Efficient Algorithms for Sequence Analysis*

David Eppstein [1] **Zvi Galil** [2,3]

Raffaele Giancarlo [4,5] **Giuseppe F. Italiano** [2,6]

[1] Department of Information and Computer Science, University of California, Irvine, CA 92717
[2] Department of Computer Science, Columbia University, New York, NY 10027
[3] Department of Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[4] AT&T Bell Laboratories, Murray Hill, NJ 07974
[5] On leave from Dipartimento di Matematica, Università di Palermo, Palermo, Italy
[6] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Rome, Italy

Abstract: We consider new algorithms for the solution of many dynamic programming recurrences for sequence comparison and for RNA secondary structure prediction. The techniques upon which the algorithms are based effectively exploit the physical constraints of the problem to derive more efficient methods for sequence analysis.

1. INTRODUCTION

In this paper we consider algorithms for two problems in sequence analysis. The first problem is sequence alignment, and the second is the prediction of RNA structure. Although the two problems seem quite different from each other, their solutions share a common structure, which can be expressed as a system of dynamic programming recurrence equations. These equations also can be applied to other problems, including text formatting and data storage optimization.

We use a number of well motivated assumptions about the problems in order to provide efficient algorithms. The primary assumption is that of concavity or convexity. The recurrence relations for both sequence alignment and for RNA structure each include an energy cost, which in the sequence alignment problem is a function of the length of a gap in either input sequence, and in the RNA structure problem is a function of the length of a loop in the hypothetical structure. In practice this cost is taken to be the logarithm, square root, or some other simple function of the length. For our algorithms we make no such specific assumption, but we require that the function be either convex or concave. The second assumption is that of sparsity. In the sequence alignment problem we need only consider alignments involving some sparse set of exactly matching subsequences; analogously, in the RNA structure problem we need only consider structures involving some sparse set of possible base pairs. We show how the algorithms for both problems may be further sped up by taking advantage of this sparsity rather than simply working around it.

Our primary motivation in developing the sequence analysis algorithms we present is their application to molecular biology, although the same sequence analysis procedures also have important uses in other fields. The reasons for the particular interest in molecular biology are, first, that it

* Work partially supported by NSF Grant CCR-9014605.

is a growing and important area of scientific study, and second, the lengths of biological sequences are such that the need for efficient methods of sequence analysis is becoming acute.

The development and refinement of rapid techniques for DNA sequencing [39, 47] have contributed to a boom in nucleic acid research. A wealth of molecular data is presently stored in several data banks (reviewed by Hobish [25]), which are loosely connected with each other. The biggest of these banks are GenBank [7] and the EMBL data library [20]. As of 1986, GenBank contained 5,731 entries with a total of more than 5 million nucleotides. Most of these data banks double in size every 8-10 months, and there are currently no signs that this growth is slowing down. On the contrary, the introduction of automatic sequencing techniques is expected to accelerate the process [36]. The existence of these nucleotide and amino acid data banks allows scientists to compare sequences of molecules and find similarities between them. As the quantity of data grows, however, it is becoming increasingly difficult to compare a given sequence, usually a newly determined one, with the entire body of sequences within a particular data bank.

The current methods of RNA structure computation have similar limitations. In this case one needs only perform computations on single sequences of RNA, rather than performing computations on entire data banks at once. However the cost of present methods grows even more quickly with the size of the problem than does the cost of sequence comparison, and so at present we can only perform structure computations for relatively small sequences of RNA.

Present algorithms for biological sequence analysis tend to incorporate sophisticated knowledge of the domain of application, but their algorithmic content is limited. Typically they act by computing a set of simple recurrences using dynamic programming in a matrix of values. Our algorithms solve the same problems, using the same assumptions about the physical constraints of the domain. However by using more sophisticated algorithmic techniques we can take better advantage of those constraints to derive more efficient methods for sequence analysis.

2. SEQUENCE ALIGNMENT

The first sequence analysis problem we study is that of sequence alignment. We only consider alignments between pairs of sequences, since the problem of multiple sequence alignment is NP-complete in general [18, 35] and therefore not likely to be solvable in polynomial time. Although many excellent surveys have been written on this topic and its significance to various fields (see for instance [51, 61, 62]), we briefly review the main algorithms for sequence analysis, omitting the technical details. Then, we introduce the algorithms that we have obtained, briefly mentioning the technicalities involved in them. Our main focus here is on fast computation of sequence alignment rather than on establishing how meaningful a given alignment is. For this latter topic, so important for molecular biology, the reader can refer to the excellent book by Waterman [62].

Sequence alignment seeks to compare the two input sequences, either to compute a measure of similarity between them or to find some core sequence with which each input sequence shares characteristics. Sequence alignment is an important tool in a wide variety of scientific applications [51, 64]. In molecular biology, the sequences being compared are proteins or nucleotides. In geology, they represent the stratigraphic structure of core samples. In speech recognition, they are samples of digitized speech.

The first important use of such comparisons is to find a common subsequence or consensus

sequence for the two input sequences. For instance, in molecular biology a common structure to a set of sequences can lead to an elucidation of the function of those sequences. A related use of sequence alignment is an application to computer file comparison, made by the widely used *diff* program [4]. The other use of sequence alignment is to compute a measure of similarity between the sequences. This can be used for instance in molecular biology, to compute the taxonomy of evolutionary descent of a set of species, or the genetic taxonomy of a set of proteins within a species [14]. It can also be used to group proteins by common structure, which also typically has a high correlation with common function.

2.1. Longest Common Subsequences and Edit Distance Computation

The simplest definition of an alignment is a matching of the symbols of one input sequence against the symbols of the other so that the total number of matched symbols is maximized. In other words, it is the longest common subsequence between the two sequences [5, 23, 26]. However in practice this definition leaves something to be desired. One would like to take into account the process by which the two sequences were transformed one to the other, and match the symbols in the way that is most likely to correspond to this process. For instance, in molecular biology the sequences being matched are either proteins or nucleotides, which share a common genetic ancestor, and which differ from that ancestor by a sequence of mutations. The best alignment is therefore one that could have been formed by the most likely sequence of mutations.

More generally, one can define a small set of *edit operations* on the sequences, each with an associated cost. For molecular biology, these correspond to genetic mutations, for speech recognition they correspond to variations in speech production, and so forth. The alignment of the two sequences is then the set of edit operations taking one sequence to the other, having the minimum cost. The measure of the distance between the sequences is the cost of the best alignment.

The first set of edit operations that was considered consisted of substitutions of one symbol for another (point mutations), deletion of a single symbol, and insertion of a single symbol. With these operations the problem can be solved in time $O(mn)$, where m and n are the lengths of the two input strings (we assume without loss of generality that $m < n$). This algorithm was independently published many times by researchers from a variety of fields [13, 21, 34, 41, 45, 46, 48, 56, 57] and is based upon dynamic programming, one of several used problem-solving technique in computer science and operations research. In particular, the following recurrence allows to find the best alignment between two given strings x and y:

$$D[i,j] = \min\{D[i-1,j-1] + sub(x_i, y_j), D[i-1,j] + del(y_j), D[i,j-1] + ins(x_i)\} \qquad (1)$$

where $D[i,j]$ is the best alignment between the prefix $x_1, x_2, \ldots, x_i$ and the prefix $y_1, y_2, \ldots y_j$, $del(y_j)$ is the cost of deleting the symbol y_j, $ins(x_i)$ is the cost of inserting symbol x_i, and $sub(x_i, y_j)$ is equal to 0 if $x_i = y_j$ and is equal to the cost of substituting symbol y_j for x_i. Further, one can find the subsequence of the first string having the best alignment with the second string in the same time.

If only the cost of the best sequence is desired, the $O(mn)$ dynamic programming algorithm for the sequence alignment problem need take only linear space: if we compute the values of the dynamic programming matrix in order by rows, we need only store the values in the single row

above the one in which we are performing the computation. However if the sequence itself is desired, it would seem that we need $O(mn)$ space, to store pointers for each cell in the matrix to the sequence leading to that cell. Hirschberg [22] showed that less storage was required, by giving an ingenious algorithm that computes the edit sequence as well as the alignment cost, in linear space, and remaining within the time bound of $O(mn)$.

Approaches other than dynamic programming are also possible. Masek and Paterson [38] have designed an algorithm that can be thought of as a finite automaton of $O(n)$ states, which finds the edit distance between the pattern and the strings in the database in $O(mn/\log n)$ time. Further speed-ups can be obtained using this automata-theoretic approach [32, 55]; however such improvements seem to require automata with exponentially many states, and an exponentially large preprocessing time to construct the automata; therefore they are only of theoretical interest, and are only applicable when m is much smaller than n.

2.2. Generalized Edit Operations and Gap Costs

Above we discussed sequence alignment with the operations of point substitution, single symbol insertion, and single symbol deletion. A number of further extensions to this set of operations have been considered. If one adds an operation that transposes two adjacent symbols, the problem becomes NP-complete [18, 58]. If one is allowed to take circular permutations of the two input sequences, before performing a sequence of substitutions, insertions, and deletions, the problem can be solved in time $O(nm \log n)$ [29]. We call this latter problem the *circular sequence alignment* problem.

An extension considered by Waterman [59] used dynamic programming to produce all the sequence alignments within a specified distance of the optimum. Waterman's algorithm allows one to determine "true" alignments, whenever these may disagree with the optimum (computed) solution because either the edit operations are not correctly weighted or some other constraints on the sequences were not taken into account.

Another important generalization includes subsequence-subsequence alignments as well as subsequence-sequence alignments [53, 54]. In this case, one is interested in aligning only pieces of the two input sequences, according to some maximization criteria; indeed the most common of such problems consists of finding maximally homologous subsequences. This is very important in molecular sequence analysis, where mutations involve changes in large blocks of a sequence. In this framework, even if it does not seem to be meaningful to look for a global alignment between two sequences, it is nevertheless important to find the pairs of maximally similar subsequences of the input sequences [54]. The mathematical formulation of this problem can be stated as follows. Given two input sequences x and y and a real valued function s expressing the similarity of each alignment, find the local maxima of s for x and y.

The solution proposed by Sellers [53] uses a technique called the intersection method. The basic idea is to solve a dynamic programming recurrence similar to (1) to compute entries in a dynamic programming matrix. Entry (i, j) of the matrix stores the score of the alignment between $x_1, x_2, \ldots, x_i$ and $y_1, y_2, \ldots, y_j$. Starting from the matrix, two graphs are constructed. The vertices of the graphs are entries in the matrix, while edges are defined according to the choice of the minimum in the recurrence. Paths in these graphs correspond to alignments of the two sequences.

The first graph G_1 contains all the alignments with nonnegative score which are not intersected from above or the left by an alignment with greater score. Analogously, the second graph G_2 contains alignments with nonnegative score which are not intersected from below or the left by an alignment with greater score. Taking the intersection of these two graphs gives all the local optimal alignments plus some spurious alignments which must be identified and removed. This can be done by finding the strongly connected components of the intersection graph by using well known algorithms [3]. As a result, the total time required to find the local maxima for the alignment of the two strings is $O(mn)$. As a last remark, we notice that the intersection method is a general technique which can be used also with the other algorithms for sequence analysis described in this paper.

A further generalization of the set of operations has been considered, as follows. We call a consecutive set of deleted symbols in one sequence, or inserted symbols in the other sequence, a *gap*. With the operations and costs above, the cost of a gap is the sum of the costs of the individual insertions or deletions which compose it. However, in molecular biology for example, it is much more likely that a gap was generated by one mutation that deleted all the symbols in the gap, than that many individual mutations combined to create the gap. Similar motivations apply to other applications of sequence alignment. Experimental results by Fitch and Smith [15] indicate that the cost of a gap may depend on its endpoints (or location) and on its length. Therefore we would like to allow gap insertions or deletions to combine many individual symbol insertions or deletions, with the cost of a gap insertion or deletion being some function of the length of the gap. The cost $w(i, j)$ of a generic gap $x_i...x_j$ that satisfies such experimental findings must be of the form

$$w(i, j) = f^1(x_i) + f^2(x_j) + g(j - i) \tag{2}$$

where f^1 and f^2 give the cost of breaking the sequence at the endpoints of the gap and g gives a cost that is proportional to the gap length.

Moreover, the most likely choices for g are linear or convex functions of the gap lengths [15, 60]. With such a choice of g, the cost of a long gap will be less than or equal to the sums of the costs of any partition of the gap into smaller gaps, so that as desired the best alignment will treat each gap as a unit. This requirement on the function g is equivalent to saying that the function w satisfies the following inequality:

$$w(i, j') + w(i', j) \leq w(i, j) + w(i', j') \text{ for all } i < i' \leq j < j' \tag{3}$$

When a function w satisfies this inequality we say that it is convex; when it satisfies the inverse of (3) we say that it is concave; and when it satisfies (3) with equality we say that it is linear. As we have said, the cost (or weight) functions that are meaningful for molecular biology are either convex or linear; however it is also natural to consider other classes of gap cost function. We call the sequence alignment problem with gap insertions and deletions the *gap sequence alignment problem*, and similarly we name special cases of this problem by the class of cost functions considered, e.g. the *convex sequence alignment problem*, etc.

To solve the gap sequence alignment problem, the following dynamic programming equation

was considered, where w' is a cost function analogous to w which satisfies (2).

$$D[i,j] = \min\{D[i-1,j-1] + sub(x_i, y_j), E[i,j], F[i,j]\} \tag{4}$$

where

$$E[i,j] = \min_{0 \le k \le j-1} \{D[i,k] + w(k,j)\} \tag{5}$$

$$F[i,j] = \min_{0 \le l \le i-1} \{D[l,j] + w'(l,i)\} \tag{6}$$

with initial conditions $D[i,0] = w'(0,i)$, $1 \le i \le m$ and $D[0,j] = w(0,j)$, $1 \le j \le n$.

The original sequence alignment problem treats the cost of a gap as the sum of the costs of the individual symbol insertions or deletions of which it is composed. Therefore if the gap cost function is some constant times the length of the gap, the gap sequence alignment problem can be solved in time $O(mn)$. This can be generalized to a slightly wider class of functions, the *linear* or *affine* gap cost functions. For these functions, the cost $g(x)$ of a gap of length x is $k_1 + k_2 x$ for some constants k_1 and k_2. A simple modification of the solution to the original sequence alignment problem also solves the linear sequence alignment problem in time $O(mn)$ [19].

For general functions, the gap sequence alignment problem can be solved in time $O(mn^2)$, by a simple dynamic programming algorithm [51]. The algorithm is similar to that for the original sequence alignment problem, but the computation of each entry in the dynamic programming matrix depends on all the previous entries in the same row or column, rather than simply on the adjacent entries in the matrix. This method was discovered by Waterman et al. [66], based on earlier work by Sellers [52]. But this time bound is an order of magnitude more than that for non-gap sequence alignment, and thus is useful only for much shorter sequences.

An early attempt to obtain an efficient algorithm for the problem of sequence alignment with convex gaps costs was made by Waterman [60]. As later shown by Miller and Myers [40], the algorithm in [60] still has an $O(mn^2)$ time behavior. However, the merit of [60] is that it proposes the use of convex gap cost functions for molecular biology. Later on, Galil and Giancarlo [16] considered the gap sequence alignment problem for both convex and concave cost functions. They reduced the problem to $O(n)$ subproblems, each of which can be expressed as a dynamic program generalizing the *least weight subsequence* problem, which had previously been applied to text formatting [24, 31] and optimal layout of B-trees [24]. Galil and Giancarlo solved this subproblem in time $O(n \log n)$, or linear time for many simple convex and concave functions such as $\log x$ and x^2. As a result they solved both the convex and concave sequence alignment problems in time $O(mn \log n))$, or $O(mn)$ for many simple functions. The algorithm by Galil and Giancarlo is very simple and thus likely to be useful also in practice. We point out that Miller and Myers [40] independently solved the same problem in similar time bounds.

Wilber [67] pointed out a resemblance between the least weight subsequence problem and a matrix searching technique that had been previously used to solve a number of problems in computational geometry [1]. He used this technique in an algorithm for solving the least weight subsequence problem in linear time. His algorithm also extends to the generalization of the least weight subsequence problem used as a subproblem by Galil and Giancarlo in their solution of the concave sequence alignment problem; however because Galil and Giancarlo use many interacting

instances of the subproblem, Wilber's analysis breaks down and his algorithm can not be used for concave sequence alignment. Klawe and Kleitman [30] studied a similar problem for the convex case and they obtained an $O(n\alpha(n))$ algorithm for it, where $\alpha(n)$ denotes a very slowly growing function, namely the inverse of Ackermann's function. The method devised by Klawe and Kleitman yields an $O(mn\alpha(n))$ algorithm for the gap sequence alignment problem with convex costs. Eppstein [9] showed that it is possible to modify Wilber's algorithm so that it can be used to solve the gap sequence alignment with concave costs in $O(mn)$ time. He used this result to solve the gap sequence alignment problem for functions which are neither convex nor concave, but a mixture of both. More precisely, if the cost function can be split into s convex and concave pieces, then the gap sequence alignment can be solved in $O(mns\alpha(n/s))$ time. This time is never worse than the time of $O(mn^2)$ for the naive dynamic programming solution [51] known for the general case. When s is small, the bound will be much better than the bound obtained by the naive solution. All these algorithms are based upon matrix searching and so the constant factors in the time bounds are quite large. Therefore, the algorithms are mainly of theoretical interest.

2.3. Sparse Sequence Alignment

All of the alignment algorithms above take a time which is at least the product of the lengths of the two input sequences. This is not a serious problem when the sequences are relatively short, but the sequences used in molecular biology can be very long, and for such sequences these algorithms can take more computing time than what is available for their computation.

Wilbur and Lipman [68, 69] proposed a method for speeding up these computations, at the cost of a small loss of accuracy, by only considering matchings between certain subsequences of the two input sequences. Let $F = \{f_1, f_2, \ldots, f_b\}$ be a given set of *fragments*, each fragment being a string over an alphabet A of size s. Let the two input sequences be $x = x_1x_2...x_m$ and $y = y_1y_2...y_n$. We are interested in finding an optimal alignment of x and y using only fragments from F that occur in both strings. A fragment $f = (i, j, k)$ of length k *occurs* in x and y if $f = x_ix_{i+1}...x_{i+k-1} = y_jy_{j+1}...y_{j+k-1}$. Given F, x and y, we can find all occurrences of fragments from F in x and y in time $O(n + m + M)$, where M denotes the number of such occurrences, by using standard string matching techniques.

An occurrence of a fragment (i', j', k') is said to be *below* an occurrence of (i, j, k) if $i + k \leq i'$ and $j + k \leq j'$; i.e. the substrings in fragment (i', j', k') appear strictly after those of (i, j, k) in the input strings. Equivalently, we say that (i, j, k) is *above* (i', j', k'). The *length* of fragment (i, j, k) is the number k. The *diagonal* of a fragment (i, j, k) is the number $j - i$. An alignment of fragments is defined to be a sequence of fragments such that, if (i, j, k) and (i', j', k') are adjacent fragments in the sequence, either (i', j', k') is below (i, j, k) on a different diagonal (a *gap*), or the two fragments are on the same diagonal with $i' > i$ (a *mismatch*). The cost of an alignment is taken to be the sum of the costs of the gaps, minus the number of matched symbols in the fragments. The number of matched symbols may not necessarily be the sum of the fragment lengths, because two mismatched fragments may overlap. Nevertheless it is easily computed as the sum of fragment lengths minus the overlap lengths of mismatched fragment pairs. The cost of a gap is some function of the distance between diagonals $w(|(j - i) - (j' - i')|)$.

When the fragments are all of length 1, and are taken to be all pairs of matching symbols from

the two strings, these definitions coincide with the usual definitions of sequence alignments. When the fragments are fewer, and with longer lengths, the fragment alignment will typically approximate fairly closely the usual sequence alignments, but the cost of computing such an alignment may be much less.

The method given by Wilbur and Lipman [69] for computing the least cost alignment of a set of fragments is as follows. Given two fragments, at most one will be able to appear after the other in any alignment, and this relation of possible dependence is transitive; therefore it is a partial order. Fragments are processed according to any topological sorting of this order. Some such orders are by rows (i), columns (j), or back diagonals ($i+j$). For each fragment, the best alignment ending at that fragment is taken as the minimum, over each previous fragment, of the cost for the best alignment up to that previous fragment together with the gap or mismatch cost from that previous fragment. The mismatch cost is simply the length of the overlap between two mismatched fragments; if the fragment whose alignment is being computed is (i, j, k) and the previous fragment is $(i-l, j-l, k')$ then this length can be computed as $\max(0, k' - l)$. From this minimum cost we also subtract the length of the new fragment; thus the total cost includes a term linear in the total number of symbols aligned. Formally, we have

$$D(i,j,k) = -k + \min \begin{cases} \min\limits_{(i-l,j-l,k')} D[i-l, j-l, k'] - \max(0, k'-l) \\ \min\limits_{(i',j',k') \text{ above } (i,j,k)} D[i',j',k'] + w(|(j-i)-(j'-i')|) \end{cases} \tag{7}$$

The naive dynamic programming algorithm for this computation, given by Wilbur and Lipman, takes time $O(M^2)$. If M is sufficiently small, this will be faster than many other sequence alignment techniques. We remark that the Wilbur-Lipman algorithm works for general cost functions w and does not take any advantage of the fact that w's used in practice satisfy inequality 3. We can show that by using this restriction on w we can compute recurrence 7 in time close to linear in M [11, 12]. This has the effect of making such computations even more practical for small M, and it also allows more exact computations to be made by allowing M to be larger.

The algorithms that we propose are quite different from the one given by Wilbur and Lipman and have the following common background. We consider recurrence 7 as a dynamic program on points in a two-dimensional matrix. Each fragment (i, j, k) gives rise to two points, (i, j) and $(i + k - 1, j + k - 1)$. We compute the best alignment for the fragment at point (i, j); however we do not add this alignment to the data structure of already computed fragments until we reach $(i+k-1, j+k-1)$. In this way, the computation for each fragment will only see other fragments that it is below. We compute separately the best mismatch for each fragment; this is always the previous fragment from the same diagonal, and so this computation can easily be performed in linear time. From now on we will ignore the distinction between the two kinds of points in the matrix, and the complication of the mismatch computation. Thus, we consider the following subproblem: Compute

$$E[i,j] = \min_{(i',j') \text{ above } (i,j)} D[i',j'] + w(|(j-i)-(j'-i')|), \tag{8}$$

where $D[i, j]$ is easily computable from $E[i, j]$.

We define the *range* of a point in which we have to compute recurrence 8 as the set of points below and to the right of it. Furthermore, we divide the range of a point into two portions, the *left influence* and the *right influence*. The left influence of (i, j) consists of those points in the range of (i, j) which are below and to the left of the forward diagonal $j - i$, and the right influence consists of the points above and to the right of the forward diagonal. Within each of the two influences, $w(|p - q|) = w(p - q)$ or $w(|p - q|) = w(q - p)$; i.e. the division of the range in two parts removes the complication of the absolute value from the cost function. Thus, we can now write recurrence 8 as:

$$E[i, j] = \min\{LI[i, j], RI[i, j]\}, \tag{9}$$

where

$$RI[i, j] = \min_{\substack{(i',j') \text{ above } (i,j) \\ j' - i' < j - i}} D(i', j') + w((j - i) - (j' - i')) \tag{10}$$

and

$$LI[i, j] = \min_{\substack{(i',j') \text{ above } (i,j) \\ j - i < j' - i'}} D(i', j') + w((j' - i') - (j - i)). \tag{11}$$

We observe that the order of computation of the points in the matrix must be the same for the two recurrences so that they can be put together into a single algorithm for the computation of recurrence 8.

Assuming that the cost function w is convex, we can compute recurrences 10 and 11 in time $O(n+m+M \log M\alpha(M))$. This time bound reduces to $O(n+m+M \log M)$ when w is concave. Our algorithm uses in a novel way an algorithmic technique devised by Bentley and Saxe [6], namely dynamic to static reduction. Matrix searching [1, 30] is also used. We remark that if the cost function is convex/concave and simple, matrix searching can be replaced by the algorithm of Galil and Giancarlo [16] to obtain an $O(n+m+M \log M)$ algorithm both for simple convex and concave cost functions. In the case that the cost function is not simple, we can still use the algorithm of Galil and Giancarlo instead of matrix searching. This results in a slow-down of our algorithm by a factor of $\log M$, but gives an algorithm which is likely to be more practical. The reader is referred to [12] for further details on the algorithm.

When the function w is linear, we can compute recurrences 10 and 11 in time $O(n + m + M \log\log \min(M, nm/M))$. This algorithm is based on the use of efficient data structures for the management of priority queues with integer keys [27]. As a by-product, we also obtain an improved implementation of the algorithm for the longest common subsequence devised by Apostolico and Guerra [5]. Our implementation runs in time $O(n \log s + d \log\log \min(d, nm/d))$. Here s is the minimum between m and the cardinality of the alphabet and d denotes the number of dominating matches defined in [23].

3. ALGORITHMS FOR COMPUTATION OF RNA SECONDARY STRUCTURE

In this section we are interested in algorithms for the computation of RNA secondary structure. More specifically, we will consider algorithms for loop dependent energy rules [70]. In order to

make the presentation of our algorithms self contained, we briefly review the biological background common to all of them.

RNA molecules are among the primary constituents of living matter. RNA is used by cells to transport genetic information between the DNA repository in the nucleus of the cell and the ribosomes which construct proteins from that information. It is also used within the process of protein construction, and may also have other important functions.

An RNA molecule is a polymer of nucleic acids, each of which may be any of four possible choices: adenine, cytosine, guanine, and uracil. Thus an RNA molecule can be represented as a string over an alphabet of four symbols, corresponding to the four possible nucleic acid bases. In practice the alphabet may need to be somewhat larger, because of the sporadic appearance of certain other bases in the RNA sequence. This string or sequence information is known as the *primary structure* of the RNA. The primary structure of an RNA molecule can be determined by gene sequencing experiments. Throughout this section we denote an RNA molecule by the string $y = y_1 y_2, ..., y_n$ and we refer to its i-th base y_i.

In an actual RNA molecule, hydrogen bonding will cause further linkages to form between pairs of bases. Adenine typically pairs with uracil, and cytosine with guanine. Other pairings, in particular between guanine and uracil, may form, but they are much more rare. Each base in the RNA sequence will pair with at most one other base. Paired bases may come from positions of the RNA molecule that are far apart in the primary structure. The set of linkages between bases for a given RNA molecule is known as its *secondary structure*.

The *tertiary structure* of an RNA molecule consists of the relative physical locations in space of each of its constituent atoms, and thus also the overall shape of the molecule. The tertiary structure is determined by energetic (static) considerations involving the bonds between atoms and the angles between bonds, as well as kinematic (dynamic) considerations involving the thermal motion of atoms. Thus the tertiary structure may change over time; however for a given RNA molecule there will typically be a single structure that closely approximates the tertiary structure throughout its changes.

The tertiary structure determines how the molecule will react with other molecules in its environment, and how in turn other molecules will react with it. Thus the tertiary structure controls enzymatic activity of RNA molecules as well as the splicing operations that take place between the time RNA is copied from the parent DNA molecule and the time that it is used as a blueprint for the construction of proteins.

Because of the importance of tertiary structure, and its close relation to molecular function, molecular biologists would like to be able to determine the tertiary structure of a given RNA molecule. Tertiary structures can be determined experimentally, but this requires complex crystalization and X-ray crystallography experiments, which are much more difficult than simply determining the sequence information of an RNA molecule. Further, the only known computational techniques for determining tertiary structure from primary structure involve simulations of molecular dynamics, which require enormous amounts of computing power and therefore can only be applied to very short sequences [44].

Because of the difficulty in computing tertiary structures, some biologists have resorted to the simpler computation of secondary structure, which also gives some information about the physical

shape of the RNA molecule. Secondary structure computations also have their own applications: by comparing the secondary structures of two molecules with similar function one can determine how the function depends on the structure. In turn, a known or conjectured similarity in the secondary structures of two sequences can lead to more accurate computation of the structures themselves, of possible alignments between the sequences, and also of alignments between the structures of the sequences [49].

3.1. Secondary Structure Assumptions and the Structure Tree

A perfectly accurate computation of RNA structure would have to include as well a computation of tertiary structure, because the secondary structure is determined by the tertiary structure. As we have said this seems to be a hard problem. Instead, a number of assumptions have been made about the nature of the structure. An energy is assigned to each possible configuration allowed by the assumptions, and the predicted secondary structure is the one having the minimum energy.

The possible base pairs in the structure are usually taken to be simply those allowed by the possible hydrogen bonds among the four RNA bases; that is, a base pair is a pair of positions (i, j) where the bases at the positions are adenine and uracil, cytosine and guanine, or possibly guanine and uracil. We write the bases in order by their positions in the RNA sequence; i.e. if (i, j) is a possible base pair, then $i < j$. Each pair has a binding energy determined by the bases making up the pair.

Define the *loop* of a base pair (i, j) to be the set of bases in the sequence between i and j. The primary assumption of RNA secondary structure computation is that no two loops cross. In other words, if (i, j) and (i', j') are base pairs formed in the secondary structure, and some base k is contained in both loops, then either i' and j' are also contained in loop (i, j), or alternately i and j are both contained in loop (i', j'). This assumption is not entirely correct for all RNA [37], but it works well for a great majority of the RNA molecules found in nature.

A base at position k is *exposed* in loop (i, j) if k is in the loop, and k is not in any loop (i', j') with i' and j' also in loop (i, j). Because of the non-crossing assumption, each base can be exposed in at most one loop. We say that (i', j') is a *subloop* of (i, j) if both i' and j' are exposed in (i, j); if either i' or j' is exposed then by the non-crossing assumption both must be.

Therefore the set of base pairs in a secondary structure, together with the subloop relation, forms a forest of trees. Each root of the tree is a loop that is not a subloop of any other loop, and each interior node of the tree is a loop that has some other subloop within it. We further define a *hairpin* to be a loop with no subloops, that is, a leaf in the loop forest, and we define a *single loop* or *interior loop* to be a loop with exactly one subloop. Any other loop is called a *multiple loop*. A base pair (i, j) such that the two adjacent bases $(i+1, j-1)$ are also paired is called a *stacked pair*. A single loop such that one base of the subloop is adjacent to a base of the outer loop is called a *bulge*.

As we have said, each base pair in an RNA secondary structure has a binding energy which is a function of the bases in the pair. We also include in the total energy of the secondary structure a *loop cost*, which is usually assumed to be a function of the length of the loop. This length is simply the number of exposed bases in the loop. The loop cost may also depend on the type of the loop; in particular it may differ for hairpins, stacked pairs, bulges, single loops, and multiple loops. The

loop costs in use today for hairpins, bulges and single loops are logarithms [70]. Therefore they are convex functions according to the definition given in the previous section. Moreover, they are simple convex functions.

With these definitions one can easily compute the total energy of a structure, as the sum of the base pair binding energies and loop costs. The optimal RNA secondary structure is then that structure minimizing the total energy.

3.2. Computation of Secondary Structure

With the definitions above, the optimum secondary structure can be computed by a three-dimensional dynamic program with matrix entries for each triple (i, j, k), where i and j are positions in the RNA sequence (not necessarily forming a base pair) and k is the number of exposed bases in a possible loop containing i and j. This computation takes time $O(n^4)$. The algorithm was recently discovered by Waterman and Smith [65] and it is the first polynomial time algorithm obtained for this problem.

Clearly, this time bound is so large that the computation of RNA structure using this algorithm is feasible only for very short sequences. Furthermore, the space bound of $O(n^3)$ also makes this algorithm impractical. Therefore, one needs further assumptions about the possible structures, or about the energy functions determining the optimum structure, in order to perform secondary structure computation more efficiently.

A particularly simple assumption is that the energy cost of a loop is zero or a constant, so that one need only consider the energy contribution of the base pairs in the structure. Nussinov et al. [42] showed how to compute a structure maximizing the total number of base pairs, in time $O(n^3)$; this algorithm was later extended to allow arbitrary binding energies for base pairs, while keeping the same time bound [43].

A less restrictive assumption, although not realistic, is that the cost of a multiple loop is a linear function of its length, rather than being a convex function of the base pairs in the loop and the length of the loop. Kruskal et al. [50] used such an assumption to derive a set of dynamic programming equations that yield the minimum energy RNA secondary structure in time $O(n^3)$. The $O(n^3)$ time bound is mainly due to the computation of internal loops and the computation of multiple loops. Indeed, each such computation takes $O(n)$ for each entry (i, j) of the dynamic programming matrix. We point out that the algorithm by Kruskal et al. is a variation of an earlier algorithm obtained by Zuker and Stiegler [71].

Instead of restricting the possible loop cost functions, one could restrict the possible types of loops. In particular, an important special case of RNA secondary structure computation is the computation of the best structure with no multiple loops. Such structures can be useful for the same applications as the more general RNA structure computation. Single loop RNA structures could be used to construct a small number of pieces of a structure which could then be combined to find a structure having multiple loops; in this case one sacrifices optimality of the resulting multiple loop structure for efficiency of the structure computation.

The single loop secondary structure computation can again be expressed as a dynamic programming recurrence relation [50, 63]. Again this relation seems to require time $O(n^4)$, but the space requirement is reduced from $O(n^3)$ to $O(n^2)$. In fact the time for solving the recurrence

can also be reduced, to $O(n^3)$, as was shown by Waterman and Smith [65]. In this paper, the authors also conjectured that the given algorithm runs in $O(n^2)$ time for convex (and concave) functions. Eppstein et al. [10] have shown how to compute single loop RNA secondary structure, for convex or concave energy costs, in time $O(n^2 \log^2 n)$. For many simple cost functions, such as logarithms and square roots, they show how to improve this time bound to $O(n^2 \log n \log\log n)$. The algorithm obtained by [10] is based on a new and fast method for the computation of internal loops for convex or concave energy costs. These results have recently been improved by Aggarwal and Park [2], who gave an $O(n^2 \log n)$ algorithm, and further by Larmore and Schieber [33], who gave an $O(n^2)$ algorithm for the concave case and an $O(n^2\alpha(n))$ algorithm for the convex case; however all these algorithms use matrix searching techniques, which lead to a high constant factor in the time bound.

We remark that the algorithms in [2, 10, 33] can also be used as a subroutine in the algorithm devised by Kruskal et al. [50]. Namely, one can compute the interior loops by using the algorithm given in [10] or [2] or [33] rather than the naive algorithm given in [50] for the same problem. Although such a modification does not yield any asymptotic speed up in the algorithm by Kruskal et al., it achieves a practical speed up since it reduces the computation time for internal loops. Similar considerations apply to the algorithm devised by Zuker and Stiegler.

Eppstein [9] has extended Aggarwal and Park's algorithm for single loop RNA structure to handle the case that the energy cost of a loop is not a convex or a concave function of the length, but can be split into a small number s of convex and concave pieces. His algorithm takes time $O(n^2 s \log n\alpha(n/s))$, or $O(n^2 s \log n \log(n/s))$ if matrix searching techniques are avoided. When s is small, these times will be much better than the $O(n^3)$ time known for general functions [65].

3.3. Sparseness in Secondary Structure

The recurrence relations that have been defined for the computation of RNA structure are all indexed by pairs of positions in the RNA sequence (and possibly also by numbers of exposed bases). For many of these recurrences, the entries in the associated dynamic programming matrix include a term for the binding energy of the corresponding base pair. If the given pair of positions do not form a base pair, this term is undefined, and the value of the cell in the matrix must be taken to be $+\infty$ so that the minimum energies computed for the other cells of the matrix do not depend on that value, and so that in turn no computed secondary structure includes a forbidden base pair.

Further, for the energy functions that are typically used, the energy cost of a loop will be more than the energy benefit of a base pair, so base pairs will not have sufficiently negative energy to form unless they are stacked without gaps at a height of three or more. Thus we could ignore base pairs that can not be so stacked, or equivalently assume that their binding energy is again $+\infty$, without changing the optimum secondary structure. This observation is similar to that of sparse sequence alignment, in which we only include pairs of matching symbols when they are part of a longer substring match.

The effect of such constraints on the computation of the secondary structure for RNA is twofold. First, they contribute to make the output of the algorithms using them more realistic from the biological point of view. This point is discussed in [70]. Second, they combine to greatly

reduce the number of possible pairs, which we denote by M, that must be considered to a value much less than the upper bound of n^2. For instance, if we required base pairs to form even higher stacks, M would be further reduced. The computation and minimization in this case is taken only over positions (i, j) which can combine to form a base pair.

The algorithms listed earlier account for the constraints just mentioned by giving a value of $+\infty$ at positions (i, j) that cannot form a base pair. Then this value can never supply the minimum energy in future computations, so (i, j) will never be used as a base pair in the computed RNA structure. Nevertheless, the time complexities of those algorithms depend on the total number of possible pairs (i, j), including those that are disallowed from pairing. Fortunately, the algorithms can be modified to ignore altogether such pairs. The net effect of such a modification is to replace n^2 by M in the time bounds for the algorithms, where by M we denote the number of base pairs that are allowed to form. That is, the time bounds for the algorithms in [50, 71] and [65] (for the single loop structure) becomes $O(Mn)$ whereas the time bound of the algorithm in [65] (for the general case) becomes $O(Mn^2)$.

Besides the constraints given by the problem itself, algorithms actually used in practice (as the one by Zuker and Stiegler) also incorporate heuristics in order to reduce even further the computational effort. One particular heuristic for the computation of interior loops is reported in [70]. Again, such heuristics boil down to reducing the number of entries (i, j) one has to consider in order to compute the secondary structure.

Based on the preceding discussion, we model the computation of a dynamic programming matrix D yielding an RNA secondary structure as follows. We are given one or more recurrences stating how to compute D and we are also given a set S of M entries (i, j) on which we have to compute D. We assume that all entries not included in S do not matter for the final result. We notice that S may be obtained by imposing the physical constraints of the problem and/or by imposing heuristic consideration on the entries of D. We next show how to take advantage of the sparsity of D in the computation of single loop RNA structure. Waterman and Smith [63] obtained the following dynamic programming equation:

$$D[i,j] = \min\{D[i-1,j-1] + b(i,j), H[i,j], V[i,j], E[i,j]\} \tag{12}$$

where

$$V[i,j] = \min_{0<k<i} D[k,j-1] + w'(k,i) \tag{13}$$

$$H[i,j] = \min_{0<l<j} D[i-1,l] + w'(l,j) \tag{14}$$

$$E[i,j] = \min_{\substack{0<k<i-1 \\ 0<l<j-1}} D[k,l] + w(k+l, i+j). \tag{15}$$

The function w corresponds to the energy cost of an internal loop between the two base pairs, and w' corresponds to the cost of a bulge. Both w and w' typically combine terms for the loop length and for the binding energy of bases i and j. Experimental results [70] show that both w and w' are convex function, i.e. they satisfy equation 3. The function $b(i, j)$ contains only the base pair binding energy term, and corresponds to the energy gain of a stacked pair. As we have said

before, recurrence 12 must be computed on all entries (i, j) in a given set S. Clearly, V, H and E must be computed on the same set of points.

First note that the computation of $V[i, j]$ within a fixed column j does not depend on that of other columns, except indirectly via the values of $D[i, j]$. We may perform this computation using the algorithm of Galil and Giancarlo [16]. If the number of points i in column j such that $(i, j) \in S$ is denoted by p_j, then the time for computing all values of $V[i, j]$ for a fixed j will be $O((p_j + p_{j-1}) \log M)$. The total time for these computations in all columns will then be $O(M \log M)$. We could achieve even better bounds using the more complicated algorithms by Klawe and Kleitman [30] or Eppstein [9] but this would not affect our total time bounds.

The computation of $H[i, j]$ is similar. Therefore the remaining difficulty is the computation of $E[i, j]$, as defined by 15. For this problem, each point in S may be considered as having a *range of influence* consisting of the region of the dynamic programming matrix E (and D) below and to the right of it. Thus, the range of each point is a quarter-plane with vertical and horizontal boundaries. For any given point (i, j), there is a point (i', j') containing (i, j) in its range and such that (i', j') provides the minimum in recurrence 15 for (i, j). Obviously, (i, j) can be contained in the range of influence of many points. Thus, when several points have intersecting ranges, we must compute which of them supply the minima in the intersection. The methods we use to perform this computation include matrix searching [1] together with binary search and divide and conquer. The reader is referred to [12] for a complete account of how such techniques yield an $O(n+M \log M \log \min(M, n^2/M))$ time algorithm for the computation of recurrence 12. When the cost function is simple, in addition to being convex, the binary search can be eliminated, and the time bound reduces to $O(n + M \log M \log \log \min(M, n^2/M))$. Since the function w typically used for the free energy of an internal loop is a logarithm and thus a simple convex function, we can solve the RNA secondary structure problem in $O(n + M \log M \log \log \min(M, n^2/M))$ time. We remark that, as in the case of the algorithm by [10], our algorithm can be used to speed up the algorithms by Kruskal et al. [50] and Zuker and Stiegler [71].

For concave cost functions w and w', we can compute recurrence 12 in the same time bounds as for convex cost function by using the same algorithm. When the cost functions are linear, we have a different algorithm that computes recurrence 12 in time $O(n + M \log \log \min(M, n^2/M))$ [11]. The concave result is of some merit from the combinatorial point of view but gives no contribution to computational biology since concave cost functions are not meaningful for the computation of RNA secondary structure. The assumption of linearity is also not as realistic as that of convexity, but it has been used in practice because of the increased efficiency of the corresponding algorithms (see for instance [28]). As can be seen from the bounds above, our new algorithms are again somewhat more efficient in the linear case than in the convex case.

We mention that very recently Larmore and Schieber [33] showed how to reduce the $O(n + M \log M \log \min(M, n^2/M))$ bound to $O(n + M \log \min(M, n^2/M))$ for the concave case and to $O(n + M\alpha(\min(M, n)) \log \min(M, n^2/M))$ for the convex case.

4. CONCLUSIONS AND OPEN PROBLEMS

We have considered two problems which benefit from mathematical methods applied to molecular biology: sequence alignment and computation of single loop RNA secondary structure. The com-

mon unifying framework for these two problems is that they both can be solved by computing a set of dynamic programming equations and that these equations need not be computed for all points in their domain of definition. Moreover, the cost functions that are typically used satisfy convexity or concavity constraints.

We have shown that it is possible to obtain asymptotically fast algorithms for both problem. Our algorithms are robust in the sense that they do not depend on any heuristic knowledge used to make the domain of the dynamic programming equations sparse. The crucial idea behind the new algorithms is to consider the computation of the given dynamic programming equations as a geometric problem that consists of identifying and maintaining a map of regions in the dynamic programming matrix. The algorithmic techniques that we use are a sophisticated upgrade of basic computer science tools such as divide-and-conquer and efficient data structures for answering queries.

Our results represent a contribution to the general problem of finding efficient algorithms for computationally intensive tasks posed by molecular biology and many interesting and hard problems remain in this area. The reader can refer to a recent paper by C. DeLisi for a lucid presentation of the computational needs that future advances in molecular biology pose [8]. Here we limit ourselves to mentioning a few open problems that are tightly related to the topic of this paper. We also feel that the solution to those problems could possibly yield new algorithmic techniques interesting both for computer science and for the mathematics of molecular biology. Indeed, in our opinion, a closer interaction between computer scientists and molecular biologists is going to be beneficial to both fields.

- Can the $O(nm\alpha(n))$ bound of the convex sequence alignment problem be improved? Can a practical algorithm achieve this goal?
- Can the space for convex or concave sequence alignment be reduced, similarly to Hirschberg's reduction for linear sequence alignment? Galil and Rabani [17] have shown that the current algorithms require space $O(nm)$, even using Hirschberg's technique, and so new methods would be needed.
- Can the bounds for our fragment alignment algorithms be reduced? In particular can we achieve the optimal time bounds of $O(M+n)$ for linear and/or convex (concave) cost functions?
- Can the times for the other RNA secondary structure computations be reduced? Can convexity or concavity of loop energy cost functions be used to speed up the computation of multiple-loop RNA secondary structure?
- Can the space of our algorithms, and the other algorithms for single loop RNA structure, be reduced below the current $O(n^2)$ bound? Can the space for efficient computation of multiple loop RNA structure with general (or convex or concave) cost functions be reduced below $O(n^3)$?
- Is dynamic programming strictly necessary to solve sequence alignment problems? Notice that algorithms based on dynamic programming will take at least $O(mn)$ time in aligning two sequences of length m and n.

Acknowledgments

We would like to thank Stuart Haber, Hugo Martinez, Peter Sellers, Temple Smith, and Michael Waterman for useful comments.

References

[1] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter Shor, and Robert Wilber, Geometric Applications of a Matrix-Searching Algorithm, Algorithmica 2, 1987, pp. 209–233.

[2] Alok Aggarwal and James Park, Searching in Multidimensional Monotone Matrices, 29th IEEE Symp. Found. Comput. Sci., 1988, pp. 497–512.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[4] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, Data Structures and Algorithms, Addison-Wesley, 1983.

[5] A. Apostolico and C. Guerra, The Longest Common Subsequence Problem Revisited, Algorithmica 2, 1987, pp. 315–336.

[6] J.L. Bentley and J.B. Saxe, Decomposable Searching Problems I: Static-to-Dynamic Transformation. J. Algorithms 1(4), December 1980, pp. 301–358.

[7] H.S. Bilofsky, C. Burks, J.W. Fickett, W.B. Goad, F.I. Lewitter, W.P. Rindone, C.D. Swindel, and C.S. Tung, The GenBank Genetic Sequence Databank, Nucl. Acids Res. 14, 1986, pp. 1–4.

[8] Charles DeLisi, Computers in Molecular Biology: Current Applications and Emerging Trends, Science, 240, 1988, pp. 47–52.

[9] David Eppstein, Sequence Comparison with Mixed Convex and Concave Costs, J. of Algorithms, 11, 1990, pp. 85–101.

[10] David Eppstein, Zvi Galil, and Raffaele Giancarlo, Speeding Up Dynamic Programming, 29th IEEE Symp. Found. Comput. Sci., 1988, pp. 488–496.

[11] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano, Sparse Dynamic Programming I: Linear Cost Functions, J. ACM, to appear.

[12] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano, Sparse Dynamic Programming II: Convex and Concave Cost Functions, J. ACM, to appear.

[13] Michael J. Fischer and R. Wagner, The String to String Correction Problem, J. ACM 21, 1974, pp. 168–178.

[14] Walter M. Fitch, Weighted Parsimony, Workshop on Algorithms for Molecular Genetics, Washington D.C., 1988.

[15] Walter M. Fitch and Temple F. Smith, Optimal Sequence Alignment, Proc. Nat. Acad. Sci. USA 80, 1983, pp. 1382–1385.

[16] Zvi Galil and Raffaele Giancarlo, Speeding Up Dynamic Programming with Applications to Molecular Biology, Theor. Comput. Sci., 64, 1989, pp. 107–118.

[17] Zvi Galil and Y. Rabani, On the Space Requirement for Computing Edit Distances with Convex or Concave Gap Costs, manuscript.

[18] Michael R. Garey and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.

[19] O. Gotoh, An Improved Algorithm for Matching Biological Sequences, J. Mol. Biol. 162, 1982, pp. 705–708.

[20] G.H. Hamm and G.N. Cameron, The EMBL Data Library, Nucl. Acids Res. 14, 1986, pp. 5–9.

[21] J.P. Haton, Practical Application of a Real-Time Isolated-Word Recognition System using Syntactic Constraints, IEEE Trans. Acoustics, Speech and Signal Proc. ASSP-22(6), 1974, pp. 416–419.

[22] D.S. Hirschberg, A Linear Space Algorithm for Computing Maximal Common Subsequences, Comm. ACM 18, 1975, pp. 341–343.

[23] D.S. Hirschberg, Algorithms for the Longest Common Subsequence Problem, J. ACM 24, 1977, pp. 664–675.

[24] D.S. Hirschberg and L.L. Larmore, The Least Weight Subsequence Problem, 26th IEEE Symp. Found. Comput. Sci., 1985, 137–143, and SIAM J. Comput. 16, 1987, pp. 628–638.

[25] M.K. Hobish, The Role of the Computer in Estimates of DNA Nucleotide Sequence Divergence, in S.K. Dutta, ed., DNA Systematics, Volume I: Evolution, CRC Press, 1986.

[26] J.W. Hunt and T.G. Szymanski, A Fast Algorithm for Computing Longest Common Subsequences, C. ACM 20(5), 1977, pp. 350–353.

[27] Donald B. Johnson, A Priority Queue in Which Initialization and Queue Operations Take $O(\log \log D)$ Time, Math. Sys. Th. 15, 1982, pp. 295–309.

[28] M.I. Kanehisi and W.B. Goad, Pattern Recognition in Nucleic Acid Sequences II: An Efficient Method for Finding Locally Stable Secondary Structures, Nucl. Acids Res. 10(1), 1982, pp. 265–277.

[29] Zvi M. Kedem and Henry Fuchs, On Finding Several Shortest Paths in Certain Graphs, 18th Allerton Conf., 1980, pp. 677–686.

[30] Maria M. Klawe and D. Kleitman, An Almost Linear Algorithm for Generalized Matrix Searching, Tech. Rep. IBM Almaden Research Center, 1988.

[31] Donald E. Knuth and Michael F. Plass, Breaking Paragraphs into Lines, Software Practice and Experience 11, 1981, pp. 1119–1184.

[32] A. G. Ivanov, Distinguishing an approximate word's inclusion on Turing machine in real time, Izv. Acad. Nauk USSR Ser. Mat. 48, 1984, pp. 520–568.

[33] L.L. Larmore and B. Schieber, On-Line Dynamic Programming with Applications to the Prediction of RNA Secondary Structure, J. Algorithms, to appear.

[34] V.I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Sov. Phys. Dokl. 10, 1966, pp. 707–710.

[35] D. Maier, The Complexity of Some Problems on Subsequences and Supersequences, J. ACM 25, 1978, pp. 322–336.

[36] T. Maniatis, Recombinant DNA, in D.M. Prescott, ed., Cell Biology, Academic Press, New York, 1980.

[37] Hugo Martinez, Extending RNA Secondary Structure Predictions to Include Pseudoknots, Workshop on Algorithms for Molecular Genetics, Washington D.C., 1988.

[38] W.J. Masek and M.S. Paterson, A Faster Algorithm Computing String Edit Distances, J. Comp. Sys. Sci. 20, 1980, pp. 18–31.

[39] A.M. Maxam and W. Gilbert, Sequencing End-Labeled DNA with Base Specific Chemical Cleavages, Meth. Enzymol. 65, 1980, p. 499.

[40] Webb Miller and Eugene W. Myers, Sequence Comparison with Concave Weighting Functions, Bull. Math. Biol., 50(2), 1988, pp. 97–120.

[41] S.B. Needleman and 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, p. 443.

[42] Ruth Nussinov, George Pieczenik, Jerrold R. Griggs, and Daniel J. Kleitman, Algorithms for Loop Matchings, SIAM J. Appl. Math. 35(1), 1978, pp. 68–82.

[43] Ruth Nussinov and A. Jacobson, Fast Algorithm for Predicting the Secondary Structure of Single-Stranded RNA, Proc. Nat. Acad. Sci. USA 77, 1980, pp. 6309–6313.

[44] G. N. Reeke, Protein Folding: Computational Approaches to an Exponential-Time Problem, Ann. Rev. Comput. Sci. 3, 1988, pp. 59–84.

[45] T.A. Reichert, D.N. Cohen, and A.K.C. Wong, An Application of Information Theory to Genetic Mutations and the Matching of Polypeptide Sequences, J. Theor. Biol. 42, 1973, pp. 245–261.

[46] H. Sakoe and S. Chiba, A Dynamic-Programming Approach to Continuous Speech Recognition, Proc. Int. Cong. Acoustics, Budapest, 1971, Paper 20 C 13.

[47] F. Sanger, S. Nicklen, and A.R. Coulson, Chain Sequencing with Chain-Terminating Inhibitors, Proc. Nat. Acad. Sci. USA 74, 1977, 5463.

[48] David Sankoff, Matching Sequences under Deletion-Insertion Constraints, Proc. Nat. Acad. Sci. USA 69, 1972, pp. 4–6.

[49] David Sankoff, Simultaneous Solution of the RNA Folding, Alignment and Protosequence Problems, SIAM J. Appl. Math. 45(5), 1985, pp. 810–825.

[50] David Sankoff, Joseph B. Kruskal, Sylvie Mainville, and Robert J. Cedergren, Fast Algorithms to Determine RNA Secondary Structures Containing Multiple Loops, in D. Sankoff and J.B. Kruskal, editors, Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, Addison-Wesley, 1983, pp. 93–120.

[51] David Sankoff and Joseph B. Kruskal, editors, Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, Addison-Wesley, 1983.

[52] P.H. Sellers, On the Theory and Computation of Evolutionary Distance, SIAM J. Appl. Math. 26, 1974, pp. 787–793.

[53] P.H. Sellers, Personal Communication, 1989.

[54] Temple Smith and Michael S. Waterman, Identification of Common Molecular Subsequences, J. Mol. Biol. 147 (1981), pp. 195–197.

[55] E. Ukkonen, On approximate string matching, J. of Algorithms, 6, 1985, pp. 132–137.

[56] V.M. Velichko and N.G. Zagoruyko, Automatic Recognition of 200 Words, Int. J. Man-Machine Studies 2, 1970, pp. 223–234.

[57] T.K. Vintsyuk, Speech Discrimination by Dynamic Programming, Cybernetics 4(1), 1968, 52–57; Russian Kibernetika 4(1), 1968, pp. 81–88.

[58] R.A. Wagner, On the Complexity of the Extended String-to-String Correction Problem, 7th ACM Symp. Theory of Computing, 1975, pp. 218–223.

[59] Michael S. Waterman, Sequence alignments in the neighborhood of the optimum with general applications to dynamic programming, Proc. Natl. Acad. Sci. USA, 80, 1983, pp. 3123–3124.

[60] Michael S. Waterman, Efficient Sequence Alignment Algorithms, J. of Theor. Biol., 108, 1984, pp. 333.

[61] Michael S. Waterman, General Methods of Sequence Comparison, Bull. Math. Biol. 46, 1984, pp. 473–501.

[62] Michael S. Waterman Editor, Mathematical Methods for DNA Sequences, CRC Press, Inc., 1988.

[63] Michael S. Waterman and Temple F. Smith, RNA Secondary Structure: A Complete Mathematical Analysis, Math. Biosciences 42, 1978, pp. 257–266.

[64] Michael S. Waterman and Temple F. Smith, New Stratigraphic Correlation Techniques, J. Geol. 88, 1980, pp. 451–457.

[65] Michael S. Waterman and Temple F. Smith, Rapid Dynamic Programming Algorithms for RNA Secondary Structure, Adv. Appl. Math. 7, 1986, pp. 455–464.

[66] Michael S. Waterman, Temple F. Smith, and W.A. Beyer, Some Biological Sequence Metrics, Adv. Math. 20, 1976, pp. 367–387.

[67] Robert Wilber, The Concave Least Weight Subsequence Problem Revisited, J. Algorithms 9(3), 1988, pp. 418–425.

[68] W.J. Wilbur and D.J. Lipman, Rapid Similarity Searches of Nucleic Acid and Protein Data Banks, Proc. Nat. Acad. Sci. USA 80, 1983, pp. 726–730.

[69] W.J. Wilbur and David J. Lipman, The Context Dependent Comparison of Biological Sequences, SIAM J. Appl. Math. 44(3), 1984, pp. 557–567.

[70] M. Zucker, The Use of Dynamic Programming Algorithms in RNA Secondary Structure Prediction, in M.S. Waterman editor, Mathematical Methods for DNA Sequences, CRC Press, 1988, pp. 159–184.

[71] M. Zuker, and P. Stiegler, Optimal Computer Folding of Large RNA Sequences using Thermodynamics and Auxiliary Information, Nucl. Acids Res. 9, 1981, pp. 133.