# Computing All Suboptimal Alignments in Linear Space†

Kun-Mao Chao

Department of Computer Science and Engineering,
The Pennsylvania State University,
University Park, PA 16802, USA.

**Abstract.** Recently, a new compact representation for suboptimal alignments was proposed by Naor and Brutlag (1993). The kernel of that representation is a minimal directed acyclic graph (DAG) containing all suboptimal alignments. In this paper, we propose a method that computes such a DAG in space *linear* to the graph size. Let $F$ be the area of the region of the dynamic-programming matrix bounded by the suboptimal alignments and $W$ the maximum width of that region. For two sequences of lengths $M$ and $N$, it is shown that the worst-case running time is $O(MN + F \log\log W)$. To exploit the computed DAG, we employ a variant of Aho-Corasick pattern matching machine (Aho and Corasick, 1975) to locate all occurrences of specified patterns, and then find a path in the DAG that maximizes the sum of the scores of the non-overlapping patterns occurring in it. An example illustrates the utility.

## 1. Introduction

Biologically significant alignments are not necessarily mathematically optimized. It has been shown that sometimes the neighborhood of an optimal alignment reveals additional interesting biological features (Waterman and Byers, 1985; Saqi and Sternberg, 1991). Besides, the most strongly conserved regions can be effectively located by inspecting the range of variation of suboptimal alignments (Vingron and Argos, 1990; Zuker, 1991; Chao *et al*., 1993). While rigorous statistical analysis for the mean and variance of an optimal alignment score is not yet available, suboptimal alignments have been successfully used to informally estimate the significance of an optimal alignment.

However, it is essentially impractical to enumerate all suboptimal alignments since the number could be enormous. Therefore, a more compact representation of all suboptimal alignments is indispensable. A 0-1 matrix can be used to indicate if a pair of positions is in some suboptimal alignment or not (Vingron and Argos, 1990; Zuker, 1991). As pointed out by Naor and Brutlag (1993), this approach misses some connectivity information among those pairs of positions. They then used a set of "canonical" suboptimal alignments to represent all suboptimal alignments. The kernel of that representation is a minimal directed acyclic graph (DAG) containing all suboptimal alignments. Although their work was based on a simple scoring scheme, it is also applicable for affine gap penalties. (“Affine” means that a gap of length $k$ is penalized $\alpha + k \times \beta$, i.e., it costs $\alpha$ to open up a gap plus $\beta$ for each symbol in the gap.)

Traditional dynamic-programming algorithms for sequence comparison require quadratic space, and hence are infeasible for long protein or DNA sequences. Fortunately, quadratic-time, linear-space methods have been successfully designed to conquer this problem (Hirschberg, 1975; Myers and Miller, 1988).

†This work was supported by grant R01 LM05110 from the National Library of Medicine.

In this paper, we propose a method that computes the DAG representing all suboptimal alignments. The space requirement is linear to the size of the DAG. The time, however, is output-sensitive. Let $F$ be the area of the region of the dynamic-programming matrix bounded by the suboptimal alignments and $W$ the maximum width of that region. For two sequences of lengths $M$ and $N$, it is shown that the worst-case running time is $O(MN + F \log\log W)$.

To exploit the computed DAG, we employ a variant of Aho-Corasick pattern matching machine (Aho and Corasick, 1975) to locate all occurrences of specified patterns, and then find a path in the DAG that maximizes the sum of the scores of the non-overlapping patterns occurring in it. This is useful in delivering a more "meaningful" alignment. For instance, if there is more than one optimal alignment, we would prefer the one revealing more motifs of interest.

The rest of the paper is organized as follows. In Section 2, we present a relatively simple linear-space algorithm for computing the DAG in time $O(MN + F \log W)$. In Section 3, the algorithm is refined to compute the DAG in time $O(MN + F \log\log W)$. In Section 4, we discuss an algorithm that finds a path in the computed DAG with the maximum pattern score. In Section 5, an example illustrates the utility. Section 6 discusses some future research directions.

## 2. A Simple Linear-Space Algorithm for Computing the DAG

Given two sequences $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$, an *alignment* of $A$ and $B$ is obtained by introducing dashes into the two sequences such that the lengths of the two resulting sequences are identical and no column contains two dashes. Let $\Sigma$ denote the input symbol alphabet. A score $\sigma(a, b)$ is defined for each $(a, b) \in \Sigma \times \Sigma$. A gap of length $k$ is penalized $\alpha + k \times \beta$. The score of an alignment is the sum of $\sigma$ scores of all columns with no dashes minus the penalties of the gaps.

It is helpful to think of an alignment as a path in the alignment graph, $G_{A,B}$, defined as follows. $G_{A,B}$ is a directed graph with $3(M+1)(N+1)$ nodes, denoted $(i, j)_D$, $(i, j)_I$ and $(i, j)_S$, where $i \in [0, M]$ and $j \in [0, N]$. Table 1 depicts all the edges in $G_{A,B}$:

| *edge* | *weight* | *aligned pair* | *range* |
|---|---|---|---|
| $(i-1, j)_D \to (i, j)_D$ | $-\beta$ | $\begin{bmatrix} a_i \\ - \end{bmatrix}$ | $i \in [1, M]$ and $j \in [0, N]$ |
| $(i-1, j)_S \to (i, j)_D$ | $-(\alpha+\beta)$ | $\begin{bmatrix} a_i \\ - \end{bmatrix}$ | $i \in [1, M]$ and $j \in [0, N]$ |
| $(i, j-1)_I \to (i, j)_I$ | $-\beta$ | $\begin{bmatrix} - \\ b_j \end{bmatrix}$ | $i \in [0, M]$ and $j \in [1, N]$ |
| $(i, j-1)_S \to (i, j)_I$ | $-(\alpha+\beta)$ | $\begin{bmatrix} - \\ b_j \end{bmatrix}$ | $i \in [0, M]$ and $j \in [1, N]$ |
| $(i-1, j-1)_S \to (i, j)_S$ | $\sigma(a_i, b_j)$ | $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$ | $i \in [1, M]$ and $j \in [1, N]$ |
| $(i, j)_D \to (i, j)_S$ | 0 | none | $i \in [0, M]$ and $j \in [0, N]$ |
| $(i, j)_I \to (i, j)_S$ | 0 | none | $i \in [0, M]$ and $j \in [0, N]$ |

**Table 1.** The weights and aligned pairs associated with edges of $G_{A,B}$.

Let $s$ denote $(0, 0)_S$ and $t$ denote $(M, N)_S$. A path is *normal* if and only if it does not contain subpaths of the form $(i-1, j)_D \to (i-1, j)_S \to (i, j)_D$ or $(i, j-1)_I \to (i, j-1)_S \to (i, j)_I$. It can be shown that alignments of $A$ and $B$ are in one-to-one correspondence with normal $s$-$t$ paths (Myers and Miller, 1989). Furthermore, define the score of an $s$-$t$ path $P$, denoted as $Score(P)$, to be the sum over the weights of its edges. $Score(P)$ is the score of the alignment corresponding to $P$.

Suppose we are given a threshold score $\Delta$ that does not exceed the optimum score. A $\Delta$-suboptimal path (or $\Delta$-path) is an $s$-$t$ path with score at least as large as $\Delta$. A $\Delta$-suboptimal grid point (or $\Delta$-point) is a grid point where at least one of its nodes appears in some $\Delta$-path. Obviously, both $(0,0)$ and $(M,N)$ are $\Delta$-points. A $\Delta$-suboptimal edge (or $\Delta$-edge) is an edge that appears in some $\Delta$-path.

Our goal is to compute a directed acyclic graph, denoted by $DAG_\Delta = (V_\Delta, E_\Delta)$, where $V_\Delta$ is the set of nodes in all $\Delta$-points and $E_\Delta$ is the set of all $\Delta$-edges. In the following, we will show how to construct $V_\Delta$ in $O(|V_\Delta|)$ space.

Let $Score^-(i,j)_X$ be the maximum score of any path from $s$ to $(i,j)_X$, where $X \in \{D, I, S\}$. With proper initializations, these scores can be computed by the following recurrence relations (Myers and Miller, 1988):

$$Score^-(i,j)_D = \max\{Score^-(i-1,j)_D - \beta,\ Score^-(i-1,j)_S - \alpha - \beta\}$$
$$Score^-(i,j)_I = \max\{Score^-(i,j-1)_I - \beta,\ Score^-(i,j-1)_S - \alpha - \beta\}$$
$$Score^-(i,j)_S = \max\{Score^-(i-1,j-1)_S + \sigma(a_i, b_j),\ Score^-(i,j)_D,\ Score^-(i,j)_I\}$$

Similarly, let $Score^+(i,j)_X$ be the maximum score of any path from $(i,j)_X$ to $t$, where $X \in \{D, I, S\}$. With proper initializations, these scores can be computed by the following recurrence relation:

$$\begin{aligned} Score^+(i,j)_S = \max\{&Score^+(i+1,j+1)_S + \sigma(a_{i+1}, b_{j+1}),\ Score^+(i+1,j)_D - \alpha - \beta, \\ &Score^+(i,j+1)_I - \alpha - \beta\} \end{aligned}$$
$$Score^+(i,j)_D = \max\{Score^+(i+1,j)_D - \beta,\ Score^+(i,j)_S\}$$
$$Score^+(i,j)_I = \max\{Score^+(i,j+1)_I - \beta,\ Score^+(i,j)_S\}$$

Define $Score(i,j) = \max\ \{Score^-(i,j)_X + Score^+(i,j)_X \mid X \in \{D, I, S\}\}$.

**Lemma 1.** A grid point $(i,j)$ is a $\Delta$-point if and only if $Score(i,j) \geq \Delta$.

**Proof.** Omitted. □

Let $[T,B] \times [L,R]$ denote the rectangle whose upper left corner is $(T,L)$ and lower right corner is $(B,R)$. We say that $[T,B] \times [L,R]$ contains $(i,j)$ (or $(i,j)$ is in $[T,B] \times [L,R]$) if $T \leq i \leq B$ and $L \leq j \leq R$.

**Lemma 2.** Let $(i,j)$ be a $\Delta$-point in $[1, M-1] \times [1, N-1]$. At least one of $(i, j-1)$, $(i-1, j)$ and $(i-1, j-1)$ is a $\Delta$-point. At least one of $(i, j+1)$, $(i+1, j)$ and $(i+1, j+1)$ is a $\Delta$-point.

**Proof.** Omitted. □

Given a rectangle, denoted by $\Pi$, let $\pi$ be the set of $\Delta$-points on $\Pi$'s boundaries. If $\pi$ is not empty, let $\pi_{i_1}$ and $\pi_{i_2}$ be the minimum and maximum index, respectively, of the rows containing some of $\pi$'s elements, and let $\pi_{j_1}$ and $\pi_{j_2}$ be the minimum and maximum index, respectively, of the columns containing some of $\pi$'s elements.

**Lemma 3.** If $\pi$ is empty, there is no $\Delta$-point in $\Pi$. Otherwise, $[\pi_{i_1}, \pi_{i_2}] \times [\pi_{j_1}, \pi_{j_2}]$ contains all $\Delta$-points in $\Pi$.

**Proof.** Suppose there are some $\Delta$-points in $\Pi$, and $\pi$ is empty. Take any such $\Delta$-point. By Lemma 2, we can always trace back from that $\Delta$-point to a boundary $\Delta$-point. A contradiction with the assumption that $\pi$ is empty.

If $\pi$ is not empty, we claim $[\pi_{i_1}, \pi_{i_2}] \times [\pi_{j_1}, \pi_{j_2}]$ contains all $\Delta$-points in $\Pi$. Indeed, suppose there exists a $\Delta$-point in $\Pi$ with a row index smaller than $\pi_{i_1}$. We can trace back from that $\Delta$-point to a boundary $\Delta$-point with a row index smaller than $\pi_{i_1}$, contradicting the assumption that $\pi_{i_1}$ is the minimum index of the rows that contain some of $\pi$'s points. Similar arguments apply to $\pi_{i_2}$, $\pi_{j_1}$ and $\pi_{j_2}$. It follows that $[\pi_{i_1}, \pi_{i_2}] \times [\pi_{j_1}, \pi_{j_2}]$ contains all $\Delta$-points in $\Pi$. $\square$

In fact, it can be shown that $[\pi_{i_1}, \pi_{i_2}] \times [\pi_{j_1}, \pi_{j_2}]$ is the smallest rectangle that contains all $\Delta$-points in $\Pi$.

The algorithm for computing all $\Delta$-points is outlined as follows. For each conducted subproblem, the invariant is that $Score^-$ are given for every grid point on the left and upper boundaries, and $Score^+$ are given for every grid point on the right and lower boundaries. With these scores, the $Score^-$ and $Score^+$ for grid points within the subproblem can be computed. Problems with one or two rows or columns, can be solved directly. In general, a larger subproblem is then divided into four non-overlapping subproblems by the middle row and middle column.

To do so, a linear-space forward pass is performed to compute $Score^-$. To maintain the invariant, $Score^-$ are stored in every grid point on the two middle rows and two middle columns. To decide a more accurate range of each subproblem, *Score* for each grid point on the right and lower boundaries is also determined and stored.

Similarly, a linear-space backward pass is performed to compute $Score^+$. To maintain the invariant, $Score^+$ are stored in every grid point on the two middle rows and two middle columns. *Score* for each grid point on the left and upper boundaries is also determined and stored.

At this point, the *Score* for each grid point on the boundaries of the four subrectangles, divided by the middle row and middle column, can be determined in constant time. Take one subrectangle for example, we determine the minimum and maximum indices of the rows and the minimum and maximum indices of the columns that contain at least one $\Delta$-point on the subrectangle's boundaries. Lemma 3 says that the rectangle bounded by these rows and columns contains all $\Delta$-points in the subrectangle. It is therefore enough to consider only the "shrunken" subrectangle. Figure 1 illustrates the approach.

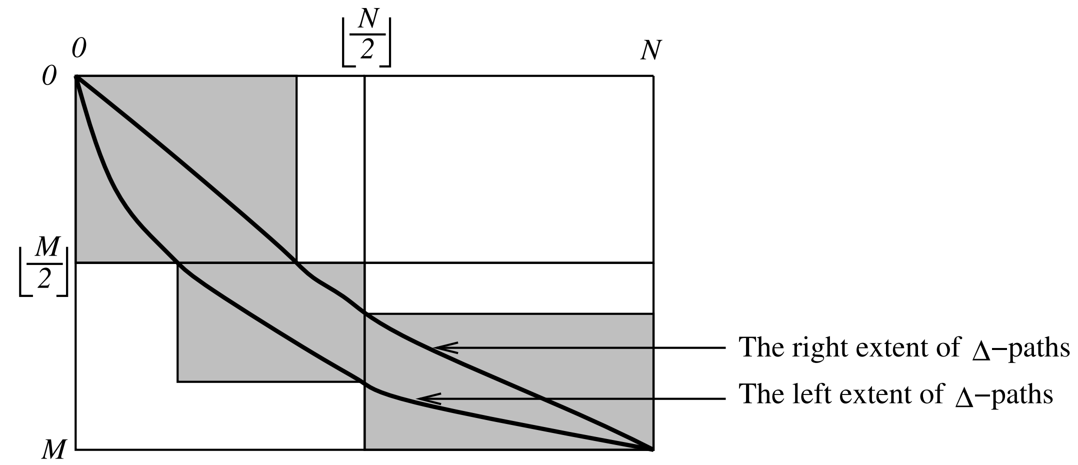


**Figure 1.** Splitting the problem into subproblems (shaded areas).

Figure 2 gives the pseudo code for constructing $V_\Delta$ in linear space. Let $Sub[i]$ be a linked list to store all $\Delta$-points in row $i$ for $0 \le i \le M$. Initially, they are set to be empty. Each time when a $\Delta$-point is found, the function *append* is called to add the point to its *Sub* list. We assume that $Score^-$ and $Score^+$ are stored in each $\Delta$-point.

```
procedure SUB_OPT(M, N)
  { Compute Score⁻ for row 0 and column 0
    Compute Score⁺ for row M and column N
    for i ← 0 to M do Sub[i] ← φ
    sub_opt(0, 0, M, N, initial boundary score vectors)
  }

recursive procedure sub_opt(I1, J1, I2, J2, boundary score vectors)
  /* Compute all Δ-points in [I1, I2] × [J1, J2] */
  { if I1 + 1 ≥ I2 or J1 + 1 ≥ J2 then
      { Compute and store Score(i, j) for each (i, j) in [I1, I2] × [J1, J2].
        for i ← I1 to I2 do
            for j ← J1 to J2 do { if Score(i, j) ≥ Δ then append(Sub[i], POINT(i, j)) }
        return
      }
    midI ← ⌊(I1 + I2)/2⌋
    midJ ← ⌊(J1 + J2)/2⌋
    A linear-space forward computation is performed to compute Score⁻:
        store Score⁻(i, j) if i = midI or midI + 1, or j = midJ or midJ + 1;
        store Score(i, j) if i = I2 or j = J2.
    A linear-space backward computation is performed to compute Score⁺:
        store Score⁺(i, j) if i = midI or midI + 1, or j = midJ or midJ + 1;
        store Score(i, j) if i = I1 or j = J1.
    /* Divide the problem by row midI and column midJ */
    Π1 ← the set of the grid points on the boundaries of [I1, midI] × [J1, midJ]
    Π2 ← the set of the grid points on the boundaries of [I1, midI] × [midJ + 1, J2]
    Π3 ← the set of the grid points on the boundaries of [midI + 1, I2] × [J1, midJ]
    Π4 ← the set of the grid points on the boundaries of [midI + 1, I2] × [midJ + 1, J2]
    for k ← 1 to 4 do
      { π ← {(i, j) | Score(i, j) ≥ Δ, (i, j) ∈ Πk }
        if π ≠ φ then
          { i1 ← min{i | (i, j) ∈ π}
            j1 ← min{j | (i, j) ∈ π}
            i2 ← max{i | (i, j) ∈ π}
            j2 ← max{j | (i, j) ∈ π}
            Compute Score⁻ for row i1 and column j1
            Compute Score⁺ for row i2 and column j2
            sub_opt(i1, j1, i2, j2, new boundary score vectors);
          }
      }
  }
```

**Figure 2.** The algorithm for constructing $V_\Delta$ in linear space.

The following lemma proves the correctness of the algorithm in Figure 2.

**Lemma 4.** For each row $i$, $Sub[i]$ contains only and all $\Delta$-points in row $i$. Moreover, those points are distinct and linked in increasing column order.

**Proof.** Omitted. □

*Space requirement*

**Theorem 5.** The space for the boundary score vectors of all pending subproblems is $O(M + N)$.

**Proof.** Let $S(m, n)$ denote the worst-case space requirement for the boundary score vectors of all pending subproblems when applying *sub_opt* to a subproblem with $m$ rows and $n$ columns. Since each of its four possible subproblems is solved independently,

$$S(m, n) \leq \begin{cases} c(m+n) & \text{for } m \leq 2 \textit{ or } n \leq 2 \\ S(\lceil m/2 \rceil, \lceil n/2 \rceil) + c(m+n) & \text{for } m > 2 \text{ and } n > 2 \end{cases}$$

where $c$ is a constant. It follows $S(M, N) = O(M + N)$. □

Since $|V_\Delta|$ is $\Omega(\max\{M, N\})$, the space for the boundary score vectors and computed $\Delta$-points is $O(|V_\Delta|)$. To see that this dominates the algorithm's space requirements, we need to consider the maximum size of the procedure activation stack, which depends on the maximum recursion depth. The number of rows (and columns) of the problem at a recursive call to *sub_opt* is at most half that of the containing problem (rounded up), so the maximum stack depth is $O(\min\{\log M, \log N\})$.

*Time analysis*

For each row $i$, define $L[i]$ and $R[i]$ to be the minimum and maximum index, respectively, of the columns where a $\Delta$-path intersects row $i$. The band width of row $i$, $R[i] - L[i] + 1$, is denoted by $W_{row}[i]$. $W_{col}[j]$ is defined in a similar way. $W$ is defined to be $\min\{\max\{W_{row}[i]\}, \max\{W_{col}[j]\}\}$. Let $F$ denote the area of the region of the dynamic-programming matrix bounded by $\Delta$-paths, i.e. $F = \sum_{i=0}^{M} W_{row}[i]$.

**Lemma 6.** If $R[i] \leq midJ$ in the current subproblem, $(i, midJ+1), \cdots, (i, J_2)$ will not be included in any subsequent subproblem. Similarly, if $L[i] > midJ$ in the current subproblem, $(i, J_1), \cdots, (i, midJ)$ will not be included in any subsequent subproblem.

**Proof.** Since the right extent of $\Delta$-paths is monotonically increasing, it is easy to see that if $R[i] \leq midJ$, $[I_1, i] \times [midJ+1, J_2]$ does not contain any $\Delta$-points. Either $[I_1, I_2] \times [midJ+1, J_2]$ does not contain any $\Delta$-points, or the minimum index of the rows that contain some $\Delta$-points in $[I_1, I_2] \times [midJ+1, J_2]$ is larger than $i$. In either case, $(i, midJ+1), \cdots, (i, J_2)$ will not be included in any subsequent subproblem. The case when $L[i] > midJ$ can be proved in a similar way. □

**Theorem 7.** Let $T$ be the total number of grid points in all the calls to *sub_opt*. $T = O(MN + F \log W)$.

**Proof.** Let subproblems with no more than two rows or two columns be trivial subproblems. Since each grid point can be included in at most one trivial subproblem, $O(MN)$ grid points are included in such subproblems.

Fix a row $i$, consider all nontrivial subproblems that include some row $i$'s grid-points. Before reaching the first subproblem with the property $J_1 \leq L[i] \leq midJ \leq R[i] \leq J_2$, all its containing subproblems include in total $O(N)$ row $i$'s grid points. This is because all its containing subproblems is either with the property $J_1 \leq L[i] \leq R[i] < midJ \leq J_2$ or $J_1 \leq midJ < L[i] \leq R[i] \leq J_2$ which will truncate half of row $i$'s grid points in the subsequent call (Lemma 6).

The subproblem is further split into at most one subproblem with the property $J_1 \leq L[i] \leq J_2 \leq R[i]$, and at most one with the property $L[i] < J_1 \leq R[i] \leq J_2$. Now we show that each of them will include $O(N + W_{row}[i] \log W_{row}[i])$ row $i$'s grid points in its subsequent calls. Indeed, consider the subproblem with $J_1 \leq L[i] \leq J_2 \leq R[i]$. If $midJ \geq L[i]$, it is easy to see that all its subsequent subproblems include in total $O(W_{row}[i] \log W_{row}[i])$ row $i$'s grid points. If $midJ < L[i]$, $(i, J_1), \cdots, (i, midJ)$ will be truncated (Lemma 6). Before reaching the subproblem with

$midJ \geq L[i]$, those containing subproblems include in total $O(N)$ row $i$'s grid points. Similar arguments apply to the case when $L[i] < J_1 \leq R[i] \leq J_2$.

It follows that all subproblems include $O(N + W_{row}[i] \log W_{row}[i])$ row $i$'s grid points. Therefore, we have

$$T = O(MN + \sum_{i=0}^{M} W_{row}[i] \log W_{row}[i]) = O(MN + F \log \max \{W_{row}[i]\})$$

In a similar way, we can derive $T = O(MN + F \log \max\{W_{col}[j]\})$. It follows $T = O(MN + F \log W)$.

□

Since $F \leq MN$ and $W \leq \min\{M, N\}$, $T = O(MN \log \min\{M, N\})$. This remains even when $DAG_\Delta$ is sparse because the width of $DAG_\Delta$ could be independent of its density. On the other hand, Theorem 7 implies that if $F = O(MN/\log W)$, $T = O(MN)$.

To complete the construction of $DAG_\Delta$, we need to build $E_\Delta$. Let $e$ be an edge from node $u$ to node $v$. Define $Score(e)$ to be $Score^-(u) + weight(e) + Score^+(v)$. It can be shown that $e$ is a $\Delta$-edge if and only if $Score(e) \geq \Delta$. Obviously, if $e$ is a $\Delta$-edge, both $u$ and $v$ are at some $\Delta$-point. Constructing all $\Delta$-edges from the *Sub* lists takes $O(|V_\Delta|)$ time.

It should be noted that not every *s-t* path in $DAG_\Delta$ has score at least $\Delta$. However, methods of Waterman and Byers (1985) or Naor and Brutlag (1993) can be applied to $DAG_\Delta$ to generate $\Delta$-paths efficiently.

As defined by Naor and Brutlag (1993), an *s-t* path $P$ is called canonical if there exists an edge $e$ in $P$ such that $Score(e) = Score(P)$. They further showed that canonical $\Delta$-paths can represent all $\Delta$-paths and their number is far less than the number of all $\Delta$-paths. It can be shown that their theorems for canonical paths also hold for $DAG_\Delta$.

**3. An Improved Linear-Space Algorithm**

For each conducted subproblem, the invariant is that $Score^-$ are given for every grid point on the left and upper boundaries, and $Score^+$ are given for every grid point on the right and lower boundaries. Instead of partitioning a subproblem into four subproblems, we partition it into a different number of subproblems, depending on the recursion depth of a subproblem. Let $L(i)$ and $W(i)$ be the number of rows and columns of a subproblem in recursion depth $i$, respectively. In general, an $L(i) \times W(i)$ subproblem at recursion depth $i$ is divided into $T^2(i)$ non-overlapping $\frac{L(i)}{T(i)} \times \frac{W(i)}{T(i)}$ subproblems, where $T(i)$ is determined by the following recurrence relation.

$$T(i) = \begin{cases} b & \text{for } i = 0 \\ T^2(i-1)/2 & \text{for } i > 0 \end{cases}$$

where $b > 2$ is a constant. It can be shown that $T(i) = \frac{b^{2^i}}{2^{2^i - 1}}$. $L(i)$ and $W(i)$ are computed as follows.

$$L(i) = \begin{cases} M + 1 & \text{for } i = 0 \\ L(i-1)/T(i-1) & \text{for } i > 0 \end{cases}$$

$$W(i) = \begin{cases} N + 1 & \text{for } i = 0 \\ W(i-1)/T(i-1) & \text{for } i > 0 \end{cases}$$

One can show that $L(i) = \frac{2^{2^i-1-i}}{b^{2^i-1}}(M+1)$ and $W(i) = \frac{2^{2^i-1-i}}{b^{2^i-1}}(N+1)$.

**Theorem 8.** The space for the boundary score vectors of all pending subproblems is $O(M+N)$.

**Proof.** Let $c(n+m)$ be the space required to store boundary score vectors for an $n \times m$ subproblem, where $c$ is a constant. Let $S(k)$ be the total space for the boundary score vectors of all pending subproblems when the recursion depth is $k$.

$$S(k) = \sum_{i=0}^{k} c(L(i) + W(i))T(i)$$

$$= c \sum_{i=0}^{k} \frac{b}{2^i}(M+N+2)$$

$$\leq 2cb(M+N+2)$$

$$= O(M+N)$$

□

Table 2 illustrates the case when $b = 4$

| Recursion depth $i$ | 0 | 1 | 2 | 3 | 4 | 5 | ... |
|---|---|---|---|---|---|---|---|
| $W(i)$ | $N+1$ | $\frac{1}{2^2}(N+1)$ | $\frac{1}{2^5}(N+1)$ | $\frac{1}{2^{10}}(N+1)$ | $\frac{1}{2^{19}}(N+1)$ | $\frac{1}{2^{36}}(N+1)$ | ... |
| $T(i)$ | $2^2$ | $2^3$ | $2^5$ | $2^9$ | $2^{17}$ | $2^{33}$ | ... |
| $T(i) \times W(i)$ | $2^2(N+1)$ | $2(N+1)$ | $N+1$ | $\frac{1}{2}(N+1)$ | $\frac{1}{2^2}(N+1)$ | $\frac{1}{2^3}(N+1)$ | ... |

**Table 2.** Rate of growth when $b = 4$.

**Lemma 9.** The maximum recursion depth is $O(\log\log\min\{M, N\})$.

**Proof.** The recursive procedure stops at recursion depth $i$ when $L(i) \leq 1$ or $W(i) \leq 1$. Since $L(i) = (\frac{2}{b})^{2^i-1}\frac{1}{2^i}(M+1)$ and $b > 2$, it can be shown that $L(i) \leq 1$ for some $i = c\log\log M$, where $c$ is a constant. Thus, $L(i)$ decreases to 1 in $O(\log\log M)$ steps. Similarly, we can show that $W(i)$ decreases to 1 in $O(\log\log N)$ steps. Therefore, the maximum recursion depth is $O(\log\log\min\{M, N\})$. □

**Theorem 10.** The total running time is $O(MN\log\log\min\{M, N\})$.

**Proof.** It takes in total $O(MN)$ time for all the subproblems at the same recursion depth. Lemma 9 shows that the recursion depth is bounded by $O(\log\log\min\{M, N\})$. It follows that the total running time is $O(MN\log\log\min\{M, N\})$. □

Again, Lemma 3 can be applied to reduce the size of each conducted subproblem. With an argument similar to the proof of Theorem 7, we have the following theorem.

**Theorem 11.** The total running time of the new divide-and-conquer algorithm augmented with shrinking operation described in Lemma 3 is $O(MN + F\log\log W)$.

**4. Finding an *s*-*t* path in $DAG_\Delta$ with the maximum pattern score**

This section discusses one way of utilizing $DAG_\Delta$. Given is a set of patterns, where each pattern $\omega$ is given a positive score $\omega_{score}$. The pattern score of a path $P$ is defined as the maximum sum of the scores of non-overlapping patterns occurring in $P$. The goal is to find an $s$-$t$ path $P_\Delta$ in $DAG_\Delta$ such that the pattern score of $P_\Delta$ is maximum among all $s$-$t$ paths in $DAG_\Delta$. Furthermore, if there are more than one $s$-$t$ paths maximizing the pattern score, $Score(P_\Delta)$ is maximum among all such paths.

A pattern $\omega$ is said to *occur* at $\Delta$-point $(i, j)$ if $a_{i-|\omega|+1} a_{i-|\omega|+2} \cdots a_i = b_{j-|\omega|+1} b_{j-|\omega|+2} \cdots b_j = \omega$, and $(i-|\omega|, j-|\omega|)_S \rightarrow (i-|\omega|+1, j-|\omega|+1)_S \rightarrow \cdots (i, j)_S$ is a path in $DAG_\Delta$. An occurrence edge from $(i-|\omega|, j-|\omega|)_S$ to $(i, j)_S$, denoted by $(i-|\omega|, j-|\omega|)_S \rightarrow_\omega (i, j)_S$, is augmented to $DAG_\Delta$ if $\omega$ occurs at $(i, j)$ for some pattern $\omega$ in the given pattern set.

In order to augment $DAG_\Delta$ with all such occurrence edges, a finite state pattern matching machine, following the scheme of Aho and Corasick (1975), is constructed. It is operated by three functions: a goto function $g$, a failure function $f$, and an output function *output* (see Aho and Corasick, 1975). Figure 3 outlines the algorithm for constructing all occurrence edges.

```
for each Δ-point (i, j) in topological order do
    if (i − 1, j − 1)_S → (i, j)_S is not a Δ-edge then
     {  state ← 0
        k ← 0
        while (i + k, j + k)_S → (i + k + 1, j + k + 1)_S is a Δ-edge
          {  if a_{i+k+1} = b_{j+k+1} then
               {  while g(state, a_{i+k+1}) = fail do state ← f(state)
                  for each pattern ω in output(state) do
                         Construct (i + k + 1 − |ω|, j + k + 1 − |ω|)_S →_ω (i + k + 1, j + k + 1)_S
               }
             else state ← 0
             k ← k + 1
          }
     }
```

**Figure 3.** The algorithm for constructing all occurrence edges.

Let $l$ be the sum of the pattern lengths. Let $Num_s$ be the number of the patterns recognized by state $s$. The time for constructing an Aho-Corasick pattern matching machine is $O(l|\Sigma| + \sum_s Num_s)$. It can be shown that the total number of state transitions made by the algorithm in Figure 3 is $O(|V_\Delta|)$. If a pattern $\omega$ occurs at a $\Delta$-point $(i, j)$, we have to construct an occurrence edge $(i-|\omega|, j-|\omega|)_S \rightarrow_\omega (i, j)_S$. A stack can be used to backtrack the starting location of the occurrences on the same diagonal. The time for constructing all occurrence edges is $O(|V_\Delta| + Occ)$, where $Occ$ is the number of occurrences.

Let $Pat\_Score(u)$ be the maximum pattern score of any path from $u$ to $t$ in $DAG_\Delta$. The following recurrence relation computes $Pat\_Score(u)$.

$$Pat\_Score(u) = \max \{ \max \{ Pat\_Score(v) \mid u \rightarrow v \text{ is a } \Delta\text{–edge.} \}, \\ \max \{ Pat\_Score(v) + \omega_{score} \mid u \rightarrow_\omega v \text{ is an occurrence edge.} \} \}$$

It can be computed in $O(|V_\Delta| + Occ)$ time for all nodes in $DAG_\Delta$. A simple backtracking method with the tie-breaking rules yields an $s$-$t$ path $P_\Delta$ in $DAG_\Delta$ such that $Score(P_\Delta)$ is maximum among all $s$-$t$ paths in $DAG_\Delta$ with the maximum pattern score. It should be noted that $Score(P_\Delta)$ may be worse than $\Delta$.

In particular, when the threshold score $\Delta$ is the optimum score, it is easy to see that every $s$-$t$ path in $DAG_\Delta$ is an optimal path. Therefore, the algorithm presented in this section can be used to deliver an optimal alignment with the maximum pattern score. It should be noted that the problem of finding optimal alignments containing patterns has been explored before. For example, Lawerence *et al*. (1986) compute the alignment score as the score of the concatenated optimal local alignments which were extended from homologies exceeding or equal to a specified minimum length.

## 5. An example

We have implemented the algorithms in Sections 2 and 4. The conducted experiments showed that with threshold score $\Delta$ close to the optimum score, $T < 2(M+1)(N+1)$. Surprisingly, in that reasonable range, it even ran faster than the quadratic-time, linear-space *left_right* program (Chao *et al*., 1993) that locates merely the left and right extents of $\Delta$-paths.

To illustrate the utility of the algorithm developed in Section 3, we aligned the $\varepsilon$-globin gene regions of human and rabbit. Identical matching nucleotides scored 1, mismatches scored -1 and $k$-symbol gaps were penalized $6 + 0.2k$. Figures 4 and 5 are a portion of two different optimal alignments. Figure 6 is a multiple alignment shown in Hardison *et al*. (1993). The human sequence is given in full, and periods denote a matching nucleotide in the other species.

The alternate optimal alignment in Figure 5 reveals three interesting features that are not in the optimal alignment in Figure 4. Box 1 contains the same gap revealed in the multiple alignment in Figure 6. Box 2 contains a gap followed by a matching block instead of two matching blocks split by a gap. Incidentally, this can be used to improve the multiple alignment in Figure 6. Finally, box 3 contains a matching block GAAGAG, a candidate for the phylogenetic footprint, which is defined as at least six consecutive invariant positions (Tagle *et al*, 1988; Gumucio *et al*., 1993). Phylogenetic footprints have been demonstrated to be useful as a guide to identifying nuclear protein binding sites. In fact, GAAGAG also appears in the corresponding region of galago.

```
                  |         |         |         |         |
19091:   TTTGTCAACTGTCACCACCTTTAAGGCAAATGTTAAATGTGCTTTGGCTGAAACTTTTTT   human
 5544:   .......-------...A.C..G.CC.....A.G...C-.A....A...CT...AC...-   rabbit

                  |         |         |         |         |
19151:   TCCTATTTTGAGATTTGCTCCTTTATATGAGGCTTTCTTGGAAAAGGAGAATGGGAGAGA   human
 5595:   -..AG.C..AT.C..A...G..C.C....AT................G........AT..   rabbit

                  |         |         |         |         |
19211:   TGGATATCATTTTGGAAGATGATGA-----------AGAGGGTAAAAAAGGGGACAAATG   human
 5654:   .....GC...C.........T.CATGGAAAAAGAAG....T.A...C.T.ATA.TGT...   rabbit
```

**Figure 4.** An optimal alignment.

```
                       1                                                    2
19091:  TTTGTCAACTGTCACCACCTTTAAGGCAAATGTTAAATGTGCTTTGGCTGAAACTTTTTT   human
 5544:  ..-------........A.C..G.CC.....A.G...C-.A....A...CT...AC--..   rabbit

19151:  TCCTATTTTGAGATTTGCTCCTTTATATGAGGCTTTCTTGGAAAAGGAGAATGGGAGAGA   human
 5594:  ...AG.C..AT.C..A...G..C.C.....AT...............G........AT..   rabbit

                                                3
19211:  TGGATATCATTTTGGAAGATGAT---------GAAGAGGGTAAAAAAGGGGACAA--ATG   human
 5654:  .....GC...C.........T.CATGGAAAAA......A..G....CAT.ATA.TGT...   rabbit
```

**Figure 5.** An alternate optimal alignment. TGTCACCA, TTTCC and GAAGAG are in the given pattern set.

```
              εNEBP
19091:  TTTGTCAACTGTCACCACCTTTAAGGCAAATGTTAAATGTGCTTTGG-CTGAAACTTTTT   human
 3960:  .......T-......T.T.CA...TC.....T.....CTCAT.....-.....G..G...   galago
 5544:  ..-------........A.C..G.CC.....A.G...CT-A....A.-..CT...AC...   rabbit
   76:                                                            ..   goat
21352:  C..A.T.T-.A.AG...TTC..TCCCT...C.G....GT.AT...A.A...G.TT.....   mouse

                                                   YRC          YY1
19150:  TTCCTATTTTGAGATTTGCTCCTTTATATGAGGCTTTCTTGGAAAAGGAG-AATGGGAGA   human
 4018:  .-....CC.GAG.C..A.T..A..---.....A..--...........G.G....A....   galago
 5595:  --..AG.C..AT.C..A...G..C.C.....AT...............G.-.......AT   rabbit
   78:  C-.TAG.C.GAT.C..A...A.AC.CC...GAT.A....G.A.G........-.........   goat
21411:  .-.....AGC.G.C.AAA.....C.T.C..T.C....G........A.T.--...A....   mouse

          YY1   GATA1      Ets
19209:  GATGGATATCATTTTGGAAGATGATGAAGAGGGTAAAAAAGGGGACAAATG           human
 4072:  A...TT.G.T......T...G.....TG.G.AAAG..G.GA.T..GT.GA.           galago
 5652:  .......GC...C.........T.CATG..AAAAG..G.GA.T..A..CAT           rabbit
  136:  .......GC...CG....G.....GAGG.GAA.A.C.G.G..T..A..TG.           goat
21468:  A....GCG.T..AC...G..T.T.CATGTG.A.AGT.G..AAT..A..GA.           mouse
```

**Figure 6.** Multiple alignment of the 5' flank of mammalian $\varepsilon$-globin genes.

## 6. Discussion

It has been shown that Δ-points are useful in speeding up the computation for multiple sequence alignment problem (Carrillo and Lipman, 1988; Altschul and Lipman, 1989). As noted by Kececioglu(1989), the $O(MN)$ space, which is required by a straightforward method for computing all Δ-points, may be the dominant space requirement for inputs consisting of a few long sequences. The linear-space algorithm presented here can be applied in this context.

It is natural to design a model that does some pattern matching to extract more information from the DAG. For instance, it is hoped that the DAG will provide a good estimate of how robust an optimal alignment is. Also, the DAG might be utilized for finding genes in a given sequence. It remains to be investigated what kind of language would be appropriate for these purposes.

A *local alignment* is an alignment where the end-nodes can be arbitrary, i.e., they are not restricted to $(0,0)_S$ and $(M,N)_S$. One can define a grid point to be a *local* Δ-point if at least one of its nodes appears in some local alignment with score at least Δ. The divide-and-conquer approach described in Section 3 yields a $O(MN \log\log \min\{M,N\})$-time, linear-space method for computing all local Δ-points. Can it be done more efficiently?

**Acknowledgements**

I would like to thank Dr. Webb Miller for valuable guidance and encouragement. His comments resulted in numerous improvements in the presentation. I would also like to thank Dr. John Kececioglu for helpful discussions during his visit at Penn State.

**References**

Aho, A. V. and Corasick, M. J. (1975) Efficient string matching: an aid to bibliographic search. *Comm. ACM*, **18**, 333-340.

Altschul, S. F. and Lipman, D. J. (1989) Trees, stars, and multiple biological sequence alignment. *SIAM J. Appl. Math.*, **49**, 197-209.

Carrillo, H., and Lipman, D. J. (1988) The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, **48**, 1073-1082.

Chao, K.-M., Hardison, R. C. and Miller, W. (1993) Locating well-conserved regions within a pairwise alignment. *CABIOS*, **9**, 387-396.

Gumucio, D. L., Shelton, D. A., Bailey, W. J., Slightom, J. L., and Goodman, M. (1993) Phylogenetic footprinting reveals unexpected complexity in trans factor binding upstream from the $\varepsilon$-globin gene. *Proc. Natl. Acad. Sci. USA*, **90**, 6018-6022.

Hardison, R. C., Chao, K.-M., Adamkiewicz, M., Price, D., Jackson, J., Zeigler, T., Stojanovic, N., and Miller, W. (1993) Positive and negative regulatory elements of the rabbit embryonic $\varepsilon$-globin gene revealed by an improved multiple alignment program and functional analysis. *DNA Sequence*, **4**, 163-176.

Hirschberg, D. S. (1975) A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, **18**, 341-343.

Kececioglu, J. D. (1989) Notes on a multiple sequence alignment cost bound of Carrillo and Lipman. Manuscript.

Lawerence, C. B., Goldman, D. A., and Hood, R. T. (1986) Optimized homology searches of the gene and protein sequence data banks. *Bull. Math. Biol.*, **48**, 569-583.

Myers, E. W. and Miller, W. (1988) Optimal alignments in linear space. *CABIOS*, **4**, 11-17.

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

Naor, D. and Brutlag, D. (1993) On suboptimal alignments of biological sequences. In Proceedings of the 4th Symposium on *Combinatorial Pattern Matching*, Lecture Notes in Computer Science, **684**, 179-196.

Saqi, M. and Sternberg, M. (1991) A simple method to generate non-trivial alternative alignments of protein sequences. *J. Mol. Biol.*, **219**, 727-732.

Tagle, D. A., Koop, B. F., Goodman, M., Slightom, J., Hess, D. L. and Jones, R. T. (1988) Embryonic $\varepsilon$ and $\gamma$ globin genes of a prosimian primate (Galago crassicaudatus): Nucleotide and amino acid sequences, developmental regulation and phylogenetic footprints. *J. Mol. Biol.*, **203**, 7469-7480.

Vingron, M. and Argos, P. (1990) Determination of reliable regions in protein sequence alignment. *Protein Engineering*, **3**, 565-569.

Waterman, M., and Byers, T. (1985) A dynamic programming algorithm to find all solutions in a neighborhood of the optimum. *Math. Biosciences*, **77**, 179-185.

Zuker, M. (1991) Suboptimal sequence alignment in molecular biology: alignment with error analysis. *J. Mol. Biol.*, **221**, 403-420.