# A Linear Time Algorithm for Finding All Maximal Scoring Subsequences

**Walter L. Ruzzo** and **Martin Tompa**

Department of Computer Science and Engineering
Box 352350
University of Washington
Seattle, WA 98195-2350, U.S.A.
{ruzzo,tompa}@cs.washington.edu

## Abstract

Given a sequence of real numbers ("scores"), we present a practical linear time algorithm to find those nonoverlapping, contiguous subsequences having greatest total scores. This improves on the best previously known algorithm, which requires quadratic time in the worst case. The problem arises in biological sequence analysis, where the high-scoring subsequences correspond to regions of unusual composition in a nucleic acid or protein sequence. For instance, Altschul, Karlin, and others have used this approach to identify transmembrane regions, DNA binding domains, and regions of high charge in proteins.

**Keywords:** maximal scoring subsequence, locally optimal subsequence, maximum sum interval, sequence analysis.

## 1 Introduction

When analyzing long nucleic acid or protein sequences, the identification of unusual subsequences is an important task, since such features may be biologically significant. A common approach is to assign a score to each residue, and then look for contiguous subsequences with high total scores. This natural approach was analyzed by Altschul and Erickson (1986a; 1986b), Karlin and Altschul (1990; 1993), (Dembo & Karlin 1991; Karlin & Dembo 1992), and (Karlin, Dembo, & Kawabata 1990), and applied to a variety of protein analyses such as the identification of transmembrane regions, DNA binding domains, and regions of high charge (Brendel *et al.* 1992; Karlin & Brendel 1992; Karlin *et al.* 1991). The method is also applicable to long genomic DNA sequences, where computation time is of more concern.

As an example of this scoring approach, consider the application to identifying transmembrane domains in proteins. Transmembrane domains are rich in hydrophobic residues, so Karlin and Brendel (1992) adapted the hydropathy index of Kyte and Doolittle (1982) to assign scores to the 20 amino acids ranging from $-5$ (least hydrophobic) to $+3$ (most hydrophobic). They then looked for those (contiguous and disjoint) subsequences of the human $\beta_2$-adrenergic receptor sequence with the highest total residue scores, and observed that they correspond to the known transmembrane domains of the receptor.

Karlin and Brendel went on to propose a scoring scheme more specific to identifying transmembrane domains than the simple hydropathy index. They determined the frequency $q_i$ of each residue $i$ among the annotated transmembrane domains from 980 entries in a protein database, and the corresponding background frequency $p_i$ of each residue in the same set of 980 proteins. They then assigned a score of $s_i = \ln(q_i/p_i)$ (resembling a log likelihood ratio) to each residue $i$ in the protein sequence to be analyzed, in this instance the human $\beta_2$-adrenergic receptor. This scoring function has the property that subsequences of high total score have amino acid composition more closely resembling that of the known transmembrane domains than that of the protein collection overall, and hence are candidate transmembrane domains themselves. In fact, Karlin and Altschul (1990) demonstrated that such log likelihood ratios form the optimal scores, assuming that the target and background frequencies are accurate. Returning to the human $\beta_2$-adrenergic receptor, Karlin and Brendel observed that the highest scoring subsequences were similar to the ones obtained with the hydropathy scores, but were more pronounced.

Karlin and Altschul (1993) applied the same scoring function to identify transmembrane domains in the *Drosophila virilis* sevenless protein, and in the human serotonin receptor. The authors' emphasis in that paper was on finding multiple disjoint high scoring subsequences corresponding, for instance, to multiple transmembrane domains in a single protein. Altschul (1997) provided additional analysis of the statistical significance of multiple disjoint high scoring subsequences. Our emphasis in this paper is on the same problem of identifying multiple high scoring subsequences.

We present an $O(n)$ time algorithm for finding *all* maximal scoring subsequences in a given sequence of length $n$. There is a classical $O(n)$ time algorithm for finding the *single* maximum scoring subsequence, from which it follows that *all* maximal scoring subsequences can be found in $O(n^2)$ time in the worst case. Worst case behavior is rare in practice. Running times of order $n \log n$ are perhaps more representative of the previously best known algorithm's behavior, and an expected running time of $\Theta(n \log n)$ is provable for an unrealistic but suggestive model. (See Section 2.) While that algorithm may be fast enough to be useful in practice, ours is an order of magnitude faster on problems of realistic size. Furthermore, it is not substantially more complex, and presents some interesting subtleties in its design and analysis. Finally, we hope better understanding of this problem will lead to improvements in performance of algorithms for more complex scoring schemes (P. Green (1997)), alternative definitions of maximal or optimal subsequences (Sellers (1984)), or for finding certain kinds of maximal subalignments (Altschul and Erickson (1986a; 1986b), Sellers (1984), Waterman and Eggert (1987)), for which the best known algorithms are quadratic or worse. Note that our algorithm immediately gives a quadratic method quite different from the usual dynamic programming methods for finding maximal (gapless) subalignments; perhaps further improvements are possible.

Section 2 defines the computational problem to be solved, and describes the best previously known algorithm for it. Section 3 provides an alternative characterization of the problem, and gives some basic facts about maximal scoring subsequences. Section 4 presents our algorithm, and discusses its correctness and analysis. Section 5 outlines some experimental results.

## 2 Problem Definition and Previously Known Algorithms

**Problem Definition.** The input is a sequence $(x_1, x_2, \ldots, x_n)$ of (not necessarily positive) real numbers, called "scores." The goal is to identify those contiguous subsequences having the greatest total scores, where the score $S_{i,j}$ of a subsequence $(x_i, x_{i+1}, \ldots, x_j)$ is simply the sum of its elements:

$$S_{i,j} = \sum_{i \le k \le j} x_k.$$

Throughout the paper, the term "subsequence" will be taken to mean "contiguous subsequence", and likewise for "supersequence". Contiguous subsequences are sometimes called *substrings*, *segments*, *intervals*, or *regions*. Likewise, maximal scoring subsequences (defined below) are sometimes called *locally optimal subsequences* or *maximum sum intervals*.

There is a subtlety in defining exactly what constitutes a maximal scoring subsequence. Temporarily setting aside the handling of tied scores, the highest scoring subsequence is simply the subsequence $(x_i, x_{i+1}, \ldots, x_j)$ that maximizes $S_{i,j}$. It is not so clear, however, what the second best subsequence should be. The subsequence with the second highest numerical score very likely will overlap the highest scoring subsequence except for the addition or deletion of a few scores at one end. Given the motivating application, this conveys no useful information. Instead, the $k^{th}$ best subsequence will be defined to be the one that maximizes $S_{i,j}$ among those subsequences *disjoint* from the $k-1$ best subsequences. Additionally, to avoid trivialities, we stop the process when the next best score is nonpositive. Returning to the matter of tied scores, a zero-scoring subsequence adjacent to a positive-scoring one creates overlapping subsequences with tied scores. We resolve these ties by disallowing nonempty, zero-scoring prefixes or suffixes in maximal scoring subsequences.

We define a *maximal scoring subsequence* (or *maximal subsequence* for short) to be any of the (positive scoring) subsequences found by the process described in the previous paragraph, and the desired output is a list of all these subsequences. In practice, of course, scores below a certain threshold might be discarded. Karlin and Altschul (1993) describe how such a threshold should be chosen to correspond to a desired level of statistical significance.

As an example, consider the input sequence $(4, -5, 3, -3, 1, 2, -2, 2, -2, 1, 5)$. The highest scoring subsequence is $M = (1, 2, -2, 2, -2, 1, 5)$, with a total score of 7. (There is another subsequence tied for this score by appending $(3, -3)$ to the left end of $M$, but this subsequence is not maximal, since it has a nonempty zero-scoring prefix.) Thus, the maximal subsequences are $(4)$, $(3)$, and $M$.

**Previously Known Algorithm.** Although the definition of the single highest scoring subsequence suggests that quadratic time would be needed to search over all combinations of $i$ and $j$, there is a well known linear time algorithm for finding the single maximum scoring subsequence; cf. Bates and Constable (1985), Bentley (1986, Column 7), or Manber (1989, Section 5.8). (It can also be found by a specialization of the Smith-Waterman algorithm (Smith & Waterman 1981).) The disjointness property immediately suggests a simple divide-and-conquer algorithm for finding *all* maximal subsequences: find the highest scoring subsequence, remove it, and then apply the algorithm recursively to the remainder of the sequence to the left of the removed portion, and then to the remainder to the right.

Analysis of this algorithm is similar to that for Quicksort. (For an analysis of Quicksort see, for example, Manber (1989).) In the worst case, it will require quadratic time, since the next highest scoring subsequence may be a short subsequence near one end of the remaining sequence in every recursive call. However, if the input is appropriately random, the expected running time will be $\Theta(n \log n)$, since the highest scoring subsequence will usually fall nearer to the middle. This result holds if (i) the residues in the sequence are chosen independently at random, and (ii) the expected score of a single random residue is negative. Assumption (ii) is a reasonable one (Karlin & Altschul 1990; 1993). Note in particular that if the expected score were positive, then the highest scoring subsequence would likely include most of the sequence, an uninteresting situation. Moreover, if log likelihood ratios are used as scores, as described in Section 1, then the expected score is the negative of a relative entropy, and is thus provably nonpositive. However, assumption (i) is decidedly unreasonable in practice: we are not interested in regions of unusual composition in random sequences. Nevertheless, the $n \log n$ result is suggestive, and in accord with the performance observed in practice. (See Section 5.)

## 3 Alternative Characterization

The procedural definition of maximal scoring subsequences presented above, while well motivated, is somewhat difficult to use for many purposes, and a more direct definition is preferable. Intuitively, the desired subsequences are "maximal" in the sense that they cannot be lengthened or shortened without reducing their scores. This simple intuition, however, is too simple. For example, in the score sequence $(5,-4.9,1,3)$, both (5) and (1, 3) are maximal: (5) is the highest scoring subsequence, and (1, 3) is the highest scoring subsequence disjoint from (5). Under the naive definition suggested above, however, (5) would be the only maximal scoring subsequence: although subsequence (1, 3) has nearly as high a score, it would be excluded because it can be extended to form a higher scoring sequence $(5,-4.9,1,3)$, which in turn can be contracted to form the still higher scoring sequence (5). The following alternative characterization of the maximal scoring subsequences, due to Green (1997), corrects this problem.

**Proposition 1** *Let K be any nonempty score sequence. A subsequence I is maximal scoring in K if and only if*

*P1. all proper subsequences of I have lower score, and*

*P2. no proper supersequence of I contained in K satisfies P1.*

For the remainder of the paper, we will take this as the definition of a maximal scoring subsequence, proving its equivalence to the original procedural definition at the end of this section. We first develop a variety of useful consequences of the new definition. For example, we show that every nonempty prefix and suffix of a maximal scoring subsequence has positive score, and that every subsequence satisfying P1 (and hence every individual positive score) is contained in some maximal scoring subsequence. Some of these properties are needed for the proof of Proposition 1, and some for the correctness of the algorithm in Section 4.

Let $|K|$ denote the length of a sequence $K$.

**Lemma 2** *Let K be any nonempty score sequence. For any subsequence I of K the following are equivalent:*

1. *I satisfies property P1.*
2. *For $0 \leq i \leq |I|$, let $S_i$ denote the cumulative total of all scores in K up to and including the $i^{th}$ score of I. Then $S_0$ is the unique minimum and $S_{|I|}$ is the unique maximum of the $S_i$'s.*
3. *All nonempty prefixes and all nonempty suffixes of I have positive total score.*

**Proof:** $1 \Rightarrow 2$: If for some $0 \leq i < |I|$ we have $S_i \geq S_{|I|}$, then the proper prefix of $I$ with length $i$ has total score at least that of $I$. Similarly, if $S_i \leq S_0$ for some $0 < i \leq |I|$, then the length $(|I|-i)$ suffix of $I$ has total score at least that of $I$.

$2 \Rightarrow 3$: The total score of the length $i$ prefix of $I$ is $S_i - S_0$, which is positive since $S_0$ is the unique minimum. Similarly, a suffix will have score $S_{|I|} - S_i$, which is positive since $S_{|I|}$ is the unique maximum.

$3 \Rightarrow 1$: The score of any proper subsequence $J$ of $I$ will be the total score of $I$ minus the scores of the prefix of $I$ to the left of $J$ and the suffix of $I$ to the right of $J$. Since nonempty prefixes and suffixes all have positive scores, $J$ will have a strictly lower score than $I$. □

**Lemma 3** *Let K be any nonempty score sequence. The maximal subsequences of K are pairwise disjoint (neither overlapping nor abutting).*

**Proof:** Suppose $I$ and $J$ are maximal and nondisjoint, with $I$'s left end at least as far left as $J$'s. Let $L$ be the union of the intervals $I$ and $J$. By Lemma 2, the minimum cumulative scores within $I$ and $J$ occur uniquely at their respective left ends, and the left end of $J$ falls within (or at the right end of) $I$, so the minimum cumulative score within $L$ occurs uniquely at its left end. Similarly, $L$'s maximum cumulative score occurs uniquely at its right end. Thus by Lemma 2, $L$ satisfies property P1, contradicting the P2 property of either $I$ or $J$. □

**Lemma 4** *Let K be any nonempty score sequence. Every subsequence I of K satisfying property P1 is contained in a maximal subsequence of K. In particular, every positive single score is contained in a maximal subsequence.*

**Proof:** Suppose not. Let $I$ be a counterexample of maximum length. $I$ satisfies property P1, but is not itself maximal, so it must be properly contained in some supersequence $J$ satisfying P1. $J$ cannot be contained in any maximal subsequence, for otherwise $I$ would be contained in a maximal subsequence. Thus $J$ is also a counterexample to the lemma, contradicting the choice of $I$ as a longest counterexample. □

**Corollary 5** *Within any subsequence that does not overlap any maximal subsequence, the cumulative scores are monotonically nonincreasing.*

**Lemma 6** *Let K be any nonempty score sequence. Among consecutive occurrences of the minimum (maximum) cumulative total of all scores in any prefix of K, the rightmost (leftmost, respectively) is either at the left (right, respectively) end of one of its maximal subsequences, or at the right (left, respectively) end of K.*

**Proof:** If a minimum cumulative score occurs in a subsequence $H$ that does not overlap any maximal subsequence, by Corollary 5 the rightmost of consecutive occurrences of this minimum must occur at the right end of $H$, which is either the right end of $K$ or the left end of some maximal subsequence. If the minimum cumulative score occurs within some maximal subsequence, it must occur at its left end, by Lemma 2. Arguments for the maximum cumulative score are dual. □

Suppose $J$ is a subsequence of a score sequence $K$. The key issue in our inductive correctness arguments turns out to be relating the maximal subsequences of $J$ to those of $K$. The following three lemmas give some of these relationships. Lemma 7 establishes the easy direction, and Lemmas 8 and 9 give useful partial converses. We sometimes say that a sequence is *I-maximal* as a shorthand for "maximal scoring subsequence of $I$."

**Lemma 7** *Let $K$ be any nonempty score sequence. If $I$ is a subsequence of $J$, which in turn is a subsequence of $K$, and $I$ is $K$-maximal, then $I$ is $J$-maximal.*

**Proof:** Since $I$ is $K$-maximal, it satisfies property P1, and no proper supersequence of $I$ in $K$ satisfies P1. Then certainly there is no proper supersequence of $I$ in $J$ satisfying P1. □

**Lemma 8** *Let $K$ be any nonempty score sequence. Suppose $J$ is a prefix of $K$, and suppose all scores in the suffix of $K$ following $J$ are nonpositive. Then the maximal subsequences of $K$ are exactly the same as those of $J$.*

**Proof:** If $I$ is $K$-maximal, then $I$ is $J$-maximal by Lemma 7. Conversely, if $I$ is $J$-maximal, then no proper supersequence of $I$ satisfying property P1 exists in $J$, and every supersequence extending outside of $J$ fails to satisfy property P1 by Lemma 2, since it has a nonempty nonpositive suffix. Thus, $I$ is $K$-maximal. □

**Lemma 9** *Let $K$ be any nonempty score sequence. Suppose $M$ is a maximal subsequence of $K$, and let $L$ and $R$ be the two (possibly empty) subsequences of $K$ formed by deletion of $M$. Then a subsequence $I$ of $L$ is $L$-maximal if and only if $I$ is $K$-maximal. Similarly, a subsequence of $R$ is $R$-maximal if and only if it is $K$-maximal.*

**Proof:** We will only give the argument for $L$, the argument for $R$ being identical.

The "if" direction is immediate from Lemma 7.

For the "only if" direction, suppose $I$ is $L$-maximal but not $K$-maximal. $I$ satisfies property P1, so by Lemma 4, $I$ is contained in some $J$ that is $K$-maximal. $J$ properly contains $I$, since $I$ is not $K$-maximal. Furthermore, $J$ cannot be contained in $L$, for otherwise $I$ is not $L$-maximal. But neither can $J$ extend past the end of $L$, because then it would overlap $M$, violating disjointness of $K$-maximal sequences (Lemma 3). □

Finally we prove Proposition 1, establishing the equivalence of the two alternative definitions of "maximal scoring subsequence" considered above. That is, we show that a subsequence is maximal according to the original procedural definition given in Section 2 if and only if it satisfies properties P1 and P2 of Proposition 1.

**Proof** (of Proposition 1): Let $\textsc{Max}(K)$ be the complete list of subsequences satisfying P1 and P2 in $K$. Any positive-scoring subsequence $M$ having a globally maximum score and no zero-scoring prefix or suffix clearly satisfies properties P1 and P2. Hence, $\textsc{Max}(K)$ is the empty list if $K$ consists only of nonpositive scores (by Lemma 2), and otherwise is $\langle \textsc{Max}(L), M, \textsc{Max}(R)\rangle$ (easily shown by Lemma 9 and induction on $|K|$), which is exactly the list produced by the recursive algorithm in Section 2. □

## 4 New Algorithm

We now present an algorithm that finds all maximal scoring subsequences in time $O(n)$. We first describe the algorithm, and then discuss its correctness and performance.

**Algorithm.** The algorithm reads the scores from left to right, and maintains the cumulative total of the scores read so far. Additionally, it maintains a certain ordered list $I_1, I_2, \ldots, I_{k-1}$ of disjoint subsequences. For each such subsequence $I_j$, it records the cumulative total $L_j$ of all scores up to but not including the leftmost score of $I_j$, and the total $R_j$ up to and including the rightmost score of $I_j$.

The list is initially empty. Input scores are processed as follows. A nonpositive score requires no special processing when read. A positive score is incorporated into a new subsequence $I_k$ of length one[1] that is then integrated into the list by the following process.

1. The list is searched from right to left for the maximum value of $j$ satisfying $L_j < L_k$.
2. If there is no such $j$, then add $I_k$ to the end of the list.
3. If there is such a $j$, and $R_j \geq R_k$, then add $I_k$ to the end of the list.
4. Otherwise (i.e., there is such a $j$, but $R_j < R_k$), extend the subsequence $I_k$ to the left to encompass everything up to and including the leftmost score in $I_j$. Delete subsequences $I_j, I_{j+1}, \ldots, I_{k-1}$ from the list (none of them is maximal) and reconsider the newly extended subsequence $I_k$ (now renumbered $I_j$) as in step 1.

After the end of the input is reached, all subsequences remaining on the list are maximal; output them.

As an example of the execution of the algorithm, consider the sample input from Section 2. After reading the scores $(4,-5,3,-3,1,2,-2,2)$, suppose the list of disjoint subsequences is $I_1 = (4), I_2 = (3), I_3 = (1,2), I_4 = (2)$, with $(L_1,R_1) = (0,4)$, $(L_2,R_2) = (-1,2)$, $(L_3,R_3) = (-1,2)$, and $(L_4,R_4) = (0,2)$. (See Figure 1.) At this point, the cumulative score is 2. If the ninth input is $-2$, the list of subsequences is unchanged, but the cumulative score becomes 0. If the tenth input is 1, Step 1 produces $j = 3$, because $I_3$ is the rightmost subsequence with $L_3 < 0$. Now Step 3 applies, since $R_3 \geq 1$. Thus $I_5 = (1)$ is added to the list with $(L_5,R_5) = (0,1)$, and the cumulative score becomes 1. If the eleventh input is 5, Step 1 produces $j = 5$, and Step 4 applies, replacing $I_5$ by $(1,5)$ with $(L_5,R_5) = (0,6)$. The algorithm returns to Step 1 without reading further input, this time producing $j = 3$. Step 4 again applies, this time merging $I_3$, $I_4$, and $I_5$ into a new $I_3 = (1,2,-2,2,-2,1,5)$ with $(L_3,R_3) = (-1,6)$. The algorithm again returns to Step 1, but this time Step 2 applies. If there are no further input scores, the complete list of maximal subsequences is then $I_1 = (4), I_2 = (3), I_3 = (1,2,-2,2,-2,1,5)$, as in Section 2.

**Correctness.** The key to proving the correctness of the algorithm is the following lemma.

[1]In practice, one could optimize this slightly by processing a consecutive series of positive scores as $I_k$.

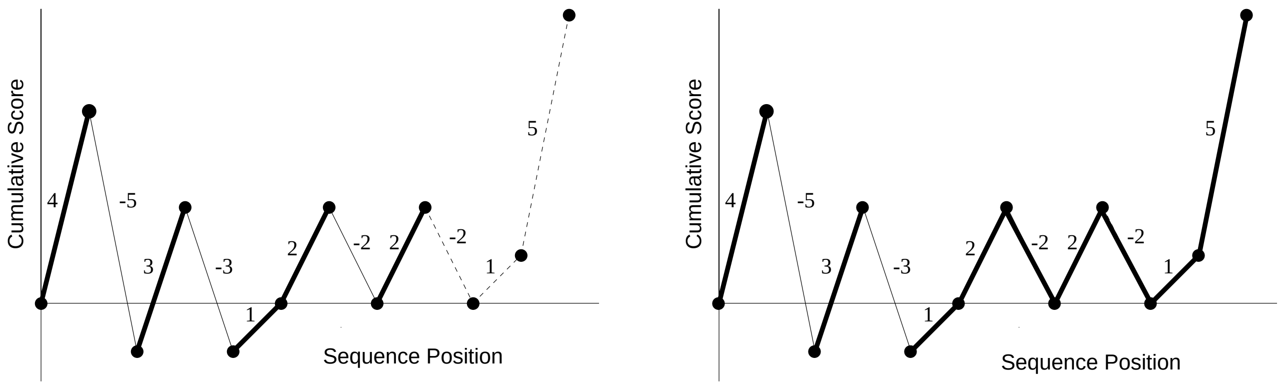

Figure 1: An example of the algorithm. Bold segments indicate score sequences currently in the algorithm's list. The left figure shows the state prior to adding the last three scores, and the right figure shows the state after.

**Lemma 10** *Let $Q$ be a score sequence. Suppose that a suffix $I_k$ of $Q$ satisfies property P1. Let $P$ be the prefix of $Q$ preceding $I_k$. Further suppose that $I_1,\ldots,I_{k-1}$ is the complete ordered list of the $P$-maximal subsequences. Then the subsequences $I'_1, I'_2,\ldots$ constructed from $I_1,\ldots,I_k$ by Steps 1–4 above form the complete ordered list of the $Q$-maximal sequences.*

The correctness of the algorithm follows from Lemma 10. Given an input sequence $T$, let $Q_i,\ 0 \le i \le |T|$, be the length $i$ prefix of $T$. We show by induction on $i$ that after processing the $i^{th}$ score, the list $I_1, I_2,\ldots$ constructed by the algorithm above is the complete ordered list of $Q_i$-maximal sequences. The basis, $i = 0$, is immediate. For the induction step, suppose the list consists of $I_1,\ldots,I_{k-1}$ when the $i^{th}$ score is read. If the $i^{th}$ score is nonpositive, then the maximal subsequences of $Q_i$ are exactly the same as the maximal subsequences of $Q_{i-1}$, by Lemma 8. If the $i^{th}$ score is positive, then it comprises the entire subsequence $I_k$ constructed by the algorithm, which clearly satisfies property P1. Furthermore, by the induction hypothesis, $I_1,\ldots,I_{k-1}$ is the complete ordered list of the $Q_{i-1}$-maximal subsequences, and so Lemma 10 implies that the algorithm correctly constructs the complete ordered list of $Q_i$-maximal sequences.

We now prove the Lemma.

**Proof** (of Lemma 10): There are three cases, paralleling Steps 2–4 of the algorithm.

CASE 1: Suppose Step 1 locates no $j$ such that $L_j < L_k$. Applying Lemma 6 to $P$ and Lemma 2 to $I_k$, $L_k$ is a minimum cumulative score in $Q$. Then applying Lemma 6 to $Q$ and Lemma 2 to $I_k$, some prefix $J$ of $I_k$ is $Q$-maximal. In order for $J$ to have property P2, $J = I_k$, so $I_k$ is $Q$-maximal. The fact that each $I_j,\ 0 < j < k$, is $Q$-maximal then follows from Lemma 9, by choosing $M = I_k$.

CASE 2: Suppose there is a $j$ such that $L_j < L_k$, and suppose for the greatest such $j$ that $R_j \ge R_k$. We claim that $I_j$ is $Q$-maximal. Suppose not. From the definition of maximality there must be some proper supersequence $J$ of $I_j$ satisfying property P1. Furthermore, this supersequence cannot lie totally within $P$ (otherwise $I_j$ would not be $P$-maximal). Consequently, $J$ must overlap $I_k$. The (nonempty) suffix of $J$ that begins at the right end of $I_j$ must have a nonpositive score (since $R_k$, and hence by Lemma 2 every cumulative total within $I_k$, is at most $R_j$). But this contradicts Lemma 2, which means $I_j$ is $Q$-maximal.

Let $S$ be the suffix of $Q$ that begins at the right end of $I_j$. Applying Lemma 6 as in Case 1, $I_k$ is $S$-maximal, since $L_k$ is a minimum cumulative score in $S$. By choosing $M = I_j$ in Lemma 9, $I_k$ is also $Q$-maximal. Then one more application of Lemma 9 with $M = I_k$ shows that $I_1, I_2,\ldots,I_{k-1}$ are all $Q$-maximal.

CASE 3: Suppose there is a $j$ such that $L_j < L_k$, and for the greatest such $j$ we have $R_j < R_k$. This is perhaps the most interesting case, in which several $P$-maximal subsequences are merged into one sequence having a greater total score. The merged sequence may not be $Q$-maximal, but will be shown to satisfy property P1, and so Lemma 10 may be applied inductively. Let $k'$ be the least index in the range $j < k' \le k$ such that $R_j < R_{k'}$. (Such a $k'$ exists since $k$ satisfies this property.) Let $J$ be the interval extending from the left end of $I_j$ to the right end of $I_{k'}$. By construction, all intervals $I_{j'},\ j < j' < k'$, have $L_j < L_{j'}$ and $R_{j'} < R_{k'}$, so by Lemma 6 (and Lemma 7), $L_j$ is the unique minimum and $R_{k'}$ the unique maximum among the cumulative totals within $J$. By Lemma 2, then, $J$ satisfies property P1. If $k' < k$, then $J$ lies wholly within $P$, contradicting the $P$-maximality of $I_j$, say. Hence we have $k' = k$, and $J$ is a suffix of $Q$ satisfying property P1. Let $P'$ be the prefix of $P$ to the left of $J$, and note that, by choosing $K = P$ and $M = I_j$ in Lemma 9, the complete list of $P'$-maximal subsequences is $I_1,\ldots,I_{j-1}$. Furthermore, $j < k$, so by induction on $k$, returning to Step 1 (as the algorithm does in Step 4) will correctly compute the maximal subsequences of $Q$. (The basis follows from Cases 1 and 2.) □

**Analysis.** There is an important optimization that may be made to the algorithm. In the case that Step 2 applies, not only are subsequences $I_1,\ldots,I_{k-1}$ maximal in $Q$, they are maximal in every sequence $R$ of which $Q$ is a prefix, and so may be output before reading any more of the input. (This is true since $L_k$ is a minimum cumulative score in $Q$, so any supersequence of $I_j$, $1 \leq j \leq k-1$, extending past the left end of $I_k$ will have a nonpositive prefix, hence is not $R$-maximal by Lemma 2.) Thus, Step 2 of the algorithm may be replaced by the following, which substantially reduces the memory requirements of the algorithm.

2.$'$ If there is no such $j$, all subsequences $I_1, I_2, \ldots, I_{k-1}$ are maximal. Output them, delete them from the list, and reinitialize the list to contain only $I_k$ (now renumbered $I_1$).

The algorithm as given does not run in linear time, because several successive executions of Step 1 might re-examine a number of list items. This problem is avoided by storing with each subsequence $I_k$ added during Step 3 a pointer to the subsequence $I_j$ that was discovered in Step 1. The resulting linked list of subsequences will have monotonically decreasing $L_j$ values, and can be searched in Step 1 in lieu of searching the full list. Once a list element has been bypassed by this chain, it will be examined again only if it is being deleted from the list, either in Step 2$'$ or Step 4. The work done in the "reconsider" loop of Step 4 can be amortized over the list item(s) being deleted. Hence, in effect, each list item is examined a bounded number of times, and the total running time is linear.

The worst case space complexity is also linear, although one would expect on average that the subsequence list would remain fairly short in the optimized version incorporating Step 2$'$: since the expected value of an individual score is negative, Step 2$'$ should occur fairly frequently. Empirically, a few hundred stack entries suffice for processing sequences of a few million residues, for either synthetic or real genomic data.

## 5 Experimental Results

We have implemented both our linear time algorithm and the previously known divide and conquer algorithm and compared their performances.

Our linear time algorithm is a factor of five faster even on sequences as short as 100 residues, and is 15 to 20 times faster on megabase sequences. It can process a one megabase score sequence in less than 100 milliseconds on a midrange PC. Obviously, saving a few seconds or tens of seconds on one analysis will not be critical in many circumstances, but may be more significant when analyzing long sequences, repeatedly analyzing sequences under varying scoring schemes, searching data bases, or when offered on a busy Web server, for example.

As noted in the introduction, the code for the linear time algorithm is somewhat more complex than that for the divide and conquer algorithm, but not substantially more complex. The core of the algorithm is well under 100 lines of C code.

Figure 2 gives comparative timing information for the two algorithms in one set of experiments. The figure plots $T/n$ versus $n$ (on a logarithmic scale), where $T$ is the running time of either algorithm and $n$ is the length of the input score sequence. As expected, the running time of the divide and conquer algorithm appears to be growing as $n \log n$, as evidenced by the linear growth of $T/n$ when plotted against $\log n$ in Figure 2. Also as expected, the running time of our algorithm is growing linearly with $n$, i.e., $T/n$ is nearly constant over the full range of lengths considered, from $n = 128$ to $n = 1,048,576$, with a mean of 80 nanoseconds per score.

The tests plotted in Figure 2 were performed on a Macintosh 9600/300, (300 MHz PowerPC 604e processor, 1MB inline cache, 64 MB RAM). The input length varied by factors of 2 from $2^7$ to $2^{20}$. Ten trials were run at each length, with integer scores drawn independently and uniformly from -5 to +4 (expectation -0.5). Each algorithm was run repeatedly on each random score sequence to compensate for limited resolution of the system clock, but we did not carefully control for cache effects. Each of the ten trials for each algorithm is plotted in Figure 2. It is interesting to note that the running time of our linear time algorithm is much less variable than that of the divide and conquer algorithm, and in fact most of its variation in these tests may be due to clock granularity.

The same tests using synthetic data were run on several platforms (Pentium II, SPARC, Alpha). Additionally, we ran tests using real genomic sequences (yeast and bacterial) of up to a few megabases in length. Relative performance of the two algorithms was similar for all these data sets on all platforms. The performance of neither algorithm appears to be particularly sensitive to the source of the data.

In the course of our timing tests, we ran a few simple experiments such as the following. In the *Haemophilus influenzae* genome, the dinucleotide $C_pG$ appears on average once every 25 bases. Karlin, Mrázek, and Campbell (1997) have reported that the $C_pG$ dinucleotide is rather uniformly distributed throughout the *H. influenzae* genome, and its frequency is generally explainable in terms of the underlying frequencies of the mononucleotides C and G. Specifically, they defined the quantity $\rho_{CG}$ to be $f_{CG}/f_C f_G$, where $f_{CG}$ is the frequency of $C_pG$ dinucleotides in a given region, and $f_C, f_G$ are the corresponding mononucleotide frequencies in that region. They observed that $\rho_{CG}$ is near 1 and nearly constant across successive 50kb contigs of the genome, in contrast to certain other dinucleotides. We ran our maximal subsequence algorithm on the complete genome, assigning a score of 20 to each $C_pG$ dinucleotide and a score of -1 to all others. As expected, since an "average" 25 residue sequence will have a net score of -4, most of the nineteen thousand maximal subsequences identified by the algorithm are quite short and have low scores. However, about 20 subsequences were found with lengths of two to twenty kilobases and (statistically significant (Karlin & Altschul 1990)) scores of several hundred to several thousand. A third of these are easily explained away as regions of unusually high C/G content, which can be expected to have more $C_pG$ dinucleotides (i.e., these regions also have $\rho_{CG}$ near 1). However, two thirds of the subsequences show a substantial enrichment in $C_pG$ dinucleotides as compared to the frequency of C/G mononucleotides in the same subsequence (high $\rho_{CG}$). Existence of these $C_pG$-rich regions does not contradict the observations

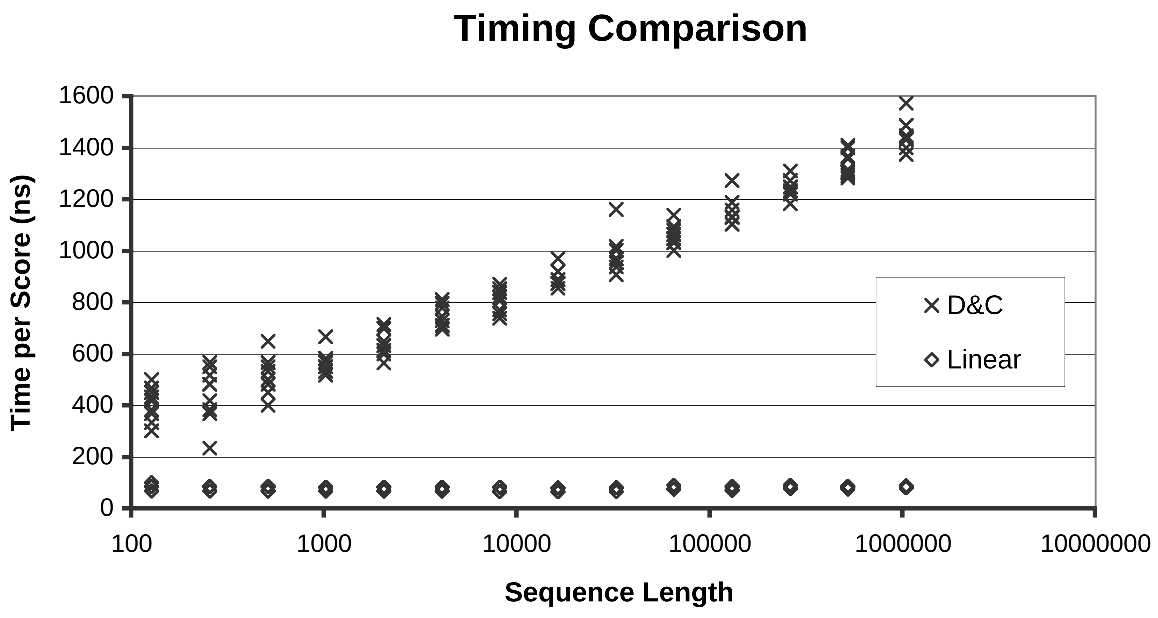

Figure 2: Timing comparison of the divide and conquer and linear time algorithms. See text for details.

of Karlin, Mrázek, and Campbell (1997) since they are only visible on a variable length scale shorter than 50kb. We do not claim that identification of these regions is novel, nor that they are biologically significant, but we do hope that availability of the fast algorithm for the very general scoring problem presented in this paper will help researchers identify features that are both novel and biologically significant.

## Acknowledgments

We thank Phil Green for presenting the problem to us, and for many enlightening discussions. Thanks to an anonymous referee for pointing out some relevant papers. This material is based upon work supported in part by the National Science Foundation and DARPA under grant #DBI-9601046.

## References

Altschul, S. F., and Erickson, B. W. 1986a. Locally optimal subalignments using nonlinear similarity functions. *Bulletin of Mathematical Biology* 48:633–660.

Altschul, S. F., and Erickson, B. W. 1986b. A nonlinear measure of subalignment similarity and its significance levels. *Bulletin of Mathematical Biology* 48:617–632.

Altschul, S. F. 1997. Evaluating the statistical significance of multiple distinct local alignments. In Suhai, S., ed., *Theoretical and Computational Methods in Genome Research*. New York: Plenum Press. 1–14.

Bates, J. L., and Constable, R. L. 1985. Proofs as programs. *ACM Transactions on Programming Languages and Systems* 7(1):113–136.

Bentley, J. 1986. *Programming Pearls*. Addison-Wesley.

Brendel, V.; Bucher, P.; Nourbakhsh, I. R.; Blaisdell, B. E.; and Karlin, S. 1992. Methods and algorithms for statistical analysis of protein sequences. *Proceedings of the National Academy of Science USA* 89(6):2002–2006.

Dembo, A., and Karlin, S. 1991. Strong limit theorems of empirical functionals for large exceedances of partial sums of IID variables. *Annals of Probability* 19(4):1737–1755.

Green, P. 1997. Detecting inhomogeniety in DNA and protein sequences. Computational Biology Seminar, Department of Molecular Biotechnology, University of Washington.

Karlin, S., and Altschul, S. F. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Science USA* 87(6):2264–2268.

Karlin, S., and Altschul, S. F. 1993. Applications and statistics for multiple high-scoring segments in molecular sequences. *Proceedings of the National Academy of Science USA* 90:5873–5877.

Karlin, S., and Brendel, V. 1992. Chance and significance in protein and DNA sequence analysis. *Science* 257:39–49.

Karlin, S., and Dembo, A. 1992. Limit distributions of maximal segmental score among Markov-dependent partial sums. *Advances in Applied Probability* 24:113–140.

Karlin, S.; Bucher, P.; Brendel, V.; and Altschul, S. F. 1991. Statistical-methods and insights for protein and DNA-sequences. *Annual Review of Biophysics and Biophysical Chemistry* 20:175–203.

Karlin, S.; Dembo, A.; and Kawabata, T. 1990. Statistical composition of high-scoring segments from molecular sequences. *Annals of Statistics* 18:571–581.

Karlin, S.; Mrázek, J.; and Campbell, A. M. 1997. Compositional biases of bacterial genomes and evolutionary implications. *Journal of Bacteriology* 179(12):3899–3913.

Kyte, J., and Doolittle, R. F. 1982. A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology* 157(1):105–132.

Manber, U. 1989. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley.

Sellers, P. H. 1984. Pattern recognition in genetic sequences by mismatch density. *Bulletin of Mathematical Biology* 46:501–514.

Smith, T. F., and Waterman, M. S. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147(1):195–197.

Waterman, M. S., and Eggert, M. 1987. A new algorithm for best subsequence alignments with applications to tRNA–rRNA comparisons. *Journal of Molecular Biology* 197:723–728.