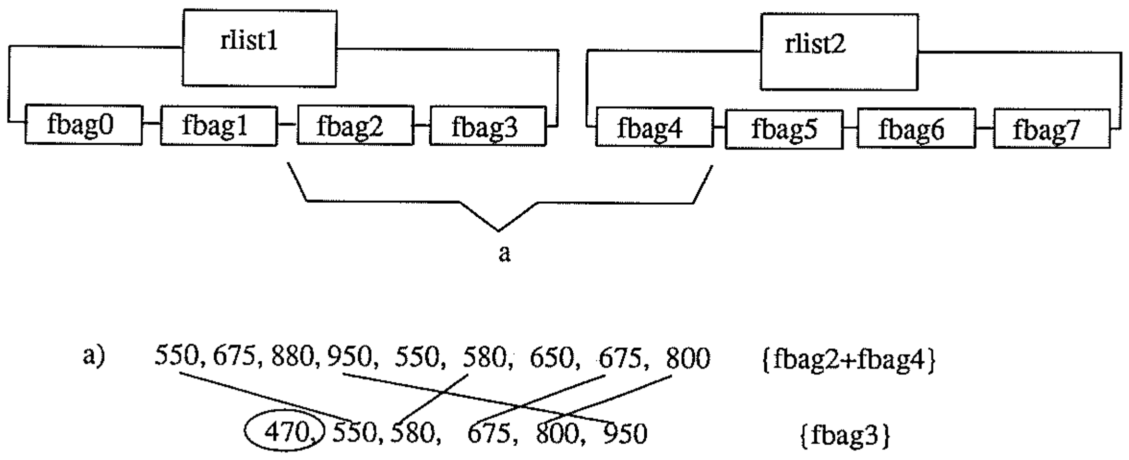

Figure 5: Subset-consecutiveness constraint applied with no swap

As a second attempt at confirming the subset-consecutiveness condition, the algorithm would attempt to swap the two fuzzybags at the end of one of the islands. Figure 6 shows this configuration, in which fbag4 and fbag5 have been swapped in the second island. Given this modification, there are four subconditions which must checked. The first three are shown in Figure 6, and the third is shown to be FALSE.

The next attempt would be to leave the second island alone and swap the fuzzybags on the end of the first island. This is shown in Figure 7, where fbag2 and fbag3 have been swapped. Again, there are four subconditions to be checked here. As shown in Figure 7, all of these subconditions are TRUE. Thus, condition (d) succeeds for this configuration. However, had condition (d) failed for this configuration, one more configuration would have been tried, that in which the bags at the ends of both islands were swapped.

A quick check confirms that condition (e) is satisfied for the configuration shown in Figure 7. Since all the conditions have been met, the two islands are fused; the result is shown in Figure 8.

3.3. MGREEDY_BW

The **MGREEDY_BW** algorithm declares the adjacency relationship between two fuzzybags to be feasible if and only if it meets the three conditions of the **MGREEDY_B**. In this algorithm, however, the order in which the adjacency relationships are selected is based on a nonincreasing *weighted sum of*

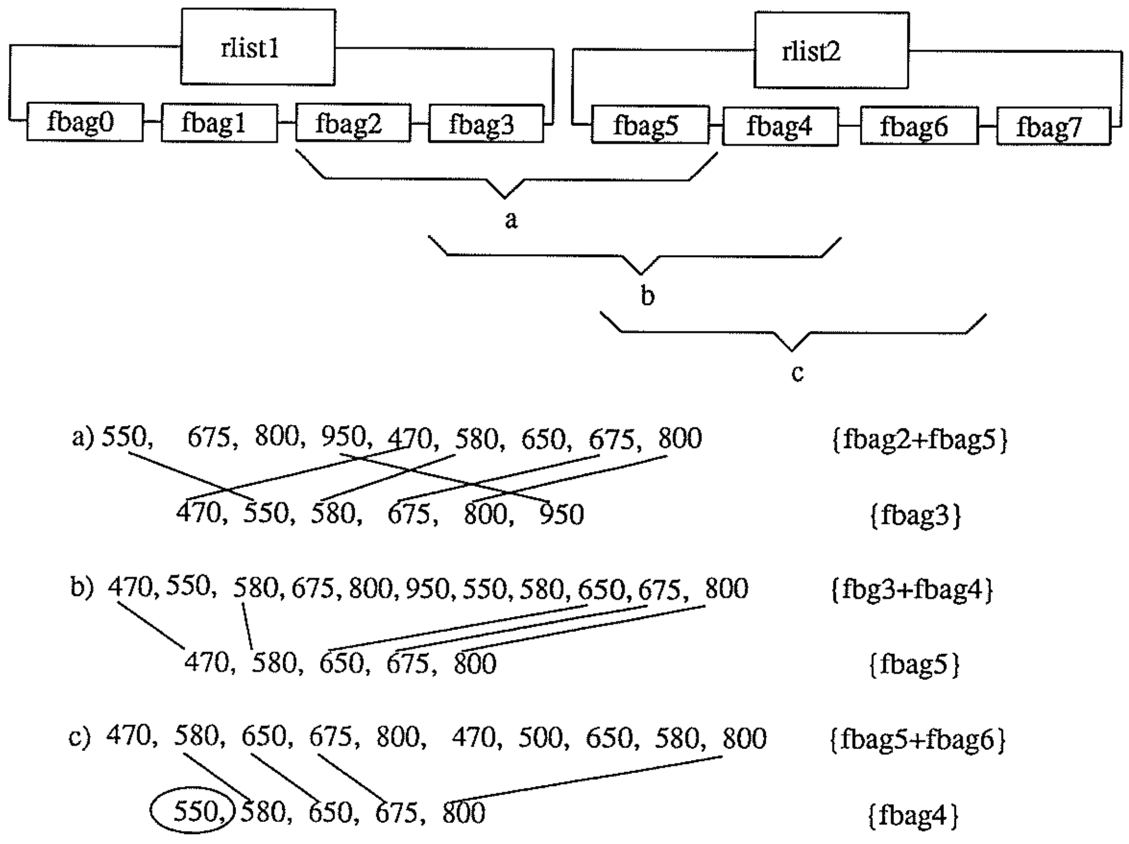

Figure 6: Subset-consecutiveness constraint applied with one swap

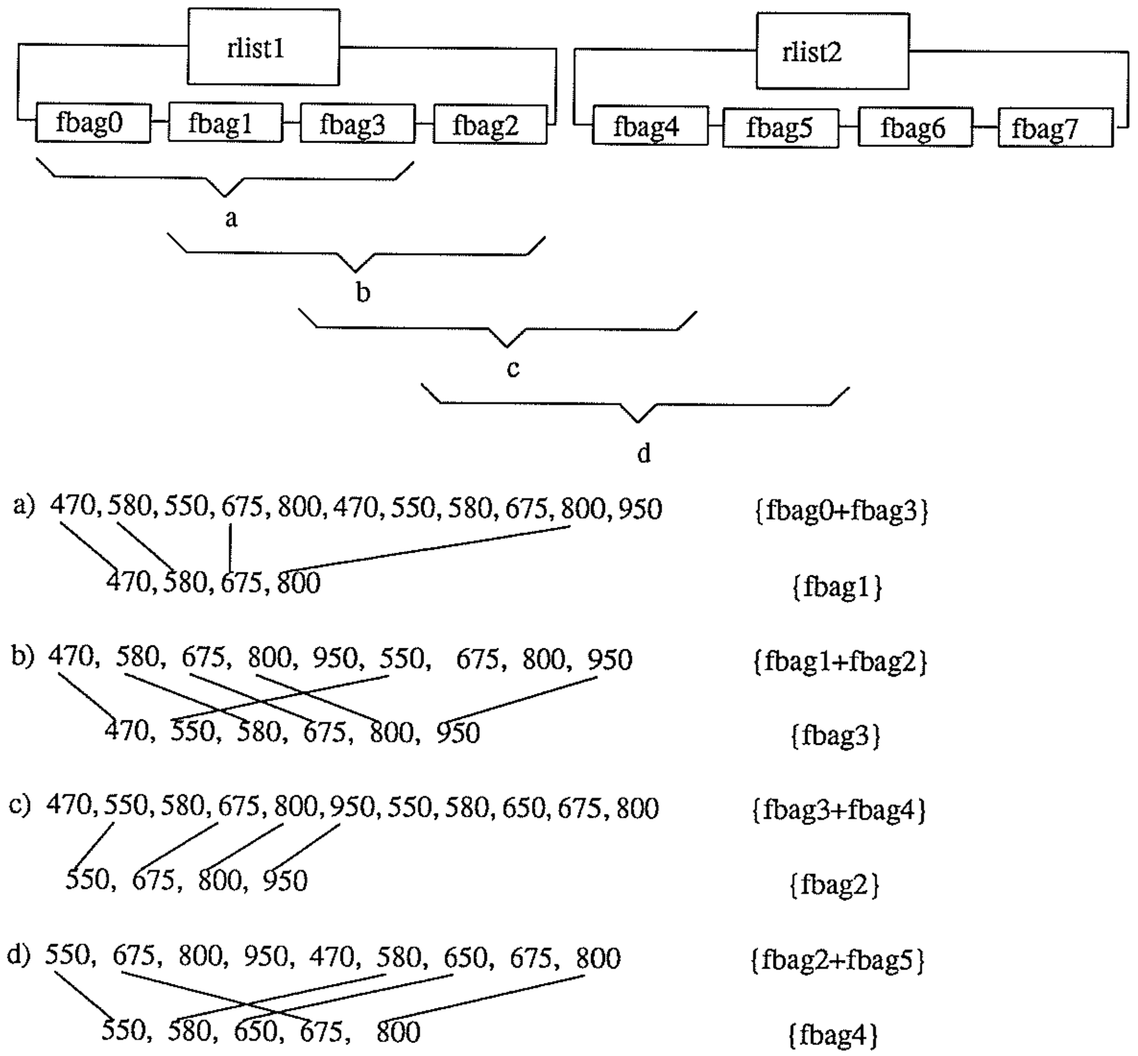

Figure 7: Subset-consecutiveness constraint applied with another swap

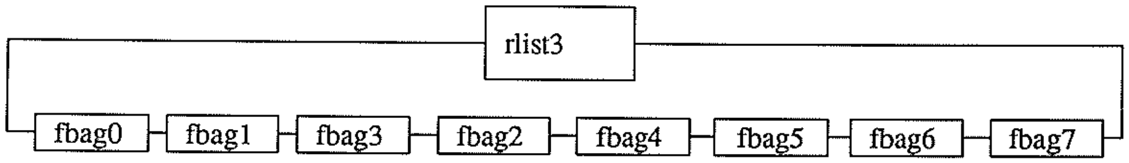

Figure 8: Resulting fused fuzzybag sequence

fuzzybag sizes value. This preprocessing step is the only difference between **MGREEDY_BW** and **MGREEDY_B**. Let *ws* represent this *weighted sum*. Then, the formula for *ws* is

$$ws(b_i, b_j) = \frac{\omega(b_i, b_j)}{w_1 * |b_i| + w_2 * |b_j|}$$

where w_1 and w_2 are constant weights, and $|b_k|$ is some metric concerning the size of the bag b_k.

Experimentally, it has been determined that the choice of $w_1 = w_2 = 1$ is no worse than any other choices for these constants. [6]

In Rhee's work, the symbols (corresponding to restriction fragments) have no length. Thus, there is only one applicable metric for the size of a bag, i.e., $|b_k|$ was taken to be the number of symbols in b_k. However, when applied to fragment length data, a second type of metric becomes applicable, i.e., one involving the lengths of fragments. Thus, the formula for calculating the weighted sum can be based on *cardinality* or *length*. In order to formalize these ideas, let:

- $\omega_c(b_i, b_j) = |b_i \cap b_j|$, i.e., the number fragments in b_i and b_j with the same apparent length,
- $\omega_l(b_i, b_j) = \sum_{f \in b_i \cap b_j} length(f)$, i.e., the sum of the lengths of the fragments in b_i and b_j with the same apparent length,
- $S_c(b_i) = |b_i|$, i.e., the cardinality of the fragments in b_i, and
- $S_l(b_i) = \sum_{f \in b_i} length(f)$, i.e., the sum of the lengths of the fragments in b_i.

Then the weighted sum based on the length is

$$ws_l(b_i, b_j) = \frac{\omega_l(b_i, b_j)}{w_1 * S_l(b_i) + w_2 * S_l(b_j)}$$

and the weighted sum based on the cardinality is

$$ws_c(b_i, b_j) = \frac{\omega_c(b_i, b_j)}{w_1 * S_c(b_i) + w_2 * S_c(b_j)}$$

The method of calculating the weighted sum is indicated in subsequent tables by the use of the term *length* (to reference ws_l) and *cardinality* (to reference ws_c).

3.4. MGREEDY_BSW

The **MGREEDY_BSW** algorithm is identical to **MGREEDY_BS**, except that the inputs are selected in order of nonincreasing weighted sums, as described for the **MGREEDY_BW** algorithm above.

4. Performance

4.1. Performance Measurements

Rhee's work [6] defined a number of performance measures against which the quality of the result obtained by his algorithms could be tested. These took the form of decision ratios.

- R_{wd} represents the ratio of wrong decisions to total decisions:

$$R_{wd} = \frac{N_{wd}}{N_{wd} + N_{rd} + N_{ud}}$$

- R_{ud} represents the ratio of unmade decision to total decision:

$$R_{ud} = \frac{N_{wd}}{N_{wd} + N_{rd} + N_{ud}}$$

Here, N_{wd} = number of wrong decisions, N_{rd} = number of right decisions, and N_{ud} = number of unmade decisions.

As an example of how to compute these ratios, consider the following example from Rhee's dissertation. [6] Assume that there are ten clones labeled 0 through 9, where the actual clone sequence is 0123456789. Since there are ten clones, there are nine decisions to be make, i.e., $N_{wd} + N_{rd} + N_{ud} = 9$. Assume that the algorithm in question produces as output a set of two islands, {123540789, 6}. Since there are two islands, one of the nine adjacency decisions was not made; thus $N_{ud} = 1$. Eight adjacency decisions were made, three of which are wrong and five of which are right. The three wrong decisions are: 3 **is_adjacent_to** 5, 4 **is_adjacent_to** 0, and 0 **is_adjacent_to** 7. Thus, $N_{wd} = 3$ and $N_{rd} = 5$. Hence, for this example $R_{wd} = 3/9 \approx 0.333$ and $R_{ud} = 1/9 \approx 0.111$.

Note that the ability to compute these ratios is based on the absolute knowledge of the actual underlying clone sequence. Such knowledge can be extracted from fragment length data produced by simulation, because the underlying reality can be determined. This is not true of data obtained from a laboratory; any mapping which has been done, from which a clone sequence can be extracted, represents only an estimate of the underlying reality.

4.2. Performance Results

4.2.1. The DNA Simulator

In order to determine the performance of the greedy algorithms as modified to address fragment length data, the four algorithms were applied to fragment length data produced by a simulator constructed within the DNA Mapping group. [1] Although the exact details of the simulator are not extremely important here, a few details may be helpful to the reader. The simulator can produce genomes of any desired size; certain statistical properties of the genome produced can be controlled by the use of input parameters. The simulator generates clones from a genome in an manner analogous to the biological production of clones. For each clone, digestion with any specific restriction enzyme (or set of restriction enzymes) can be performed, and the fragment length data can be obtained. Many postprocessing steps can be applied to these fragment length data prior to mapping processing, the most important of which is the introduction of random error (normally distributed). In the specific experiments simulated in these data, the clones were digested three different times: (a) the first time with restriction enzyme *Eco*RI, a 6-cutter with recognition site GAATTC, (b) the second time with restriction enzyme *Hin*dIII, a 6-cutter with recognition site AAGCTT, and (c) the third time with RH, which is a combination of the two (i.e., when digesting the clones, both enzymes are present so that the DNA is cleaved at both *Eco*RI sites and *Hin*dIII sites).

The performance of these algorithms over a range of conditions is of interest, including genome size, minimum overlap required, average fragment size, and introduction of random error. Most of the following tables deal with four basic data sets whose genome sizes range from 100 kilobasepairs (kb) to 1600kb. The designation of *length* or *cardinality* underneath the name of a weighted algorithm refers to the formula used for the weighted sum of the bag sizes, as discussed in Section 3.3.

4.2.2. Comparison with Performance on Abstract Data

Table 1 presents the performance of these four algorithms over four different genomes with no random error introduced and a minimum required overlap of 3 fragments. The use of data with no random error introduced is as close to Rhee's abstracted (symbolic) data as can be approximated with fragment length data. In the execution of the algorithms in these runs, the error window was reduced to 0%, so that only fragments of *identical* length were identified with one another. This reduces the probability of confusing two different fragments to the probability that two different fragments have identical lengths. Since the probability of two different fragments having exactly the same length is relatively small, this significantly reduces the chances of fragment confusion and roughly approximates abstracted symbolic data in which the symbols are the discrete lengths of the fragments themselves.

The four different genomes are indicated in the first column. Their sizes are 100kb, 400kb, 800kb, and 1600kb; the basic input parameters which produced them were identical. The second column indicates the enzyme(s) used during the digestion. The third column indicates the number of clones produced from the genome; this includes both **subclones** and **maximal clones**. Subclones correspond to subbags, and maximal clones correspond to fuzzybags for which no subbag is present. The number of apparent maximal clones for each kind of digestion is shown in the fourth column.

The last four columns present the performance of the four greedy algorithms as applied to the specific data set. Each entry contains a 2-tuple. The first component is the ratio of wrong decisions; the

Table 1
Performance Data for NO Random Error, Minimum required overlap = 3

Performance Results of Greedy Algorithms Applied to Simulated Fragment Length Data
NO Random Error, Minimum required overlap = 3

				Performance Calculations (R_{wd}, R_{ud})			
Genome Size	Enzyme	Total Clones	Maximal Clones	MGREEDY_B	MGREEDY_BS	MGREEDY_BW (length)	MGREEDY_BSW (length)
100kb	*Eco*RI	35	6	(0.000,0.400)	(0.000,0.600)	(0.000,0.400)	(0.000,0.600)
	*Hind*III		7	(0.000,0.500)	(0.000,0.500)	(0.000,0.500)	(0.000,0.500)
	RH		9	(0.000,0.125)	(0.000,0.125)	(0.000,0.125)	(0.000,0.125)
400kb	*Eco*RI	192	40	(0.000,0.436)	(0.000,0.436)	(0.000,0.436)	(0.000,0.436)
	*Hind*III		45	(0.000,0.273)	(0.000,0.273)	(0.000,0.273)	(0.000,0.273)
	RH		61	(0.033,0.083)	(0.033,0.100)	(0.033,0.083)	(0.033,0.100)
800kb	*Eco*RI	342	74	(0.000,0.438)	(0.000,0.438)	(0.000,0.438)	(0.000,0.438)
	*Hind*III		77	(0.000,0.447)	(0.000,0.513)	(0.000,0.447)	(0.000,0.513)
	RH		110	(0.000,0.046)	(0.000,0.082)	(0.000,0.046)	(0.000,0.073)
1600kb	*Eco*RI	606	130	(0.000,0.380)	(0.000,0.419)	(0.000,0.380)	(0.000,0.419)
	*Hind*III		133	(0.000,0.364)	(0.000,0.386)	(0.000,0.364)	(0.000,0.386)
	RH		197	(0.000,0.107)	(0.000,0.138)	(0.000,0.107)	(0.000,0.138)

second component is the ratio of unmade decisions. There are five important observations to note from this table. (1) In these "perfect" data, there are virtually no wrong decisions made. (The 0.033 entry for the RH digestion of the 400kb genome represent two erroneous decisions out of the 60 decisions that were to be made.) (2) The number of unmade decisions is very high, especially for *Eco*RI and *Hin*dIII. This ratio may be artificially high, do to the natural occurrence of islands in the simulated input data. (3) The **MGREEDY_BS** algorithm causes more unmade decisions to occur than does the **MGREEDY_B** algorithm. This is to be expected because the subset-consecutiveness constraint constitutes an extra condition which must be met, within the context of there being virtually no wrong decisions. (4) The number of unmade decisions is much smaller for the RH digestion than for the *Eco*RI and *Hin*dIII digestions. This is because the average number of fragments per clone in the *Eco*RI and *Hin*dIII digestions is approximately half of that in the RH digestion. This does not supply enough data to make useful overlap decisions, given a minimum overlap of 3. (5) In these data, the weighted form of the algorithm has virtually the identical performance as that of the unweighted form.

The statistics of the simulated data produced here are similar to those found for actual biological fragment length data produced in a genetics laboratory, while the statistics of the simulated data used by Rhee are not similar to actual biological fragment length data. Thus, it is difficult to compare the performance results presented here with those generated by Rhee. However, in those few instances where the statistics are not too divergent, the results seem to be roughly comparable. In general, the DNA Mapping Group does not conclude that the results indicated here refute those found by Rhee. However, nor does the group conclude that these results conclusively corroborate his results.

4.2.3. Performance on Fragment Length Data with Random Error

Although it is of interest to verify Rhee's results, as applied to idealized fragment length data (i.e., fragment length data containing no error), the real reason to implement fragment length modifications of these algorithms is to apply them to actual laboratory data. As an intermediate step, they are applied to simulated fragment length data, to which random error has been introduced.

Table 2 presents the performance of these four algorithms over four different genomes *with* random error introduced and a minimum required overlap of 3. Although it is of interest to peruse all of the data, the discussion here will focus on the performance obtained for the 800kb genome; this is the approximate size that corresponds to a YAC (Yeast Artificial Chromosome), the unit of DNA to which the DNA Mapping software will most likely be applied. Also, the RH digestion (a double digestion) is the most pertinent to analyze, since this is the type of digestion most often done in the laboratory. It supplies enough data, in the form of average number of fragments per clone (roughly 6.5 fragments per clone) to attempt a significant mapping activity. The data extracted from single digestions often is too sparse (roughly 3.3 fragments per clone) to perform mapping with a high degree of certainty.

The most striking difference between the performance of the algorithms when applied to data with and without random error is the difference in wrong decisions. This is to be expected, of course, since the major source of fragment confusion is caused by measurement error producing inappropriate apparent overlap. Note that the percentage of wrong decisions for **MGREEDY_B** has risen from 0% to the 11-16% range, and that the percentage of unmade decisions has decreased slightly.

Table 2
Performance Data for 3% Random Error, Minimum required overlap = 3

Performance Results of Greedy Algorithms Applied to Simulated Fragment Length Data 3% Random Error, Minimum required overlap = 3							
				Performance Calculations (R_{wd},R_{ud})			
Genome Size	Enzyme	Total Clones	Maximal Clones	MGREEDY_B	MGREEDY_BS	MGREEDY_BW (length)	MGREEDY_BSW (length)
100kb	*Eco*RI	35	6	(0.000,0.400)	(0.000,0.600)	(0.200,0.400)	(0.000,0.600)
	*Hind*III		6	(0.000,0.400)	(0.000,0.400)	(0.000,0.400)	(0.000,0.400)
	RH		6	(0.000,0.000)	(0.000,0.000)	(0.000,0.000)	(0.000,0.000)
400kb	*Eco*RI	192	38	(0.054,0.324)	(0.000,0.378)	(0.081,0.324)	(0.000,0.378)
	*Hind*III		43	(0.048,0.310)	(0.024,0.310)	(0.048,0.286)	(0.048,0.310)
	RH		56	(0.091,0.054)	(0.000,0.091)	(0.073,0.018)	(0.000,0.010)
800kb	*Eco*RI	342	73	(0.111,0.375)	(0.056,0.417)	(0.153,0.347)	(0.069,0.430)
	*Hind*III		75	(0.122,0.392)	(0.054,0.459)	(0.203,0.351)	(0.094,0.500)
	RH		108	(0.159,0.019)	(0.019,0.065)	(0.243,0.019)	(0.075,0.131)
1600kb	*Eco*RI	606	122	(0.165,0.240)	(0.041,0.372)	(0.314,0.223)	(0.124,0.421)
	*Hind*III		125	(0.194,0.282)	(0.089,0.371)	(0.226,0.258)	(0.105,0.363)
	RH		193	(0.182,0.031)	(0.047,0.114)	(0.240,0.016)	(0.083,0.114)

Comparing **MGREEDY_B** with **MGREEDY_BS**, it is clear that the introduction of the subset-consecutiveness condition significantly reduces the number of wrong decisions; unfortunately, it causes the number of unmade decisions to rise slightly. Focusing in on the RH digestion shows extremely good results for the **MGREEDY_BS** algorithm — less than 2% wrong decisions and 7% unmade decisions. Translating these figures to a more concrete form yields the following analysis. The 0.065 value for R_{ud} implies that 7 decisions (out of the 107 to be made) were not made. This implies that there are 8 islands in the result computed by **MGREEDY_BS**, yielding an average of approximately 14 clones per island. (This is the correct order of magnitude for the desired size of contigs.) The 0.019 value for R_{wd} implies that only 2 of the adjacency decisions made within all of the 8 islands are wrong. This is a surprisingly small error rate for algorithms as simple as the **MGREEDY** family.

In comparing the performance of the weighted form of the algorithms to the unweighted form, note that the weighed form (at least when based on length) has worse performance than the unweighted form. This seemed surprising, since Rhee's work indicated that the weighted form should improve the performance.

In Rhee's abstraction, the possibility of a weighting metric based on length was excluded; however, as fragment length data were introduced, the possibility became evident. In the translation of Rhee's work to fragment length data, it was inferred that the sum of the lengths of the fragments with apparent overlap was a better metric for the likelihood of the adjacency of two clones than the cardinality of the fragments with apparent overlap. Thus, the original algorithms were coded to used the metric of length, with the counter-intuitive results shown in Table 2. In order to investigate the difference between Rhee's results and those indicated by Table 2, the algorithms were recoded to allow the option of using a metric based on cardinality. The results (only for the 800kb genome) of this experiment are shown in Table 3. Clearly, the cardinality metric significantly improves the performance of the weighted algorithms, thus supporting Rhee's results.

The question of how the minimum required overlap would affect the performance of these algorithms was investigated. (All of Rhee's work used a minimum required overlap of 3.) The results of reducing the minimum required overlap to 2 are shown in Table 4, and the results of increasing the minimum

Table 3
Performance Data Based on Cardinality Weighting

Performance Results of Greedy Algorithms Applied to Simulated Fragment Length Data 3% Random Error, Minimum required overlap = 3							
				Performance Calculations (R_{wd}, R_{ud})			
Genome Size	Enzyme	Total Clones	Maximal Clones	MGREEDY_B	MGREEDY_BS	MGREEDY_BW (cardinality)	MGREEDY_BSW (cardinality)
800kb	*Eco*RI	342	73	(0.111,0.375)	(0.056,0.417)	(0.069,0.375)	(0.056,0.403)
	*Hind*III		75	(0.122,0.392)	(0.054,0.459)	(0.081,0.365)	(0.027,0.459)
	RH		108	(0.159,0.019)	(0.019,0.065)	(0.112,0.000)	(0.028,0.065)

required overlap to 4 are shown in Table 5. The analysis of the data confirms the expected outcome. As the minimum required overlap is decreased, R_{wd} tends to increase while R_{ud} tends to decrease; as the minimum required overlap is increased, R_{wd} tends to decrease while R_{ud} tends to increase.

Given the encouraging results obtained on simulated data, it was decided to apply the algorithms to actual laboratory data. Data were obtained from the Olson laboratory. [4] These data correspond to RH digestions of 218 clones (92 maximal clones) from the CF YAC, so named because the cystic fibrosis gene lies somewhere in this region of DNA. This is the only human DNA data currently available from the

Table 4
Performance Data for 3% Random Error, Minimum required overlap = 2

Performance Results of Greedy Algorithms Applied to Simulated Fragment Length Data 3% Random Error, Minimum required overlap = 2							
				Performance Calculations (R_{wd}, R_{ud})			
Genome Size	Enzyme	Total Clones	Maximal Clones	MGREEDY_B	MGREEDY_BS	MGREEDY_BW (length)	MGREEDY_BSW (length)
100kb	*Eco*RI	35	6	(0.000,0.200)	(0.000,0.400)	(0.400,0.200)	(0.000,0.400)
	*Hind*III		6	(0.000,0.200)	(0.000,0.200)	(0.000,0.200)	(0.000,0.200)
	RH		6	(0.000,0.000)	(0.000,0.000)	(0.000,0.000)	(0.000,0.000)
400kb	*Eco*RI	192	38	(0.135,0.081)	(0.000,0.216)	(0.189,0.081)	(0.027,0.216)
	*Hind*III		43	(0.190,0.048)	(0.048,0.143)	(0.167,0.071)	(0.095,0.166)
	RH		56	(0.109,0.018)	(0.000,0.073)	(0.073,0.000)	(0.018,0.073)
800kb	*Eco*RI	342	73	(0.222,0.167)	(0.097,0.306)	(0.333,0.167)	(0.153,0.306)
	*Hind*III		75	(0.257,0.122)	(0.108,0.230)	(0.473,0.122)	(0.257,0.392)
	RH		108	(0.168,0.009)	(0.019,0.065)	(0.271,0.000)	(0.084,0.131)
1600kb	*Eco*RI	606	122	(0.298,0.091)	(0.116,0.256)	(0.479,0.066)	(0.264,0.273)
	*Hind*III		125	(0.306,0.113)	(0.153,0.234)	(0.379,0.064)	(0.194,0.226)
	RH		193	(0.203,0.010)	(0.068,0.094)	(0.266,0.000)	(0.088,0.114)

Table 5
Performance Data for 3% Random Error, Minimum required overlap = 4

Performance Results of Greedy Algorithms Applied to Simulated Fragment Length Data
Simulated Data with 3% Random Error, Minimum required overlap = 4

				Performance Calculations (R_{wd}, R_{ud})			
Genome Size	Enzyme	Total Clones	Maximal Clones	MGREEDY_B	MGREEDY_BS	MGREEDY_BW (length)	MGREEDY_BSW (length)
100kb	*Eco*RI	35	6	(0.000,0.800)	(0.000,0.800)	(0.000,0.800)	(0.000,0.800)
	*Hin*dIII		6	(0.000,0.400)	(0.000,0.400)	(0.000,0.400)	(0.000,0.400)
	RH		6	(0.000,0.125)	(0.000,0.125)	(0.000,0.125)	(0.000,0.125)
400kb	*Eco*RI	192	38	(0.000,0.540)	(0.000,0.541)	(0.000,0.540)	(0.000,0.540)
	*Hin*dIII		43	(0.048,0.548)	(0.024,0.548)	(0.024,0.548)	(0.024,0.548)
	RH		56	(0.073,0.073)	(0.000,0.109)	(0.036,0.073)	(0.000,0.109)
800kb	*Eco*RI	342	73	(0.042,0.556)	(0.028,0.583)	(0.028,0.556)	(0.028,0.583)
	*Hin*dIII		75	(0.027,0.649)	(0.000,0.676)	(0.014,0.649)	(0.000,0.676)
	RH		108	(0.103,0.084)	(0.009,0.084)	(0.168,0.084)	(0.037,0.140)
1600kb	*Eco*RI	606	122	(0.041,0.537)	(0.000,0.587)	(0.041,0.545)	(0.016,0.587)
	*Hin*dIII		125	(0.073,0.540)	(0.040,0.548)	(0.040,0.540)	(0.032,0.548)
	RH		193	(0.130,0.083)	(0.031,0.135)	(0.188,0.073)	(0.078,0.135)

Olson laboratory. These data had been mapped by the Olson laboratory, producing 21 islands of fused clones, thus supplying a set of *candidate* clones sequences. These clone sequences are only candidates, because they were created by software known to produce maps with minor errors. The underlying *reality* of how the data are *actually* related is not known; their relationship is inferred from indirect observations, such as apparent overlap. However, if these laboratory-created maps are taken as "ground truth", the performance analysis can be made between them and the results obtained from the **MGREEDY** family of algorithms. The results for the RH digestion, using a number of different values for the minimum required overlap, are shown in Table 6.

Table 6
Performance Data for Laboratory Data from the CF YAC

Performance Results of Greedy Algorithms Applied to Laboratory Fragment Length Data
CF YAC -- 218 total clones, 92 maximal clones

	Performance Calculations (R_{wd}, R_{ud})			
Required Overlap	MGREEDY_B	MGREEDY_BS	MGREEDY_BW (length)	MGREEDY_BSW (length)
2	(0.330,0.033)	(0.121,0.176)	(0.385,0.033)	(0.110,0.198)
3	(0.242,0.121)	(0.077,0.231)	(0.264,0.132)	(0.066,0.253)
4	(0.099,0.385)	(0.044,0.440)	(0.110,0.407)	(0.022,0.450)

The results here are based on a single ordering of the 21 islands produced in the Olson laboratory. This ordering was chosen to be the one most compatible with the results obtained from **MGREEDY** algorithms. This is not extremely significant, because most of the "gaps" between islands produced from the **MGREEDY** algorithms correspond to the "gaps" between the 21 islands produced in the Olson laboratory. These "gaps" seem to be actual gaps in the data collected from the CF YAC. In collecting the clone data from the CF YAC, the redundancy factor was approximately 3.5 (i.e., on the average, a specific region of the underlying genome will appear in 3.5 clones). This is a relatively low redundancy factor and statistically allows regions of the underlying genome to be covered by *no* clone in the sample. In contrast, the DNA simulator collected clones with a redundancy factor of 5.0. Although this does not eliminate the possibility of creating gaps, it does tend to reduce the number of gaps which occur.

These results are relatively encouraging. Focusing in on the weighted and unweighted **BS** algorithms for high minimum required overlap (3 and 4), the number of wrong decisions made is relatively low (between 2 and 7, out of 91). In fact, some of these decisions may actually be correct, since the "ground truth" used is only an approximation to reality; of course, the converse is also true, and there may be more wrong decisions than indicated.

The number of unmade decisions seems uncomfortably high. However, these ratios are artificially high because of the gaps in the data being analyzed. For example, consider the R_{ud} value of 0.450 shown for the **MGREEDY_BSW** algorithm with a minimum required overlap of 4. This value indicates that there were 41 unmade decisions, i.e., there are 42 islands in the result reported. However, most of these islands are caused by the actual gaps in the clone data between the 21 islands in the input data. Thus, many of the unmade decisions were not made because it was not possible for them to be made.

5. Discussion and Conclusion

The preliminary performance results reported here for the **MGREEDY** family of algorithms are very encouraging. The modification of these algorithms to allow them to address fragment length data did not significantly deteriorate the results reported by Rhee. These algorithms are relatively simple, and run in polynomial (of low degree) time. The weighted and unweighted forms of the **BS** algorithm seem to perform best. In the data space region of most interest (RH digestion data on genomes of approximately 800kb), these algorithms seem to give outstanding performance.

We are interested in using these algorithms for producing **ordered contigs**. Thus, we are interested in algorithms within the **MGREEDY** family which produce results with R_{wd} as low as possible, and R_{ud} in a moderate range, say between 5 to 15%. These statistics seem to be obtainable from the **MGREEDY_BS** algorithm for genomes in the 800kb range, with minimum required overlap between 3 and 4. Once the cardinality weighting metric is firmly in place, the **MGREEDY_BSW** algorithm may provide the best performance.

References

1. W. Gillett and J. Heidemann, DNA Mapping Algorithms: The DNA Simulator, Washington University, Dept. of Computer Science, Technical Report WUCS-90-37, October, 1990.

2. W. Gillett and L. Hanks, DNA Mapping Algorithms: Abstract Data Types - Concepts and Implementation, Washington University, Dept. of Computer Science, Technical Report WUCS-91-33, June 1991.

3. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, Maryland, 1984.

4. M. V. Olson, J. E. Dutchik, M. Y. Graham, G. M. Brodeur, C. Helms, M. Frank, M. MacCollin, R. Scheinman and T. Frank, *Random-Clone Strategy for Gemonic Restriction Mapping in Yeast*, Genomics, Vol. 83, October, 1986.

5. G. Rhee, DNA Restriction Mapping From Random-clone Data, Washington University, Dept. of Computer Science, Technical Report WUCS-88-18, June, 1988.

6. G. Rhee, Computational Models for DNA Restriction Mapping, Doctoral Dissertation, Washington University, Dept. of Computer Science, May, 1990.

7. J. S. Turner, Approximation Algorithms for the Shortest Common Superstring Problem, Washington University, Dept. of Computer Science, Technical Report WUCS-86-16, July 1986.

8. J. S. Turner, The Complexity of the Shortest Common Matching String Problem, Washington University, Dept. of Computer Science, Technical Report WUCS-86-9, April 1986.