AUTOMATIC CONSTRUCTION

OF

RESTRICTION SITE MAPS BY COMPUTER

BY

NORBERT E. BAUMGARTNER

THESIS

For The

DEGREE OF BACHELOR OF SCIENCE

IN

LIBERAL ARTS AND SCIENCES

College of Liberal Arts and Sciences
University of Illinois
Urbana, Illinois

1984

UNIVERSITY OF ILLINOIS

May 8, 19 84

THIS IS TO CERTIFY THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Norbert E. Baumgartner

ENTITLED Automatic Construction of Restriction Site Maps by Computer

IS APPROVED BY ME AS FULFILLING THIS PART OF THE REQUIREMENTS FOR THE

DEGREE OF Bachelor of Science in Liberal Arts and Sciences

Stephen G. Sligar
Instructor in Charge

APPROVED: Lawrence Hunter

HEAD OF DEPARTMENT OF Biochemistry

O-1364

ACKNOWLEDGEMENTS

I would like to thank the two people without whose help this work would not have been possible. First of all I would like to thank Ben P. Unger for his valuable suggestions, technical advice, and for teaching me the laboratory methods of restriction mapping. I would also like to thank my project advisor, Dr. Stephen G. Sligar, for all his helpful advice and suggestions for improving the programs, and for his constant support and encouragement.

TABLE OF CONTENTS

INTRODUCTION

Many laboratories are currently engaged in the analysis and manipulation of various genetic sequences. One of the most valuable tools for manipulating these segments of DNA is a class of enzymes known as restriction endonucleases, and the development of a restriction enzyme cleavage map is often the first step in the analysis and base sequencing of an isolated gene. The goal of this project is to develop an algorithm and a computer program that will automatically generate these restriction site maps from experimental data. In order to understand the methods of developing restriction site maps, it would be useful to first review some of the properties of restriction enzymes and the methods of separating fragments of DNA.

General Properties of Restriction Endonucleases

Restriction enzymes are endodeoxyribonucleases that recognize specific nucleotide sequences in double stranded DNA and cleave both strands of the duplex. Restriction enzymes are found in many bacterial strains as part of a restriction-modification system (1). This system consists of the restriction endonuclease and a matched modification enzyme which recognizes the same nucleotide sequence recognized by the restriction enzyme and modifies (usually by methylating) the cellular DNA. This modification protects cellular DNA from degradation by the restriction enzyme. Unmodified DNA, such as foreign DNA that enters the cell via viral transduction, is quickly destroyed by the restriction enzyme. It is thought that this is the function of the enzyme in the host organism.

Restriction enzyme nomenclature is based on the name of the organism from which the enzyme is isolated (2) and the enzymes are generally separated into two classes. Class I enzymes are non-specific in their cleavage and are there-

fore of limited usefulness in molecular biology. Class II enzymes (3) recognize specific sequences in DNA, usually 4-6 base pairs possessing twofold rotational symmetry (4), and require only Mg^{2+} as a cofactor. Cleavage positions within the recognition sequence are either "blunt" or "staggered". Staggered cleavage results in the formation of identical self-complementary cohesive termini. This property is utilized to insert a DNA fragment into a vector to produce a recombinant molecule. A relatively large number of restriction enzymes share a much smaller set of recognition sequences. Enzymes which share a common recognition sequence are known as isoschizomers. Since these isoschizomers yield identical cleavage patterns for a given DNA, the most stable and easily purified enzyme can be selected for use. Other properties of restriction enzymes are reviewed in reference (4).

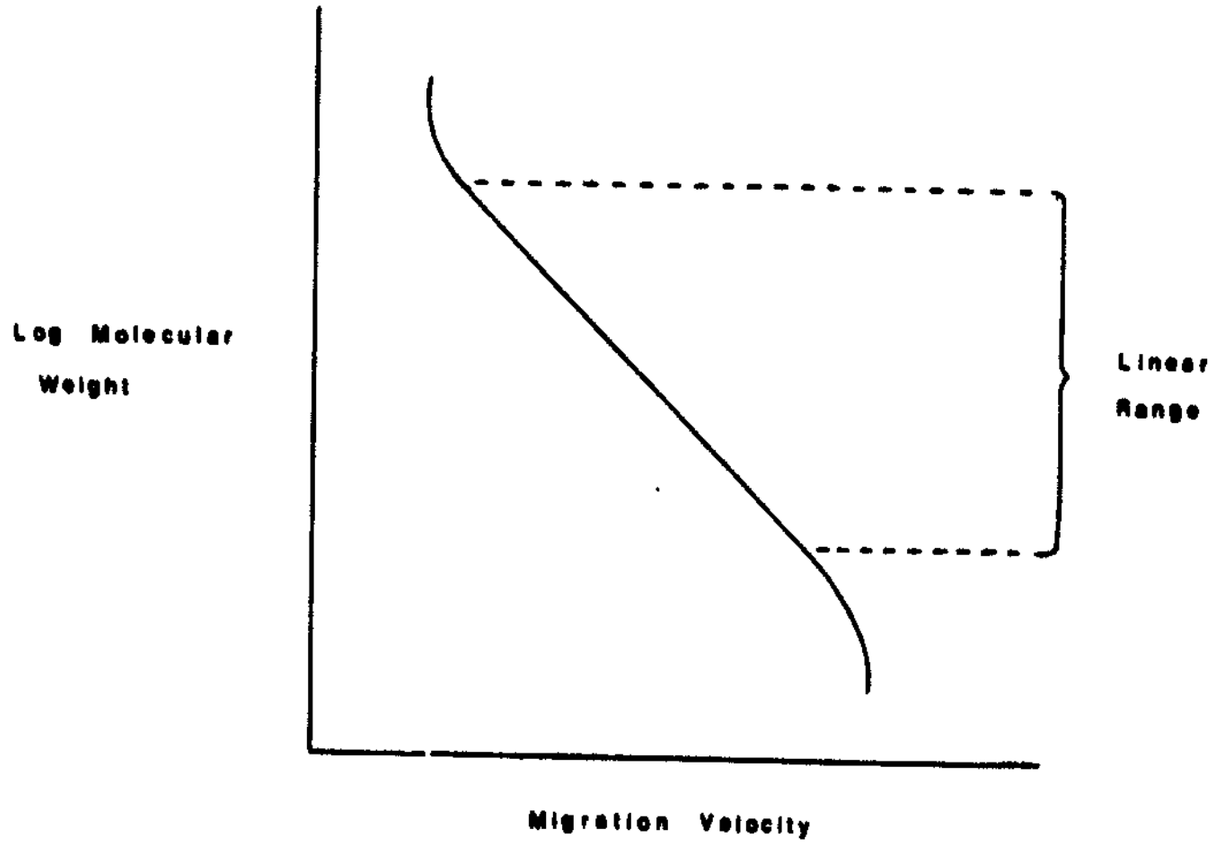
Separation of DNA Fragments and Fragment Size Determination

It is often necessary to separate a heterogenous population of DNA on the basis of size. This is especially important in some restriction site mapping techniques where the cleavage products must be resolved and the sizes of the fragments determined. Probably the easiest, most inexpensive, and most accurate method of separation by size is gel electrophoresis using a polymerized slab of either agarose or polyacrylamide. The methods of agarose and polyacrylamide gel electrophoresis are described in reference (5). By varying the composition of the gels, various separation ranges can be obtained (Table I). In each of these ranges (with the exception of the 20% polyacrylamide gel) there exists a region in which the logarithm of a molecule's length is proportional to its migration velocity (Figure 1). At either extreme of a range this relationship breaks down and the length cannot be accurately determined from the migration velocity. By running the unknown sample alongside standards of known

Table 1: Separation ranges produced by agarose and polyacrylamide gels of various composition. Modified from (5).

Figure 1: Relationship between molecular weight and migration velocity on agarose or polyacrylamide gels. The linear region is indicated. Modified from (5).

Gel	Separation Range (base pairs)
0.3% agarose	50,000 to 1,000
0.7% agarose	20,000 to 300
1.4% agarose	6,000 to 200
4% polyacrylamide	1,000 to 100
10% polyacrylamide	500 to 25
20% polyacrylamide	50 to 1



size, the sizes of the unknown fragments can be determined (using the standards and the linear relationship between size and migration velocity) to within 1%. This accuracy does not apply over the entire range of a given gel. In the case in which fragments differ in size over the entire range of the gel, this accuracy is closer to 10%. Some workers have described a method of relating molecular weight to mobility using a cubic exponential function (6). This method allows the relative molecular weight of a fragment to be determined to within ±1.5-2.5% without the introduction of standards. The use of composite agarose-polyacrylamide gels or linear gradient gels may permit a wider range of sizes to be separated on a single gel and increase the sharpness of the bands (4).

Bands of DNA separated by gel electrophoresis may be visualized by any one of several methods. Regardless of which method is used, it is desirable to be able to quantitate the DNA in each band. This will permit detection of low frequency partial digests and bands consisting of two or more fragments of equal or similar size. One frequently used method of visualizing bands involves treating the gel with either ethidium bromide, methylene blue, toluidine blue, or other stain. Ethidium bromide is a fluorescent that is excited by short or long wavelength ultraviolet light and is sensitive to the level of a few nanograms (4,5). Stained gels may be optically scanned for quantitation. Another commonly used method for visualizing electrophoresis bands is autoradiography. To utilize this technique, DNA must be radiolabeled and then either exposed to a photographic emulsion, which can be quantitated by a densitometer tracing of the exposed film, or, more accurately, the band is excised and the radioactivity measured by scintillation counting. The autoradiographic technique is also very sensitive to small amounts of DNA (4).

The methods of gel electrophoresis provide a rapid and convenient method

for separating a mixture of DNA fragments, as in a restriction enzyme digest, and for determining the sizes of the resolved fragments and the number of fragments in each size class. The methods are also fairly accurate if the range of fragment sizes are within the linear portion of the fragment size/mobility curve. This also presupposes that the purine/pyrimidine ratio is fairly constant. G+C bias alters mobility in gel electrophoresis (6) and a DNA sample with a large G+C bias will significantly affect the size determination. The application of gel electrophoresis to restriction enzyme digests will become apparent when the methods of restriction mapping are discussed.

Probability and Combinatorics Associated with Restriction Mapping

When using restriction enzymes to cleave fragments of DNA for gene isolation, base sequencing, etc. it is very useful to be able to predict approximately how large the resulting fragments will be for a given restriction enzyme. As previously described, the recognition sequence for most restriction enzymes is either 4 or 6 base pairs. These are referred to as "4-cutters" and "6-cutters" respectively. Given a recognition frame of 4 base pairs, each of which can be any one of the 4 bases (A, T, C, or G), and an essentially random distribution of bases in the DNA to be cleaved, a given recognition sequence would be expected to occur every 4^4 or 256 base pairs (bp). Thus the average fragment length for a 4-cutter restriction enzyme is 256 bp. Likewise for a recognition frame of 6 bases, a given recognition sequence would be expected to occur every 4^6 or 4096 bp, and the average fragment length for a 6-cutter would be approximately 4.1 Kb. From this information the number of fragments produced by a digest can be predicted. For example, a 2.2 Kb gene (perhaps encoding a protein molecule of interest) would be cut into 9 fragments by a 4-cutter restriction enzyme that cuts every 256 bp. It should be emphesized that these approxima-

tions assume a random distribution of bases in the source DNA; non-random sequences such as poly-purine or poly-pyrimidine regions would obviously result in either more or fewer cuts than expected, depending on the recognition sequence.

Restriction site mapping often involves the ordering of fragments produced by complete restriction digestion of a segment of DNA. In order to appreciate the magnitude of the problem, it is necessary to consider the combinatorics involved in ordering the fragments. If ν represents the number of fragments produced by a given restriction digest, then the number of possible orderings of the ν fragments, ρ, is given by

$$\rho = \nu!$$

For small values of ν the number of orderings is likewise relatively small, however this number rises rapidly with larger values of ν (e.g. ν=9 in the previous example) commonly encountered in restriction mapping. The goal of a restriction mapping algorithm, therefore, should be to reduce the number of possible orderings in some way. For example, if the number of fragments to be ordered in a 10 fragment digest could be reduced by 1 (perhaps by end-labeling the DNA so that a terminal fragment could be identified), the number of permutations would be reduced from 3.63×10^6 to 3.63×10^5 - a tenfold reduction. Successive elimination of fragments by assignment would further reduce the number of possible orderings. It is evident, therefore, that a mapping algorithm based on a "brute force" generation of possible orderings is both time consuming and inefficient and that a better approach would be to somehow successively eliminate fragments, thereby successively decreasing the number of possible permutations.

Existing Methods, Algorithms, and Computer Programs

A number of laboratory methods, algorithms, and computer programs have been developed to generate restriction site maps (4,6,7,8,9,10,11). One method uses single digestions of two or more different enzymes and a combined digestion, hereafter referred to as an n-digest, of n (where $n \geq 2$) different enzymes. The fragments in the n-digest are combined in ways so as to generate fragments consistent with the single digest data. This is often a trial and error problem and rarely, if ever, are all the possible solutions examined for large data sets when done by hand. It allows for the possibility that not all solutions are found and may result in an incorrect solution, since multiple solutions are sometimes possible for a given set of data, and, at the very least, it is a tedious process. In an attempt to overcome these problems, computer programs have been developed (6) to examine all possible combinations of n-digest fragments. This method assures that all solutions possible are found, however it is very slow (because of the number of permutations) on all but the fastest computers. Algorithms have been developed (7) that allow this problem to be solved with or without the aid of a computer. One such algorithm uses a "branch and bound" technique that examines various alternatives in order to minimize the remaining alternatives. The difficulty with this algorithm is that it is based on a large number of rules for eliminating alternatives and that it does not completely reduce all of the alternatives. Other computer programs (8) use a model-driven algorithm and a large set of canonical form and pruning rules in order to eliminate incorrect classes and generate a solution by negative inference.

Various laboratory methods have also been developed to generate restriction site maps. One technique uses end labeled DNA and partial digestion with a single restriction enzyme (9). This method is similar in concept to that us-

ed by Maxam and Gilbert for DNA sequencing. Another method uses a two-dimensional hybridization technique (10) to deduce the order of restriction sites. DNA to be mapped is treated with one restriction enzyme and electrophoresed in one dimension. Additional DNA is treated with a second enzyme and electrophoresed in the other dimension. From the hybridization pattern of the two sets of fragments, the map order of the enzymes can be determined. Finally, a cleaved permutated linear method (11) has been developed in which a circular DNA molecule is singally cleaved by one enzyme to give a complete set of permutated linears. These permutated linears are then cleaved by a second enzyme into fragments from which the mapping order of the single-hitting enzyme can be determined. All of these laboratory techniques have the disadvantage of being much more difficult to carry out and much more time consuming. Because some require only one cut by a restriction enzyme, conditions must be chosen to fulfill this requirement. Under the conditions that result in only single cuts, however, some sites may not be cleaved and therefore will be missed. This represents a serious problem and makes these methods far from perfect.

Clearly the present techniques and algorithms for restriction site mapping are not adequate to meet the needs and requirements of all those engaged in restriction mapping. What is needed is a technique that uses simple, reliable laboratory methods and that quickly and exhaustively generates all possible solutions from the available data. Such a technique has been developed and is described in the following pages.

MATERIALS AND METHODS

After a review of existing restriction mapping methods and algorithms, the method of multiple single digests and a single n-digest was selected for data aquisition. This decision was based primarily on the simplicity of this method relative to the other methods previously described and also its reliability. A model of the solution space was constructed, and from this model a method of checking the validity of the data was developed. This model, along with a consideration of the data's characteristics, allowed a recursive method of eliminating incorrect solutions in a top-down (i.e. more general to more specific) fashion to be developed. From this, a pair of mapping algorithms quickly followed: one for linear DNA and one for circular (plasmid) DNA.

The algorithms were then implemented in a computer program written in Microsoft BASIC-80 for an Osborne Z-80 based microcomputer running under a CP/M operating system. The program was debugged and tested using hypothetical digest data. For reasons of accessability, the program was also translated into VAX-11 FORTRAN Version 3.0 (based on ANSI X3.9-1978 FORTRAN-77) for use on a VAX-11 timesharing computer system running under the VAX/VMS Version 3.0 operating system. The program was also tested on several well characterized vectors (12) and some recently analyzed molecules (Unger, B.P. unpublished data) in order to assure that the correct solutions obtained during testing were not merely a spurious result of the hypothetical data selected.

RESULTS

ALGORITHMS

Before presenting the algorithms, it would be useful to consider some of the properties of the data that allowed the algorithms to be developed and some ways of checking the validity of the data that follow from these properties. A number of assumptions have been proposed (7) which must be satisfied by the data generated by the single digest/n-digest method:

1. The DNA being digested is pure (i.e. free from contaminating species).
2. The DNA has been fully digested and contains no partial digests.
3. Each enzyme cuts the DNA at least once.
4. There are no fragments missing.
5. If there are two or more fragments of the same size, they are detected as such.
6. The error in estimating the restriction fragment lengths is either known or has an upper limit.

Assumption 1 is important in that a contaminating species may contribute fragments that will interfere with the ordering of the desired species' fragments. The validity of this assumption can be tested by electrophoresing undigested DNA preparations or by quantitating the DNA in each fragment band (since a contaminating species will most likely be present in lower concentrations than the desired species and hence the resulting fragments from this species will also show a lower concentration). The validity of assumption 2 can be assured by allowing a long incubation period with the restriction enzyme (provided it is sufficiently free of contaminating nucleases) or tested by end labeling. End labeling should only produce one labeled fragment if the digestion is complete. This assumption is also important because it may introduce erroneous fragment

sizes. Assumption 3 is easily varifiable by examining the single enzyme digests, and its significance is obvious in that it is useless to try to map a restriction site that does not exist. Assumption 4 must also be true in order to derive solutions from the data. This assumption can be tested by utilizing properties of the data structure. For a circular DNA structure, the sum of the number of single digest fragments, f_i, equals the number of n-digest fragments, f_n

$$\sum_{i=0}^{n} f_i = f_n \qquad \text{(where n = number of enzymes)}$$

For a linear DNA structure, this relationship is

$$\sum_{i=0}^{n} f_i = f_n + (n-1)$$

Therefore, by comparing the number of single digest fragments and n-digest fragments, missing fragments in either the single digest or n-digest class can be detected. If equal numbers of fragments are missing from each class, however, these will cancel each other and go undetected by this method. This situation will be discussed later. Assumption 5 is important in that if it is not true, a missing fragment will result. This assumption can also be tested, and multiple fragments of the same length can be found, again by quantitating the DNA in each of the electrophoresis bands. Finally, assumption 6 becomes important when the sizes of the fragments are not known exactly, as is invariably the case in electrophoresis techniques. This error can be determined empirically for a given set of reaction conditions by running two different sets of standards on the separation gel. Once all of these assumptions have been satisfied, the data is in a form suitable for mapping by the algorithms that follow.

Linear DNA Restriction Site Mapping Algorithm

The linear restriction mapping algorithm will be discussed first because a linear segment's property of having a defined beginning and end results in a simpler ordering algorithm. The algorithms both use a top-down approach which enumerates the solution space by refining general hypotheses. Rather than proposing complete solutions and then ruling out the incorrect candidates, as is the case in a data-driven approach, the algorithms recursively generate and test branches and eliminate those branches of the solution space that are inconsistent with the model of the solution space. For this reason, this type of approach is termed "model-driven" (8). The various branches of the solution space are joined at each level to a more general branch by "nodes". When diagrammed, this model of the solution set resembles a tree. The single, most general branch at the bottom of the structure is termed the "root" and the more specific branches at the top of the structure are the branches proper.

The linear algorithm begins with the assignment of the root. Because there are non-cleaved ends in a linear DNA segment, there are at least two fragments (one at each end of the molecule) in the single digests that do not have any other restriction sites within them (i.e. there must be a first site and a last site in the segment) and hence appear in both a single digest and the n-digest. Therefore, all fragments that appear in both a single digest and the n-digest (within the allowable error range) are potential roots until proven otherwise. The number of tree structures that must be examined in finding a solution, therefore, is equal to the number of potential roots generated. The node that terminates the root can also be identified and is assigned the enzyme in whose single digest the root fragment was found. Hence the branches of the solution space are the n-digest fragments and the nodes are enzyme cleavage sites. After assigning a fragment to the root (and an enzyme to the first node) the num-

ber of possible orderings of the remaining fragments has been reduced from $f_n!$ to $(f_n-1)!$. In general, the number of possible orderings remaining at any given time is $(f_n-\ell)!$ where ℓ is the node level (how high up in the "tree" a given node is). The node level then ranges from 1 to the number of n-digest fragments (f_n).

The next step is the recursive generation cycle. This involves proposing branches for every "open" node at the current level. The branches proposed at each node are those n-digest fragments that have not already been assigned (i.e. that do not appear in the path traced from that node back to the root). The number of branches possible at each node is a function of the node level, ℓ, and is given by $(f_n-\ell)$. Each proposed branch is then tested by successively assigning each enzyme to the terminating node. The fragments from that node are summed back to the last occurance of that enzyme or the end of the DNA segment, whichever comes first. This sum, which gives a hypothetical fragment flanked on both sides by that enzyme, is compared to remaining single digest fragments for that enzyme. If found in the single digest list, within the error range, this branch and node are assigned at this node level and the remainder of the enzymes are tested. If the sum (hypothetical single digest fragment) is not found in the single digest list, the remainder of the enzymes are tested and if none can be assigned, the node is considered "closed" and need not be considered at subsequent node levels.

This process repeats itself at the next node level, successively assigning or eliminating branches, until either all nodes in the tree terminate, in which case there are no solutions for that tree, or until the top of the tree is reached (the last remaining fragment is simply checked against the single digest lists to verify that it is indeed an end fragment) and one or more paths, now solutions, can be traced back to the root. This method of eliminating branches of the sol-

ution set until only one or more completed solutions remain is a form of negative inference and is much more efficient than a method that must generate all possibilities and select correct solutions by positive inference.

Once a tree is completed, the solutions (if any) are collected and the next tree is examined. Before entering into the next generation cycle, however, the root fragment is compared to the last fragment in each previous solution. If a match is found (within error limits) the tree is skipped because it will only generate the reverse of a previous solution. In space these solutions are equivalent (degenerate) and it is therefore not necessary to examine a tree that will not generate any new solutions.

The solutions that remain after all trees have been examined are all possible non-degenerate solutions for the given data. An example of the linear algorithm, showing the tree structures, is given in Figure 2. This example uses hypothetical data for clarity.

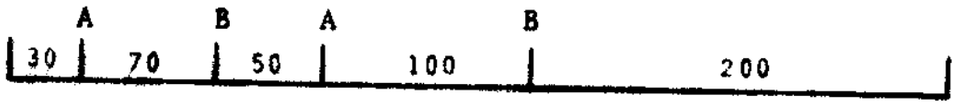
Examination of the algorithm suggests that in the best possible case (the case in which only one tree is considered, and each node level has only one branch assignment as in Figure 3) the number of incomplete orderings examined, ϵ, is given by

$$\epsilon = \sum_{i=1}^{f_n-2} (f_n-i)n$$

where f_n = number of n-digest fragments
and n = number of enzymes ($n \geq 2$)

Substituting 10 for f_n and 2 (the simplest case) for n, the value obtained for ϵ is 88. Compared to a previous example in which the number of permutations of 10 fragments was found to be 3.63×10^6 (which does not even take into account the permutations of cleavage sites) this represents a tremendous savings of computational effort.

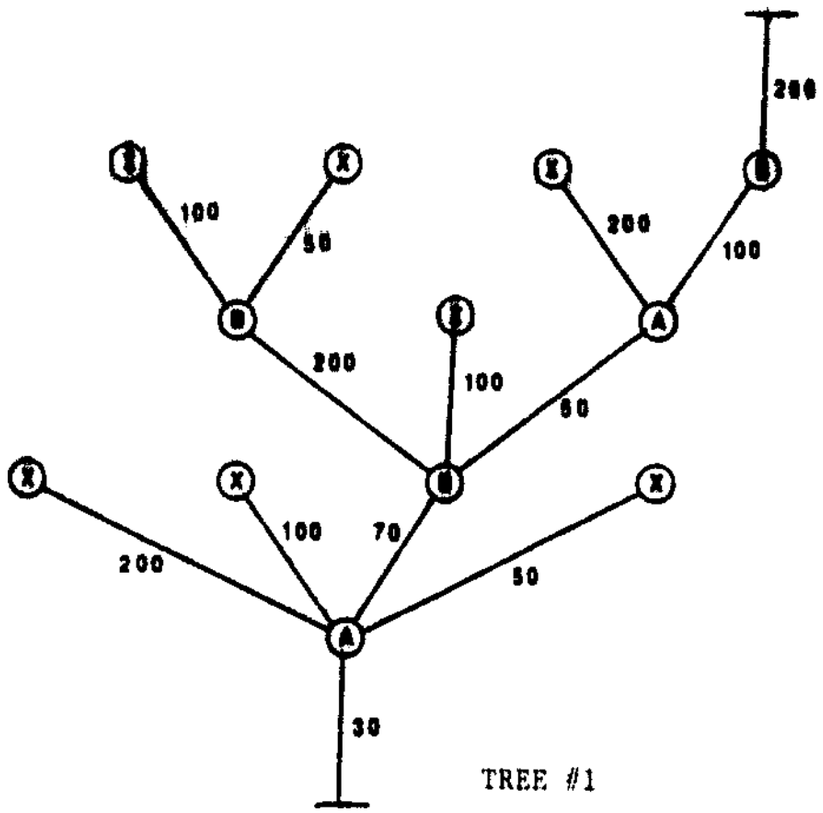
Figure 2: Example of linear DNA mapping algorithm using two enzymes and five n-digest fragments. Hypothetical map and digest data is given for enzymes A and B. Numbers on trees are sizes of fragments, letters inside of nodes (Ⓐ) indicate enzymes for assigned restri tion sites. Terminated nodes are indicated by Ⓧ.



HYPOTHETICAL SOURCE RESTRICTION MAP

DIGEST DATA

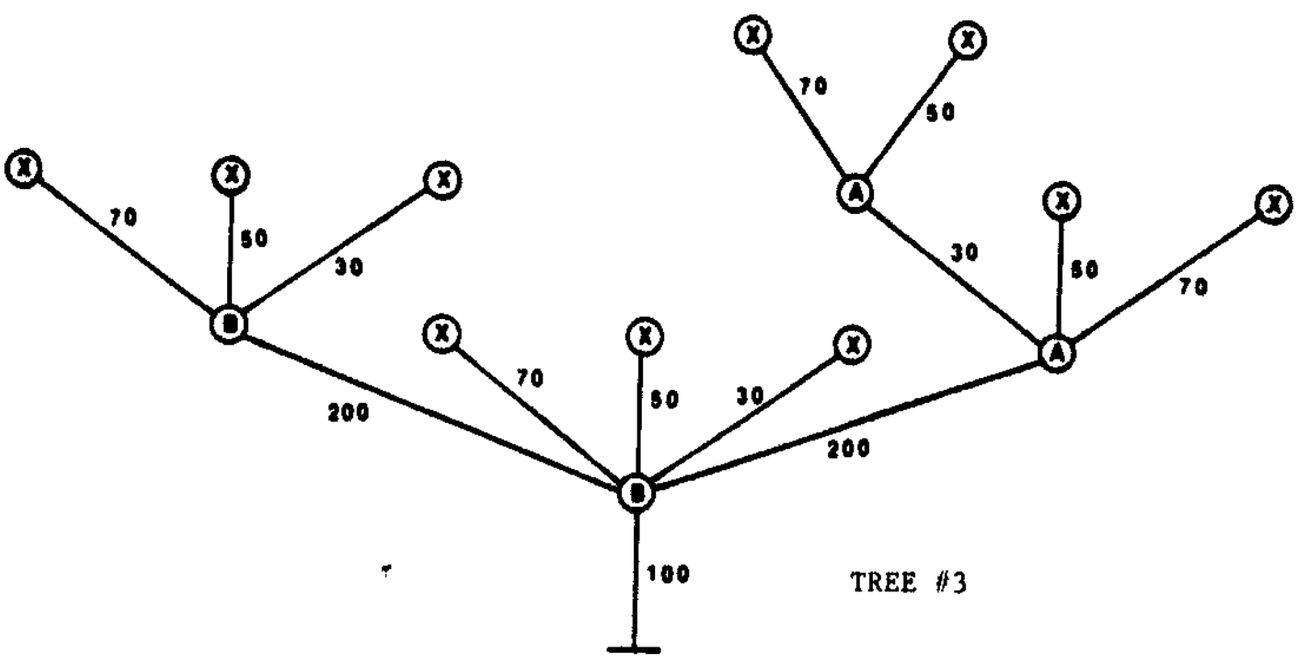
A	B	A+B (n-digest)
30	100	30
120	150	50
300	200	70
		100
		200



TREE #1

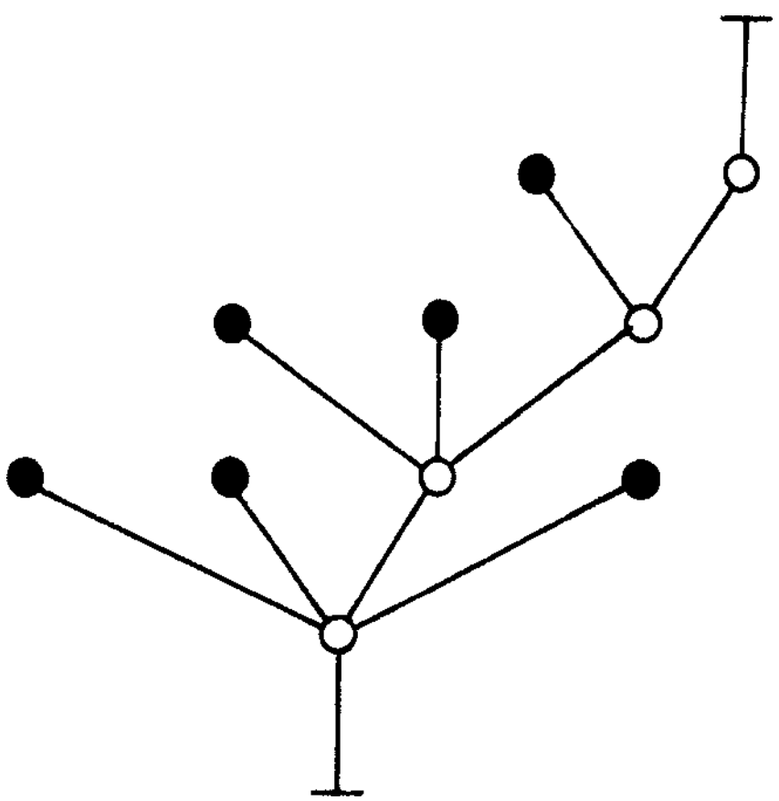
LINEAR TREE STRUCTURES

TREE #2



TREE #3

Figure 3: Ideal case for linear map. Lines indicate fragments, open circles (○) indicate nodes (restriction sites), and closed circles (●) indicate "pruned" branches. Example is for five fragment n-digest.



Circular DNA Restriction Site Mapping Algorithm

The algorithm for the mapping of circular DNA is essentially the same as that for linear DNA with a few differences because of the unique topological properties of circular DNA. First of all, because there is no beginning or end, there is no root fragment from which to begin. Therefore an arbitrary point is chosen from which to open the plasmid, such as a cleavage site for the first enzyme. If this is done hypothetically, a linear DNA molecule would be created with one half of the same cleavage site on each end. Because no unique root fragments can be found in the data, each n-digest fragment must be tried as a potential root fragment for this hypothetical linear segment until a solution is found. This amounts to searching for a fragment anywhere in the circular molecule that is adjacent to a cleavage site for the first enzyme (enzyme #1). The number of such fragments, t, is given by

$$t = 2s - a$$

where s = number of sites for enzyme #1

and a = number of adjacent enzyme #1 sites

If t is maximized (by selecting the enzyme with the most cuts to be enzyme #1) the probability of finding one of these adjacent fragments is much greater and therefore fewer trees need to be generated before a solution is found.

Once inside a tree structure, the first fragment is successively assigned enzymes as potential nodes, just like any other open branch. However, if the paths are traced back and no previous occurance of the enzyme is found, rather than stopping at the end the path must "wrap-around" to the other end (because it is really still a circular molecule as far as non-enzyme #1 single digests are concerned). Because the other end of the map is as yet undetermined, the node is tentatively assigned that enzyme (for lack of evidence that could exclude the possibility) and the usual process continues. After the last frag-

ment has been assigned, the fragments on either side of the opening site (wrap-around fragments) are summed until the first occurence of each enzyme, successively. These sums are then compared to the single digest data for the respective enzymes and if a discrepancy is found within the error range, the path containing that sum is rejected. If no solutions are found the next tree is examined, otherwise all paths through the solution space that remain are all the possible solutions for the given data. If all trees are examined and no complete paths are found, then there are no solutions possible. An example of the circular algorithm using hypothetical data is found in Figure 4.

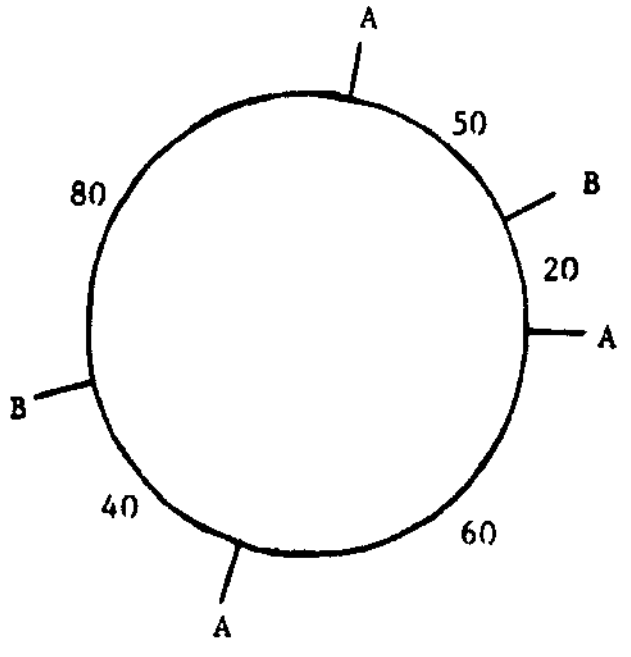
The circular algorithm is not as efficient as the linear algorithm because nodes are often assigned tentitively and may not be rejected until the wrap-around is tested. However, the number of orderings examined may still be very small because only one tree need be examined to find all the solutions if the first fragment tried is adjacent to an enzyme #1 site.

COMPUTER PROGRAMS

The computer programs (Appendices A and B) written to implement the algorithms are essentially the same, so they will be described in general first and then specifics for each will be given.

The major problem encountered in developing the software was organization and allocation of memory for various storage functions. Arrays of various dimensions were chosen to represent various structures in the construction of the maps. The original digest data supplied to the program is stored in the two-dimensional array, F. The first subscript (i.e. rows) corresponds to the digest number. A digest number of 0 refers to the n-digest, while single digests are given the numbers 1 through n (where n = number of enzymes) in the order they

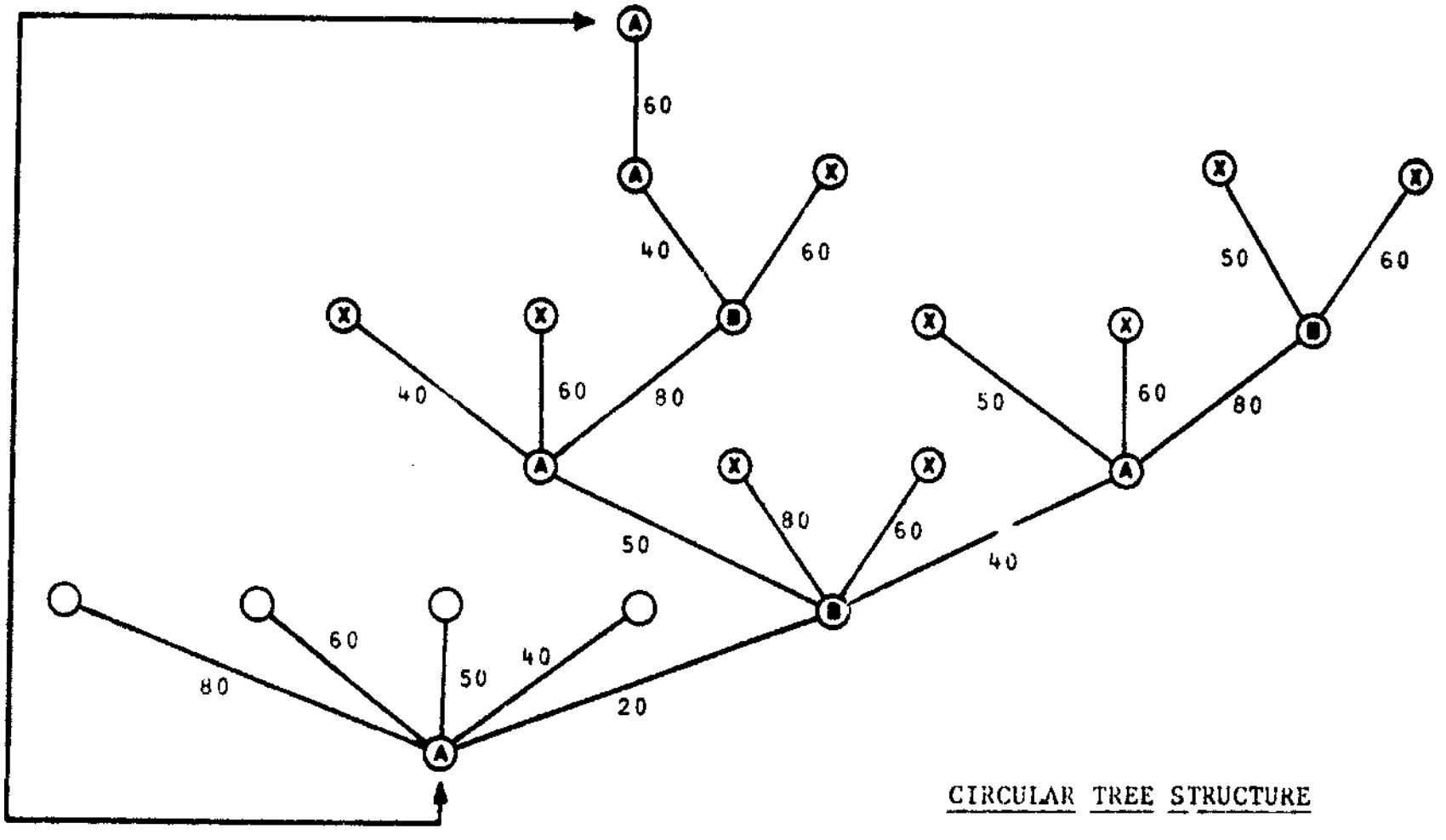
Figure 4: Example of circular DNA mapping algorithm using two enzymes and five n-digest fragments. Hypothetical map and digest data is given for enzymes A and B. Numbers on tree are sizes of fragments, letters inside of nodes (Ⓐ) indicate enzymes for assigned restriction sites. Terminated nodes are indicated by Ⓧ, unassigned nodes are indicated by open circles (○).



HYPOTHETICAL SOURCE RESTRICTION MAP

DIGEST DATA

A	B	A+B (n-digest)
60	120	20
70	130	40
120		50
		60
		80



CIRCULAR TREE STRUCTURE

are entered. The second subscript (i.e. columns) then references the fragment number within each digest. Column 0 of each digest contains the number of fragments in that digest, so that the fragment numbers in digest x range from 1 to F[x,0]. The list of roots for a linear DNA map is stored in the two-dimensional array, ROOT. This array contains two columns of data: a list of root fragments and a list of node enzymes. Rows range from 1 to the number of potential roots, whereas columns are referenced by a fragment/node code. A value of 1 for this code indicates that the column of fragment sizes is to be accessed, while a value of 2 indicates that the column of node enzymes is to be used. The three dimensional array BLDMAP is where the maps are assembled. The first subscript references a map (or path) number. Each unique path through the solution space can therefore be accessed individually. The second subscript gives the fragment or node number number within each path and ranges from 1 to f_n (the number of fragments in the n-digest). This corresponds to the order in which the fragments and nodes are assigned. The third subscript is the fragment/node code as described for ROOT. For a given value of the second subscript (i.e. node level) the value of the third subscript accesses either the fragment size at that level or the node enzyme at that level. A new BLDMAP is generated for each tree. MAPSOL is a list of completed solutions, copied from BLDMAP after all fragments are ordered, and has the same organization as BLDMAP. This array collects the solutions from all the trees. LAST is a two dimensional array that contains a list of last node levels assigned to each enzyme. The first subscript references a map number (as for BLDMAP) so that each path has its own list of last nodes assigned, and the second subscript references the enzyme number (ranging from 1 to n). A temporary copy of LAST, TMPLST, is made when creating new branches for each path. STACK, like F, contains digest data, however STACK is volatile and fragments are deleted as they are assigned to the maps.

This way it is easy to keep track of which fragments remain t signed. STACK is a three dimensional array: the first subscript refere he path number, the second subscript references the digest number (as f), and the third subscript references the fragment number within a particular digest. A temporary copy of STACK, TMPSTK, is also created when new branches are generated for a given path. All other program variables are fairly obvious and are described in the programs.

The programs essentially consist of four parts: a control section and three subprograms. The control program simply displays a menu and calls the appropriate subprogram based on the user's selection. The subprograms handle all of the data entry, map generation, and output.

The first subprogram is the data entry routine. This section solicits information from the user as to source DNA topography, number of enzymes used, enzyme names, fragment lists for each digest, error in fragment size measurement, and a line of text to be displayed at output. After all fragment data has been entered, the subroutine sorts the fragments in each digest from smallest to largest using a standard bubble sort. This is not absolutely necessary but it makes the data more presentable and makes the solutions generated independant of the order in which the data is entered. Once the fragments are sorted, the data is checked for missing fragments using the method described earlier. If data is missing, the user is alerted to this fact and asked to supply the missing data. After the data is checked for missing fragments, the program sums the fragments for each digest and computes an average. Each total is then compared to this average and if a discrepancy is found outside the allowable error range, which is chosen to be a fixed percentage of the fragment size (given the linear relationship between the log fragment size and the gel mobility), the user is alerted to this descrepancy and is asked to supply a new er-

ror value or re-evaluate the data. Some causes for this discrepancy might be incomplete digestions, impure DNA, or simply too small of an error range which causes one or more of the digests to be too large or too small than the average. Once the data has been checked for size inconsistency, data entry is complete and program control is passed back to the control section.

The data output subprogram simply prints out a summary of the data and a list of solutions. The solutions are given as a linear list of alternating fragment sizes and restriction sites. The fragment sizes indicate distances between adjacent sites. Linear maps have terminal fragments, circular maps have terminal restriction sites. The two terminal restriction sites represent the same site in the circular form and should be drawn as such on a circular diagram of the maps.

The map generator subprogram contains the actual restriction mapping algorithm. This routine contains both the linear and circular algorithms and by checking the topography skips over those sections that are not relavent for one or the other type of DNA structure. The subprogram begins by finding all possible roots if the DNA is linear. Next it sets a pointer for the final solution array and sets up a loop for examining trees based on the topography. If the DNA is linear, the last fragment of completed maps is compared to the root. If a match is found, that tree is skipped. The BLDMAP array is then cleared and, if the topography is circular, the LAST array is also cleared. Next, the digest data is copied from F into the fragment stack, STACK. If the topography of the DNA is circular the last node pointer for enzyme #1 is set to node level 1 (because this will be the arbitrary starting point for the circular algorithm), otherwise the last node pointer for every enzyme is set to the beginning of the linear DNA. A loop is then set up to examine node levels within the tree. The program next begins looking for open nodes (i.e. nodes that have not yet been

assigned a branch). When such a node is found, a temporary copy of the STACK and LAST for the path corresponding to that node are created. Branches (selected from the temporary stack of remaining fragments) are generated at that node, unless it is the first branch in the tree (root) in which case it is immediately stored. For each branch, each enzyme is tested as a terminating node. A loop sums all the fragments in the path back to the last occurance of that enzyme (or the beginning of the linear DNA), unless it is a circular DNA molecule with a previously unassigned node in which case it is immediately considered a possible solution and stored. If the sum is found in the single digest corresponding to that enzyme that fragment and node are stored as a solution for that level, otherwise the next enzyme is checked. If a solution (fragment + node) is to be stored, BLDMAP is checked to see if a branch has already been assigned to this path. If true, the path (minus the assigned branch) and LAST are first copied into free memory (found by searching BLDMAP). Next, a flag is set to indicate that a branch has been stored at the current open node, the fragment and node are added to the solution, the LAST pointer for the enzyme is updated, and the STACK is recopied from TMPSTK into free memory (if necessary). The fragment assigned is then removed, or "popped", from the n-digest STACK, and the sum of fragments is removed from the single digest STACK in which it was found. Once all enzymes and branches have been tried, the flag is tested to see if a new branch has been assigned to the open node. If not, the BLDMAP, LAST, and STACK for that node are erased (which amounts to terminating or pruning that node) so that the memory can be reclaimed. Once all open nodes have been examined, the next node level is considered. Once all node levels have been considered the tree is completed. If the topography is circular, the wrap-around fragments are first checked against the single digest stacks for each path and the path cleared if a discrepancy is found, otherwise the completed

paths are copied into MAPSOL. If the topography is linear or the topography is circular and no solutions are found, the next tree is examined. After all trees are examined the subprogram prints out the number of non-degenerate solutions calculated and returns control to the main program.

Samples of the program's execution for the linear and circular examples previously given are found in Appendices C and D.

The only major difference between the BASIC program, RESTRC.BAS (Appendix A), and the FORTRAN program, RESTRIC.FOR (Appendix B), is in regard to the allocation of memory to array variables. The BASIC program does not dimension array variables until it has obtained various parameters of the data. This allows for optimization of scarce memory available to the microcomputer. Before a new set of data is entered, the variables are erased so that the memory can be reallocated. Because FORTRAN does not allow dynamic reallocation of variable memory, the array variables are set to an arbitrary large size (taking advantage of the much larger memory available to the VAX computer). The maximum number of enzymes allowed was set at 20, the maximum number of fragments/digest was set at 20, and the maximum number of paths was set at 100. These values can be changed by simply changing the dimension statements in the program.

DISCUSSION

The algorithms have proved, in practice, to be very quick and efficient. The time needed to solve maps of medium size (about 10 n-digest fragments, 3 enzymes) by computer was well under 5 minutes on the slower microcomputer and less than a second on the much faster VAX. The time needed to generate solutions does not appear to be so much a function of size, but rather one of complexity. Complexity involves the number of enzymes used (since each branch is tested with each enzyme), the number of possible solutions (because each solution represents a path through the entire structure), and a large number (>2) of adjacent sites for one enzyme (since these can be permutated and each permutation will result in a different solution). Therefore, the fastest solution will be found for those maps using only two enzymes and having only one unique solution (sizes being equal).

Multiple solutions often present a problem. If certain information not available to the computer, such as knowing a terminal fragment, is known this may help to eliminate some of the solutions. Other ways of eliminating multiple solutions are to include more enzymes so that more complex and unique data results or to decrease the error range. If the error range is too large, fragments of approximately equal size become indistinquishable and if present in different digests will result in multiple solutions. Also if some fragments are smaller than the error range of larger fragments, these may be incorrectly placed. Obviously a large error should be avoided. However, if the error value is too small there is a chance that correct solutions will be discarded or that no solution will be found. Therefore, the error value should not be reduced to eliminate solutions unless this reduction is justified by an actual reduction in the error of the fragment measurements.

Incorrect maps will also result if very small fragments in both a single and the n-digest run off the gel during data aquisition and are not detected when the total digest lengths are tested. This may be prevented by using a gel with a wide separation range that will detect both very large and very small fragments.

The restriction site mapping method and computer programs described here provide a rapid and accurate tool for generating cleavage maps from as many enzymes as desired. As long as sources of error in fragment measurement are minimized, the computer should be able to generate at least one solution. By using appropriate combinations of restriction enzymes, a unique solution can be derived for any linear or circular DNA molecule.

APPENDIX A

```
REM  ------------------------------------------------------------
REM
REM       RESTRC   AUTOMATIC RESTRICTION SITE MAPPING PROGRAM
REM                Version 1 a  Microsoft BASIC-80 CP/M
REM
REM                                by
REM
REM                          Norbert E  Baumgartner
REM                University of Illinois at Urbana-Champaign
REM                     Department of Biochemistry
REM                          March, 1984
REM
REM               COPYRIGHT (C) 1984 BY NORBERT E  BAUMGARTNER
REM                          ALL RIGHTS RESERVED
REM
REM  ------------------------------------------------------------
REM
REM                P R O G R A M    V A R I A B L E S
REM
REM        Variable Name              Function
REM
REM  AVDIG                      Average digest length
REM  BRANCH                     Pointer to branches at current open node
REM  C                          Program command variable
REM  ENUM                       Enzyme number (1 - N)
REM  ER                         Error in fragment measurement
REM  FLAG                       Flag variable
REM  I,J,K,L,M                  Looping variables
REM  MAPPNT                     Pointer to map in use
REM  MXITM                      Maximum number of storage items
REM  MXSTOR                     Maximum storage used so far
REM  N                          Number of restriction enzymes
REM  NODLVL                     Pointer to node level in tree
REM  OPNNOD                     Pointer to search for open nodes
REM  POPDIG                     Digest to pop from
REM  POPFRG                     Fragment to pop from stack
REM  SOLPNT                     Pointer to next open solution storage
REM  SUM                        Sum of fragments
REM  TC,T1,T2                   Temporary storage variables
REM  TRACE                      Pointer for tracing back in tree
REM  TYPE                       Specifies source DNA type
REM                                  TYPE=0   Circular
REM                                  TYPE=1   Linear
REM  ELEMAP(a,b,c)              Storage for maps under construction
REM                                  a = map number:
REM                                        a = 1 to MXITM
REM                                  b = fragment number
REM                                  c = fragment/node code
REM                                        c=1  fragment length
REM                                        c=2  node (enzyme site)
REM  F(n,b)                     List of digest fragments
REM                             Note
REM                                  F(n,0) = number of fragments in
REM                                           digest #n
REM                                  n = digest number
REM                                        n = 0   n-digest (all enzymes)
REM                                        n = 1-N single digests of
REM                                                enzymes 1 through N
REM                                  b = fragment number
REM  LAST(a,d)                  Node level last assigned for each enzyme
REM                                  a = map number
REM                                  d = enzyme number
REM                                        d = 1 through N
REM  MAPSOL(a,b,c)              Solution storage matrix
REM                                  a = map number
REM                                  b = fragment number
REM                                  c = fragment/node code
REM  ROOT(a,c)                  Root fragment & node for linear maps
REM                                  a = root number
REM                                  c = fragment/node code
REM  STACK(a,n,b)               Stack of unassigned fragments
REM                                  a = map number
REM                                  n = digest number
REM                                  b = fragment number
REM  T(n)                       Total digest lengths
REM                                  n = digest number
REM  TMPLST(d)                  Temporary list of node level last
REM                             assigned for each enzyme
REM                                  d = enzyme number
REM  TMPSTK(n,b)                Temporary stack of unassigned fragments
```

```
1800 REM                                   n = digest number
1810 REM                                   b = fragment number
1820 REM  A$                            Answer character for input
1830 REM  P$                            Plural character for output
1840 REM  R$()                          List of restriction enzyme names
1850 REM                                   d = enzyme number
1860 REM  TMP$                          Temporary string variable
1870 REM  T$                            Title line for solution display
1880 REM  ------------------------------------------------------------------------
1890 REM
1900 REM
1910 REM            **** P R O G R A M   C O N T R O L ****
1920 REM
1930 PRINT CHR$(12)
1940 PRINT TAB([illegible]) "RESTRICTION SITE MAPPING PROGRAM"
1950 PRINT CHR$(10)
1960 PRINT "Select program function"
1970 PRINT
1980 PRINT "          1 = Create new data"
1990 PRINT "          2 = Calculate restriction site maps from
2000 PRINT "              entered data"
2010 PRINT "          3 = Print results of calculations"
2020 PRINT "          4 = Exit program"
2030 PRINT
2040 INPUT "Command";C
2050 IF C<1 OR C>4 THEN 2040
2060 ON C GOSUB [illegible] 3470 6240
2070 GOTO 1930
2080 REM
2090 REM            **** D A T A   E N T R Y ****
2100 REM
2110 PRINT
2120 PRINT "DATA ENTRY"
2130 PRINT
2140 PRINT "Topography of source DNA"
2150 PRINT "       1 = CIRCULAR (Plasmid)"
2160 PRINT "       2 = LINEAR"
2170 INPUT "Type (1 or 2)";TYPE
2180 IF TYPE<1 OR TYPE>2 THEN 2170 ELSE TYPE=TYPE-1
2190 PRINT
2200 IF N>0 THEN FLAG=1 ELSE FLAG=0
2210 INPUT "Number of restriction enzymes used";N
2220 IF FLAG=0 THEN 2240
2230 ERASE R$,F
2240 DIM R$(N),F(N,20)
2250 FOR J=1 TO N
2260     PRINT
2270     PRINT "Restriction enzyme #";J;
2280     INPUT R$(J)
2290     INPUT "Number of fragments in digest";F(J,0)
2300     PRINT "Enter fragments sizes"
2310     FOR K=1 TO F(J,0)
2320         PRINT "#";K;
2330         INPUT F(J,K)
2340     NEXT K
2350 NEXT J
2360 PRINT
2370 PRINT "n-digest = ";
2380 FOR J=1 TO N
2390     PRINT R$(J);
2400     IF J=N THEN 2420
2410     PRINT "/";
2420 NEXT J
2430 R$(0)="n"
2440 PRINT
2450 INPUT "Number of fragments obtained";F(0,0)
2460 PRINT "Enter fragment sizes"
2470 FOR J=1 TO F(0,0)
2480     PRINT "#";J;
2490     INPUT F(0,J)
2500 NEXT J
2510 REM BUBBLE SORT FRAGMENT LISTS (SMALLEST TO LARGEST)
2520 FOR I=0 TO N
2530     FOR J=1 TO F(I,0)-1
2540         FOR K=F(I,0)-1 TO J STEP -1
2550             IF F(I,K)<=F(I,K+1) THEN 2590
2560             TMP=F(I,K)
2570             F(I,K)=F(I,K+1)
2580             F(I,K+1)=TMP
2590         NEXT K
```

```
        NEXT J
    NEXT I
REM SUM DIGESTS
PRINT
PRINT " Digest";TAB(15);" Fragment list";TAB(40);" Total length"
FOR J=0 TO N
    T(J)=0
    PRINT R$(J);TAB(15);
    FOR K=1 TO F(J,0)
        PRINT F(J,K);
        T(J)=T(J)+F(J,K)
    NEXT K
    PRINT TAB(45);T(J)
NEXT J
REM TEST FOR MISSING FRAGMENTS
T1=0
FOR I=1 TO N
    T1=T1+F(I,0)
NEXT I
IF T1=F(0,0)+TYPE*(N-1) THEN 3150
T1=F(0,0)+TYPE*(N-1)-T1
IF ABS(T1)>1 THEN P$="s" ELSE P$=""
PRINT
PRINT "DATA MISSING"
PRINT
PRINT "Fragment analysis indicates";ABS(T1);"missing
PRINT "fragment";P$;"    Missing fragment";P$;"occurs
PRINT "in"
IF T1<0 THEN PRINT "n-digest" ELSE PRINT "single enzyme digest"
PRINT
PRINT "Data available to correct missing fragment";P$
INPUT "Y or N";A$
IF A$="Y" THEN 2980
IF A$="N" THEN 2910
PRINT
PRINT "Insufficient data - unable to continue"
GOTO 5040
PRINT
PRINT "For each of the missing fragments, enter the
PRINT "digest name (enzyme name for single digests or n
PRINT "for n-digest) and the missing fragment size s
PRINT "separated by a comma"
PRINT
FOR I=1 TO ABS(T1)
    PRINT "#";I;
    INPUT TMP$(I)
    FOR J=0 TO N
        IF R$(J)=TMP$(I) THEN 3120
    NEXT J
    PRINT "Unrecognized digest name - reenter"
    GOTO 3050
    F(J,0)=F(J,0)+1
    F(J,F(J,0))=T2
NEXT I
GOTO 2510
REM  FIND AVERAGE DIGEST LENGTH, TEST FOR DIGESTS
REM       OUTSIDE ERROR RANGE
AVG=0
FOR I=1 TO N
    AVG=AVG+T(I)
NEXT I
AVG=AVG/(N+1)
PRINT
PRINT "Relative error in fragment size measurement
INPUT "(% total fragment length)";ER
ER=ER/100
REM SEARCH FOR DIGESTS OUTSIDE ERROR RANGE
PRINT
PRINT "Length of source DNA with ";A$;" assumed to be
PRINT AVG;"BASE PAIRS (";ER*AVG;")"
FOR J=0 TO N
    IF T(J)<(AVG-ER*AVG) OR T(J)>(AVG+ER*AVG) THEN 3360
    PRINT
    PRINT "Length of ";R$(J);" digest is outside
    PRINT "error range.  Select new error value or abort
    PRINT "program and reevaluate data"
    GOTO 4130
NEXT J
PRINT
```

```
3400 INPUT "Title line for display" ; T$
3410 PRINT
3420 PRINT "DATA ENTRY COMPLETE"
3430 RETURN
3440 REM
3450 REM            **** O U T P U T   S E C T I O N ****
3460 REM
3470 PRINT CHR$(10);CHR$(10);CHR$(10);CHR$(10)
3480 PRINT "NOTE    The following are a... possible restriction
3490 PRINT          site maps from the given data.  Letters
3500 PRINT          indicate restriction sites (see KEY) num-
3510 PRINT          bers indicate distance between sites
3520 PRINT
3530 PRINT "DATA SUMMARY"
3540 PRINT
3550 PRINT "Source DNA topography = ";
3560 IF TYPE=0 THEN PRINT "CIRCULAR" ELSE PRINT "LINEAR"
3570 PRINT
3580 PRINT "Digest";TAB(20);"Fragment size";TAB(40);"Total length"
3590 FOR I=0 TO N
3600     PRINT R$(I);TAB(10);
3610     FOR J=1 TO F(I,0)
3620         PRINT F(I,J);
3630     NEXT J
3640     PRINT TAB(44);T(I)
3650 NEXT I
3660 PRINT
3670 PRINT "Error in fragment measurement = ";ER*100;"%"
3680 PRINT CHR$(10);CHR$(10);
3690 PRINT TAB(40-LEN(T$)/2);T$
3700 PRINT CHR$(10);CHR$(10)
3710 PRINT "KEY"
3720 PRINT
3730 FOR K=1 TO N
3740     PRINT CHR$(64+K);"  =  ";R$(K)
3750 NEXT K
3760 FOR I=1 TO SOLPNT-1
3770 PRINT CHR$(10);CHR$(10);
3780     PRINT "SOLUTION #";I;":"
3790     PRINT
3800     IF TYPE=1 THEN PRINT "|" ELSE PRINT "A"
3810     FOR J=1 TO F(0,0)
3820         PRINT "-";MAPSOL(I,0,J);"-";
3830         IF J=F(0,0) THEN 3850
3840         PRINT CHR$(64+MAPSOL(I,0,J));
3850     NEXT J
3860     IF TYPE=1 THEN PRINT "|" ELSE PRINT "A"
3870 NEXT I
3880 RETURN
3890 REM
3900 REM            **** M A P   G E N E R A T O R ****
3910 REM
3920 MXITM=F(0,0)+4
3930 IF SOLPNT>1 THEN 3950
3940 ERASE MAPSOL
3950 DIM BLDMAP(MXITM,F(0,0)),LAST(MXITM,N),MAPSOL(MXITM,F(0,0)),Z
3960 DIM ROOT(F(0,0),2),STACK(MXITM,N,F(0,0)),TMPLST(N),TMPSTK(N,F(0,0))
3970 IF TYPE=0 THEN 4130
3980 REM  FIND ALL POSSIBLE ROOTS
3990 REM  ROOT = DOUBLE DIGEST FRAGMENT THAT ALSO
4000 REM          APPEARS IN SINGLE DIGEST
4010 T0=1
4020 FOR J=1 TO F(0,0)
4030    FOR K=1 TO N
4040        FOR L=1 TO F(K,0)
4050            IF F(0,J)<(F(K,L)-F(K,L)*ER) OR F(0,J)>(F(K,L)+F(K,L)*ER) THEN 4090
4060            ROOT(T0,1)=F(K,L)
4070            ROOT(T0,2)=K
4080            T0=T0+1
4090        NEXT L
4100    NEXT K
4110 NEXT J
4120 REM  SET POINTER FOR FINAL SOLUTION SET
4130 SOLPNT=1
4140 REM  START EXAMINING TREES
4150 FOR TREE=1 TO ABS(T0-1)*(TYPE=1)+F(0,0)*(TYPE=0)
4160     IF TYPE=0 THEN 4230
4170     REM  CHECK LAST FRAGMENT OF COMPLETED MAPS FOR
4180     REM    DUPLICATE SOLUTION
4190     FOR J=1 TO SOLPNT-1
```

```
4100        IF MAP$(J,F(0,0),1)=ROOT(TREE,1) THEN 5080
4110    NEXT J
4120    REM  CLEAR TEMP  SOLUTION MATRIX
4130    FOR J=1 TO MXITM
4140        IF BLDMAP(J,1,1)=0 THEN 4280
4150        FOR K=1 TO F(0,0)
4160            BLDMAP(J,K,1)=0
4170        NEXT K
4180    NEXT J
4190    MXSTOR=1
4200    IF TYPE=1 THEN 4360
4210    REM  CLEAR LAST NODE MATRIX
4220    FOR J=1 TO N
4230        LAST(1,J)=0
4240    NEXT J
4250    REM  COPY DIGEST FRAGMENTS INTO FRAGMENT STACK
4260    FOR J=0 TO N
4270        FOR K=1 TO F(J,0)
4280            STACK(1,J,K)=F(J,K)
4290        NEXT K
4300    NEXT J
4310    IF TYPE=1 THEN 4460
4320    REM  SET LAST NODE POINTER FOR ENZYME 1 TO NODELEVEL 1
4330    LAST(1,1)=1
4340    GOTO 4530
4350    REM  SET LAST NODE FOR EACH ENZYME TO BOTTOM OF TREE
4360    FOR J=1 TO N
4370        LAST(1,J)=1
4380    NEXT J
4390    REM  SET PARAMETERS FOR STORING ROOT
4500    SUM=ROOT(TREE,1)
4510    ENZM=ROOT(TREE,2)
4520    REM  START EXAMINING NODE LEVELS
4530    FOR NODLVL=1 TO F(0,0)
4540        REM  SEARCH FOR OPEN NODES AT CURRENT LEVEL
4550        FOR OPNNOD=1 TO MXSTOR
4560            IF NODLVL>1 AND (BLDMAP(OPNNOD,1,1)=0 OR BLDMAP(OPNNOD,NODLVL-1,1)>0) THEN 5330
4570            FLAG=0
4580            REM  CREATE TEMPORARY STACK AND LAST FOR THIS NODE
4590            FOR J=0 TO N
4600                FOR K=1 TO F(J,0)
4610                    TMPSTK(J,K)=STACK(OPNNOD,J,K)
4620                NEXT K
4630            NEXT J
4640            FOR J=1 TO N
4650                TMPLST(J)=LAST(OPNNOD,J)
4660            NEXT J
4670            REM  CREATE BRANCHES AT CURRENT OPEN NODE
4680            FOR BRANCH=1 TO F(0,0)
4690                IF TMPSTK(0,BRANCH)=0 THEN 5360
4700                IF NODLVL>1 THEN 4750
4710                IF TYPE=1 THEN 4730
4720                IF TMPSTK(0,BRANCH)=F(0,TREE) THEN 4750 ELSE GOTO 5360
4730                IF TMPSTK(0,BRANCH)=ROOT(TREE,1) THEN 4890 ELSE GOTO 5360
4740                REM  CONSIDER EACH ENZYME AS SOLUTION
4750                FOR ENZM=1 TO N
4760                    REM  SUM BRANCHES BACK TO LAST NODE FOR EACH ENZYME
4770                    SUM=TMPSTK(0,BRANCH)
4780                    IF TYPE=1 THEN 4800
4790                    IF TMPLST(ENZM)=0 THEN 4880
4810                    FOR TRACE=NODLVL-1 TO TMPLST(ENZM) STEP -1
4820                        SUM=SUM+BLDMAP(OPNNOD,TRACE,1)
4830                    NEXT TRACE
4840                    REM  CHECK SUM AGAINST SINGLE DIGEST STACK
4850                    FOR J=1 TO F(ENZM,0)
4860                        T1=TMPSTK(ENZM,J)
4865                        IF SUM<(T1-T1*ER) OR SUM>(T1+T1*ER) THEN 5340
4870                        REM  STORE SOLUTION & POP OFF STACKS
4880                        IF BLDMAP(OPNNOD,NODLVL,1)>0 THEN 4920
4895                        MAPPNT=OPNNOD
4900                        GOTO 5120
4910                        REM  RECOPY CURRENT PATH INTO FREE MEMORY
4920                        FOR MAPPNT=1 TO MXITM
4930                            IF BLDMAP(MAPPNT,1,1)=0 THEN 5010
4940                        NEXT MAPPNT
4950                        PRINT "MEMORY OVERFLOW ERROR
4960                        PRINT "Current memory allocation =";MXITM
4970                        INPUT "Change allocation to? ";MXITM
4980                        PRINT "Retrying with new allocation
4990                        ERASE BLDMAP,LAST,ROOT,STACK,TMPLST,TMPSTK
```

```
                        GOTO 5930
                        FOR L=1 TO 2
                            FOR M=1 TO NODLVL-1
                                BLDMAP(MAPPNT,M,L)=BLDMAP(OPNNOD,M,L)
                            NEXT M
                        NEXT L
                        FOR L=1 TO N
                            LAST(MAPPNT,L)=TMPLST(L)
                        NEXT L
                        IF MAPPNT<=MXSTOR THEN 5120
                        MXSTOR=MAPPNT
                        REM  STORE & UPDATE
                        FLAG=1
                        BLDMAP(MAPPNT,NODLVL,1)=TMPSTK(0,BRANCH)
                        BLDMAP(MAPPNT,NODLVL,2)=ENDM
                        IF NODLVL=P(0,0) THEN 5530
                        LAST(MAPPNT,ENDM)=NODLVL+1
                        FOR K=0 TO N
                            FOR L=1 TO P(K,0)
                                STACK(MAPPNT,K,L)=TMPSTK(K,L)
                            NEXT L
                        NEXT K
                        REM  POP N-DIGEST FRAGMENT
                        POPDIG=0
                        POPFRG=TMPSTK(0,BRANCH)
                        GOSUB 5950
                        IF TYPE=1 THEN 5290
                        IF TMPLST(ENDM)=0 THEN 5350
                        REM  POP SINGLE DIGEST FRAGMENT
                        POPDIG=ENDM
                        POPFRG=SUM
                        GOSUB 5950
                        IF NODLVL=1 THEN 5540
                        GOTO 5350
                    NEXT J
                NEXT ENDM
            NEXT BRANCH
            REM  IF NEW BRANCH ASSIGNED, LEAVE NODE OPEN
            REM  ELSE CLEAR PATH AND STACKS FOR CURRENT NODE
            IF FLAG=1 THEN 5530
            FOR J=1 TO 2
                FOR K=1 TO NODLVL
                    BLDMAP(OPNNOD,K,J)=0
                NEXT K
            NEXT J
            FOR J=1 TO N
                LAST(OPNNOD,J)=0
            NEXT J
            FOR J=0 TO N
                FOR K=1 TO P(J,0)
                    STACK(OPNNOD,J,K)=0
                NEXT K
            NEXT J
        NEXT OPNNOD
    NEXT NODLVL
    IF TYPE=1 THEN 5770
    REM  CHECK WRAP-AROUND FRAGMENTS AGAINST SINGLE DIGEST STACKS
    FOR J=1 TO MXITM
        IF BLDMAP(J,1,1)=0 THEN 5750
        FOR ENDM=2 TO N
            SUM=0
            FOR K=1 TO P(0,0)
                SUM=SUM+BLDMAP(J,K,1)
                IF BLDMAP(J,K,2)=ENDM THEN 5650
            NEXT K
            FOR K=P(0,0) TO 1 STEP -1
                SUM=SUM+BLDMAP(J,K,1)
                IF BLDMAP(J,K-1,2)=ENDM THEN 5690
            NEXT K
            FOR L=1 TO P(ENDM,0)
                T1=STACK(J,ENDM,L)
                IF SUM>=T1*(1-ER) AND SUM<=T1*(1+ER) THEN 5740
            NEXT L
            BLDMAP(J,1,1)=0
        NEXT ENDM
    NEXT J
    REM  COPY COMPLETED MAPS INTO FINAL SOLUTION SET
    FOR J=1 TO MXITM
        IF BLDMAP(J,1,1)=0 THEN 5850
        FOR K=1 TO 2
```

```
5800            FOR L=1 TO F(0,0)
5810                MAPSOL(SOLPNT,L,K)=BLDMAP(J,L,K)
5820            NEXT L
5830        NEXT K
5840        SOLPNT=SOLPNT+1
5850    NEXT J
5860    IF TYPE=1 THEN 5880
5870    IF SOLPNT>1 THEN 5890
5880 NEXT TREE
5890 PRINT CHR$(10);"Number of non-degenerate solutions calculated =";
5900 PRINT SOLPNT-1;CHR$(10)
5910 ERASE BLDMAP,LAST,ROOT,STACK,TMPLST,TMPSTK
5920 RETURN
5930 REM            SUBROUTINES
5940 REM  POP FRAGMENT OFF STACK
5950 FOR J=1 TO F(POPDIG,0)
5960    IF POPFRG(J)<>STACK(MAPPNT,POPDIG,J) THEN 5990
5970    STACK(MAPPNT,POPDIG,J)=0
5980    GOTO 6000
5990 NEXT J
6000 RETURN
6010 REM
6020 REM            **** E N D   P R O G R A M ****
6030 REM
6040 PRINT CHR$(10);"DONE"
```

APPENDIX B

```
C     ------------------------------------------------------------------

C           RESTRIC  AUTOMATIC RESTRICTION SITE MAPPING PROGRAM
C                    Version 1.0  VAX-11 FORTRAN-77 VAX/VMS

C                                      by

C                                Norbert E. Baumgartner
C                          University of Illinois at Urbana-Champaign
C                                Department of Biochemistry
C                                      April, 1984

C                       COPYRIGHT (C) 1984 BY NORBERT E. BAUMGARTNER
C                                  ALL RIGHTS RESERVED

C     ------------------------------------------------------------------

C                    P R O G R A M    V A R I A B L E S

C        Variable Name                  Function

C     AVDIG                       Average digest length
C     BRANCH                      Pointer to branches at current open node
C     C                           Program command variable
C     ENZM                        Enzyme number
C     ER                          Error in fragment measurement
C     FLAG                        Flag variable
C     I,J,K,L,M                   Looping variables
C     MAPPNT                      Pointer to map in use
C     MXITM                       Maximum number of storage items
C     MXSTOR                      Maximum storage used so far
C     N                           Number of restriction enzymes
C     NODLVL                      Pointer to node level in tree
C     OPNNOD                      Pointer to search for open nodes
C     POPDIG                      Digest to pop from
C     POPFRG                      Fragment to pop from stack
C     SOLPNT                      Pointer to next open solution storage
C     SUM                         Sum of fragments
C     T0,T1,T2                    Temporary storage variables
C     TRACE                       Pointer for tracing back in tree
C     TYPE                        Specifies source DNA type
C                                      TYPE=0   Circular
C                                      TYPE=1   Linear
C     BLDMAP(a,b,c)               Storage for maps under construction
C                                      a = map number
C                                           a = 1 to MXITM
C                                      b = fragment number
C                                      c = fragment/node code
C                                           c=1  fragment length
C                                           c=2  node (enzyme site)
C     F(n,b)                      List of digest fragments
C                                 Note
C                                      F(n,0) = number of fragments in
C                                               digest n
C                                      n = digest number
C                                           n = 0   n-digest (all enzymes)
C                                           n = 1-N single digests of
C                                                   enzymes 1 through N
C                                      b = fragment number
C     LAST(a,d)                   Node level last assigned for each enzyme
C                                      a = map number
C                                      d = enzyme number
C                                           d = 1 through N
C     MAPSOL(a,b,c)               Solution storage matrix
C                                      a = map number
C                                      b = fragment number
C                                      c = fragment/node code
C     ROOT(e,c)                   Root fragment & node for linear maps
C                                      e = root number
C                                      c = fragment/node code
C     STACK(a,n,b)                Stack of unassigned fragments
C                                      a = map number
C                                      n = digest number
C                                      b = fragment number
C     T(n)                        Total digest lengths
C                                      n = digest number
C     TMPLST(d)                   Temporary list of node level last
C                                 assigned for each enzyme
C                                      d = enzyme number
```

```
C   TMPSTK(n,b)              Temporary stack of unassigned fragments
C                                 n = digest number
C                                 b = fragment number
C   A                       Answer character for input
C   P                       Plural character for output
C   R(d)                    List of restriction enzyme names
C                                 d = enzyme number
C   TMP                       Temporary string variable
C   TL                      Title line for solution display
C   ------------------------------------------------------------

C              **** P R O G R A M   C O N T R O L ****

      PROGRAM  RESTRIC

      COMMON ER,MXITM,N,SOLFNT,TYPE,F(0:20,0:20),MAPSOL(100,20,2),
     1       T(0:20),R,TL
      INTEGER C,MXITM,N,SOLFNT,TYPE
      REAL*4  ER,F,MAPSOL,T
      CHARACTER R(0:20)*10
      CHARACTER*72 TL

      MXITM=100
      PRINT 2040
2040  FORMAT (///T10,'  RESTRICTION SITE MAPPING PROGRAM'//)
2050  PRINT *,'Select program function'
      PRINT *,'       1 = Create new data'
      PRINT *,'       2 = Calculate restriction site maps from
      PRINT *,'           entered data'
      PRINT *,'       3 = Print results of calculations'
      PRINT *,'       4 = Exit program'
2140  PRINT 2141
2141  FORMAT (/'$Command? ')
      ACCEPT *,C
      IF ( C .LT. 1 .OR. C .GT. 4) GOTO 2140
      GOTO (2160,2161,2162,2163), C
2160  CALL DATENT
      GOTO 2140
2161  CALL MAPGEN
      GOTO 2140
2162  CALL DATOUT
      GOTO 2140
2163  PRINT *,'DONE'
      STOP
      END

C             **** D A T A   E N T R Y ****

      SUBROUTINE DATENT

      COMMON ER,MXITM,N,SOLFNT,TYPE,F(0:20,0:20),MAPSOL(100,20,2),
     1       T(0:20),R,TL
      INTEGER I,J,K,N,TYPE
      REAL*4  AVDIG,ER,T1,F,T
      CHARACTER*1 A,P
      CHARACTER R(0:20)*10,TMP*10
      CHARACTER*72 TL
      PRINT *,' '
      PRINT *,'Topography of source DNA:'
      PRINT *,'       1 = CIRCULAR (Plasmid)'
      PRINT *,'       2 = LINEAR'
2270  PRINT 2271
2271  FORMAT ('$Select 1 or 2 ? ')
      ACCEPT *,TYPE
      IF (TYPE .LT. 1 .OR. TYPE .GT. 2) THEN
         GOTO 2270
      ELSE
         TYPE=TYPE-1
      END IF
      PRINT 2310
2310  FORMAT (/'$Number of restriction enzymes used? ')
      ACCEPT *,N

      DO J=1,N
         PRINT 2370,J
2370     FORMAT (/'$Restriction enzyme #',I2,' ? ')
         ACCEPT 2380,R(J)
2380     FORMAT (A)
         PRINT 2390
2390     FORMAT ('$Number of fragments in digest? ')
```

```
            ACCEPT *,F(J,0)
            PRINT *, Enter fragment sizes.
            DO K=1,F(J,0)
               PRINT 2420,K
2420           FORMAT ( 2H ,I2, ' ' )
               ACCEPT *,F(J,K)
            END DO
         END DO
         PRINT 2470
2470     FORMAT (//'In-digest = ')
         DO J=1,N
            PRINT 2490,R(J)
2490        FORMAT ( ' ',A,$)
            IF (J .EQ. N) GOTO 2520
            PRINT 2510
2510        FORMAT ( ' +',$)
2520     END DO
         R(0)='n'
         PRINT 2550
2550     FORMAT (/ 'Number of fragments obtained' )
         ACCEPT *,F(0,0)
         MXITM=4*F(0,0)
         PRINT *,'Enter fragment sizes '
         DO J=1,F(0,0)
            PRINT 2420,J
            ACCEPT *,F(0,J)
         END DO

C    BUBBLE SORT FRAGMENT LISTS - SMALLEST TO LARGEST

2610     DO I=0,N
            DO J=1,F(I,0)-1
               DO K=F(I,0)-1,J,-1
                  IF (F(I,K) .LE. F(I,K+1)) GOTO 2690
                  T1=F(I,K)
                  F(I,K)=F(I,K+1)
                  F(I,K+1)=T1
2690           END DO
            END DO
         END DO

C    SUM DIGESTS

         PRINT 2740
2740     FORMAT (//' Digest',T21,'Fragment list',T45,'Total length',
         DO J=0,N
            T(J)=0
            PRINT 2770,R(J)
2770        FORMAT ( ' ',A,X)
            DO K=1,F(J,0)
               PRINT 2790,F(J,K)
2790           FORMAT ('+ ',F8.2,X,$)
               T(J)=T(J)+F(J,K)
            END DO
            DO I=1,54-(F(J,0)*8)
               PRINT 2800
2800           FORMAT ('+ ',$)
            END DO
            PRINT 2801
2801        FORMAT ( ' = ',$)
            PRINT 2820,T(J)
2820        FORMAT ( ' ',F8.2/)
         END DO

C    TEST FOR MISSING FRAGMENTS

         T1=0
         DO I=1,N
            T1=T1+F(I,0)
         END DO
         IF (T1 .EQ. (F(0,0)+TYPE*(N-1.))) GOTO 3290
         T1=F(0,0)+TYPE*(N-1)-T1
         IF (ABS(T1) .GT. 1) THEN
            P='s'
         ELSE
            P=
         END IF
         PRINT *,' DATA MISSING'
         I=ABS(T1)
         PRINT 2950,I
```

```
2950    FORMAT    Fragment analysis indicates ,I2, missing ,
        PRINT 2951,P,F
2951    FORMAT (  fragment ,A,    Missing fragment ,A,  occurs ,
        IF (T1 .LT. 0) THEN
           PRINT *,'in n-digest.'
        ELSE
           PRINT *,' in single enzyme digest
        END IF
3010    PRINT 3011,F
3011    FORMAT (  Data available to correct missing fragment ,A,
        PRINT 3012
3012    FORMAT (  Y(es or N,) ')
        ACCEPT 3020,A
3020    FORMAT (A,
        IF (A  EQ  'Y') GOTO 3030
        IF (A  NE  'N') GOTO 3010
        PRINT *,' Unable to continue due to insufficient data'
        STOP
3030    PRINT *,' For each of the missing fragments, enter the'
        PRINT *,' digest name (enzyme name for single digests or "n'
        PRINT *,'for n-digest) and the missing fragment size
        PRINT *,'separated by a comma'
        DO I=1,ABS(T1)
3150       PRINT 3420,I
           ACCEPT 3160,TMP,T2
3160       FORMAT (A,F8.3)
           DO J=0,N
              IF (R(J)  EQ  TMP) GOTO 3220
           END DO
           PRINT *,' Unrecognised digest name, reenter'
           GOTO 3150
3220       F(J,0)=F(J,0)+1
           F(J,F(J,0))=T2
        END DO
        GOTO 2620

C   FIND AVERAGE DIGEST LENGTH, TEST FOR DIGESTS OUTSIDE
C        OF ERROR RANGE

3240    AVDIG=0
        DO I=0,N
           AVDIG=AVDIG+T(I)
        END DO
        AVDIG=AVDIG/(N+1)
3340    PRINT *,'Relative error in fragment size measurement'
        PRINT 3360
3360    FORMAT ('0,% total fragment length)'  )
        ACCEPT *,ER
        ER=ER/100
        PRINT *,' Length of source DNA will be assumed to be'
        PRINT 3410,AVDIG,ER*AVDIG
3410    FORMAT (F8.2,' +/- ',F8.2,
        DO J=0,N
           IF (T(J)  GE  AVDIG*(1-ER) .AND. T(J)  LE  AVDIG*(1+ER))
     1  GOTO 3490
           PRINT 3450,R(J)
3450       FORMAT (' Length of ',A,  digest is outside error range ')
           PRINT *,'Select a new error value or stop program and'
           PRINT *,' reevaluate data
           GOTO 3340
3490    END DO
        PRINT 3510
3510    FORMAT (//'sTitle line for display' )
        ACCEPT 3520,TL
3520    FORMAT (A)
        PRINT *,' '
        PRINT *,'DATA ENTRY COMPLETE'
        RETURN
        END

C               **** O U T P U T   S E C T I O N ****

        SUBROUTINE DATOUT

        COMMON ER,MXITM,N,SOLFNT,TYPE,F(0:20,0:20),MAPSOL(100,20,2),
     1         T(0:20),R,TL
        INTEGER I,J,K,N,SOLFNT,TYPE
        REAL*4  ER,F,MAPSOL,T
        CHARACTER R(0:20)*10
        CHARACTER*72 TL
```

```
      PRINT 3580
3580  FORMAT (//' NOTE.  The following are all possible restriction'
     1 ,
      PRINT *,'        site maps from the given data.  Letters'
      PRINT *,'        indicate restriction sites (see KEY), num-'
      PRINT *,'        bers indicate distance between sites.'
      PRINT 3640
3640  FORMAT (/' DATA SUMMARY '//)
      PRINT 3660
3660  FORMAT (' Source DNA topography = ')
      IF (TYPE .EQ. 0) THEN
         PRINT *,'CIRCULAR'
      ELSE
         PRINT *,'LINEAR'
      END IF
      PRINT 3700
3700  FORMAT (/' Digest',T21,'Fragment list',T65,'Total length')
      DO I=0,N
         PRINT 3701,R(I)
3701     FORMAT (' ',A,X)
         DO J=1,F(I,0)
            PRINT 3702,F(I,J)
3702        FORMAT ('+',F8.2,X,$)
         END DO
         DO K=1,56-(F(I,0)*8)
            PRINT 3703
3703        FORMAT ('+ ',$)
         END DO
         PRINT 3704
3704     FORMAT ('+ a ',$)
         PRINT 3750,T(I)
3750     FORMAT ('+',F8.2)
      END DO
      PRINT 3780,ER*100
3780  FORMAT (/' Error in fragment measurement = ',F8.2,' %'///)
      DO I=1,(80-LEN(TL))/2
         PRINT 3790
3790     FORMAT ('+ ',$)
      END DO
      PRINT 3800,TL
3800  FORMAT ('+',A)
      PRINT 3810
3810  FORMAT (/' KEY '//)
      DO K=1,N
         I=64+K
         PRINT 3850,CHAR(I),R(K)
3850     FORMAT (' ',A,' = ',A)
      END DO
      DO I=1,SOLPNT-1
         PRINT 3880,I
3880     FORMAT (///' SOLUTION #',I2,'  '//)
         IF (TYPE .EQ. 1) THEN
            PRINT 3910
3910        FORMAT ('$|')
         ELSE
            PRINT 3911
3911        FORMAT ('$A')
         END IF
         DO J=1,F(0,0)
            PRINT 3930,MAPSOL(I,J,1)
3930        FORMAT ('+-',F8.2,'-',$)
            IF (J .EQ. F(0,0)) GOTO 3960
            K=MAPSOL(I,J,2)+64
            PRINT 3950,CHAR(K)
3950        FORMAT ('+',A,$)
3960     END DO
         IF (TYPE .EQ. 1) THEN
            PRINT 3961
3961        FORMAT ('+|')
         ELSE
            PRINT 3962
3962        FORMAT ('+A')
         END IF
      END DO
      RETURN
      END

C              **** M A P   G E N E R A T O R ****
```

```
      SUBROUTINE MAPGEN

      COMMON ER,MXITM,N,SOLPNT,TYPE,F(0:20,0:20),MAPSOL(100,20,2,
     1       T(0:20),R,TL
      INTEGER BRANCH,ENTM,I,J,K,L,M,MAPPNT,MXITM,MXSTOR,N,NODLVL
      INTEGER OPNNOD,SOLPNT,TRACE,TYPE,LAST,TMPLST
      REAL*4  ER,SUM,T0,T1,T2,BLDMAP,F,MAPSOL,ROOT,STACK,TMPSTK
      LOGICAL*1 FLAG

      DIMENSION LAST(100,20),TMPLST(20),BLDMAP(100,20,2)
      DIMENSION ROOT(20,2),STACK(100,0:20,20),TMPSTK(0:20,20)

      IF (TYPE  EQ. 0) GOTO 4240

C   FIND ALL POSSIBLE ROOTS.  ROOT = DOUBLE DIGEST FRAGMENT THAT
C      ALSO APPEARS IN SINGLE DIGEST

      T0=1
      DO J=1,F(0,0)
         DO K=1,N
            DO L=1,F(K,0)
               IF (F(0,J) .LT. (F(K,L)*(1-ER)) .OR. F(0,J) .GT. (F(K,L)
     1 *(1+ER))) GOTO 4200
               ROOT(T0,1)=F(K,L)
               ROOT(T0,2)=K
               T0=T0+1
4200        END DO
         END DO
      END DO

C   SET POINTER FOR FINAL SOLUTION SET

4240  SOLPNT=1

C   START EXAMINING TREES

      DO TREE=1,ABS((T0-1)*(TYPE  EQ  1)+F(0,0)*(TYPE  EQ. 0))
         IF (TYPE  EQ  0) GOTO 4340

C   CHECK LAST FRAGMENT OF COMPLETED MAPS FOR DUPLICATE SOLUTION

         DO J=1,SOLPNT-1
            IF (MAPSOL(J,F(0,0),1)  EQ. ROOT(TREE,1)) GOTO 9990
         END DO

C   CLEAR TEMPORARY SOLUTION MATRIX

4340     DO J=1,MXITM
            IF (BLDMAP(J,1,1) .EQ. 0) GOTO 4390
            DO K=1,F(0,0)
               BLDMAP(J,K,1)=0
            END DO
4390     END DO
         MXSTOR=1
         IF (TYPE .EQ. 1) GOTO 4470

C   CLEAR LAST NODE MATRIX

         DO J=1,N
            LAST(1,J)=0
         END DO

C   COPY DIGEST FRAGMENTS INTO FRAGMENT STACK

4470     DO J=0,N
            DO K=1,F(J,0)
               STACK(1,J,K)=F(J,K)
            END DO
         END DO
         IF (TYPE .EQ. 1) GOTO 4570

C   SET LAST NODE POINTER FOR ENZYME #1 TO NODE LEVEL 1

         LAST(1,1)=1
         GOTO 4444

C   SET LAST NODE FOR EACH ENZYME TO BOTTOM OF TREE

4570     DO J=1,N
```

```
            LAST(I,J)=I
         END DO

C   SET PARAMETERS FOR STORING ROOT

         SUM=ROOT(TREE,I)
         ENZM=ROOT(TREE,I)

C   START EXAMINING NODE LEVELS

4640     DO NODLVL=1,F(0,0)

C   SEARCH FOR OPEN NODES AT CURRENT LEVEL

            DO OPNNOD=1,MXSTOR
               IF ((NODLVL .GT. 1) .AND. ((BLDMAP(OPNNOD,I,1) .EQ. 0.
     1  OR. (BLDMAP(OPNNOD,NODLVL,1) .GT. 0.)) GOTO 4630
               FLAG=.FALSE.

C   CREATE TEMPORARY STACK AND LAST FOR THIS NODE

               DO J=0,N
                  DO K=1,F(J,0)
                     TMPSTK(J,K)=STACK(OPNNOD,J,K)
                  END DO
               END DO
               DO J=1,N
                  TMPLST(J)=LAST(OPNNOD,J)
               END DO

C   CREATE BRANCHES AT CURRENT OPEN NODE

               DO BRANCH=1,F(0,0)
                  IF (TMPSTK(0,BRANCH) .EQ. 0) GOTO 5480
                  IF (NODLVL .GT. 1) GOTO 4870
                  IF (TYPE .EQ. 1) GOTO 4850
                  IF (TMPSTK(0,BRANCH) .EQ. F(0,TREE)) THEN
                     GOTO 4870
                  ELSE
                     GOTO 5480
                  END IF
4850              IF (TMPSTK(0,BRANCH) .EQ. ROOT(TREE,1)) THEN
                     GOTO 5010
                  ELSE
                     GOTO 5480
                  END IF

C   CONSIDER EACH ENZYME AS A SOLUTION

4870              DO ENZM=1,N

C   SUM BRANCHES BACK TO LAST NODE

                     SUM=TMPSTK(0,BRANCH)
                     IF (TYPE .EQ. 1) GOTO 4910
                     IF (TMPLST(ENZM) .EQ. 0) GOTO 5000
4890                 DO TRACE=NODLVL-1,TMPLST(ENZM),-1
                        SUM=SUM+BLDMAP(OPNNOD,TRACE,1)
                     END DO

C   CHECK SUM AGAINST SINGLE DIGEST STACK

                     DO J=1,F(ENZM,0)
                        T1=TMPSTK(ENZM,J)
                        IF ((SUM .LT. T1*(1-ER)) .OR. (SUM .GT.
     1 T1*(1+ER))) GOTO 5460

C   CHECK IF CURRENT PATH ALREADY ASSIGNED AT THIS NODE LEVEL

5000                    IF (BLDMAP(OPNNOD,NODLVL,1) .GT. 0) GOTO 5040
5010                    HAPPNT=OPNNOD
                        GOTO 5140

C   RECOPY CURRENT PATH INTO FREE MEMORY

5040                    DO HAPPNT=1,MXSTR
                           IF (BLDMAP(HAPPNT,1,1) .EQ. 0) GOTO 5100
                        END DO
                        PRINT *, 'MEMORY OVERFLOW'
                        GOTO 5110
```

```
5.30                        DO L=1,3
                               DO M=1 NODLVL-1
                                  BLDMAP(MAPPNT,M ( )=BLDMAP(OPNNOD,M,L)
                               END DO
                            END DO
                            DO L=1,N
                               LAST(MAPPNT,L,=TMPLST (.
                            END DO
                            IF (MAPPNT  LE  MXSTOR) GOTO 5140
                            MXSTOR=MAPPNT

C   STORE AND UPDATE

5140                        FLAG= TRUE.
                            BLDMAP(MAPPNT,NODLVL,1)=TMPSTK(0,BRANCH)
                            BLDMAP(MAPPNT,NODLVL,2)=ENZM
                            IF (NODLVL  EQ  F(0,0)) GOTO 5650
                            LAST(MAPPNT,ENZM)=NODLVL+1
                            DO K=0,1
                               DO L=1,F(K,0)
                                  STACK(MAPPNT,K,L,=TMPSTK(K,L
                               END DO
                            END DO

C   POP N-DIGEST FRAGMENT

                            DO I=1,F(0,0)
                               IF (TMPSTK(0,BRANCH)  NE  STACK(MAPPNT,
     1 0,I)) GOTO 5300
                               STACK(MAPPNT,0,I)=0
                               GOTO 5301
5300                        END DO
5301                        IF (TYPE  EQ  1) GOTO 5410
                            IF (TMPLST(ENZM)  EQ  0) GOTO 5470

C   POP SINGLE DIGEST FRAGMENT

5410                        DO I=1,F(ENZM,0)
                               T1=STACK(MAPPNT,ENZM,I)
                               IF ((SUM  NE  T1*(1-BR))  OR  (SUM  GE  T1*
     1 (1+ER))) GOTO 5310
                               STACK(MAPPNT,ENZM,I)=0
                               GOTO 5311
5310                        END DO
5311                        IF (NODLVL  EQ  1) GOTO 5440
                            GOTO 5470
5440                     DO
5470                  END DO
5480               END DO

C   IF NO BRANCH ASSIGNED, LEAVE NODE OPEN FOR SIGNAL PATH
C      AND STACKS FOR CURRENT NODE

                   IF (FLAG  EQ  TRUE.) GOTO 5650
                   DO J=1,2
                      DO K=1,NODLVL
                         BLDMAP(OPNNOD,K,J)=0
                      END DO
                   END DO
                   DO J=1,2
                      F(J,0)=0
                   END DO
                   DO K=1,3
                      DO L=1,F(K,0)
                         STACK(OPNNOD,J,K,=0
                      END DO
                   END DO
5650            END DO
5660         END DO
             IF (TYPE  EQ  1) GOTO 5880

C   CHECK WRAP-AROUND FRAGMENTS AGAINST SINGLE DIGEST STACKS

             DO J=1,NODLVL
                IF (BLDMAP(J,1,1)  EQ  0) GOTO 5860
                DO ENZM=2,N
                   SUM=0
                   DO K=1,F(0,0)
                      SUM=SUM+BLDMAP(J,K,1)
                      IF (BLDMAP(J,K,2)  EQ  ENZM) GOTO 5770
```

```
                  END DO
5770              DO K=F(0,0),1,-1
                     SUM=SUM+BLDMAP(J,K,1)
                     IF (BLDMAP(J,K-1,1) .EQ. ENZM) GOTO 5810
                  END DO
5810              DO L=1,F(ENZM,0)
                     T1=STACK(J,ENZM,L)
                     IF ((SUM .GE. T1*(1-ER)) .AND. (SUM .LE. T1*(1+ER)))
     1 GOTO 5850
                  END DO
                  BLDMAP(J,1,1)=0
5850           END DO
5860        END DO

C    COPY COMPLETED MAPS INTO FINAL SOLUTION SET

5880        DO J=1,MX1TM
               IF (BLDMAP(J,1,1) .EQ. 0) GOTO 5940
               DO K=1,2
                  DO L=1,F(0,0)
                     MAPSOL(SOLPNT,L,K)=BLDMAP(J,L,K)
                  END DO
               END DO
               SOLPNT=SOLPNT+1
5940        END DO
            IF (TYPE .EQ. 1) GOTO 5990
            IF (SOLPNT .GT. 1) GOTO 6000
5990     END DO
6000     PRINT 6001,SOLPNT-1
6001     FORMAT (/' Number of non-degenerate solutions calculated = ',I2)
6010     RETURN
         END

C                ****  E N D   P R O G R A M  ****
```

```
$
: RUN RESTRIC

          RESTRICTION SITE MAPPING PROGRAM

Select program function
      1 = Create new data
      2 = Calculate restriction site maps from
          entered data
      3 = Print results of calculations
      4 = Exit program

Command? 1

Topography of source DNA
     1 = CIRCULAR (Plasmid)
     2 = LINEAR
Select 1 or 2 ? 2

Number of restriction enzymes used? 2

Restriction enzyme # 1 ? A
Number of fragments in digest? 3
Enter fragment sizes.
# 1 ? 30
# 2 ? 120
# 3 ? 300

Restriction enzyme # 2 ? B
Number of fragments in digest? 3
Enter fragment sizes.
# 1 ? 100
# 2 ? 150
# 3 ? 200

n-digest = A          /B

Number of fragments obtained? 5
Enter fragment sizes
# 1 ? 30
# 2 ? 50
# 3 ? 70
# 4 ? 100
# 5 ? 200

 Digest          Fragment list                                  Total length
A              30.00   50.00   70.00  100.00  200.00             =   450.00

A              30.00  120.00  300.00                             =   450.00

B             100.00  150.00  200.00                             =   450.00

Relative error in fragment size measurement
(% total fragment length)? 1
Length of source DNA will be assumed to be
 450.00 +/-      4.50

Title line for display? Linear DNA Test Data

DATA ENTRY COMPLETE

Command? 3

Number of non-degenerate solutions calculated =  1

Command? 3

NOTE:  The following are all possible restriction
```

site maps from the given data. Letters indicate restriction sites (see KEY), numbers indicate distance between sites.

DATA SUMMARY.

Source DNA topography =
LINEAR

Digest	Fragment list					Total length
B	30.00	50.00	70.00	100.00	200.00	= 450.00
A	30.00	120.00	300.00			= 450.00
B	100.00	150.00	200.00			= 450.00

Error in fragment measurement = 1.00%

Linear DNA Test Data

KEY

A = A
B = B

SOLUTION # 1

-- 30.00-A- 70.00-B- 50.00-A- 100.00-B- 200.00-

Command? q
DONE
FORTRAN STOP

```
$
$ RUN RESTRIC

          RESTRICTION SITE MAPPING PROGRAM

Select program function:
     1 = Create new data
     2 = Calculate restriction site maps from
         entered data
     3 = Print results of calculations
     4 = Exit program

Command? 1

Topography of source DNA:
     1 = CIRCULAR (Plasmid)
     2 = LINEAR
Select 1 or 2 ? 1

Number of restriction enzymes used? 2

Restriction enzyme # 1 ? A
Number of fragments in digest? 3
Enter fragment sizes:
# 1 ? 40
# 2 ? 70
# 3 ? 120

Restriction enzyme # 2 ? B
Number of fragments in digest? 2
Enter fragment sizes:
# 1 ? 120
# 2 ? 130

n-digest = A          /B

Number of fragments obtained? 5
Enter fragment sizes:
# 1 ? 20
# 2 ? 40
# 3 ? 50
# 4 ? 60
# 5 ? 80

 Digest              Fragment list                                  Total length
n              20.00   40.00   50.00   60.00   80.00                =   250.00

A              40.00   70.00  120.00                                =   230.00

B             120.00  130.00                                        =   250.00

Relative error in fragment size measurement
(% total fragment length)? 1
Length of source DNA will be assumed to be
  250.00 +/-      1.50

Title line for display? Circular DNA Test Data

DATA ENTRY COMPLETE

Command? 2

Number of non-degenerate solutions calculated =  1

Command? 3

NOTE:  The following are all possible restriction
       site maps from the given data.  Letters
```

indicate restriction sites (see KEY). numbers indicate distance between sites.

DATA SUMMARY

Source DNA topography =
CIRCULAR

Digest	Fragment list					Total length
A	20 00	40 00	50 00	60 00	80 00	= 250 00
B	60 00	70 00	120 00			= 250 00
AB	120 00	130 00				= 250 00

Error in fragment measurement = 1 00%

Circular DNA Test Data

KEY

A = A
B = B

SOLUTION # 1

A- 20 00-B- 50 00-A- 80 00-B- 40 00-A- 60 00-A

Command: 4
DONE
FORTRAN STOP

REFERENCES

1. Arber, W. (1974) Prog. Nucl. Acids Res. Mol. Biol. 14, 1

2. Smith, H.O. and Nathans, D. (1973) J. Mol. Biol. 81, 419

3. Boyer, H.W. (1971) Ann. Rev. Microbiol. 25, 153

4. Nathans, D. and Smith, H.O. (1975) Ann. Rev. Biochem. 44, 273-293

5. Schleif, R.F. and Wensink, D.C. (1981) Practical Methods in Molecular Biology 114-127, Springer-Verlag, New York

6. Pearson, W.R. (1982) Nuc. Acids Res. 10, 217-227

7. Fitch, W.M., Smith, T.F., and Ralph, W.W. (1983) Gene 22, 19-29

8. Stefik, M. (1978) Artificial Intelligence 11, 85-114

9. Smith, H.O. and Birnstiel, M.L. (1976) Nuc. Acids Res. 3, 2387-2398

10. Sato, S.S., Hutchinson, C.A., and Harris, J.J. (1977) Proc. Natl. Acad. Sci. 74, 542-546

11. Parker, R.C., Watson, R.M., and Vinograd, J. (1977) Proc. Natl. Acad. Sci. 74, 851-855

12. Sutcliffre, G. (1978) Nuc. Acids Res. 5, 2721-2728