On Building an Interactive Alignment Tool†

Kun-Mao Chao

Department of Computer Science and Information Management
Providence University
Shalu, Taichung, Taiwan 43309
e-mail: kmchao@csim.pu.edu.tw

Abstract.

Sequence comparison has wide applications to computer science, molecular biology, speech processing, and so on. In molecular biology, it is often used to reveal similarities among sequences, study gene regulation, or even infer evolutionary trees. This paper summarizes the techniques that are useful for building an interactive sequence alignment tool. In order to analyze very long DNA sequences, we have put a considerable emphasis on space-efficient algorithms that possess a high degree of sensitivity. Based on the preprocessed data from fast heuristic alignment methods (e.g. BLAST), users can further draw arbitrary regions to be aligned using more sensitive alignment methods.

Key Words. Computational biology, Divide-and-conquer, Dynamic programming, Linear-space algorithm, Sequence comparison.

†This work was supported in part by grant R01 LM05110 from the National Library of Medicine, National Institutes of Health, USA, and grant NSC84-2213-E-126-002 from the National Science Council, Taiwan.

1. Introduction.

Molecular biology is rapidly becoming a data-rich science with extensive computational needs. More and more computer scientists are working together on developing efficient software tools for molecular biologists. One major area of potential interaction between computer scientists and molecular biologists arises from the need for analyzing biological information. In particular, sequence alignments have been used to reveal similarities among biological sequences, study gene regulation, or even infer evolutionary trees [8, 9, 11, 15, 18].

Sequence alignment generally can be classified as either global or local [13]. Local alignment algorithms seek only conserved regions, whereas global methods align entire sequences including unconserved regions. For alignment scores that are popular with molecular biologists, dynamic-programming alignment of two sequences requires quadratic time, *i. e.*, time proportional to the product of the two sequences lengths [6, 13, 14]. Quadratic time is necessitated by the inspection of every pair (i, j), where i is a position in the first sequence and j is a position in the second sequence. Alternatively, far faster heuristic methods have been developed, such as FASTA [16, 19, 20] and BLAST [1]. BLAST employs a measure based on well-defined mutation scores, and directly approximates the results that would be obtained by a dynamic programming algorithm for optimizing this measure.

Currently, we are involved in an on-going project to build an interactive alignment tool. This project is intended to help molecular biologists align sequences *interactively*. Specifically, based on the preprocessed data from fast heuristic alignment methods (e.g. BLAST), users can further draw arbitrary regions to be aligned using more sensitive alignment methods [2-7, 13]. We will also develop tools to browse alignment data and prepare publication-quality alignment output [17].

This paper summarizes the techniques that are useful for building an interactive alignment tool. In order to analyze very long DNA sequences, we have put a considerable emphasis on space-efficient algorithms that possess a high degree of sensitivity.

2. Methods.

To simplify the presentation, we assume the following simple alignment-scoring scheme. For each possible aligned pair $\begin{bmatrix} x \\ y \end{bmatrix}$, where each of x and y is either a normal sequence entry or the symbol "–", there is an assigned score $\sigma(\begin{bmatrix} x \\ y \end{bmatrix})$. The score of a pairwise alignment is defined to be the sum of the σ-values of its aligned pairs (i.e., columns). It should be noted that the techniques described in this paper hold for *affine gap costs*, that is scoring schemes under which a gap of length k is penalized $g + ek$, where g is a fixed "gap-opening penalty" and e is a "gap-extension penalty" [10, 14].

For sequences $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$, the *alignment graph*, $G_{A,B}$, is an edge-labeled directed graph. The nodes of $G_{A,B}$ are the pairs (i, j) where $i \in [0, M]$ and $j \in [0, N]$. The edge set for $G_{A,B}$ consists of the following edges, labeled as indicated.

1. $(i-1, j) \rightarrow (i, j)$ for $i \in [1, M]$ and $j \in [0, N]$, labeled $\begin{bmatrix} a_i \\ - \end{bmatrix}$
2. $(i, j-1) \rightarrow (i, j)$ for $i \in [0, M]$ and $j \in [1, N]$, labeled $\begin{bmatrix} - \\ b_j \end{bmatrix}$

3. $(i-1, j-1) \rightarrow (i, j)$ for $i \in [1, M]$ and $j \in [1, N]$, labeled $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$

Figure 1 provides an example of the construction.

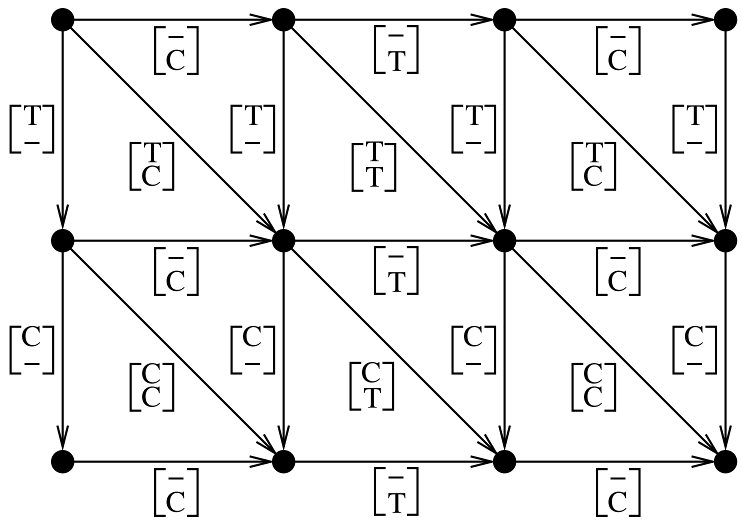

Figure 1. Alignment graph $G_{A,B}$ for the sequences $A = TC$ and $B = CTC$.

An alignment can be interpreted as a path through the rectangular dynamic programming matrix, from the upper left to lower right. Furthermore, if the score $\sigma(\pi)$ is assigned to each edge of $G_{A,B}$, where π is the aligned pair labeling that edge, then a path's score is exactly the score of the corresponding alignment. Let $S[i, j]$ denote the maximum score of a path from $(0, 0)$ to (i, j). The highest alignment score can be computed as follows:

$$S[i, j] \leftarrow \max \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ S[i-1, j] + \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix}) & \text{if } 1 \le i \le M \text{ and } 0 \le j \le N \\ S[i-1, j-1] + \sigma(\begin{bmatrix} a_i \\ b_j \end{bmatrix}) & \text{if } 1 \le i \le M \text{ and } 1 \le j \le N \\ S[i, j-1] + \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix}) & \text{if } 0 \le i \le M \text{ and } 1 \le j \le N \end{cases}$$

It is apparent that the scores in row i depend only on those in row $i-1$. Thus, the optimal alignment score for A and B can be computed in linear space. In the following, we describe methods for *delivering* optimal alignments using linear space in time proportional to the total number of the dynamic-programming matrix's grid-points.

2.1. Delivering an optimal alignment within a specified region

Rectangle

To construct an alignment in linear space, the score-only algorithm is modified so that it determines the point where an optimal path crosses the middle row (i.e., it finds the aligned pair involving $a_{M/2}$), which leaves two subproblems to be solved to complete the alignment. The critical observation for the time analysis is that the total area (i.e., time) of the subproblems is always exactly half that of the original problem [12, 14].

Repeating this procedure on each of the subproblems produces more aligned pairs and generates more "subsubproblems" that total one fourth of the original problem's area, as depicted in Figure 2. Letting T denote the score-only time, the time for solving all the problems that arise in this "divide-and-conquer" approach, and hence produce the complete alignment, is $T \times (1 + ½ + ¼ + \cdots) \approx 2T$.

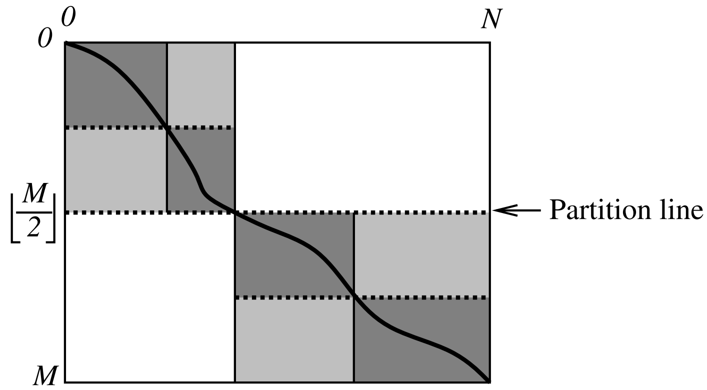

Figure 2. Hirschberg's divide-and-conquer approach.

Parallelogram

There are situations where the ith entry of the first sequence can be aligned to the jth entry of the second sequence only if $L[i] \le j \le U[i]$, for given lower and upper bounds L and U. In this case, a direct application of Hirschberg's approach might incur additional log factor in running time [2]. A new approach based on partition-line paradigm was proposed in [2, 3]. The basic idea is to define a *partition line* that bisects the region into two nearly equal parts. Unlike the earlier approach, which recursively divides the problem into two subproblems, the new method determines a variable number of subproblems. The total size of the subproblems is restricted to at most α times the size of the containing problem, for some $\alpha < 1$.

Consider the case when the region is a parallelogram, i.e., $U[i] = L[i] + C$, for some constant C. A grid point (i, j) is a *partition point* if $j = ½(L[i] + U[i])$. In the score-only backward pass we retain, for each grid-point on the current row and partition line, previous partition point where an optimal path from that grid-point crosses the partition line. When the backward pass reaches the upper left corner, this information allows us trace along an optimal path to determine all the crossing points. (Of course, we also determine which side of the partition line contains the subproblem.) Each subproblem is then solved recursively. Figure 3 depicts the approach.

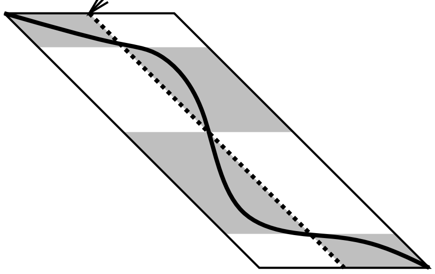

Figure 3. Aligning within a parallelogram.

Band

Consider the situation where the ith entry of the first sequence can be aligned to the jth entry of the second sequence only if $l \le j - i \le u$, for given band limits l and u. Lemma 1 immediate suggests that mid-diagonal is a proper partition line. That is, define a grid point (i, j) to be a *partition point* if $j = ½(l + u)$.

Lemma 1 Any diagonal band can be covered by a parallelogram whose area is at most double the size of the band.

Proof. Let P be the area of any given diagonal band (see Figure 4). Let a, b, c and d be corner sides and A, B, C and D be the corresponding triangle areas. Let h_1 and h_2 be the sides of the bounding rectangle.

Without loss of generality, assume $0 < A + D \le B + C$. (If $A + D = 0$, the band is already a parallelogram.)

$$
\begin{aligned}
& A + D \le B + C \\
\equiv\ & A + D + P \le B + C + P \\
\equiv\ & (a + b)h_1 \le (a + b)h_2 \\
\equiv\ & h_1 \le h_2
\end{aligned}
$$

We now prove that $P \ge A + D$. Indeed,

$$
\begin{aligned}
P &= h_1 h_2 - ½(h_1 - d)^2 - ½(h_1 - a)^2 \\
&\ge h_1^2 - ½h_1^2 - ½h_1^2 + (a + d)h_1 - ½(a^2 + d^2) \\
&\ge ½(a^2 + d^2) \qquad (h_1 \ge \max\{a, d\})
\end{aligned}
$$

□

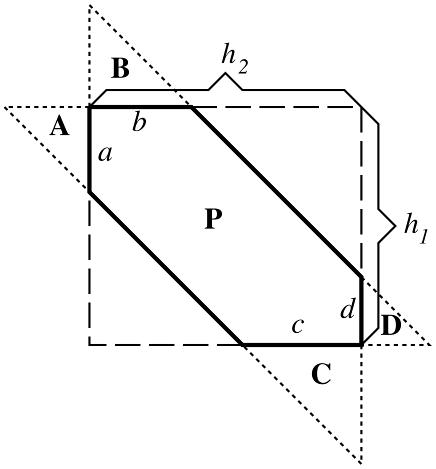

Figure 4. Covering a diagonal band with a parallelogram.

Arbitrary boundary limits

In [3], the partition-line approach is extended to general L and U. A grid point (i, j) is defined to be a *partition point* if either (1) $\max\{L[i], mid(i-1)\} \le j \le mid(i)$ or (2) $mid(i) < j \le \min\{L[i+1], U[i]\}$. Let R be the number of grid points in the feasible region, let P be number of aligned pairs in the computed optimal alignment, and let T be the total number of grid points in all computed subproblems. It is proved that T is no more than $2R + 4P$ [3].

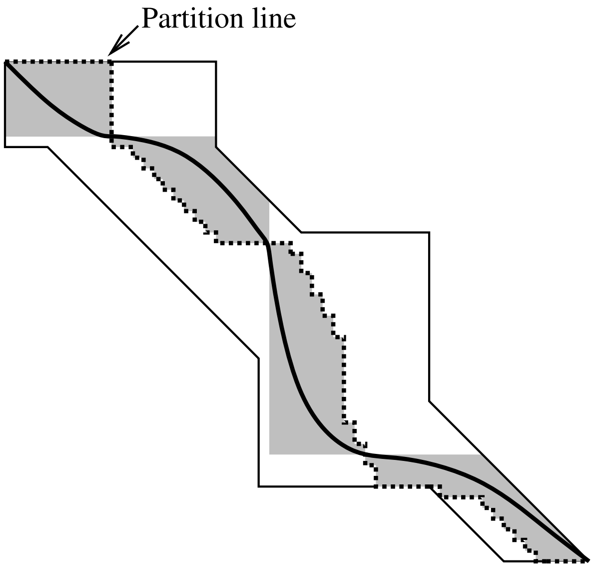

Figure 5. Aligning within a region.

2.2. Constructing a compact representation for all suboptimal alignments

Unconstrained case (rectangle)

Suppose we are given a threshold score Δ that does not exceed the optimum score. A Δ-path is a path from $(0,0)$ to (M, N) with score at least as large as Δ. A Δ-point is a grid point where at least one of its nodes appears in some Δ-path. A Δ-edge is an edge that appears in some Δ-path. In [5], a linear-space algorithm is proposed to compute a directed acyclic graph, denoted by $DAG_\Delta = (V_\Delta, E_\Delta)$, where V_Δ is the set of nodes in all Δ-points and E_Δ is the set of all Δ-edges. The time, however, is output-sensitive. Let F be the area bounded by the suboptimal alignments and W the minimum of the maximum column width and maximum row width spanned by the suboptimal alignments. For two sequences of length M and N, it is shown that the worst-case running time is $O(MN + F \log\log W)$. The basic idea is to divide the dynamic-programming matrix by both rows and columns. As the subproblem size is reduced, more partition rows and columns can be used.

Computing all suboptimal paths in a constrained region

There are situations where the ith entry of the first sequence can be aligned to the jth entry of the second sequence only if $L[i] \le j \le U[i]$, for given lower and upper bounds L and U. A straightforward modification of the algorithm of [5] results in a $O(R \log\log \min\{M, N\})$-time algorithm, where R is the area of the feasible region. The following approach yields a $O(R + F \log\log W)$-time algorithm.

The strategy is to cover the region with a small number of non-intersecting upright rectangles whose total area does not exceed a constant p (>1) times that of the rectangle's intersection with the given region. By "area" we mean the number of grid points in the interior or on the border. These rectangles are positioned in a greedy fashion; we repeat the process of placing a new rectangle that covers the first exposed and extends as far as possible. The algorithm of Figure 6 computes the set I containing the indices of first columns of all rectangles except the first. See Figure 7, where the dark line segments indicate the grid points of the leftmost column of a rectangle that can be reached along a single edge from the previous rectangle.

$I \leftarrow$ empty
$i_0 \leftarrow 0$
sum_widths $\leftarrow U[0] - L[0] + 1$
for $i \leftarrow 1$ **to** M **do**
{ *width* $\leftarrow U[i] - L[i] + 1$
  sum_widths $\leftarrow$ *sum_widths* + *width*
  if $(U[i] - L[i_0] + 1) \times (i - i_0 + 1) > p \times$ *sum_widths* **then**
  { $I \leftarrow I \cup \{i\}$
    $i_0 \leftarrow i$
    sum_widths $\leftarrow$ *width*
  }
}

Figure 6. Algorithm to partition a region of grid points.

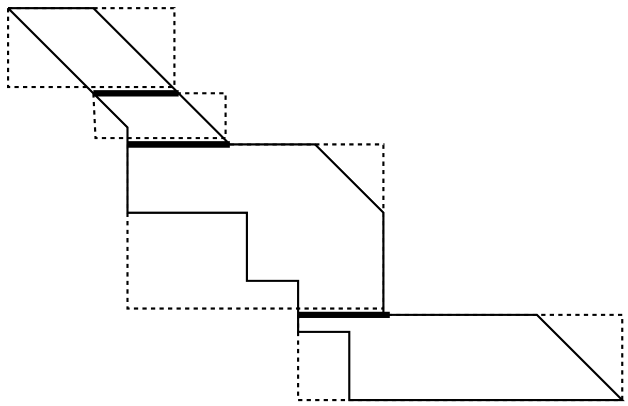

Figure 7. Covering the region defining a constrained alignment problem with non-intersecting rectangles.

Lemma 2 $\sum_{i \in I} (U[i] - L[i] + 1) < \frac{p+1}{p-1} N$.

Proof. Let $i_1, i_2, \cdots, i_n$ be the elements in I in increasing order. It is obvious that the lemma holds for the case $n \leq 1$. Suppose $n > 1$,

$$\begin{aligned}
&\sum_{s=1}^{n} (U[i_s] - L[i_s] + 1) \\
&= U[i_n] - L[i_1] + 1 + \sum_{s=1}^{n-1} (U[i_s] - L[i_{s+1}] + 1) \\
&< U[i_n] - L[i_1] + 1 + \frac{1}{p-1} \sum_{s=1}^{n-1} (U[i_{s+1}] - U[i_s] + L[i_{s+1}] - L[i_s]) \\
&= \frac{p}{p-1} (U[i_n] - L[i_1]) + \frac{1}{p-1} L[i_n] + (1 - \frac{1}{p-1} U[i_1]) \\
&< \frac{p}{p-1} (U[i_n] - L[i_1]) + \frac{1}{p-1} L[i_n] \\
&\leq \frac{p+1}{p-1} N
\end{aligned}$$

□

Lemma 2 implies that the sum of the segment widths $U[i] - L[i] + 1$ for $i \in I$ is less than $\frac{p+1}{p-1} N$.

This observation allows us to efficiently apply the algorithm of [5] to constrained alignment problems. The algorithm of Figure 6 is first used to break the problem into sections. A backward pass through the entire region allows us to compute and retain values for all nodes on dark line segments of Figure 7, which by Lemma 2 involves only linear space.

The sections of the region are then treated from top to bottom. Each section is first reduced to a special case of the general subproblem depicted in Figure 8, then the algorithm of [5] for computing all DAG_Δ is applied, with some extra code that keeps all computations to feasible grid points.

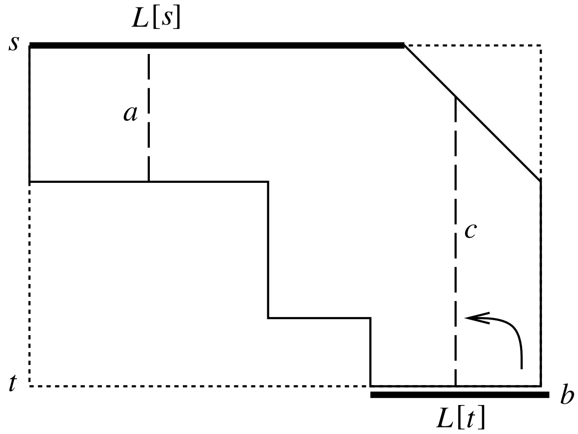

Figure 8. Conversion of a section of the region to a constrained subproblem, where a is the lowest Δ-point in the first column and b is the highest Δ-point in the last column.

It takes $O(R)$ time to initialize the score vectors for all sections. Let r_i and f_i be the area of the feasible region and the area bounded by the suboptimal alignments in the i^{th} section, respectively. With a similar argument to the proof of Theorem 7 in [5], it can be shown that the time for computing all Δ-points in the i^{th} section is $O(p \times r_i + f_i \log \log W)$. Therefore, the running time of entire process (breaking into sections and solving the pieces) is $O(R) + \sum_i O(p \times r_i + f_i \log \log W)$, which is $O(R + F \log \log W)$.

3. Discussion

We have implemented the linear-space methods in ANSI C. Currently, a graphical user interface written in Tcl/Tk is under construction. The expected product will be able to allow users to perform fast heuristic alignment methods, and then refine selected regions interactively. We will also build tools to browse pairwise alignment data, and to prepare publication-quality pictorial representations of alignments.

References

[1] Altschul, S., W. Gish, W. Miller, E. Myers and D. Lipman (1990) A basic local alignment search tool. *J. Mol. Biol.* **215**, 403-410.

[2] Chao, K.-M., W. R. Pearson and W. Miller (1992) Aligning two sequences within a specified diagonal band. *CABIOS* **8**, 481-487.

[3] Chao, K.-M., R. Hardison and W. Miller (1993) Constrained sequence alignment. *Bull. Math. Biol.* **55**, 503-524.

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

[5] Chao, K.-M. (1994) Computing all suboptimal alignments in linear space. *Combinatorial Pattern Matching '94*, Lecture Notes in Computer Science **807**, 31-42.

[6] Chao, K.-M., R. C. Hardison and W. Miller (1994) Recent developments in linear-space alignment methods: a survey. *Journal of Computational Biology* **1**, 271-291.

[7] Chao, K.-M. and W. Miller (1995) Linear-space algorithms that build local alignments from fragments. *Algorithmica*, **13**, 106-134.

[8] Doolittle, R. F. (1990) *Methods in Enzymology, Vol. 183*, "Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences," R. F. Doolittle, ed.

[9] Fitch, W. M. and T. F. Smith (1983) Optimal sequence alignments. *Proc. Natl. Acad. Sci. USA* **80**, 1382-1386.

[10] Gotoh, O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* **162**, 705-708.

[11] Hardison, R. C., K.-M. Chao, S. Schwartz, N. Stojanovic, M. Ganetsky and W. Miller (1994) Globin Gene Server: a prototype E-mail database server featuring extensive multiple alignments and data compilation for electronic genetic analysis. *Genomics* **21**, 344-353.

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

[13] Huang, X. and W. Miller (1991) A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics* **12**, 337-357.

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

[15] Needleman, S. B. and C. D. Wunsch (1970) A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.* **48**, 443-453.

[16] Pearson, W. R. and D. J. Lipman (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci.* **85**, 2444-2448.

[17] Schwartz, S., W. Miller, C.-M. Yang and R. C. Hardison (1991) Software tools for analyzing pairwise alignments of long sequences. *Nucleic Acids Res.* **17**, 4663-4667.

[18] Waterman, M. S. (1984) Efficient sequence alignment algorithms. *J. theor. Biol.* **108**, 333-337.

[19] Wilbur, W. J. and D. J. Lipman (1983) Rapid similarity searches of nucleic acid and protein sequence data banks. *Proc. Natl. Acad. Sci. U.S.A.* **80**, 726-730.

[20] Wilbur, W. J. and D. J. Lipman (1984) The context dependent comparison of biological sequences. *SIAM J. Appl. Math.* **44**, 557-567.