INFORMATION PROCESSING 83, R.E.A. Mason (ed.)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1983

ALGORITHMS FOR SOME STRING MATCHING PROBLEMS ARISING IN MOLECULAR GENETICS

Hannu PELTOLA*, Hans SÖDERLUND**, Jorma TARHIO* and Esko UKKONEN*

* Department of Computer Science, University of Helsinki
Tukholmankatu 2, SF-00250 Helsinki 25, Finland

** Recombinant DNA-laboratory, University of Helsinki
Valimotie 7, SF-00380 Helsinki 38, Finland

Analysis of DNA molecules and other biopolymers is a natural application area for computer methods. For example, with current laboratory techniques it is possible to determine the nucleotide order for relatively short fragments of a long DNA molecule while the total order for long molecules must be reconstructed from the fragments. It is almost impossible to solve this combinatorially difficult task without computer assistance. We give for this problem a simple formulation as a string matching problem, and develop efficient algorithms for finding good approximate solutions. The rapid expansion of recombinant DNA technology is making such algorithms increasingly important.

1. INTRODUCTION

The genetic information of any organism is encoded in its DNA. At the beginning of the 1950's Watson and Crick showed that the DNA is a large double-stranded molecule. Each strand consists of a long sequence of building-blocks called nucleotides. Four different nucleotides are alternating in the sequences. The nucleotides are adenosine (abbreviated A), cytosine (=C), guanine (=G) and thymine (=T). The strands are twisted together such that A and T and, respectively, C and G form pairs (fig 1).

```
—//—— A ——— T ——— C ——— G ——— T —//—
      :     :     :     :     :
      :     :     :     :     :
—//—— T ——— A ——— G ——— C ——— A —//—
```

Fig. 1. A schematic fragment of DNA.

The individual carriers of information, the genes, consist of a segment of such a DNA molecule, and their information is encoded into the relative order of the nucleotides. To determine the order for different genes, or to *sequence* them, is therefore a crucial step towards understanding the utilization of genetic information. Recently the discovery of so-called restriction enzymes which allow cleavage of the long molecule into shorter fragments, and revolutionary advances in recombinant DNA technology essentially improved possibilities to sequence often thousands of nucleotides long DNA molecules. Currently there are fast sequencing techniques with which one person can sequence about 10 DNA fragments of length 250-300 per day thus producing a data of total length of 2000 or 3000 characters {1},{2}.

It is almost inevitable to use computers for manipulation and analysis of the rapidly accumulating sequence data, and indeed, several software packages for this purpose have been developed during past 3 or 4 years {3}. Computers are used not only for simple tasks such as for collecting and storing the primary sequence data, but also for more subtle problems such as for assembling long sequences from short fragments and for finding homologies in different DNA's. It even is possible to develop computer programs for overall assistance of a DNA sequencing project. Such a program which is a typical *expert system* in the sense of Feigenbaum {4}, {14} has also been constructed {5}.

In this paper we propose algorithms for some computationally interesting key-problems encountered in computer analysis of DNA sequences when the entire DNA sequence is assembled from the smaller fragments produced experimentally. The problem becomes combinatorially difficult because the locations of the fragments in the long sequence are not known exactly.

Our point of view is that of computer science and algorithms design. To achieve a sufficiently concise mathematical formulation of the problem we will omit almost all biological details.

2. DNA SEQUENCE ASSEMBLING PROBLEM

To motivate the abstract form of the problem to be considered, we first sketch some main steps of a laboratory procedure for DNA sequencing. Restriction enzymes are the central tools. Each enzyme has a characteristic pattern c of nucleotides (typically 4-6 nucleotides long) such that enzyme cuts the DNA molecule at the occurrences of c in the sequence of the molecule.

The DNA molecule to be sequenced, G, is first split with some enzyme into smaller pieces F_1, $F_2,\ldots,F_n$, called *fragments*. It is then possible to determine the first 200 or even 400 nucleotides from each end of F_i. For each F_i we get two sequences, g_i and $g_i^{\prime}$, called the *gels* (because they are read from an autoradiograph of a sequencing gel); each gel is simply a string of characters A, C, G and T and possibly some additional characters denoting uncertain identification of A, C, G or T. The gels can typically be read at least 95 % correct. They form the primary data used in assembling the whole sequence of G. Fig. 2 (first splitting) illustrates the situation; gels are shown as arrows.

* This work was supported by the Academy of Finland under grant 01/096 to E.Ukkonen and H.Söderlund

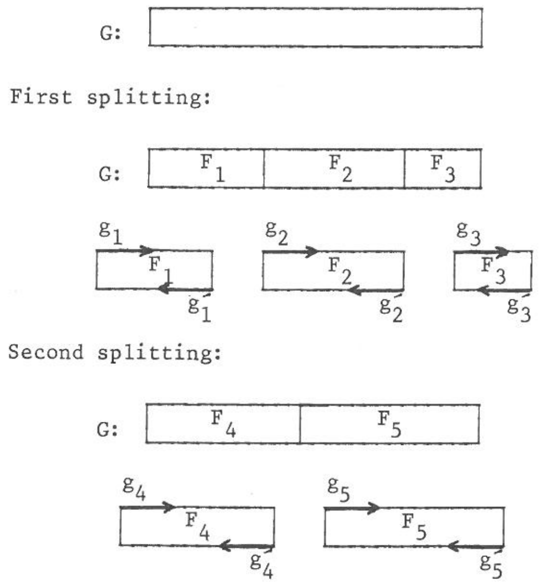

Fig. 2: DNA sequencing.

Cutting G into fragments only once does not usually give sufficient information to infer the whole sequence G. In the example of Figure 2, although all of fragment F_3 is sequenced since g_3 and g_3' overlap, fragments F_1 and F_2 have a gap between the gels. What still increases the difficulty of assembling G from the gels is that the lengths of fragments F_i is known only approximately and their order in G is not known when the gels are read. Sometimes it is possible to separately determine the locations of F_i in G, but we assume that the construction of G is based solely on the gels as character strings without using additional information on their relative positioning (the "shotgun strategy" of sequencing).

So it is necessary to produce additional sets of fragments and the corresponding gels (second splitting in Figure 2) until every position of G is covered with so many gels that G can assembled with sufficient confidence. Thus we are playing a sort of "puzzle" with strings (the gels) that are a few hundred characters long to assemble the overall picture, the DNA molecule. Noting that one sequencing project may produce hundreds of gels, this is almost an impossible task without computer assistance.

The abstract *DNA sequence assembling problem* considered in this paper is as follows: Let the *basic sequence*, G, be a string over some finite alphabet, and let strings $g_1,\dots,g_k$, the *gels*, be approximate substrings of G. This means that for some fixed *error ratio* δ, each g_i can be transformed in at most $\delta|g_i|$ steps to a substring of G, that is, to a string q_i such that $G = xq_iy$ for some strings x and y. Possible transformation steps include insertion and deletion of a character, change of a character and transposition of two adjacent characters. The problem is, given gels $g_1,\dots,g_k$ and error ratio δ, to find the best possible approximation of the basic sequence G. By the best approximation we mean a shortest string G' containing for each g_i a substring which can be transformed in at most $\delta|g_i|$ steps to g_i. (The requirement that G' must be shortest possible seems the most natural condition to make the problem non-trivial.)

In this formulation of the problem we assume no a priori information on the locations of gels in G although it is not difficult to use such information in the algorithms to be presented. Moreover, our formulation simplifies the real problem by fixing the orientation of the gels with respect to G. Thus we require for each g_i that it is q_i and not possibly the reversal of q_i that is a substring of G. This restriction is made only to simplify the presentation.

It is most likely that the DNA sequence assembling problem cannot have a general solution algorithm whose running time would depend only polynomially on the total length of the gels. This is because a related problem (see Section 4) is NP-complete. In addition, the internal repeats and other internal homologies found in actual DNA sequences suggest that the instances of the problem arising in real sequencing projects are not essentially simpler than the general case. Therefore we'll describe an approximate polynomial time solution having the following main steps:

M 1. For each pair g_i and g_j of gels, compute all possible overlaps between g_i and g_j. Store all acceptable overlaps. The result can be understood as a graph whose nodes are the gels and there is one or more edges between two gels if and only if the gels overlap. The edges are labelled with detailed description of the structure of the corresponding overlap.

M 2. Using the graph of the step M1, compute a global alignment of the gels. This means that for each gel its most likely position "below" the imagined basic sequence G is determined.

M 3. From the alignment of step M2, compute the corresponding approximate G. Announce uncertain sections and gaps in the approximation.

This procedure is interactively used in a sequencing project as follows. After producing the first set of gels g_i, steps M1 and M2 are performed. The global alignment is shown to the biologist who inspects the positioning and makes the necessary corrections; this may lead to an iteration of step M2. Finally step M3 is performed, after which the biologist may manually edit the approximation into its final form. If the approximation is not good enough, additional gels are determined and the process is repeated from the step M1. To improve efficiency, it may be useful to replace some of the old gels by the section of approximate G they determined.

The delineated solution strategy is similar to the procedures implemented in some recent DNA analysis programs {2}, {15}. The detailed algorithms for steps M1, M2 and M3, given in Sections 3, 4 and 5, are our main contribution.

3. FINDING OVERLAPS BETWEEN GELS

To refine step M1, let g_1 and g_2 be gels. If they represent overlapping sections of G, and assuming that the errors in gels are uniformly distributed, g_1 and g_2 must have an approximate overlap. Hence there is an alignment, as shown in fig. 3, such that $g_1 = xy$, $g_2 = y'z$, and y can be transformed in at most $\delta|y|$ steps and y' in at most $\delta|y'|$ steps into the same substring of G. This means that at most $2\delta\cdot\max(|y|,|y'|)$ steps are needed to transform y into y'. Of course, the symmetric case where the right end of g_2 is to the left of the right end of g_1 can occur, too. Also is possible that the whole g_2 is an approximate substring of g_1, and vice versa. In the more general case of the problem where the relative orientation of the gels is free, g_2 should also be replaced by its mirror image (more precisely, by its so-called reversed complement).

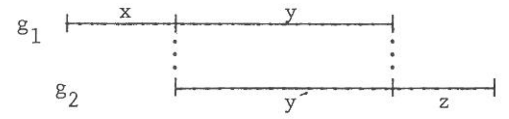

Fig. 3. Alignment of two gels.

The only mathematically satisfactory algorithm to find such an alignment can be based on a well-known dynamic programming method for computing so called *evolutionary distance* between two strings. Let the possible "mutation" operations of character strings be, as already mentioned, insertion, deletion, change and transposition. The *penalties* for mutations are given by function d. For all characters a and b, $d(a,b) \geq 0$ is the penalty of changing a to b; d(a,b) gives the penalty for inserting b, if $a=\varepsilon$ (ε denotes the empty string), and for deleting a, if $b=\varepsilon$. We also require that $d(a,b)=0$, if $a=b$. The penalty for transposition is given as $d(ab,ba) \geq 0$ for all (non-empty) characters a and b; for technical reasons (see {10}) we define that $d(ab,cd)=\infty$ whenever $ab \neq dc$ and that $2d(ab,ba) \geq \max(\ d(a,\varepsilon)+d(\varepsilon,a),\ d(b,\varepsilon)+d(\varepsilon,b)\ )$. The evolutionary distance between two strings is defined as the minimum total penalty of the mutation steps that transform one sequence to the other.

It is natural to select d such that the induced evolutionary distance function is a metric. This is true, for example, if

$$d(a,b) = \begin{cases} 0, & \text{if } a = b \\ 1, & \text{otherwise} \end{cases}$$

$$d(ab,cd) = \begin{cases} 1, & \text{if } ab = dc \\ \infty, & \text{otherwise} \end{cases}$$

We call this metric *step-counting metric* because now the evolutionary distance simply means the minimum number of mutation steps.

As implicitly proposed by Sellers {11}, the alignments of $g_1 = a_1a_2\ldots a_m$ and $g_2 = b_1b_2\ldots b_n$ can be computed by first constructing an $(m+1)\times(n+1)$ matrix f(i,j) from the recurrence

$$f(0,j) = f(i,0) = 0, \text{ for all } 0 \leq i \leq m,\ 0 \leq j \leq n\ ;$$

$$\begin{aligned} f(i,j) = \min[\ & f(i-1,j-1) + d(a_i,b_j)\ , \\ & f(i-1,\ j\ ) + d(a_i,\ \varepsilon)\ , \\ & f(\ i\ ,j-1) + d(\ \varepsilon\ ,b_j)\ , \\ & f(i-2,j-2) + d(a_{i-1}a_i,b_{j-1}b_j)\ ] \end{aligned}$$

This method is a variation of the well-known algorithm for computing the evolutionary distance as invented several times in various contexts {6}, {7}, {8}, {9}.

The computation of values f(i,j) determines a directed acyclic graph with nodes (i,j) and with an arc from $(i',\ j')$ to (i,j) if and only if the minimization gives f(i,j) from $f(i',j')$. The subgraph that consists of directed paths leading to (i,j) is called the *history graph* for (i,j). Fig. 4 shows an example where g_1 = GCATAT and g_2 = AGCAC, and the step-counting metric is used. The arcs in the history graph of the element (6,4) are also given.

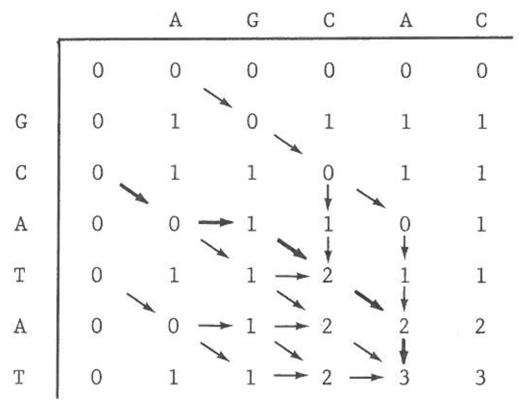

Fig. 4. A matrix f.

To find an alignment as shown in fig. 3, choose some (i_1,j_1) from the last row or column of matrix f, i.e., $i_1=m$ or $j_1=n$. Let some path in the history graph of (i_1,j_1) start from (i_o,j_o). Then necessarily $i_o=0$ or $j_o=0$. The path from (i_o,j_o) to (i_1,j_1) describes an alignment of g_1 and g_2 such that, using the notations of fig. 3, $y = a_{i_o+1}\ldots a_{i_1}$ and $y' = b_{j_o+1}\ldots b_{j_1}$ and the distance between y and y' is $f(i_1,j_1)$. For example, there is in fig. 4 a path from (2,0) to (6,4). Hence GCATAT and AGCAC can be aligned such that ATAT and AGCA overlap. The distance between these strings is f(6,4)=3. Each individual path specifies a detailed alignment. So the path given with double arcs gives the alignment

```
    G C A   T A T
        A G C A   A C        (1)
```

When (i_o,j_o) varies on the first row and column and (i_1,j_1) varies on the last row and column, several alignments are found. However, the alignment has to be good enough to be a potential overlap occurring in the correct positioning of the gels. Noting in addition that for relative small values $f(i_1,j_1)$, the length of the overlapping portions of g_1 and g_2 must be about $\min(i_1,j_1)$, we obtain with the step-counting metric that the expected upper bound on $f(i_1,j_1)\ /\ \min(i_1,j_1)$ is 2δ. From all (i_1,j_1) such that $f(i_1,j_1)/\min(i_1,j_1) \leq 2\delta$ we select those (i_1,j_1) which locally minimize $f(i_1,j_1)/\min(i_1,j_1)$. Finally choose some start node (i_o,j_o) from the history graph of each selected (i_1,j_1) as well as some path from (i_o,j_o) to (i_1,j_1); the details of selection are not important. Each path found in this way gives an alignment of g_1 and g_2 used later in assembling the basic sequence G.

A straightforward implementation of this procedure runs in time and space $O(mn)$. We now describe some improvements of implementation that essentially decrease the time and space requirements.

First, to choose the right end (i_1,j_1) of the alignment path, only the last row and column of the f matrix has to be known. Because a new row can be computed from two previous rows, only these must be stored, and hence the last row and column can be computed in space $O(m+n)$.

Second, if the evolutionary distance is a metric (as we assume), one easily proves that $f(i+1,j+1) \geqslant f(i,j)$. This means that if $f(i,j) > 2\delta\cdot\min(m,n)$ for some (i,j), then an acceptable alignment path cannot go through $(i+k,j+k)$ for any $k \geqslant 0$. If in the evalution of matrix f, some $f(i,j)$ is greater than $2\delta\cdot\min(m,n)$, no $f(i+k,j+k)$ need to be evaluated. Evaluation of $f(i+k,j+k)$ can simply be skipped and if the evaluation of some $f(i',j')$ asks for $f(i+k,j+k)$, we may assume $f(i+k,j+k) = \infty$. This considerably improves time-efficiency because only about $400\cdot\delta$ % of values $f(i,j)$ need explicit evaluation according to our experience.

Third, the computation of the history graph for (i_1,j_1) can be made more efficient by noting that this graph has nodes in a relatively narrow diagonal band of matrix f. Suppose that we are using the step-counting metric or another metric in which the penalty for insertion and deletion is at least 1. Then the band has at most $2\cdot f(i_1,j_1)+1$ diagonal rows with the row (i_1-k,j_1-k), $k \geqslant 0$, in the middle. Because $f(i_1,j_1) \leqslant 2\delta\cdot\min(m,n)$, the band needs at most space propotional to $\delta\cdot\min(m,n)^2$. After selecting (i_1,j_1), only the restriction of matrix f to this band must be recomputed to find (i_o,j_o) and the corresponding alignment path; c.f. {3,p.190}. Clearly, the computation can be done in time and space propotional to $\delta\cdot\min(m,n)^2$

Summarizing, the alignment procedure can be implemented such that it decides in time $O(\delta nm)$ and in space $O(n+m)$ whether there are acceptable alignments between g_1 and g_2. In most cases there are none and we are done. If there are acceptable alignments, each can be found in time and space $O(\delta\cdot\min(n,m)^2)$. Our experience shows that this procedure is reasonably efficient and has no ad hoc features usual in procedures used for same purpose so far; see {3}.

The procedure finds for every pair of gels g_1 and g_2 zero or more alignments. Each alignment is stored for later use as a record which contains the starting point of the overlapping section and a list of locations where an insertion of a blank character is to be made in either gel to align them. For example, the overlap in (1) is described by a record (3;3;4) indicating that the overlap starts at third character of the first gel, and that a blank character is to be inserted after the third character of the first gel and after the fourth character of the second gel.

We describe the final output of step M1 as graph, called the *alignment graph*. Such a graph is defined as a directed multigraph with the gels $g_1,\ldots,g_k$ as nodes and with a directed arc between g_i and g_j for each alignment found by our procedure with given δ. An arc between g_i and g_j is directed from g_i to g_j if and only if the corresponding overlap starts properly to the right of the first character of g_i or the first characters of g_i and g_j are aligned and $i < j$.

4. FINDING GLOBAL ALIGNMENT

To refine step M2, we must give a procedure for finding a global arrangement of the gels such that the approximation for G computed from the arrangement is best possible. As already mentioned, by the best approximation derivable from gels $g_1,\ldots,g_k$ we mean a shortest sequence G' which contains for each g_i a substring q_i whose distance from g_i in the step-counting metric is at most $\delta\cdot|g_i|$. Unfortunately, this is an NP-complete problem because restricted to the case $\delta=0$, finding G' is the so-called shortest common superstring problem, which is known to be NP-complete {13}. Hence a polynomial time procedure for finding G' is highly improbable, and a reasonable approximate solution should be used.

Our solution is based on the alignment graph found in step M1. The graph may contain arcs which are wrong in the sense that the corresponding overlaps do not occur in the correct global arrangement of the gels. The probability of finding wrong arcs increases rapidly with δ. To locate wrong arcs we can examine whether the arcs are inconsistent, that is, the corresponding overlaps cannot occur simultaneously in the global arrangement. Having found an inconsistency, the preference is given to the arc with longer overlap thus using the heuristic argument that a longer overlap is wrong with smaller likelihood. For finding a consistent subgraph of the alignment graph we also use some properties of directed interval graphs.

For our purposes, a *directed interval graph* {12} is a directed graph whose every node g_i corresponds to some substring (=interval) of some basic string, and there is a directed arc from g_i to g_j if and only if strings g_i and g_j overlap such that the overlap starts either properly to the right of the first character of g_i or at the first character of g_i and g_j and $i < j$. Such graphs have properties:

P1. The graph is acyclic.

P2. The graph has a directed spanning tree whose every node has at most one son which is not a leaf.

P3. The graph satisfies the *triangle condition*.

It is easy to verify P1 and P2. As regards P3, the triangle condition is the following consistency condition (i)-(ii) between arcs and overlaps:

(i) Suppose that there is an arc from g_i to g_j and the overlap between g_i and g_j starts at the (p_j+1)th character of g_i, and that there is an arc from g_i to g_k and the overlap between g_i and g_k starts at the (p_k+1)th character of g_i. If $p_k < p_j$ and $p_k+|g_k| > p_j$ then the graph has an arc from g_k to g_j which is consistent with the two arcs in the sense that the corresponding overlap between g_k and g_j must start at the (p_j-p_k+1)th character of g_k. Otherwise, if $p_j \leq p_k$ and $p_j+|g_j| > p_k$ then the graph has a consistent arc from g_j to g_k (from g_k to g_j if $p_j=p_k$ and $k<j$) such that the overlap between g_j and g_k must start at the (p_k-p_j+1)th character of g_j.

(ii) Suppose that there is an arc from g_i to g_k and from g_j to g_k. Then g_i and g_j necessarily overlap and the graph has a consistent arc from g_i to g_j or from g_j to g_i.

It should be clear that if the alignment graph contains only correct arcs then it is in fact a directed interval graph. However, the actual alignment graph formed by step M1 only approximates the correct graph, but it is reasonable to assume that the approximation is relatively good. This suggests that we can base our global alignment procedure on finding for the alignment graph a subgraph with properties P1-P3. The procedure has the alignment graph as its input and proceeds in steps A1 - A4 as follows.

A 1. To construct for the alignment graph a directed spanning tree of the form mentioned in P2, select a node (=a gel) r_o with no entering arcs. Node r_o will be the root of the tree. Also set i := 1.

A 2. Select node $r_i \notin \{r_o,\ldots,r_{i-1}\}$ from all nodes to which there is an arc from r_{i-1} by requiring that the right end of r_i is to the right of the right end of r_{i-1} in the overlap corresponding to arc (r_{i-1},r_i) and that the overlap is longest possible. Then (r_{i-1},r_i) is the next arc of the tree and r_i the next node of the tree. Set i := i+1. Repeat step A2 until no new r_i can be found.

A 3. For each r_i, $i=k,k-1,\ldots,0$, selected so far do the following: if (r_i,r) is an arc and r is not yet in the tree, add node r and arc (r_i,r) to the tree.

A 4. Check that the tree constructed in steps A1 - A3 satisfies the triangle condition by verifying for each pair of arcs in the tree which satisfies the premises of the triangle condition (in fact, only case (i) may occur), that a good approximation of the implied third arc is in the original alignment graph.

This procedure can be implemented in a depth-first traversal starting from r_o. We call the tree constructed in this way an *alignment tree*. The tree fixes in an obvious way a consistent global arrangement for its nodes. It may happen that every gel is not included in the tree, that is,the tree spans only a subgraph of the alignment graph. Then steps A1 - A4 are repeated for the subgraph induced by the remaining gels. Each tree found corresponds to a connected section of the basic sequence G.

If in step A1 all nodes have entering arcs, the construction can be started, say, from the node with minimum number of entering arcs. In step A4 all violations of the triangle condition are reported to the user who should first interactively remove or change the inconsistent arcs and overlaps in the alignment graph and then find a consistent alignment by repeating steps A1 - A4. The user must also be notified if there are arcs in the alignment graph that join two nodes in different trees. Such arcs indicate that some tree arcs are possibly wrong and should be removed.

Step M2 of the main procedure gives in this way a forest of alignment trees which satisfy the triangle condition. Since the procedure examines each arc of the alignment graph only a bounded number of times, the time requirement to produce a forest is linear in the size of the graph. Compared to the procedures proposed in {2}, {15} for finding global arrangements of gels, the use of interval graphs is the main novel feature of our method.

5. COMPUTING THE FINAL APPROXIMATION

To finally refine step M3, let T be some alignment tree produced by step M2. Tree T gives a unique global arrangement for its gels but the detailed alignment of the gels is known only pairwise, as computed in step M1. Since more than two gels may overlap the same region, we must give a procedure for aligning arbitrary many gels character by character. A many-dimensional generalization of the alignment procedure of Section 3 would give the best solution but it is useless in practice because its time and space requirements are exponential in the number of sequences to be aligned.

Our practical solution utilizes the alignment information associated with each arc (g_i,g_j) of the alignment tree in step M1. We call this information the *arcwise insertion lists*, and denote it by $(K_{ij};L_i;L_j)$. An example of such lists was given already at the end of Section 3. To describe an additional example, let

```
g1 = A A C A A C T G G G A T A   and
g2 = C A C T T G G A T A A
```

be gels connected by an arc. Then $(K_{12};L_1;L_2)$ might be (3 ; 7 ; 2,6) indicating that the overlap starts at the third character of g_1, and that the insertion list is 7 for g_1 and 2,6 for g_2. The encoded alignment is

```
g1 = A A C A A C T   G G G A T A
g2 =     C A   C T T G   G A T A A
```

From arcwise lists we compute a *global insertion list* M_i for each gel g_i in tree T.

Initially, each M_i is empty. The effect of each arcwise list on M_i can be calculated, for example, in the same order as the arcs were originally added to the tree. What is essential is that when processing the list for arc (g_i,g_j), the list for the arc leading to g_i in T has already been processed. So if an arcwise list $(K_{ij};L_i;L_j)$ is the next in the order, we set first $M_j := L_j$. Moreover each s such that $s + K_{ij}$ is in M_i but not in L_i is added to M_j. In addition, each element s' which is in L_i but not in M_i is added to M_i. Finally, if any gel g_k inserted to the tree before gel g_i overlaps the position s' of g_i (this can be decided because the global alignment is fixed), then position s" which in g_k overlaps position s' of g_i is added to M_k.

To illustrate the method, we continue the above example. Let g_3 = ACTGTGAATAACC. When g_1, g_2 and g_3 are aligned together by first joining g_2 to g_1 as above and then g_3 to g_2 using arcwise list $(K_{23};L_2;L_3)$ = (1 ; 6,8 ; 2), we obtain

```
g1 = A A C A A C T   G G G A   T A
g2 =     C A   C T T G   G A   T A A
g3 =       A   C   T G T G A A T A A C G
```

and the global insertion lists are $M_1 = (7,11)$, $M_2 = (2,6,8)$ and $M_3 = (1,2)$. Note that the global alignment is achieved by inserting blank symbols. Hence no information is lost.

Having found the final global alignment we must still generate the implied approximation G´ for the segment of the basic sequence G represented by T. This is a simple voting process. Consider some position s. If every gel overlapping s has the same character X in s, then G´ will have X in position s. Otherwise, if X is the character that occurs most often in s, then G´ will have $\underline{X}$ (= uncertain X) in s. If X is not unique, notation $\underline{?}$ is used. The above alignment for g_1, g_2 and g_3 would yield in this way

$$G' = A\ A\ C\ A\ \underline{A}\ C\ \underline{T}\ \underline{T}\ G\ \underline{?}\ G\ A\ \underline{A}\ T\ A\ A\ C\ G$$

The time needed in producing G´ is clearly linear in the total langth of the gels in tree T. Of cource, different improvements of the voting procedure can be imagined. However, the final confirmation of G´ must in every case be done interactively by the user.

6. CONCLUSION

Starting with some concepts from string matching algorithms and from graph algorithms, we developed a solution method for the DNA sequence assembling problem. The solution has a reasonable balance between computational efficiency and accuracy. Also important is that we have avoided using ad hoc ideas by working with a precise mathematical formulation of the problem.

Our algorithm consists of main steps M1, M2 and M3. The time requirement is dominated by M1, which performs the pairwise comparison of the gels. In fact, step M1 needs time $\mathcal{O}(\delta N^2)$ while steps M2 and M3 are linear in N where $N = \Sigma|g_i|$ denotes the total length of the gels.

We have constructed a prototype FORTRAN77 implementation of the algorithm running on Burroughs B7800. To test the program we used a data of 123 gels from a (not yet completed) sequencing project. The total length N of the gels was about 17500. The program produced automatically about 5800 characters long approximation for the DNA sequence. The quality of the approximation was comparable to what was obtained ealier by a more conventional program requiring frequent user interaction. The total CPU time for M1, M2, M3 was about 26 minutes including about 5 seconds for step M2 and less than 5 seconds for step M3. Thus the dominance of M1 is very clear. It should emphasized, however, that the time requirement of M1 decreases rapidly with δ (for example, the time was about 20 minutes for δ=5%) and that the time spent by M1 is also very useful: new approximations can be generated in a few seconds with steps M2 and M3 from the information computed by M1.

REFERENCES

{1} T.R.Gingeras & R.J.Roberts, Steps toward computer analysis of nucleotide sequences. Science 209(19 Sept.1980), 1322-1328.
{2} R.Staden, Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. Nucleic Acids Research 10, 15(1982), 4731-4751.
{3} Nucleic Acids Research 10, 1(January 11, 1982). Special issue devoted to the applications of computers to research on nucleic acids.
{4} E.A.Feigenbaum, The art of artificial intelligence: I. Themes and case studies of knowledge engineering. Proc. Fifth Int. Joint Conf. Artificial Intelligence (1977), 1014-1029.
{5} P.Friedland, L.Kedes, D.Brutlag, Y.Iwasaki, R.Bach, GENESIS, a knowledge-based genetic engineering simulation system for representation of genetic data and experiment planning. Nucleic Acids Research 10(1982), 323-340.
{6} V.I.Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. 10(1966), 707-710.
{7} R.Wagner & M.Fisher, The string-to-string correction problem.J.ACM 21(1974), 168-178.
{8} S.B.Needleman & C.D.Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48(1970), 443-453.
{9} D.Sankoff, Matching sequences under deletion/insertion constraints. Proc. Nat. Acad. Sci. 69(1972), 4-6.
{10} R.Lowrance & R.Wagner, An extension to the string-to-string correction problem. J.ACM 22(1975), 177-183.
{11} P.H.Sellers, The theory and computation of evolutionary distances:Pattern recognition. Journal of Algorithms 1(1980), 359-373.
{12} M.C.Columbic, Algorithmic Graph Theory and Perfect Graphs. Academic Press, 1980.
{13} M.R.Garey & D.S.Johnson, Computers and Intractability. Freeman, 1979.
{14} M.Stefik & al., The organization of expert systems, a tutorial. Artificial Intelligence 18(1982), 135-173.
{15} G.Osterburg, K.H.Glatting, J.Buchert, J. Wolters, A fast method for arranging DNA sequence fragments. Manuscript, German Cancer Research Center, 1982.