Efficient Optimal Decomposition of a Sequence into Disjoint Regions, Each Matched to Some Template in an Inventory

DAVID SANKOFF
Centre de recherches mathématiques, Université de Montréal, Montréal, Québec, H3C 3J7 Canada

Received 20 September 1990; revised 21 April 1992

ABSTRACT

Given an amino acid sequence, we discuss how to find efficiently an optimal set of disjoint regions (substrings, domains, modules, etc.), each of which can be matched to some element of a predefined inventory containing, for example, consensus sequences, protosequences, or protein family profiles. A two-stage approach to sequence decomposition, consisting of the detection of all acceptable matches followed by the construction of an optimal subset of compatible matches, leads to computational difficulties. When the problem is reformulated in terms of network comparisons, it can be solved in time quadratic in the length of the sequence and linear with the number of templates in the inventory, by a single pass of a dynamic programming algorithm. This method has the advantage that the criterion for acceptable matches can be relaxed without materially affecting computing time. Except under special conditions it is more efficient than previous segmentation methods based on dynamic programming.

INTRODUCTION

An increasingly important problem in sequence comparison is as follows: Given an experimental sequence **a**, how can we find, in an efficient way, an optimal set of disjoint regions (substrings, domains, modules, etc.) of **a**, each of which can be matched to some element **b** of an predefined set or inventory $B = (\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(D)})$. There may be unmatched regions in **a**, and several regions in **a** may be matched to the same inventory element. We will focus on amino acid sequences, though the methodology we develop can also be adapted to the study of nucleic acid sequences.

The inventory may contain, for example, (i) consensus sequences, (ii) reconstructed protosequences, (iii) probabilistic profiles (sequences of distributions, each distribution specifying a probability for each of the 20 amino acids), (iv) sequences of nodes from a classificatory hierarchy

655 Avenue of the Americas, New York, NY 10010 0025-5564/92/$5.00

or lattice defined on the basis of amino acid properties (hydrophobic, positively charged, etc.), or (v) motifs in the form of acyclic directed networks each compactly representing a number of alternative versions of a sequence. We will refer to the elements of the inventory as *templates*.

This problem has a counterpart in the study of speech recognition, where dynamic programming for sequence comparison is known as "time warping." This approach was extended from single-word recognition to the much more difficult problem of recognizing *continuous* speech, through the introduction of the "level-building" method [5] for finding the optimal decomposition of an utterance into words drawn from a predefined vocabulary. This technique was recognized [3] as a special case of the comparison of two acyclic directed networks. See [4] for an even more general framework.

In this paper we will first show how a two-stage approach to sequence decomposition consisting of the detection of all acceptable matches followed by the construction of an optimal subset of compatible matches leads to computational difficulties. We then show how the protein sequence decomposition problem can be formulated as a network comparison problem and solved relatively efficiently (quadratic in the length of the sequence, linear with inventory size) by a single pass of a dynamic programming algorithm. The latter method has the advantage that the criterion for acceptable matches can be relaxed without materially affecting computing time, whereas the former approach becomes impracticable due to the explosion in the number of marginally acceptable matches. Finally, we show how the network approach improves on previous methods also based on dynamic programming.

FORMALIZATION

We are given an experimental sequence $\mathbf{a} = a_1 a_2 \cdots a_n$ of amino acid residues and an inventory $B = \{\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(D)}\}$. If the $\mathbf{b}^{(i)}$ are simply amino acid sequences themselves [cases (i) and (ii) above], let their average length be m, so that the total size of the inventory is Dm. Note that in all of cases (i)–(v), it is not difficult to define an effective template "length" such that any template $\mathbf{b}^{(i)}$ of length m_i can be compared to any sequence $\mathbf{a}$ of length r in time proportional to $m_i r$. Again, let m be the average length, so that in all cases we can say that the size of the inventory is Dm.

We will enter into the details here only of the case where the elements of the inventory are themselves sequences, but all of cases (i)–(v) can be treated in essentially analogous ways. Later we will discuss some aspects of case (v) more fully.

We are also given a set of individual match scores $d(\alpha,\alpha)>0$ and penalties $d(\alpha,\beta)<0$ for amino acids α and β (where $\alpha\neq\beta$), as well as deletion and insertion penalties $d(\alpha,0)<0, d(0,\beta)<0$. By convention we speak of deletion when the right-hand argument of (α,β) is zero and insertion when the left-hand one is. As we will be trying to maximize scores, no generality will be lost in requiring $d(\alpha,\beta)\geqslant d(\alpha,0)+d(0,\beta)$. We can also assume $d(\alpha,\beta)\geqslant d(\alpha,\gamma)+d(\gamma,\beta)$ for any three different amino acids α,β,γ. For simplicity, we assume that the penalty for the deletion or insertion of several contiguous amino acids in a sequence is equivalent to the sum of their individual deletion or insertion penalties. A match between an inventory element $\mathbf{b}=b_1\cdots b_r$ and the experimental sequence $\mathbf{a}$ consists of a region or fragment $\mathbf{a}'=a_i\cdots a_j$ of $\mathbf{a}$ and a set of correspondences (a_h,b_k). Two correspondences (a_h,b_k) and (a_u,b_v) can be in the same match only if $i\leqslant h<u\leqslant j \Leftrightarrow 1\leqslant k<v\leqslant r$. The overall score of the match is

$$f(\mathbf{a}',\mathbf{b})=\sum_{\substack{(a_h,b_k)\\ \text{in match}}} d(a_h,b_k)+\sum_{\substack{a_h \text{ in no}\\ \text{correspondence,}\\ i<h<j}} d(a_h,0)+\sum_{\substack{b_k \text{ in no}\\ \text{correspondence,}\\ 1\leqslant k\leqslant r}} d(0,b_k).$$

Since f is to be maximized, the first term a_i and the last term a_j in a matched region $\mathbf{a}'$ are never deleted or mismatched, because that would only lower f—it would always be better to use a region that started at a_{i+1} (or terminated at a_{j-1}); hence the $i<h<j$ under the deletion penalty summation. It is essential to the very concept of a template $\mathbf{b}$, however, that all its terms be accounted for; hence the $1\leqslant k\leqslant r$ under the insertion penalty summation.

The individual match scores and penalties $d(\alpha,\alpha), d(\alpha,\beta), d(\alpha,0)$, and $d(0,\beta)$ are designed so that matches with overall scores f of zero or less can be considered accidental and hence should be discarded. Increasing the individual match scores $d(\alpha,\alpha)$ and/or decreasing the size of the penalties $d(\alpha,\beta), d(\alpha,0)$, and $d(0,\beta)$ effectively relaxes the criterion for match acceptability because fewer matches will be discarded.

The optimal decomposition problem, then, is to find a set of disjoint regions of $\mathbf{a}$ such that each region is matched to some template in the inventory and such that the sum of the overall match scores is maximal. Recall that not every term of $\mathbf{a}$ need be in some matched region, in which case it will neither add to nor subtract from this sum of the overall scores.

A TWO-STAGE SOLUTION AND ITS DIFFICULTIES

The most direct way of approaching this problem is to first find all suitably close matches of inventory elements in the experimental sequence and then piece together an optimal subset of nonoverlapping matched regions. We will show that this method has serious shortcomings.

For the first step, we sketch a version of the well-known dynamic programming solution [3, 7]. For each inventory element $\mathbf{b} = b_1 \cdots b_r$, the following recurrence is calculated for each cell (i, j) of the $(n+1) \times (r+1)$ matrix f, using the shorthand notation $f(i,j) = f(a_1 \cdots a_i, b_1 \cdots b_j)$:

$$f(i,j) = \max\big[f(i-1,j) + d(a_i,0),$$
$$f(i-1,j-1) + d(a_i,b_j),$$
$$f(i,j-1) + d(0,b_j)\big],$$

with initial conditions $f(0,0) = f(i,0) = 0$; $f(0,j) = \sum_{k=1}^{j} d(0,b_k)$. Positive values of $f(i,r)$ indicate acceptable matches of **b** with a region of **a** ending in a_i. This match can be found by tracing back the maximizing step in the recurrence that led to (i,r) in the usual way until a cell of form $f(h,0)$ is encountered, as illustrated in Figure 1, where all $d(\alpha,\alpha) = 3$, other $d(\alpha,\beta) = -1$, and $d(\alpha,0) = d(0,\beta) = -2$. The asymmetry in the initial conditions ensures that all sequence terms in a template **b** will be accounted for but that all terms preceding and following a matched region in the experimental sequence **a** will be ignored.

Unfortunately for this method, if $f(i,r)$ is large, $f(i-1,r), f(i+1,r)$, etc., are often large also, and the corresponding matches differ only with respect to the last few substitutions, deletions, and insertions. In addition, there are often several traceback paths from (i,r) leading to different cells in the zeroth column. This leads to a proliferation in the number of matches to be considered in the second stage of the algorithm. This is illustrated in Figure 2, where the eight acceptable matches detected by the recurrence for the simple example in Figure 1 are listed.

Nevertheless it can be shown that one pass of the dynamic programming recurrence plus traceback can find all acceptable matches between **a** and all D templates $\mathbf{b}^{(i)}$ in the inventory (or at least a compact representation of these matches, as in the highlighted traceback paths in Figure 1) in time roughly proportional to $(Dm)n$.

We now turn to the second step of the method. Given a set of matches output from the first step, there are generally many ways of piecing together a subset that contains only nonoverlapping regions in

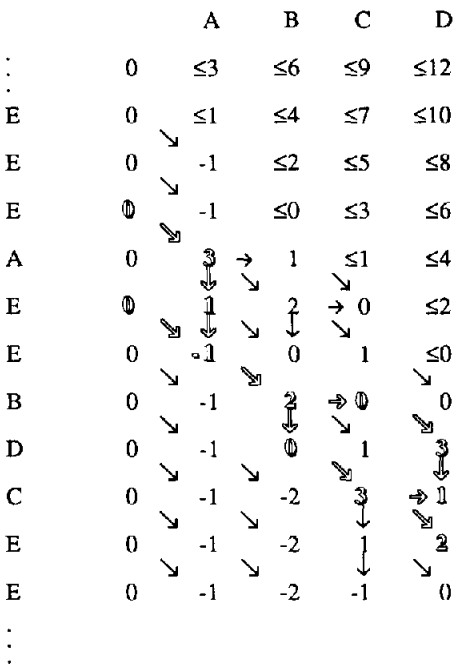

FIG. 1. Dynamic programming recurrence applied to the search for matches to $\mathbf{b} = ABCD$ in the experimental string $\mathbf{a} = \cdots EEEAEEBDCEE \cdots$. All $d(\alpha, \alpha) = 3$, other $d(\alpha, \beta) = -1$, and $d(\alpha, 0) = d(0, \beta) = -2$. Arrows to cell (i, j) come from cell(s) that provided the maximizing term(s) in calculating $f(i, j)$. Traceback for acceptable matches indicated by highlighted terms and arrows.

the experimental sequence. This is illustrated in Figure 3. Multiple solutions can become a serious problem, especially if the match scores have been adjusted (i.e., penalties have been reduced) to increase the sensitivity of the search.

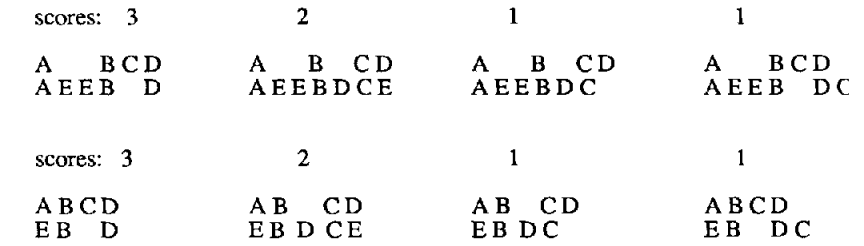

FIG. 2. Eight acceptable matches detected by the recurrence in Figure 1.

template inventory: {AB, BCD, DABABB, BBC}

experimental sequence: A B C D A B A B B B C D

solution 1: ----M_1---- --M_2-- --M_3-- ---M_4---

solution 2: --M_5-- --------M_6---------- -----M_7----

solution 3: --M_5-- --M_2-- --M_3-- ---M_4----

FIG. 3. Three solutions to a decomposition problem. M_2, M_3, and M_5 match template *AB*, M_1 and M_7 match *BCD*, M_4 matches *BBC*, and M_6 matches *DABABB*. Substitution, insertion, and deletion penalties are set at $-\infty$, so that only identical terms may be matched.

One way to sort this out (i.e., to find the optimal solution) algorithmically requires setting up a compatibility matrix for the set of matches found in the search. Two matches are compatible if they do not overlap, as illustrated in Figure 4 for the data of Figure 3.

The selection of an optimal subset of compatible matches is a computationally difficult problem, in a worst-case sense, though algorithms exist that usually operate efficiently with reasonably large compatibility matrices. See [6] for this type of treatment of the RNA folding problem.

Thus the two-stage approach (first find all matches; then pick optimal subset), while generally feasible, has some theoretical disadvantages, one of which is the exponential worst-case behavior of compatible subset algorithms. In practice, this may prevent us from experimenting with values of d that can pick up marginally acceptable matches: If matches that involve many insertions, deletions, and substitutions be-

	M_1	M_2	M_3	M_4	M_5	M_6	M_7
M_1	1	1	1	1	0	0	1
M_2	1	1	1	1	1	0	1
M_3	1	1	1	1	1	0	1
M_4	1	1	1	1	1	0	0
M_5	0	1	1	1	1	1	1
M_6	0	0	0	0	1	1	1
M_7	1	1	1	0	1	1	1

FIG. 4. Compatibility matrix for matches found in Figure 3. 1 in cell (i, j) indicates M_i compatible with M_j, 0 indicates incompatibility. Compatible subsets include the three solutions in Figure 3: $\{M_1, M_2, M_3, M_4\}$, $\{M_5, M_6, M_7\}$, $\{M_3, M_2, M_3, M_4\}$ as well as $\{M_1, M_2, M_3, M_7\}$ and $\{M_5, M_2, M_3, M_7\}$. Of course, any subset of any of these is also compatible but could not maximize f.

come acceptable in the search step, the number of matching regions found can proliferate rapidly, and the compatible subset algorithm is overloaded.

A SINGLE-PASS ALGORITHM

The alternative approach we propose is based on the dynamic programming comparison of directed acyclic networks [3] instead of sequences. This type of algorithm is designed to find two paths, one in each network, such that the match between these paths, considered as sequences, is optimal.

In a sequence **a**, each term a_i except for a_1 has one immediate predecessor, a_{i-1}. In most quadratic dynamic programming recurrences for comparing sequences **a** and **b**, such as the recurrence in the previous section, filling the (i,j)th cell requires the examination of three previously filled cells involving the predecessors of a_i and b_j, namely $(i-1,j),(i-1,j-1),(i,j-1)$, as illustrated in Figure 1.

In a network, each element a_i may have $N(a_i)$ immediate predecessors, where $N(a_i) \geq 1$, as in Figure 5, except for *source* elements such as a_t or b_h, where $N(a_t) = N(b_h) = 0$. Note that a term like a_u, which is predecessor to no other term, is called a *sink*.

In dynamic programming for comparing networks **a** and **b**, filling the "cell" corresponding to nodes a_i and b_j requires the examination of $[N(a_i)+1][N(b_j)+1]-1$ previously filled cells. In the example in Figure 4, each of the cells a_ib_h, a_ib_k, a_rb_j, a_rb_h, a_rb_k, a_sb_j, a_sb_h, a_sb_k, a_tb_j, a_tb_h, and a_tb_k must be examined (and previously filled) in order to calculate the value for a_ib_j. To find acceptable matched paths through

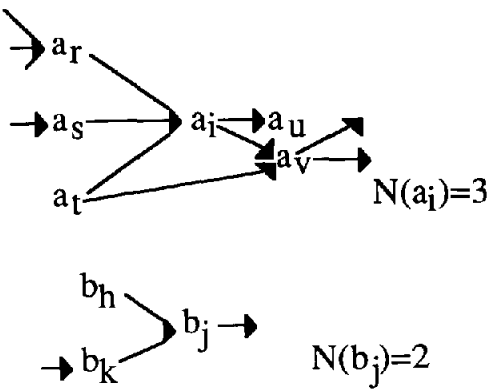

FIG. 5. Parts of two networks illustrating multiple predecessors.

the two networks, the recurrence becomes

$$f(i,j) = \max_{\substack{a_h \text{ predecessor of } a_i \\ b_k \text{ predecessor of } b_j}} \left[f(h,j) + d(a_i,0), \; f(h,k) + d(a_i,b_j), \; f(i,k) + d(0,b_j) \right]$$

with appropriate initial conditions.

To digress from the main point of this paper, if we wanted to search, in analogy with the sequence searches of the previous section, for matches of a template network **b** with an experimental sequence **a** in which only the template paths were required to be source-to-sink, the asymmetric initial conditions would be $f(i,0) = 0$ for all a_i, where 0 is introduced as a new predecessor to all sources in the **b** network, and $f(0,j) = \Sigma d(0,b_k)$, where the sum is taken over all b_k in some maximizing path in **b**, from a source b_h to b_j. Again, 0 is introduced as a new predecessor to all sources in the **a** network. Here the traceback would commence from any positive $f(i,k)$, where b_k is a sink.

The key to the method we will propose, however, is that it does not search for matches to one template entry **b** at a time. Instead, it uses the network recurrence to match an entire set of compatible regions in **a** to appropriate templates simultaneously. The idea is to construct a network such that the paths through this network consist of all possible sequences of templates, and then find the path that, when compared to **a**, maximizes f.

APPLICATION TO SEQUENCE DECOMPOSITION INTO DISJOINT REGIONS

What, then, are the networks to be compared in out method? The first network is simply the experimental sequence **a** of amino acids. The second, $\mathscr{B}$, has the form of Figure 6. Here the Φ is a "spacer" sequence $\sigma\sigma\cdots\sigma$ of length approximately n/K, and each of the K boxes is identical, containing the D elements of the inventory.

As in the expanded view of Figure 7, each template sequence or

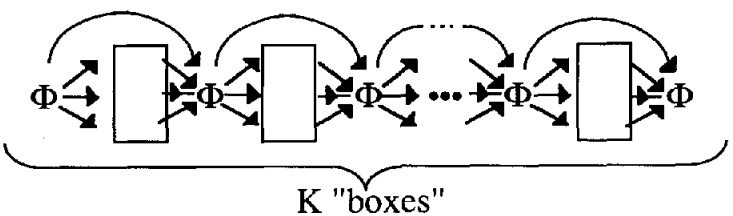

FIG. 6. Directed network $\mathscr{B}$ based on K copies of template inventory.

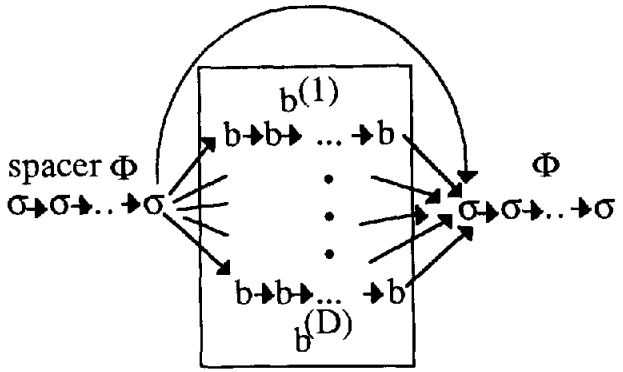

FIG. 7. Structure of each "box" in the $\mathscr{B}$ network.

expression in a box of $\mathscr{B}$ has an arrow leading to it from the preceding Φ and an arrow leading from it to the following Φ. Of course, within each template in the inventory, assuming the templates are sequences, there are also arrows connecting each term with its successor, but there are no arrows connecting terms of different inventory templates. If the templates in the inventory are not simply sequences but more general networks, then the configuration of arrows within each template may be more complicated, but there are still no connections between different templates. Note that there is an arrow directly connecting the Φ preceding a box to the Φ following it, allowing the box itself to be "skipped."

The basis of the method is a search for a match of the entire experimental sequence **a** to some source-to-sink path through $\mathscr{B}$. The initial conditions for the recurrence are thus $f(0,0)=0$; $f(i,0)=\sum_{h=1}^{i} d(a_h,0)$, which will be made irrelevant through the use of the spacers; and, considering the jth term in the pth template in the uth box, $f(0,j)=\sum d(0,b_k)$, where the sum is taken over all b_k in some maximizing path in $\mathscr{B}$, from the first σ in the first Φ to $b_j^{(p)}$ in the uth box; the zero-cost option of skipping boxes means that only $u=1$ is meaningful.

Then the comparison of the experimental sequence **a** with network $\mathscr{B}$ will match at most K successive regions of the experimental sequence with inventory items, and may match fewer by skipping boxes and proceeding from one spacer directly to another.

THE ROLE OF THE SPACERS

The main technical difficulty with our approach is how to deal with the regions of **a** that are not matched. If they were treated as deletions,

each such a_i would contribute $d(a_i, 0) < 0$, and hence the overall score for a given set of compatible matches would not be evaluated in the same way as in the previous two-stage algorithm, where unmatched regions contributed nothing to the overall score. Our method would be biased toward longer matched regions in **a** that are only marginally acceptable, rather than shorter regions that match more closely to inventory items. It is to avoid this, and to ensure that overall match scores are evaluated in the same way as in the two-stage method, that we have introduced the spacer sequences $\Phi = \sigma\sigma\cdots\sigma$. We set $d(\alpha, \sigma) = 0$ for all amino acids α. Thus, although every term of **a** must be matched (or mismatched, or deleted) in an optimizing pair of source-to-sink paths through the two networks **a** and $\mathscr{B}$, those terms that are matched with a σ have no direct effect on f, just as in the two-stage algorithm. We also assume $d(0, \sigma) = 0$ so that each unmatched region between two matched regions in **a** may be of any length—from 0 to n/K from any one Φ; but more than this is possible using spacers from two or more successive Φ's, with upper limit $(K+1)n/K > n$.

HOW MANY DOMAINS?

The value of K can be fixed beforehand at some reasonable value. In most applications, it should be approximately $n/(1-\epsilon)m'$, where m' is the shortest inventory item $\mathbf{b}'$ and ϵ is the maximum proportion of inserted terms (i.e., inserted into $\mathbf{b}'$) in an acceptable match between a region of **a** and $\mathbf{b}'$. Obviously no more than $n/(1-\epsilon)m'$ inventory items can be matched to disjoint regions in **a**, so there is no point in using a larger K.

Equally important, it can be shown that K should not be much smaller than n/m', because the network comparison between **a** and $\mathscr{B}$ does not then necessarily produce the optimizing pair of source-to-sink paths containing K or fewer matched regions. This is illustrated in the example in Figure 8, where the optimal solution involves $K = n/2m'$ regions of length m' all contiguous in **a**, that is, each one separated from the next by no unmatched terms.

This exhausts the K boxes in $\mathscr{B}$, "wastes" the intervening spacer sequences, and leaves only the initial and final spacer sequences in $\mathscr{B}$, of total length $2n/K$, to accommodate the remaining unmatched regions of **a**, of total length $n - Km'$. But in this example $n - Km' = n/2$, whereas the two spacer sequences have total length only $4m'$. For sufficiently large n, this is not enough, and so the optimal path will not be found by the algorithm.

There is way to ensure that any path with K matched regions is evaluated by the algorithm: We simply use a $\mathscr{B}$ network with $2K$ boxes and $2K+1$ spacer sequences, each with n/K terms σ. The matched

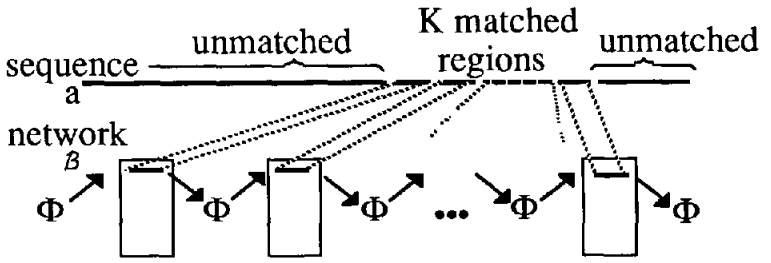

FIG. 8. Case where optimal solution cannot be realized in terms of matched source-to-sink paths in sequence **a** and network $\mathscr{B}$.

regions take up K boxes and can "waste" no more than K spacer sequences, leaving K boxes, which are skipped, and the remaining $K+1$ spacer sequences, of total length greater than n, to account for the unmatched regions of **a**. Of course, with $2K$ boxes in $\mathscr{B}$, the optimal path may well involve more than K boxes, so a solution that is optimal only among those with K or fewer matched regions may not be detected.

In any case, setting K large enough, such as $K = n/(1-\epsilon)m'$, avoids all these problems.

COMPUTING TIME

If the templates are sequences, each of the K boxes contains Dm terms. Each of these terms b_j must be compared to each a_i of the experimental sequence. The number of cells previously filled that must be examined in the recurrence is three for each pair (a_i, b_j). For pairs of form (a_i, σ), only two previously filled cells need be examined, as only substitution or insertion of σ are of interest, unless the σ is the first term of a spacer sequence, in which case $2(D+1)-1$ cells must be considered, because that σ has D predecessors in the $\mathscr{B}$ network.

Thus the total number of previously filled cells to be examined is

$$\begin{aligned} &3KnDm + 2(K+1)n(n/K-1) + (2D+1)(K+1)n \\ &\quad \leqslant 3(K+1)n(D[m+1]+n/K) \\ &\quad \leqslant cn^2Dm/m'(1-\epsilon), \end{aligned}$$

using $K = n/m'(1-\epsilon)$. For all large enough n, the constant c may be as low as 3. This basically describes the time complexity of the algorithm. In most applications, we can assume that m' is considerably greater than 1 and that m/m' does not materially change as new templates are added to the inventory. In addition, though the score

function d may change, we can assume that ϵ is bounded above by $1/2$ or some other constant less than 1. In these contexts, then, computing time is essentially proportional to n^2D, *independent of* m.

By our definition of m in the case when the templates are themselves networks, the computing time is still proportional to n^2Dm/m' or simply to n^2D, despite the fact that

$$[N(a_i)+1][N(b_j)+1]-1=2[N(b_j)+1]-1$$

may be much larger than 3 for many b_j.

Memory requirements are of the same order as computing time, though the techniques of [7] may be applicable to reduce this considerably.

INSENSITIVITY OF COMPUTING TIME TO NUMBER OF MATCHING REGIONS

One of the problems with the two-stage algorithm discussed earlier is its failure to cope with the rapid increase in the number of matched regions as the penalties for insertions, deletions, and substitutions are lessened. In contrast, the number of steps in the network comparison algorithm does not depend on the penalties or scores. Thus, in Figure 8 there are only two matched regions possible in the first run, but in the second run there are three variants for each of the matches shown as well as many different ways in which the first template may match. There are hundreds of different sets of compatible matches (decompositions), but the second run takes no more computing time than the first.

COMPARISON WITH OPTIMAL SEGMENTATION METHODS

The problem of optimally decomposing a sequence into disjoint segments arises in a number of fields besides continuous speech recognition and template recognition in proteins and nucleic acids. Dynamic programming as a second step in piecing together previously evaluated segments has been proposed in geology [2] and in the recognition of structural and functional regions of proteins [1]. In contrast to two-sequence comparison algorithms, these optimal segmentation procedures involve a series of one-dimensional recurrences:

$$f_K(i)=\max_{j<i}\{f_K(i-1),f_{K-1}(j)+v(j+1,i)\}$$

for $i \leqslant n$ and $K \leqslant n$, with appropriate initial conditions, where v measures the quality of the sequence interval $[j,i]$ according to some criterion and has been precalculated and stored for all such intervals as an initial step.

To compare these two-step optimal segmentations to the network comparison algorithm proposed here, we will first make some simplifications favorable to the former. We assume that all inventory items have length m. Then the maximum value of K that must be considered is of the order of n/m. We also assume that no interval $[j,i]$ need be considered where $i-j>m$. Thus carrying out the entire series of recurrences will require computing time proportional to $nmn/m=n^2$, while the precalculation, again assuming that $v(j,i)$ need not be calculated for $i-j>m$, is of the order of $m^2Dnm=nm^3D$, where m^2D term represents the cost of a dynamic programming comparison between the $[j,i]$ interval and the D inventory items. Thus the network comparison is more efficient where $n<m^3$, that is, where the precalculation dominates the segmentation recurrence with respect to computing time. When $n>m^3$, the segmentation recurrence dominates, and both it and the network algorithm are quadratic in n, though the network algorithm is more sensitive to the number of inventory items.

The assumption of inventory homogeneity in the foregoing calculation was favorable to the two-step segmentation algorithm. If the inventory consisted half of long [length $=m''=m(1+\theta)$] templates and half of short [length $=m'=m(1-\theta)$] ones, we would have to consider K of size n/m' and intervals $[j,i]$ of length m''. The computing times for the precalculation of v and for the recurrences would be increased by factors of $(1+\theta)(1+\theta^2)$ and $(1+\theta)/(1-\theta)$, respectively, whereas the order n^2D time of the network algorithm would be increased by a factor of $1/(1-\theta)$ only. Moreover, if the distribution of template lengths were highly skewed, the precalculation and segmentation recurrences would both be disadvantaged through the increase in m'', but the network algorithm would not.

In summary, the two-step optimal segmentation algorithm is preferable only in the case where n is very large in comparison to m (e.g., $m<10$, $n>10^3$) and then only if there is a large inventory containing templates all of about the same size. The network algorithm has the advantage for $n<m^3$, for example, when $m=30$ and $n=10^4$, or when the inventory contains a wide range of template sizes (e.g., $m'=10$, $m''=50$).

We note that in comparison with the two-stage algorithms discussed earlier in this paper, the precalculation step necessary for the optimal segmentation recurrence is much more costly, by a factor of m^2, though, of course, finding the decomposition through the recurrence is far more efficient than general compatibility methods of the latter algorithms.

DISCUSSION

In adapting the general directed network comparison algorithm to the sequence decomposition problem, we have had to develop a specific

treatment for the unmatched regions of the experimental sequence. This has involved the use of an appropriate number and configuration of spacer symbols in the network separating the copies of the inventory, and definition of penalties in such a way as to allow unpenalized deletions and insertions between matched regions while penalizing deletions from within, and insertions into, such regions. We have undertaken a detailed analysis of the complexity of the algorithm and compared this with two-step optimal segmentation procedures published previously.

Increasing the sensitivity of the match search step of the first two-stage algorithm we discussed at the beginning of this paper, by reducing substitution, insertion, and deletion penalties, leads to a proliferation of acceptable matches involving a high proportion of mismatched terms. This can lead to excessive computing requirements in the second step. In our network-based method, regions will be identified only if they form part of a larger, high-scoring, decomposition. More important, computing time depends only slightly (through ϵ) on the match scoring scheme.

Experimental (undocumented) Fortran code for the network comparison algorithm, suitable for small problems such as the one in Figure 9,

experimental sequence:	template inventory:
DCBABECDCBADCBABECDCBA	{ABCD, DCCCAB, CDBBBA}
ϵ=0.3, K= 8, spacer	sequences of length 3
first run	**second run**
match identity score: 3	match identity score: 3
insertion, deletion and substitution score: -7	insertion, deletion and substitution score: -2
best decomposition (score=10):	best decomposition (score=32):
DCBABECDCBADCBABECDCBA	D CBABECD CBAD CBABECDC BA
AB CD AB CD	DCCCAB CDBBBA DCCCAB CDBBBA
matches: 8	matches: 16
substitutions (true): 0	substitutions (true): 4
substitutions (with σ): 12	substitutions (with σ): 2
deletions: 2	deletions: 0
insertions (true): 0	insertions (true): 4
insertions (of σ): 0	insertions (of σ): 7

FIG. 9. Two runs of network algorithm for different values of penalty parameter.

is available from the author. As with other dynamic programming algorithms for sequence comparison, the algorithm proposed here could easily be vectorized for rapid execution.

This research was supported in part by operating and infrastructure grants from the Natural Sciences and Engineering Research Council of Canada and a team grant from the Fonds pour la formation de chercheurs et l'aide à la recherche of the government of Québec. The author is a fellow of the Canadian Institute for Advanced Research.

REFERENCES

1 I. E. Auger and C. E. Lawrence, Algorithms for optimal identification of segment neighborhoods, *Bull. Math. Biol.* 51:39–54 (1989).

2 T. R. Bement and M. S. Waterman, Locating maximum variance segments in sequential data, *Math. Geol.* 9:55–61 (1977).

3 J. B. Kruskal and D. Sankoff, An anthology of algorithms and concepts for sequence comparison, in *Time Warps, String Edits, and Macromolecules*, D. Sankoff and J. B. Kruskal, Eds., Addison-Wesley, Reading, Mass., 1983, pp. 265–310.

4 E. W. Myers and W. Miller, Approximate matching of regular expressions, *Bull. Math. Biol.* 51:5–37 (1989).

5 C. S. Myers and L. R. Rabiner, A level-building dynamic time-warping algorithm for connected word recognition, *IEEE Trans. Acoust., Speech Signal Proc.* ASSP-29(2):284–297 (1981).

6 J. M. Pipas and J. E. McMahon, Methods for predicting RNA secondary structure, *Proc. Natl. Acad. Sci. USA* 72:2017–2021 (1975).

7 T. F. Smith and M. S. Waterman, Identification of common molecular subsequences, *J. Mol. Biol.* 147:195–197 (1981).