Progressive Multiple Alignment with Constraints

Gene Myers[*] Sanford Selznick[*] Zheng Zhang[†] Webb Miller[†]

June 18, 1996

Abstract

A progressive alignment algorithm produces a multi-alignment of a set of sequences by repeatedly aligning pairs of sequences and/or previously generated alignments. We describe a method for guaranteeing that the alignment generated by a progressive alignment strategy satisfies a user-specified collection of constraints about where certain sequence positions should appear relative to others. Given a collection of C constraints over K sequences whose total length is N, our algorithm takes $O(K(N^2 + KC))$ time. An alignment of the β-like globin gene clusters of several mammals illustrates the practicality of the method.

Key words: Multiple sequence alignment, constrained alignment, dynamic programming

1 Introduction

It is straightforward to extend the dynamic programming alignment algorithm (Needleman and Wunsch 1970) to the simultaneous alignment of $K > 2$ sequences. However, the $O(2^K N^K)$ execution time for sequences of length N makes it impractical to align more than three sequences this way, unless the sequences are extremely short. In practice, it is common to compute multiple-sequence alignments with a

[*]Department of Computer Science, University of Arizona, Tucson, AZ 85721. Supported by grant LM04960 from the National Library of Medicine and by the Aspen Center for Physics.

[†]Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. Supported by grant LM05110 from the National Library of Medicine.

progressive alignment strategy. Although the computed alignment cannot be guaranteed to be mathematically optimal, large numbers of sequences can be aligned.

Progressive alignment works as follows. Two of the K sequences are aligned together, and the resulting pairwise alignment replaces the two sequences. This gives an alignment problem with $K - 1$ "sequences", one of which is a sequence of aligned pairs (i.e., each of its "symbols" is a column of the pairwise alignment). Two of those sequences are chosen, aligned together, and replaced by that alignment, until only a K-way alignment remains. What makes this possible is that the dynamic programming algorithm for aligning two sequences can be made to work when one or both of the input sequences is itself an alignment of some of the originally-given sequences. This strategy can be traced to a paper of Waterman and Perlwitz (1984), and has been implemented, with numerous variations on the basic theme, by many investigators (Feng and Doolittle 1987, Taylor 1987, Corpet 1988, Higgins and Sharpe 1988).

Alignment algorithms, even theoretically optimal ones, are typically designed to perform well on a wide range of datasets. On the other hand, a program's user frequently has knowledge of their particular data that should override the conventional wisdom embodied in the program. For example, it might be known that in all members of a certain protein family a particular cysteine residue enters into a disulfide bond, so an alignment of members of that family should place all those cysteines in the same column even if the program awards a higher score to some other configuration.

This paper presents an algorithm for progressive alignment under a very general class of constraints. The last section describes an example of constrained alignment of genomic DNA sequences.

2 Constraints

Consider sequences $A^1, A^2, \ldots, A^K$, where $A^s = a_1^s a_2^s \ldots a_{N_s}^s$ has length N_s, and let $N = \sum_s N_s$. An *alignment* of the K sequences is a rectangular matrix with K rows, each composed of sequence entries and dashes (indicating gaps introduced into a sequence), such that (1) removing dashes from row s leaves sequence A^s for each $s \in [1, K]$ and (2) no column consists entirely of dashes.

A constraint denoted by $i_s \prec j_t$ asserts that the symbol a_i^s should occur in a column of the alignment strictly before the column containing the symbol a_j^t. We call i_s a *sequence position*, since it designates the i^{th} position in the sequence A^s. We also consider constraints of the form $i_s \preceq j_t$, which asserts that the symbol a_i^s should occur in the same column as, or before the column containing, a_j^t. Given a collection $\mathcal{C}$ of such constraints, we wish to produce a multi-alignment that satisfies all of the constraints in $\mathcal{C}$.

The two types of constraints are not equivalent. One might be tempted to think that $(i-1)_s \prec j_t$ is the same as $i_s \preceq j_t$, but this is not so as the former permits a_i^s to occur in a column after a_j^t. On the other hand, a constraint that i_s and j_t are in the same column is equivalent to the two constraints $i_s \preceq j_t$ and $j_t \preceq i_s$. Indeed an assertion that $k \leq K$ sequence positions in distinct sequences should be in the same column is readily expressed by a cyclic chain of k $\preceq$-constraints. Similarly, negation of a constraint can be expressed since, e.g., $i_s \not\prec j_t$ is equivalent to $j_t \preceq i_s$. We can thus restrict our attention to a collection $\mathcal{C}$ of $\prec$- and $\preceq$-constraints.

Note that any multi-alignment satisfies the constraints $(i-1)_s \prec i_s$ for all $s \in [1, K]$ and $i \in [2, N_s]$, since the entries of sequence s occur in order within the corresponding alignment row. We call such constraints *implicit* and denote the set of implicit constraints by $\mathcal{I}$. In contrast, the constraints of $\mathcal{C}$ are *explicit*.

3 A Consistency Test

We say that the set $\mathcal{C}$ of constraints is *consistent* if there exists at least one multi-alignment of the K sequences satisfying all of the constraints. This section develops efficient algorithms to test for consistency.

Given $\mathcal{C}$, consider the graph G whose vertices are the N sequence positions and whose edges are directed between vertex pairs that are related either explicitly or implicitly. Each edge has an associated *type*, which is either $\prec$ or $\preceq$. In particular, implicit edges are always of type $\prec$, while each explicit edge has the type of the constraint defining the edge. If more than one constraint exists between an ordered pair of sequence positions, then the edge is of type $\prec$ if at least one of the constraints is of type $\prec$, and of type $\preceq$ otherwise.

Theorem 1 *Let G be the graph constructed as above for a given set of constraints. There exists an alignment that is consistent with the constraints if and only if each cycle in G involves only edges of type $\preceq$.*

Proof. First suppose that there exists a cycle in G containing a $\prec$ edge, say $i_s \prec j_t$. The remainder of the cycle from j_t back to i_s demands that $j_t \preceq i_s$, an impossibility.

For the converse direction, suppose that every cycle of G involves only $\preceq$ edges. Form the graph H whose nodes are the strongly connected components of G and where there is an edge in H from u to v iff there is an edge from some G-node contained in u to some G-node in v. Edges in H are not assigned a type (i.e. $\prec$ or $\preceq$) and by definition H is acyclic. A node of H cannot contain two distinct positions in the same sequence, since otherwise there would be a cycle in G containing an implicit $\prec$ edge. For each node of H, form an alignment column whose non-dash entries are precisely the sequence positions in the node, and order the columns according to some topological sort of H. This ordering guarantees that all of the constraints in $\mathcal{C} \cup \mathcal{I}$ are met. Thus the resulting matrix is an alignment and it satifies all of the explicit constraints. □

Theorem 1 leads directly to the following algorithm to check for consistency. Given a set of C explicit constraints, the graph G is constructed in $O(N+C)$ time and space. Then the graph's strongly connected components are determined in time proportional to size of the graph, using any of several known algorithms (e.g., Cormen et al. 1990, pp. 488-493). By definition two vertices are in the same component if and only if there is a cycle containing both of them. Thus there is a cycle involving a $\prec$ edge if and only if some $\prec$ edge connects two vertices in the same component. In the final step, the algorithm inspects each $\prec$ edge and thereby checks consistency. Thus consistency can be determined in $O(N + C)$ time and space.

The algorithm can be improved in cases where C is much smaller than N, in particular, when $C \log C = o(N)$. Term a sequence position that occurs in some constraint of $\mathcal{C}$ *active*. Consider the graph G_c whose vertices are just the active sequence positions. Let there be an edge $v \rightarrow w$ from vertex v to vertex w iff either (1) $v \preceq w$ or $v \prec w$ is a constraint in $\mathcal{C}$, or (2) $v = i_s$, $w = j_s$ (i.e., v and w are positions in the same sequence), $i < j$, and there does not exist an active position k_s such that $i < k < j$. Such an edge $v \rightarrow w$ has type $\prec$ if either $v \prec w$ is in $\mathcal{C}$ or case (2) applies, and type $\preceq$ otherwise. Observe that G_c has at most $2C$ vertices and $3C$ edges. In essence, G_c is a sparse encoding of G where chains of implicit edges whose internal vertices are not active have been collapsed into a single edge. It then follows that checking this graph for a $\prec$ edge connecting two vertices in the same strongly connected component also determines the consistency of the constraints. Using a comparison-based method to sort the active positions in each sequence, G_c can be constructed in $O(C \log C)$ time and $O(C)$ space. Computing strongly connected components and checking their edges takes an additional $O(C)$ time. Thus consistency can be determined in $O(C \log C)$ time and $O(C)$ space.

4 Transitively Implied Constraints

Given that $\mathcal{C}$ is consistent, our goal is to build up a multi-alignment of the K sequences by progressively aligning pairs of sequences or smaller alignments in a way that results in a multi-alignment satisfying the constraints. Clearly, when aligning two of the given sequences, one must select only from pairwise alignments that satisfy the constraints between the two sequences. However, because of the transitivity of the constraint relations, one must be even more careful. For example, if $\mathcal{C}$ contains $i_s \prec k_u$ and $k_u \preceq j_t$, then one must ensure that $i_s \prec j_t$ in any pairwise alignment between A^s and A^t, since otherwise it would be impossible to align the resulting alignment of A^s and A^t with A^u to produce a three-way alignment satisfying $\mathcal{C}$.

The following two relations capture the transitive implications of a given set $\mathcal{C}$ of constraints. Define the constraint $p \preceq^+ q$ iff either $p = q$ or there exists a chain $p = r_0 \lhd_1 r_1 \lhd_2 \ldots \lhd_L r_L = q$ such that $r_{k-1} \lhd_k r_k$ is in $\mathcal{C} \cup \mathcal{I}$ for each k, and define the constraint $p \prec^+ q$ iff there exists such a chain with at least one (possibly implicit) $\prec$ constraint. (*Note:* In this context, r_k *does not* mean the rth position in the kth sequence, but instead denotes the kth sequence position in the chain.) We denote by $\mathcal{C}^+$ the set of $\preceq^+$ and $\prec^+$ constraints arising from constraints $\mathcal{C}$. Note that $\mathcal{C} \cup \mathcal{I} \subseteq \mathcal{C}^+$. Because the relationships in $\mathcal{C}^+$ model every transitive chain of $\mathcal{C} \cup \mathcal{I}$ it certainly follows that every alignment satisfying the constraints of $\mathcal{C}$ satisfies those of $\mathcal{C}^+$ and vice versa.

The following definitions will be used to formulate Theorem 2, which formalizes the key observation concerning transitively implied constraints. The *restriction to* A^s and A^t of a set $\mathcal{D}$ of constraints is the set of all $p \lhd q$ in $\mathcal{D}$ such that p and q are positions in either A^s or A^t. An alignment $\mathcal{B}$ is a *subalignment* of alignment $\mathcal{A}$

if $\mathcal{B}$ can be obtained from $\mathcal{A}$ by deleting zero or more rows and then removing any columns that consist entirely of dashes. A sample use of these notions is provided by the following straightforward converse to Theorem 2: the subalignment consisting of rows s and t of an alignment of sequences $A^1, A^2, \ldots, A^K$ satisfying $\mathcal{C}$ is a pairwise alignment satisfying the restriction to A^s and A^t of $\mathcal{C}^+$.

Theorem 2 *Let constraints $\mathcal{C}$ for sequences $A^1, A^2, \ldots, A^K$ be consistent, and let $\mathcal{B}$ be a pairwise alignment of A^s and A^t that satisfies the restriction to A^s and A^t of $\mathcal{C}^+$. Then there exists an alignment $\mathcal{A}$ of $A^1, A^2, \ldots, A^K$ satisfying $\mathcal{C}$ such that $\mathcal{B}$ is the subalignment consisting of rows s and t of $\mathcal{A}$.*

Proof. Let $\mathcal{D}$ be the set of all constraints satisfied by $\mathcal{B}$. To be more precise, let $col_{\mathcal{B}}(i_s)$ denote the index of the column of $\mathcal{B}$ in which i_s occurs. Then $i_s \preceq j_t$ is in $\mathcal{D}$ iff $col_{\mathcal{B}}(i_s) \leq col_{\mathcal{B}}(j_t)$, and $i_s \prec j_t$ is in $\mathcal{D}$ iff $col_{\mathcal{B}}(i_s) < col_{\mathcal{B}}(j_t)$. Note that an alignment of $A^1, A^2, \ldots, A^K$ satisfies $\mathcal{D}$ if and only if the subalignment consisting of rows s and t is exactly $\mathcal{B}$.

We want to verify the existence of an alignment that satisfies all constraints in $\mathcal{C} \cup \mathcal{D}$. By Theorem 1, it is sufficient to examine cycles $r_0 \lhd_1 r_1 \lhd_2 \ldots \lhd_L r_L = r_0$, where each $r_{k-1} \lhd_k r_k$ is in $\mathcal{C} \cup \mathcal{D} \cup \mathcal{I}$. Within the cycle, each run of consecutive constraints from $\mathcal{C} \cup \mathcal{I}$ can be replaced by a single constraint in $\mathcal{C}^+$. Thus the cycle can be replaced by one where the successive constraints, $r_{k-1} \lhd_k r_k$, alternate between a member of $\mathcal{C}^+$ and a member of $\mathcal{D}$, whence every sequence position in the cycle is in either A^s or A^t. But since $\mathcal{B}$ satisfies the restriction to A^s and A^t of $\mathcal{C}^+$, that restriction is a subset of $\mathcal{D}$, i.e., all constraints in the cycle are in $\mathcal{D}$.

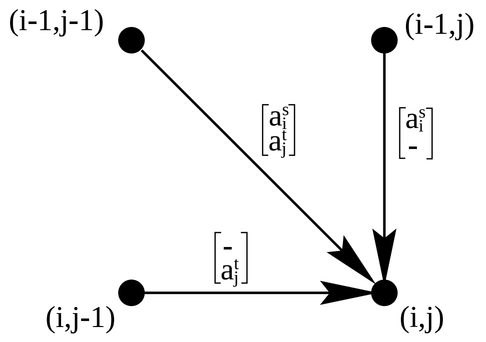

(i-1,j-1)
(i-1,j)
(i,j-1)
(i,j)
$\begin{bmatrix} a_i^s \\ a_j^t \end{bmatrix}$
$\begin{bmatrix} a_i^s \\ - \end{bmatrix}$
$\begin{bmatrix} - \\ a_j^t \end{bmatrix}$

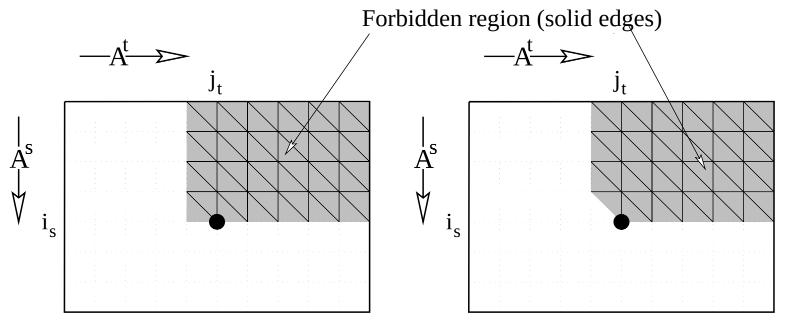
Figure 1: Edges entering node (i, j) in the edit graph and their labels.

Consistency of $\mathcal{D}$ implies that the cycle contains only $\preceq$ constraints, so the original cycle (with constraints in $\mathcal{C} \cup \mathcal{D} \cup \mathcal{I}$) contains only $\preceq$ constraints. □

5 Constrained Pairwise Alignment

We now address the problem of computing a pairwise alignment subject to a constraint of the form $p \preceq^+ q$ or $p \prec^+ q$. It is helpful to think in terms of an *edit graph* for sequences A^s and A^t. We will frame the discussion in terms of the simplest kind of edit graph, which is appropriate when the pairwise alignment is optimized with respect to the sum of scores for each column (including those containing dashes). The discussion carries over with only minor changes to graphs that are based on scores more appropriate for progressive alignment of biological sequences (e.g., Chao et al. 1994, pp. 280-282).

The simplest kind of edit graph for A^s and A^t consists of an (N_s+1)-by-(N_t+1) matrix of vertices $V[0..N_s][0..N_t]$. The vertex at gridpoint (i, j) with $i > 0$,



(a) $i_s \prec^+ j_t$ (b) $i_s \preceq^+ j_t$

Figure 2: Edges forbidden by $\prec$ and $\preceq$ constraints.

$j > 0$ has three entering edges, each of which is labeled by an alignment column, as pictured in Figure 1. (Vertices in row 0 and column 0, except for $(0, 0)$, have only one entering edge.) To each path from $(0, 0)$ to (N_s, N_t) there corresponds the alignment of concatenated edge labels; this gives a one-to-one correspondence between such paths and alignments of A^s and A^t. Note that the alignment column containing a^s_i comes from the edge where the path enters row i, and a^t_j comes from the edge entering column j.

The constraint $i_s \prec^+ j_t$ requires that the alignment's path reach row i strictly before it reaches column j. Thus the constraint is equivalent to requiring that the path avoid edges in the interior of the *forbidden region* $V[0..i_s][j_t - 1..N_t]$. The constraint $i_s \preceq^+ j_t$ differs only in allowing the column aligning a^s_i and a^t_j, so its forbidden region differs from that for $i_s \prec^+ j_t$ by permitting use of the diagonal edge from $(i - 1, j - 1)$ to (i, j). Figure 2 illustrates these forbidden regions.

6 Prime Constraints

This section and the next describe how to delimit, for all sequence pairs A^s and A^t, the forbidden regions corresponding to the restriction to A^s and A^t of $\mathcal{C}^+$. The number of these constraints can be quadratic in the sequence lengths; for instance, this happens if $\mathcal{C}$ contains a constraint $i_s \lhd j_t$ with i not too near 1 and j not too near N_t. Fortunately, we need only consider the constraints whose forbidden region is not contained in the forbidden region of another constraint.

A constraint $i_s \prec^+ j_t$ will be called *prime* if (1) no $h > i$ satisfies $h_s \prec^+ j_t$ or $h_s \preceq^+ j_t$ and (2) no $h < j$ satisifies $i_s \prec^+ h_t$ or $i_s \preceq^+ h_t$. Furthermore, a constraint $i_s \preceq^+ j_t$ is *prime* if it is not the case that $i_s \prec^+ j_t$. It follows directly from these definitions that the forbidden region of a non-prime constraint is contained within that of some other constraint. For example, if $i_s \prec^+ j_t$ is not prime because (1) is violated, then there exists an $h > i$ such that $h_s \prec^+ j_t$ or $h_s \preceq^+ j_t$. The forbidden region of either of these two constraints properly contains that of $i_s \prec^+ j_t$. Henceforward, we say that a constraint *dominates* another if its forbidden region properly contains that of the other. Now every non-prime constraint c_0 is dominated by some other constraint c_1. If c_1 is non-prime then it is dominated by some other constraint c_2. Continuing this chain inductively, one must eventually reach a prime constraint c_n or the chain must loop on itself. But the later is impossible as this would imply that a forbidden region properly contains itself. Thus every non-prime constraint is dominated by a prime constraint.

Let $L_{s,t}$ be the s-ordered list of prime constraints between s and t, i.e., $\langle i_s^1 \lhd_1$

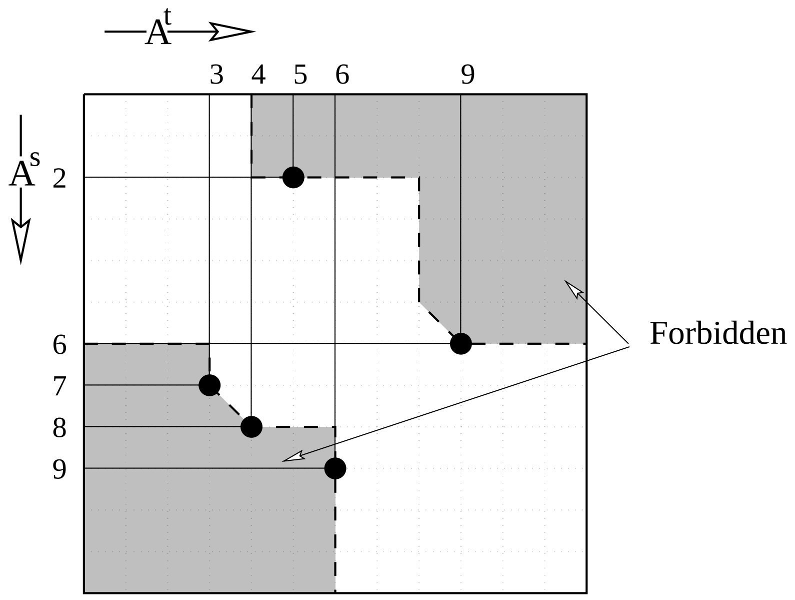


Figure 3: Forbidden regions when $L_{s,t} = \langle 2 \prec^+ 5,\ 6 \preceq^+ 9\rangle$ and $L_{t,s} = \langle 3 \prec^+ 7,\ 4 \preceq^+ 8,\ 6 \prec^+ 9\rangle$.

$j^1_t, i^2_s \lhd_2 j^2_t, \ldots, i^{len_{s,t}}_s \lhd_{len_{s,t}} j^{len_{s,t}}_t\rangle$ where $i^k_s < i^{k+1}_s$ and $\lhd_k$ is either $\preceq^+$ or $\prec^+$. Note that by primality it follows that $j^k_t < j^{k+1}_t$ for all k. The union of the forbidden regions of all constraints in the restriction to A^s and A^t of $\mathcal{C}^+$ equals the union of the forbidden regions for the prime constraints (this follows from the previous paragraph), and hence the union is properly specified by $L_{s,t}$ and $L_{t,s}$. Further note that there is at most one prime constraint whose right position is j_t for a given j, that this constraint can be uniquely charged to a constraint of $\mathcal{C}$ whose right position is j_t, and thus that the size of $L_{s,t}$ is bounded by the number of constraints in $\mathcal{C}$ whose right position is in A^t. Hence, for a fixed s, $\sum_{t \neq s} len_{s,t} \leq C$, where $\mathcal{C}$ contains C constraints.

7 An O(K+C) Prime Constraint Algorithm

It now remains to design an efficient algorithm for determining the ordered list $L_{s,t}$ for all $s \neq t$. Consider the sparse graph model G_c of the constraints in $\mathcal{C} \cup \mathcal{I}$ described in the last paragraph of Section 3. For algorithmic purposes assume that every vertex v is annotated with a $.pos$ and $.seq$ attribute, so that $v.pos_{v.seq}$ is the sequence position represented by v. Further assume that V_s is a sorted list of the set of vertices v for which $v.seq = s$ in increasing order of their $.pos$ attributes. Recall that the lists V_s are needed to construct G_c in the first place. Finally, to distinguish the types of edges in G_c we will write $v \rightarrow w$ if the constraint between v and w is of type $\prec$, and $v \Rightarrow w$ if the constraint is of type $\preceq$.

For a given sequence A^s, the algorithm of Figure 4 computes $L_{s,t}$ for all $t \neq s$ in $O(K + C)$ time and space. It does so by computing the prime constraint (if one exists) from each position in V_s to every other sequence in reverse order of V_s.

Theorem 3 *The algorithm of Figure 4 correctly determines $L_{s,t}$ for fixed s and all $t \neq s$.*

Proof. It suffices to prove the invariant that when one is about to enter the loop of lines 6-14 then (1) $L_{s,t}$ is an increasing sorted list of all prime constraints from s to t originating at a successor of v in V_s, (2) every vertex reachable from a successor of v in V_s has its $.mark$ field set to $\prec^+$, and all others have their's set to $?$, (3) $prime[t]$ is the leftmost position in sequence t that is reachable from a successor of v in V_s, and (4) $ctype[t] =?$ for all $t \neq s$. This is certainly true before the first

```
1.  procedure $L\_Lists(s : [1..K])$
2.  { $L_{s,t} \leftarrow \emptyset \; \forall t \neq s$
3.      $prime[t] \leftarrow \infty \; \forall t$
4.      $ctype[t] \leftarrow ? \; \forall t$
5.      $v.mark \leftarrow ?$ $\forall$ vertices $v$
6.      for $v \in V_s$ in reverse order do
7.      { $cnt \leftarrow 0$
8.          Search$(v, \preceq^+)$
9.          for $i \leftarrow 1$ to $cnt$ do
10.         { $t \leftarrow list[i]$
11.             $L_{s,t} \leftarrow (v.pos \; ctype[t] \; prime[t]) \bullet L_{s,t}$
12.             $ctype[t] \leftarrow ?$
13.         }
14.         $Markup(v)$
15.     }
16. }

17. procedure Search($v$: vertex, $type$: { $\prec^+$, $\preceq^+$ } )
18. { if $v.mark = ?$ or $(v.mark = \preceq^+$ and $type = \prec^+)$ then
19.     { $v.mark \leftarrow type$
20.         if $prime[v.seq] \geq v.pos$ then
21.         { if $ctype[v.seq] = ?$ and $v.seq \neq s$ then
22.                 $list[cnt \leftarrow cnt + 1] \leftarrow v.seq$
23.             $prime[v.seq] \leftarrow v.pos$
24.             $ctype[v.seq] \leftarrow type$
25.         }
26.         for $v \rightarrow w$ do
27.             Search$(w, \prec^+)$
28.         for $v \Rightarrow w$ do
29.             Search$(w, type)$
30.     }
31. }

31. Markup($v$: vertex )
32. { if $v.mark \neq \prec^+$ then
33.     { $v.mark \leftarrow \prec^+$
34.         for $v \Rightarrow w$ do
35.             Markup$(w)$
36.     }
37. }
```

Figure 4: Algorithm to determine the lists $L_{s,t}$ for fixed s.

execution of line 8 by the action of lines 2-5. Lines 18, 19, and 26-29 implement a traversal from the originating s vertex that traverses a vertex once if it is first reached via a $\prec^+$ chain and possibly twice if it is first reached via a $\preceq^+$ chain. An additional traversal realized by *Markup* advances the mark field of all vertices marked with $\preceq^+$ to $\prec^+$ so that clause (2) of the invariant is true upon completion of the loop body. Since all vertices reachable from a successor of v have been visited, the test at line 20 guarantees clause (3). Similarly, cnt and the $list$ array are manipulated in lines 21-22 and lines 9-12, so that $ctype[t]$ is reset if it gets set during the marked traversal. Thus clause (4) is also true upon completion.

What remains is to prove the invariance of clause (1). We first show that after executing line 8, if there is a prime constraint originating at $v.pos_s$ and ending in sequence t then it is the constraint $v.pos_s$ $ctype[t]$ $prime[t]_t$, and otherwise $prime[t] = \infty$. Suppose that $v.pos_s \prec^+ j_t$ is *the* prime constraint originating at $v.pos_s$. Then the vertex for j_t is marked $?$, for otherwise there is a successor w of v on V_s for which $w.pos_s \prec^+ j_t$ or $w.pos_s \preceq^+ j_t$. In either case w contradicts the primality of $v.pos_s \prec^+ j_t$. By primality, $j < h$ for any other position h_t for which $v.pos_s \prec^+ h_t$. Thus when j_t is reached, $prime$ will be properly set to j. The only other possibility is that $v.pos_s \preceq^+ j_t$, but because *Search* retraverses vertices marked $\preceq^+$ when reached by a $\prec^+$-chain the $\prec^+$ constraint will take precedence. Thus $ctype$ will be properly set. The argument for $v.pos_s \preceq^+ j_t$ is much the same, save that we need not worry about the possibility of $v.pos_s \prec^+ j_t$ as it would imply the constraint is not prime.

The final observation is that only prime constraints are added to L-lists. If $prime[t] = p$ before executing line 8, and if $v.pos_s \prec j_t$ is added to list $L_{s,t}$ at line 11, then $j < p$ because of invariant (2) and the test at line 20. Invariant (4) then guarantees that the constraint is prime. □

8 A Multiple Alignment Algorithm

An algorithm for progressive alignment of sequences $A^1, A^2, \ldots, A^K$ subject to constraints $\mathcal{C}$ is given in Figure 5. As noted in Section 3, the constraint graph, G_c, for the original set of sequences can be determined in time $O(\min\{N+C, C \log C\})$, where N is the sum of the sequence lengths and C is the size of $\mathcal{C}$. At each subsequent iteration of the **while** loop, G_c can be updated as follows. Let s and t be the sequences selected at line 6 in the previous iteration, let $v_1, v_2, ..., v_n$ be the vertices of s and let $u_1, u_2, ..., u_m$ be the vertices of t. The two lists can be merged to get the implicit constraint edges for vertices of s', which replace the implicit contraints between two u or two v vertices and explicit contraints between a u vertex and a v vertex (or vice versa). A u vertex and a v vertex that correspond to the same column of s' are represented as a single node, with the appropriate changes to adjacency lists. Thus subsequent executions of line 3 take time $O(C)$.

Each execution of line 4 involves $|\mathcal{S}|$ executions of Figure 4, and hence requires time $O(K^2 + KC)$. Computation of all pairwise alignments takes time $O(N^2)$. (For the time being, assume that pairwise alignment of sequences of lengths N_s and N_t takes time $O(N_s N_t)$, even if each sequence is an alignment of several A^i. See

```
1.  $\mathcal{S} \leftarrow \{A^1, A^2, \ldots, A^K\}$
2.  **while** $|\mathcal{S}| > 1$ **do**
3.  {   $G_c \leftarrow$ the constraint graph for $\mathcal{S}$
4.      compute $L_{s,t}$ for all $s, t \in \mathcal{S}$ with $s \neq t$
5.      compute all constrained pairwise alignments for $s, t \in \mathcal{S}$
6.      $s, t \leftarrow$ the most similar pair in $\mathcal{S}$
7.      $s' \leftarrow$ the alignment of $s$ and $t$
8.      $\mathcal{S} \leftarrow \mathcal{S} - \{s, t\} \cup \{s'\}$
9.  }
```

Figure 5: An algorithm for progressive multiple alignment with constraints.

the next section for a different analysis that is appropriate for a particular alignment-scoring scheme.) It follows that the algorithm of Figure 5 runs in time $O(K(N^2 + KC))$.

In practice, it may be possible to reduce the running time by avoiding some of the pairwise-alignment computations. For example, after $L_{s,t}$ and $L_{t,s}$ are recomputed, one can check in linear time whether the previous alignment of s and t satisfies the new constraints. If so, then no recomputation is required.

9 An Implementation

$Yama2$ (Chao et al. 1994) is a progressive multi-alignment program used for genomic DNA sequences. The order in which sequences are aligned is predetermined according to the assumed evolutionary relationships among the sequences. $Yama2$ scores a multiple alignment as the sum of scores of the implied pairwise alignments, using quasi-nataral gap costs (Altschul 1989). With these scores, progressive alignment of K sequences takes time $O(K^2L^2)$ (Chao et al. 1994), where the final alignment has L columns.

We modified $yama2$ to handle constraints, calling the new program $yama3$. The implementation task was simplified by the fact that $yama2$ already limited each progressive step to a region of the dynamic-programming grid determined by a lower and an upper column bound for each row. The earlier algorithm (Chao et al. 1993) defined a "forbidden region" strictly in terms of vertices, and required modification to disallow an edge in the case of a $\prec$ constraint. Each of the $K-1$ merge steps requires time $O(C)$ to update G_c and time $O(K+C)$ to determine the forbidden regions. (Only the regions for the two sequences being aligned at this step are required.) Thus the program runs in time $O(K^2L^2+KC)$.

10 An Example

One $yama2$ alignment, which includes a 73,308-nucleotide sequence from the human β-like globin gene cluster, can be accessed by electronic mail (Hardison et al. 1994) or the WorldWide Web (`http://globin.cse.psu.edu`). It is intended that the alignment can be inspected to locate conserved regions falling outside of protein-coding segments, which may indicate signals associated with the regulation of gene expression. Of course, it is possible that regulatory sites which are homologous (i.e., descended from the same DNA region in the species' common ancestor) may not correspond under the current alignment. For any such cases, it would be desirable to force them to align using $yama3$.

A potential example of this phenomenon concerns GATA1 binding sites lying

```
                           -207                -187      -177
                  GATA1      |     GATA1         |         |
61915:    ACTGATGGTATGG----GGCCAAGAGATATATCTTAGAGGGAGGGCTGAG     human
34701:    G...C...G....----.A...T...C..C.C---.C............C     galago
30703:    ....C.........----.T.TGG......C..---...A...A.......     rabbit
10556:    .T..C...A...ACTGT.AT.T......GCCC---...AA.........T     cow
50533:    T...CACAG..AA----..A...ACAT...TC---.......TA.CC..     mouse
```

Figure 6: Portion of a $yama2$ alignment of the β-like globin gene clusters of several mammals. A dot indicates agreement with the entry of the human sequence in that column. Boxes are drawn around runs of six or more successive columns with at most one mismatch per column. GATA1 binding sites in the human and mouse sequences, as discussed in the text, are underlined.

about 200 basepairs before the transcription initiation site of the β-globin gene. GATA1 is a transcription factor known to be critical in regulation of the globin genes. Its consensus binding site is WGATAR, where W designates either A or T, and R designates either A or G. An imperfect match, AGATAT, at -200 (relative to the transcription start site) in the human sequence has been shown experimentally (deBoer et al. 1988) to bind GATA1, as has a conforming AGATAA at -215 in the mouse sequence (Macleod and Plumb 1991). However, these two regions do not align (Fig. 6), even when the alignment-scoring scheme is varied.

There is some experimental evidence that these sites play a functional role in regulation of the β-globin gene. They appear to have an effect on laboratory protocols that chemically induce expression of the β-globin gene in cultured MEL (mouse erythroleukemia) cells, which may indicate that the sites play a role in regulation of that gene *in vivo*. In particular, an experiment involving deletion of the human GATA1 site at -200 indicated that it can act with other sites to confer inducibility (deBoer et al. 1988), whereas point mutations in the -215 mouse GATA1 site re-

```
               -213      -207                -187
                 |         |    GATA1   GATA1  |         |
61915:  ACTGATGGTATGG----GGCCAAGAGATATATCTTAGAGGGA--------GGGCTGAG    human
34701:  G...C...G....----.A...T...C..C.C---.C.....--------.......C    galago
30703:  ....C........----.T.TGG......C..---...A...--------A.......    rabbit
10556:  .T..C...A...ACTGT.AT.T......GCCC---...AA..--------.......T    cow
50533:  T...C-----------------------.C.G---.T.A...CAAACATTATT.A...    mouse
```

Figure 7: Portion of the $yama3$ alignment computed with the single constrant: $61935_{\mathbf{human}} \preceq 50540_{\mathbf{mouse}}$. Note that the two GATA1 sites have reversed their order.

```
               -213      -207                -187      -179
                 |         |    GATA1          |         |
61915:  ACTGATGGTATGG----GGCCAAGAGATATATCTTAGAGGGA--GGGCTGAG          human
34701:  G...C...G....----.A...T...C..C.C---.C.....--.......C          galago
30703:  ....C........----.T.TGG......C..---...A...--A.......          rabbit
10556:  .T..C...A...ACTGT.AT.T......GCCC---...AA..--.......T          cow
50533:  T...C-----------------.C.....AGG---.C.AAC.TTATT.A...          mouse
```

Figure 8: Portion of the $yama3$ alignment computed with the two constrants: $61935_{\mathbf{human}} \preceq 50540_{\mathbf{mouse}}$ and $50540_{\mathbf{mouse}} \preceq 61935_{\mathbf{human}}$. Note that the two GATA1 sites are now aligned.

lieved repression prior to induction (Macleod and Plumb 1991). On the other hand, different point mutations in the mouse GATA1 site had no effect either before or after induction by the hormone erythropoietin in two different cell lines (Taxman and Wojchowski 1995), and we feel that the putative homology and functional roles of the two GATA1 sites have yet to be fully verified.

In any case, assume for the moment that further experiments will confirm that the two GATA1 binding sites are homologous and that they play a role in regulating expression of the β-globin gene. With only one constraint given, $yama3$ fails to produce the desired alignment (Fig. 7), but it succeeds when given two constraints (Fig. 8).

References

[1] Altschul, S. 1989. Gap costs for multiple sequence alignment. *J. theor. Biol.* 138, 297–309.

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

[3] Chao, K.-M., Hardison, R., and Miller, W. 1994. Recent developments in linear-space alignment methods: A survey. *J. Computational Biology* 13, 271–291.

[4] Cormen, T., Leiserson, C., and Rivest, R. 1990. *Introduction to Algorithms*, MIT Press.

[5] Corpet, F. 1988. Multiple sequence alignment with hierarchical clustering. *Nucl. Acids. Res.* 16, 10881–10890.

[6] deBoer, E., Antoniou, M., Mignotte, V., Wall, L., and Grosveld, F. 1988. The human β-globin promoter; nuclear factors and erythroid specific induction of transcription. *The EMBO Journal* 7, 4203–4212.

[7] Feng, D., Doolittle, R. 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* 25, 351–360.

[8] Hardison, R., Chao, K.-M., Schwartz, S., Stojanov, N., Ganetsky, M., and Miller, W. 1994. Globin Gene Server: a prototype e-mail database server fea-

turing extensive multiple alignments and data compilation for electronic genetic analysis. *Genomics* 21, 344–353.

[9] Higgins, D., and Sharpe, P. 1988. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene* 73, 237–244.

[10] Macleod, K., and Plumb, M. 1991. Derepression of mouse β-major-globin gene transcription during erythroid differentiation. *Mol. and Cell. Biol.* 11, 4324–4332.

[11] Needleman, S., and Wunsch, C. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48, 443–453.

[12] Taylor, W. 1987, Multiple sequence alignment by a pairwise algorithm. *CABIOS* 3, 81–87.

[13] Taxman, D., and Wojchowski, D. 1995. Erythropoietin-induced transcription at the murine β^{maj}-globin promoter: a central role for GATA1. *J. Biol. Chem* 270, 6619–6627.

[14] Waterman, M., and Perlwitz, M. 1984. Line geometries for sequence comparison. *Bull. Math. Biol.* 46, 567–577.