The maximum weight trace problem in multiple sequence alignment

John Kececioglu

Computer Science Department, University of California at Davis, Davis, California 95616

Abstract. We define a new problem in multiple sequence alignment, called *maximum weight trace*. The problem formalizes in a natural way the common practice of merging pairwise alignments to form multiple sequence alignments, and contains a version of the minimum sum of pairs alignment problem as a special case.

Informally, the input is a set of pairs of matched characters from the sequences; each pair has an associated weight. The output is a subset of the pairs of maximum total weight that satisfies the following property: there is a multiple alignment that places each pair of characters selected by the subset together in the same column. A set of pairs with this property is called a *trace*. Intuitively a trace of maximum weight specifies a multiple alignment that agrees as much as possible with the character matches of the input.

We develop a branch and bound algorithm for maximum weight trace. Though the problem is NP-complete, an implementation of the algorithm shows we can solve instances on as many as 6 sequences of length 250 in a few minutes. These are among the largest instances that have been solved to optimality to date for any formulation of multiple sequence alignment.

1 Introduction

Multiple sequence alignment is among the outstanding problems of computational biology for which a satisfactory solution is unknown.

Briefly, multiple alignment may be defined as the following general problem. Given a set of sequences S_1, S_2, ..., S_k and a scoring function d, find a *multiple sequence alignment* $\mathcal{A} = (a_{ij})_{1 \le i \le k}$ that minimizes

$$\sum_j d(a_{1j}\, a_{2j} \cdots a_{kj}).$$

The entries a_{ij} of matrix $\mathcal{A}$ are either symbols from the sequence alphabet Σ or the *null character* ε, which is the identity under concatenation. The only constraint on $\mathcal{A}$ is that, concatenating characters $a_{i1}\, a_{i2} \cdots$ in any row i of the matrix, we must obtain the ith sequence S_i. Function d assigns a score to a column based on the characters that appear in it. In words, we seek an alignment $\mathcal{A}$ that minimizes the sum of the scores of its columns.

All the standard multiple sequence alignment problems may be cast in this form. For example in the *shortest common supersequence problem*, d counts the number of distinct symbols from Σ that appear in a column. The score of a multiple alignment is then the length of the supersequence it encodes.

In the *longest common subsequence problem*, d assigns the value -1 to a column consisting of k copies of a symbol from Σ, and assigns any other column the value 0. Minimizing the alignment score then maximizes the length of the common subsequence.

In the *minimum sum of pairs alignment problem* of Carrillo and Lipman [2],

$$d(a_{1j}\, a_{2j} \cdots a_{kj}) = \sum_{p<q} \delta(a_{pj}, a_{qj}),$$

where δ is a scoring function for pairs of symbols, including ε, satisfying $\delta(\varepsilon, \varepsilon) = 0$. In this problem the score of a multiple alignment is the sum of the scores of its pairwise alignments.

Even the problem of *multiple alignment under a fixed evolutionary tree*, in which we have a tree T with leaves labelled $S_1, \ldots, S_k$ and we seek ancestral sequences to label the internal nodes of T so as to minimize the sum of the edit distances along its edges, can be reduced to this general form, as shown by Sankoff [14].

As can be expected for a problem of such generality, multiple sequence alignment, as formulated above, is NP-complete. This follows from the NP-completeness of the longest common subsequence problem for a set of sequences [12], which we have noted is a special case of general multiple alignment.

On the other hand, by the early 1970s it was known that multiple sequence alignment could be solved by dynamic programming for k sequences of length n using $O(2^k n^k)$ evaluations of the scoring function d and $O(n^k)$ space. Thus for a fixed number of sequences, the problem is polynomial-time solvable.

This is hardly a practical solution, however. For a reasonable sized problem such as five sequences of length 100, the space requirements for dynamic programming are already more than ten gigabytes, even if the user could afford to wait for completion of the algorithm.

Consequently biologists have opted for suboptimal methods, such as the following. For every pair of sequences an optimal pairwise alignment is computed; then a set of pairwise alignments is selected that connect the sequences, and the pairwise alignments are merged to form a multiple alignment. Such an approach is fast, usually $O(k^2 n^2)$ time to compute the pairwise alignments and $O(k^2 \log k + kn)$ time to select a subset and merge them, and there is a well-developed theory for alignment of two sequences.

In the method of Feng and Doolittle [4] for instance, a maximum weight spanning tree of pairwise alignments is selected, where the weight of a pairwise alignment is its similarity score. This exploits the fact that any tree T of pairwise alignments can be merged into a multiple alignment that agrees with the alignments in T.

The main drawback of this and similar methods is that less than k of all $\binom{k}{2}$ pairwise alignments are used. Indeed, it is not hard to construct instances in which the merged alignment agrees with *only* the $k-1$ alignments of the tree, and none of the remaining alignments, even when there is a multiple sequence alignment that agrees with more than $\binom{k}{2} - k$ of the pairwise alignments [11].

This leads to the question we address in this paper, namely,

> *Given a set of pairwise alignments, how can we find a multiple alignment that is as close as possible to* all *pairwise alignments in the set?*

2 The maximum weight trace problem

To answer our question we must define "as close as possible."

The set of pairwise alignments may be represented by a graph, whose vertices are the characters[1] of the sequences, and whose edges are the pairs of characters matched in the alignments. The interpretation of vertices as characters in a sequence can be maintained by a vertex relation that captures the ordering of characters within a sequence.

Definition 1. An *alignment graph* $(V, E, \prec)$ for a set $\mathcal{S}$ of sequences is a graph (V, E) whose vertices V correspond to the characters of the sequences in $\mathcal{S}$, together with a relation $\prec$ on V. Relation $v \prec w$ holds if and only if character v immediately precedes character w in a sequence of $\mathcal{S}$. □

Consider a path of edges in an alignment graph. Each edge on the path is a pair of aligned characters; transitively the path is a set of characters to be aligned in one column. The connected components[2] then of a set of edges correspond to the columns of a multiple alignment. These columns form a valid alignment if they can be ordered so as to respect $\prec$ character by character.

We can represent this ordering of columns by shrinking each component to a supervertex, and directing an edge from one supervertex to another if a character in one component immediately precedes a character in the other. We express this formally by extending $\prec$ to a relation $\prec^*$ on sets of vertices. For subsets X and Y of V,

$$X \prec^* Y \text{ if and only if } (\exists x \in X)\ (\exists y \in Y) \text{ such that } x \prec y.$$

Relation $\prec^*$, unlike $\prec$, may not be a partial order. Figure 1 gives an example. The point is that a multiple alignment exists that agrees with a set E of edges precisely when the components of E have a linear ordering under $\prec^*$.

Definition 2. A multiple sequence *trace* of an alignment graph $(V, E, \prec)$ is a subset $T \subseteq E$ of the edges such that $\prec^*$ on the connected components of T is acyclic. □

A trace of maximum cardinality agrees with as many of the matches in an alignment graph as possible. We gain a little more flexibility by weighting edges and seeking a trace of maximum total weight.

Definition 3. The *maximum weight trace problem* is, given alignment graph $(V, E, \prec)$ and edge weight function w, find a trace $T \subseteq E$ maximizing $\sum_{e \in T} w(e)$. □

Maximum weight trace was originally defined in [11] as a way of forming a multiple alignment from pairwise overlaps to determine a consensus sequence for DNA sequence reconstruction. We make a few remarks on the problem.

---

[1] Throughout the paper, a *character* of a sequence S is a position in S together with the symbol at that position.

[2] A *connected component* of a graph (V, E) is a maximal set $C \subseteq V$ such that every pair of vertices in C is joined by a path in E. *Maximal* means C is not contained in a larger set with this property.

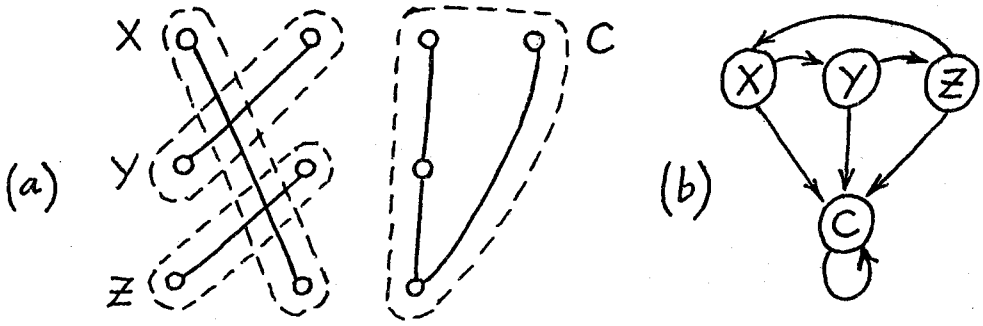

Fig. 1. (a) An alignment graph on three sequences. We use the convention of drawing the characters in a sequence horizontally left to right. (b) Relation $\prec^*$ on its connected components.

First, notice that multiple sequence trace, when restricted to two sequences, coincides with the definition of trace in sequence comparison [15, page 12]. It generalizes the definition and conforms to established terminology.

Second, when the alignment graph is a complete graph, maximum weight trace contains the minimum sum of pairs alignment problem [2], for symbol-independent insertion and deletion costs, as a special case. To see this, note that we can treat the minimum sum of pairs problem as a maximization problem by negating the cost function, and then by adding twice the insertion or deletion cost to all substitution scores we can transform the objective function so that the set of solutions is unchanged, while making all insertion and deletion scores zero. Weighting the edges of the complete alignment graph by the resulting substitution scores reduces this version of the minimum sum of pairs problem to maximum weight trace. When the alignment graph is not complete, we can score the multiple alignments considered by our maximum weight trace algorithm with the minimum sum of pairs objective function by performing the same transformation and simply taking the transitive closure of the input alignment graph, as will become clear in Section 4.1.

Third, we note that the alignment graph in the definitions may be a hypergraph, in which edges can be triples or quadruples of vertices, rather than simply pairs. The connected components for a set of edges are then given by the partition of vertices induced by successive union of intersecting edges. As before a trace is a collection of edges such that $\prec^*$ on its connected components is acyclic. This accomodates a very general notion of trace.

One advantage of the trace formulation is that a combinatorial structure is imposed on the problem from the start. When the edges of the graph come from an optimal alignment for each pair of sequences, this structure can often be exploited to speed up the computation. We call a graph whose edges between pairs of sequences form pairwise traces, a *pairwise alignment graph.*

In this paper we design an exact algorithm for the class of pairwise alignment graphs. Much of what we develop can be extended to more general inputs, and as

we shall see in the next section, even for pairwise alignment graphs the problem remains hard.

3 The complexity of maximum weight trace

Theorem 4. *Maximum weight trace is NP-complete.*

Proof. Certainly maximum weight trace is in NP, as a trace can be guessed, tested for acyclicity, and its weight determined, all in polynomial time. The NP-hardness of maximum weight trace can be shown by a simple reduction from the *feedback edge set problem* [5]. In feedback edge set we are given a directed graph (V, E) together with an integer m, and we ask whether there is a subset $F \subseteq E$ called a *feedback edge set* with at most m edges such that $(V, E-F)$ is acyclic.

Given an instance (V, E) of feedback edge set, we construct an instance $(V', E', \prec)$ of maximum weight trace as follows. For every vertex $v \in V$ we create a vertex sequence S_v, and for every edge $(v, w) \in E$ we create an edge sequence S_{vw}. Sequence S_v contains one character, while sequence S_{vw} contains two. Edges in E' connect the first character of S_{vw} to S_v, and the second character of S_{vw} to S_w. The effect is that the connected components of (V', E') are star graphs centered on vertex sequences, which are ordered under $\prec^*$ by the edge sequences.

It is straightforward to show that, giving the edges of E' unit weight, (V, E) has a feedback edge set of at most m edges if and only if $(V', E', \prec)$ has a trace of weight at least $|E| - m$. □

We remark that a consequence of this reduction is that maximum weight trace is NP-complete even if every sequence has only two characters and the edges between every pair of sequences form a pairwise trace. In other words, maximum weight trace is NP-complete for pairwise alignment graphs over sequences of bounded length.

4 A branch and bound algorithm

As with the general alignment problem of the introduction, maximum weight trace can be solved by dynamic programming, which in turn can be expressed as a shortest or longest path problem. The standard breadth-first search solution of longest path on an acyclic graph is the starting point of our branch and bound algorithm. In order to describe the algorithm, we quickly review dynamic programming and its expression as a path problem.

Recall that a multiple alignment given by a maximum weight trace ends in a column containing a subset of the characters $S_1[n] \cdots S_k[n]$, where for convenience all sequences have length n. Remove this last column from the alignment. The remaining columns correspond to a maximum weight trace over the prefix of the sequences obtained by deleting the characters of the final column.

This gives a recurrence for the weight of an optimal trace. Writing $D(x_1 \cdots x_k)$ for the weight of an optimal trace over prefixes $S_1[1, x_1], \ldots, S_k[1, x_k]$, the recurrence

has the form[3]

$$D(x_1 \cdots x_k) \;=\; \max_{(b_1 \cdots b_k)} \Big\{ D\big((x_1 - b_1) \cdots (x_k - b_k)\big) \;+\; d\big((S_1[x_1])^{b_1} \cdots (S_k[x_k])^{b_k}\big) \Big\},$$

where

- $(b_1 \cdots b_k)$ is a binary vector from the set $\{0,1\}^k - \{0\}^k$,
- a^1 denotes the character a, and
- a^0 denotes the null character ε.

Function d gives the score of a column, which is the total weight of the edges in the subgraph induced by the column. Value $D(n \cdots n)$ is the weight of an optimal trace of the alignment graph.

One can evaluate D by filling in a k-dimensional table in order of lexicographically increasing coordinates. Each entry involves a maximum over $2^k - 1$ terms, and evaluating d for a term takes $O(k^2)$ time. As there are $O(n^k)$ entries, evaluating the dynamic program takes $O(2^k n^k k^2)$ time and $O(n^k)$ space.

In turn one can view the dynamic program as finding a longest path through an acyclic graph. Each vertex of the graph is associated with a subproblem $(x_1 \cdots x_k)$. Edges correspond to columns, and are weighted by the score of the column. An edge is directed from vertex $(x_1 \cdots x_k)$ to $(y_1 \cdots y_k)$ if $y_i - x_i \in \{0,1\}$ for each coordinate, and the vertices are distinct. A maximum weight trace alignment corresponds to a longest path from the source vertex $(0 \cdots 0)$ to the sink vertex $(n \cdots n)$.

This longest path problem may be solved by a breadth-first search from the source. The search maintains a queue of vertices with the property that the length of a longest path is known from the source to the vertex at the head of the queue. The generic step removes vertex v from the head of the queue, examines all edges (v, w) leaving v, compares the length of the best known path to w with the length of the longest path to v followed by (v, w), and adds w to the queue in lexicographic order if it is not already queued.

We can recast this breadth-first search in the framework of a *branch and bound algorithm.* We view examining the edges (v, w) leaving v as a *branch step*, in which a set of subproblems, namely the vertices w, are generated.[4] The dynamic program generates $2^k - 1$ subproblems by considering all possible columns with which to extend an alignment. Section 4.1 describes how to consider a smaller set of subproblems and still guarantee optimality.

We can add a *bound step* before placing w on the rear of the queue. Suppose we have a lower bound L on the length of a longest path from the source to the sink, for instance the weight of a multiple sequence trace computed by a heuristic. If we can quickly determine an upper bound $U(w)$ on the length of the longest path from w to the sink, or equivalently, an upper bound on the weight of an optimal trace over a suffix of the sequences, we may be able to avoid adding subproblem w to the queue. Writing $\ell(v)$ for the length of the longest path from the source to v, and $d(v, w)$ for

[3] To avoid cluttering the equation we have left out the boundary conditions. The conditions are that no term with a coordinate less than zero is evaluated and that $D(0 \cdots 0) = 0$.

[4] Instead of the computation tree of a standard branch and bound algorithm, we have a computation dag.

the weight of the column represented by edge (v, w), we can avoid adding w to the queue if

$$\ell(v) + d(v, w) + U(w) \leq L.$$

Section 4.2 describes how to compute bounds L and $U(w)$.

4.1 The branch step

Given vertex $v = (x_1 \cdots x_k)$, which edges (v, w) do we have to consider? Recall that v represents the problem $S_1[1, x_1], \ldots, S_k[1, x_k]$, and (v, w) represents a column over $\{S_i[x_i+1]\}$.

Call set $\{S_i[x_i+1]\}$ the *exposed characters* on the *frontier* $(x_1 \cdots x_k)$, as shown in Figure 2. The alignment subgraph exposed by the frontier consists of all edges that touch an exposed character at one end, and touch a character on or to the right of the frontier at the other end.

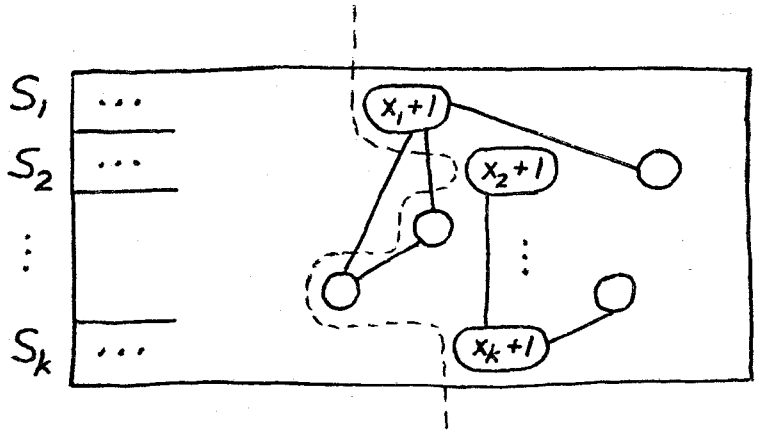

Fig. 2. The alignment subgraph exposed by frontier $(x_1 \cdots x_k)$.

Any column that extends frontier $(x_1 \cdots x_k)$ is a subset of the exposed characters. Our first observation is that we never have to consider subsets that are disconnected.

Observation 1. *It suffices to consider columns of characters that are connected in the alignment graph.*

Proof. Any column that contains two or more connected components can be divided into columns of one component without changing the weight of the alignment. □

Given that we can consider columns over each connected component of the exposed subgraph separately, which connected components do we have to consider?

Figure 3 shows a set of components on the frontier, together with edges of the exposed subgraph that leave the components. A maximum weight trace avoids cutting edges as much as possible, where an edge is *cut* if it is not included. Trying to avoid cutting exposed edges sets up a precedence among components.

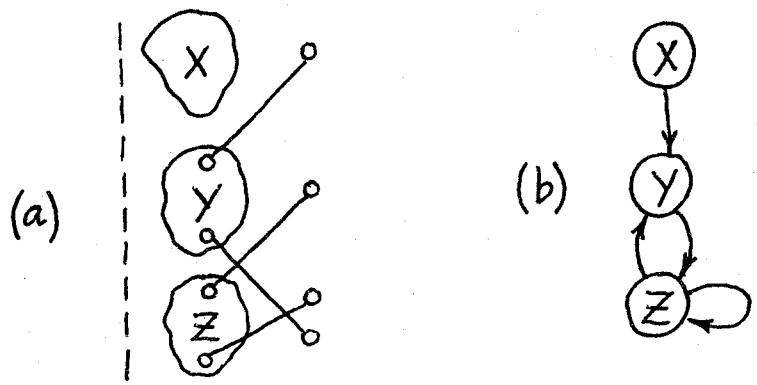

Fig. 3. (a) Components on the frontier, together with their exposed edges. (b) Avoiding cutting the exposed edges imposes an ordering on components.

For a set X of exposed characters, let us write X^* for set X together with the characters strictly to the right of the frontier that are joined to X by an exposed edge. Imagine shrinking each component of exposed characters to a supervertex in a graph G, and directing an edge in G from component X to component Y if $X \prec^* Y^*$, as in part (b) of the figure.

Consider a component C with no in-edges in G. A column consisting of the characters in C cuts no edges of the alignment graph, which is optimal. As any alignment over the characters to the right of the frontier must eventually output a column containing characters from C, we might as well output the column consisting of C, and advance the frontier. The maximum weight trace alignment can do no better.

But what if every component has an in-edge in G? In that case we can generalize C to a set of components with the property that no in-edges enter the set in G. In the following, a *strongly connected supercomponent* is a maximal set $\mathcal{C}$ of components such that, for every pair X, Y of components in $\mathcal{C}$, there is a path in G from X to Y and from Y to X. We use the term supercomponent to distinguish this set from a component, which is a connected set of exposed characters.

Observation 2. *It suffices to consider columns over the components in a strongly connected supercomponent of G with no incoming edges.*

Proof. Let $\mathcal{C}_1 \mathcal{C}_2 \cdots$ be the strongly connected supercomponents of G. Certainly considering all columns over all components in the $\mathcal{C}_i$ is correct.

Notice however that columns from supercomponents that are incomparable in G may be arbitrarily ordered without affecting the weight of an alignment. Thus considering columns from one supercomponent is also correct, as long as supercomponents are considered in topological order. Choosing a supercomponent with no incoming edges effectively chooses one of the many topological orders. □

Now that we know we can guarantee optimality by considering columns over each component X in one supercomponent, which columns over X do we have to consider?

If there are no exposed edges leaving component X, simply outputing the column consisting of X is correct. But if there are edges leaving the component, we are forced to cut an edge. Figure 4 shows the three kinds of edges that affect a component.

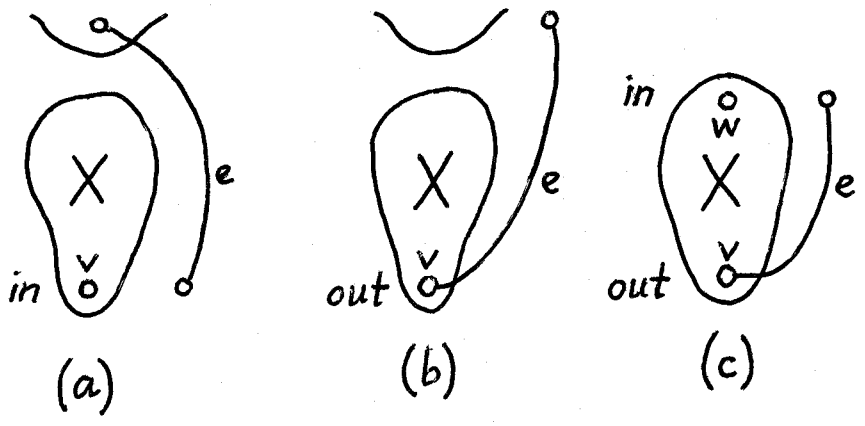

Fig. 4. The three types of edges that affect a component, and the constraints they induce.

Unless we are willing to look arbitrarily far past the frontier, we cannot tell in general which of the edges leaving a component will be cut in a maximum weight trace. If an edge of type (a) is not cut, we must output a column from component X that contains a character from the sequence to which e is incident; in the figure, this is character v. We represent this by placing an *in-constraint* on vertex v, which means we will consider a column that contains v.

Now consider an edge of type (b). If edge e is not cut, we must reserve character v so that it may be aligned with the other endpoint of e. This is represented by an *out-constraint* on v, which means we will consider a column that does not contain v.

Finally, an edge of type (c) is handled similarly to types (a) and (b). We place an in-constraint on vertex w and an out-constraint on vertex v.

The combination of edges not cut by an optimal trace assigns some set of in- and out-constraints to the characters of X. For a given constraint assignment, we can find the optimal column over X that meets the constraints by solving a minimum cut problem.[5]

Figure 5 shows the construction. Every character of the component has either the in-constraint, the out-constraint, or it has no constraint, in which case we say it is *free*. We construct a graph H that contains a vertex for every free character, and an additional source and sink vertex. An exposed edge between a free character and an in-contrained character is mapped to an edge of H touching the source, and an exposed edge between a free character and an out-constrained character is mapped to an edge touching the sink. Any parallel edges that arise are collapsed to a single edge that is given the sum of the weights of the parallel edges.

[5] A *minimum weight cut* of a graph is a partition of the vertices into two non-empty sets, such that the total weight of the edges that span the partition is minimized. A minimum cut can be found in polynomial time by a maximum flow computation [6].

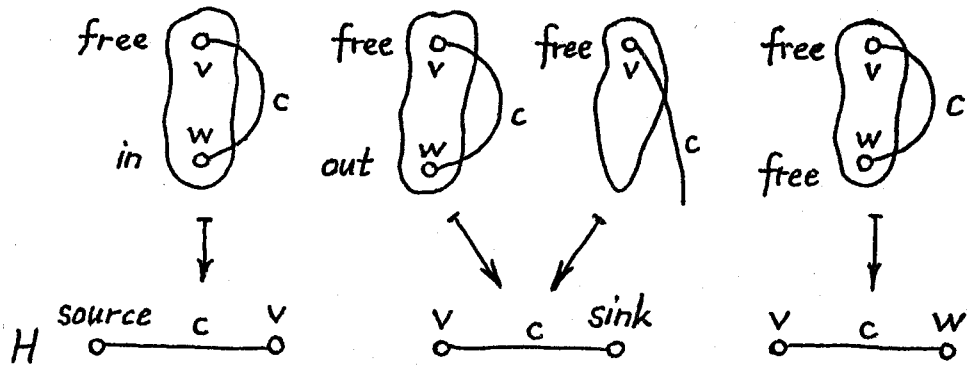

Fig. 5. Meeting a constraint assignment by constructing a minimum cut problem H.

Let F be the set of exposed edges that leave components on the frontier and are not cut by a maximum weight trace, and consider the assignment of constraints that F induces on component X. The effect of the construction is that the set of free characters that are in the source-half of a minimum weight source-sink cut of H, together with the in-constrained characters, form an optimal column over X that does not cut the edges of F. This gives our final observation.

Observation 3. *For each combination of constraints on the vertices in a component, it suffices to consider the column given by a minimum weight cut of H.*

Proof. Let E be the set of edges that leave components on the frontier, X be the component we are considering columns over, and C be a set of characters from X that are together in a column of a maximum weight trace alignment.

The maximum weight trace uses some subset F of the edges in E, and cuts the rest. Using F constrains some characters of X to be in or out of C, as in Figure 4. Solving the associated minimum cut problem of Figure 5 over H yields a column $\widetilde{C}$ that satisfies the same constraints.

Since $\widetilde{C}$ is given by a minimum cut, the weight of the edges cut by $\widetilde{C}$ is no more than the weight of the edges cut by C. Thus $\widetilde{C}$ is also optimal for the given component X and the given choice of edges F. □

Combining our observations, we get the following branch step.

1. Collect the connected components of exposed characters on the frontier for v.
2. Construct G and determine its strongly connected supercomponents.
3. Find a supercomponent $\mathcal{C}$ with no incoming edges in G.
4. For each connected component $C \in \mathcal{C}$, and for every combination of constraints on C, branch[6] on the column (v, w) given by a minimum cut of H.

[6] By *branch* we mean update the length of the best path to the new frontier w, and add w to the queue if not pruned by the bound step.

Theorem 5. *The set of columns considered by the branch step is sufficient to guarantee optimality.*

Proof. By Observations 1, 2, and 3. □

4.2 The bound step

The bound step requires a lower bound L on the weight of an optimal trace and an upper bound U on the weight of an optimal trace over a suffix of the sequences.

We can obtain L by the following *greedy heuristic.* Instead of computing a minimum cut for every assignment of constraints to the characters in a component, compute one cut for each component, in which every character is free. From all the components in the chosen supercomponent, select that column that has the minimum cut weight, and advance the frontier. This chooses the column that gives up the least immediate trace weight.

For upper bound U in a pairwise alignment graph, we can take the total weight of the edges in the alignment graph over the suffixes of the sequences. More elaborate upper bounds are also possible, such as computing a maximum weight trace over the suffixes for all triples of sequences, summing the weights of the optimal traces, and dividing by the number of sequences minus two, which is the number of times a pair of sequences is counted in triples.

5 Preliminary computational results

We have implemented a version of the branch and bound algorithm that exploits branching to reduce the search space, but at present does no bounding.

The current implementation is roughly 2,000 lines of C. The code computes minimum cuts using the push-relabel maximum flow algorithm of Goldberg and Tarjan [6]. Vertices of the longest path graph are stored and looked up by coordinate using the lexicographic splay tree of Sleator and Tarjan [16]. Since this effectively factors vertex coordinates in a trie, the space per vertex in practice is less than $O(k)$ for k sequences.

We give an example in Figure 6 of a maximum weight trace alignment of 6 tyrosine kinase protein sequences of length 273 to 285. The alignment graph consisted of one optimal alignment from each pair of sequences. Pairwise alignments were computed using a standard scoring scheme: the PAM 250 matrix of Margaret Dayhoff to score substitutions, which contains integer similarities in the range 0 to 25, with insertions and deletions receiving similarity 0 and a length-independent gap penalty of 8. Edges of the alignment graph were weighted by the substitution matrix. Notice that, as is common with protein sequences, there are many substitutions and few gaps in the alignment.

This problem was solved to optimality in 176.6 seconds (roughly 3 minutes) and 2.6 megabytes on a NeXT machine running at 33 megahertz. The branch and bound algorithm explored a subgraph of 119,046 vertices, out of a dynamic programming graph of 4.14×10^{14} vertices. This is less than .00000003% of the dynamic programming graph.

```
-----GLA-K-DAWEIPRESLRLEAKLGQGCFGEVWMGT-WND-TTRVAIKTLK-PGTMSPEA-FLQEAQVMKKLRHEK
-----GLA-K-DAWEIPRESLRLEVKLGQGCFGEVWMGT-WNG-TTKVAIKTLK-LGTMMPEA-FLQEAQIMKKLRHDK
TIY--GVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGV-WKKYSLTVAVKTLKE-DTMEVEE-FLKEAAVMKEIKHPN
-VLNRAVP-K-DKWVLNHEDLVLGEQIGRGNFGEVFSGRLRAD-NTLVAVKSCRETLPPDIKAKFLQEAKILKQYSHPN
-VLTRAVL-K-DKWVLNHEDVLLGERIGRGNFGEVFSGRLRAD-NTPVAVKSCRETLPPELKAKFLQEARILKQCNHPN
-------S-S-YYWKMEASEVMLSTRIGSGSFGTVYKGK-WHG-DVAVKILKVVDPTPEQLQA-FRNEVAVLRKTRHVN

LVQLYAVVSE-EPIYIVIEYMSKGSLLDFLKGEMGKYLRLPQLVDMAAQIASGMAYVERMNYVHRDLRAANILVGENLV
LVPLYAVVSE-EPIYIVTEFMTKGSLLDFLKEGEGKFLKLPQLVDMAAQIADGMAYIERMNYIHRDLRAANILVGDNLV
LVQLLGVCTREPPFYIITEFMTYGNLLDYLRECNRQEVSAVVLLYMATQISSAMEYLEKKNFIHRDLAARNCLVGENHL
IVRLIGVCTQKQPIYIVMELVQGGDFLTFLRT-EGARLRMKTLLQMVGDAAAGMEYLESKCCIHRDLAARNCLVTEKNV
IVRLIGVCTQKQPIYIVMELVQGGDFLSFLRS-KGPRLKMKKLIKMMENAAAGMEYLESKHCIHRDLAARNCLVTEKNT
ILLFMGYMTK-DNLAIVTQWCEGSSLYKHLHV-QETKFQMFQLIDIARQTAQGMDYLHAKNIIHRDMKSNNIFLHEGLT

CKVADFGLARLIEDNEYTARQGAK-FPIKWTAPEA--ALY-GRFTIKSDVWSFGILLTELTTKGRVPYPGMVNRE-VLD
CKIADFGLARLIEDNEYTARQGAK-FPIKWTAPEA--ALY-GRFTIKSDVWSFGILLTELVTKGRVPYPGMVNRE-VLE
VKVADFGLSRLMTGDTYTAHAGAK-FPIKWTAPES--LAY-NKFSIKSDVWAFGVLLWEIATYGMSPYPGIDLSQ-VYE
LKISDFGMSREAADGIYAASGGLRQVPVKWTAPEA--LNY-GRYSSESDVWSFGILLWETFSLGASPYPNLSNQQ-TRE
LKISDFGMSRQEEDGVYASTGGMKQIPVKWTAPEA--LNY-GWYSSESDVWSFGILLWEAFSLGAVPYANLSNQQ-TRE
VKIGDFGLATVKSRWSGSQQVEQPTGSVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIF

QVERGY---RMPCP-PECPESLHDLMCQCWRKDPEERPTFKYLQAQLLPACVLEVAE-------
QVERGY---RMPCP-QGCPESLHELMKLCWKKDPDERPTFEYIQS-FLEDYFTAAEPSG-----
LLEKDY---RMERP-EGCPEKVYELMRACWQWNPSDRPSFAEIH-----Q-AFETMFQESS-IS
FVEKGG---RLPCP-ELCPDAVFRLMEQCWAYEPGQRPSFSAIY-----Q-ELQSIRKRHR---
AIEQGV---RLEPP-EQCPEDVYRLMQRCWEYDPHRRPSFGAVH-----Q-DLIAIRKRHR---
MVGRGYASPDLSRLYKNCPKAIKRLVADCVKKVKEERPLFPQIL-----S-SIELLQHSLPKIN
```

Fig. 6. An optimal maximum weight trace alignment of six tyrosine kinase sequences. The alignment is wrapped to fit lines of a fixed width.

On this data set we could not add a seventh tyrosine kinase sequence and solve the problem to optimality within 32 megabytes of memory. Incorporating the bound step should extend the range of solution.

6 Conclusion

In summary, we have introduced a new problem in multiple sequence alignment, maximum weight trace, which we believe is a natural formulation of merging partial alignments to form multiple alignments. We have studied the problem from the point of view of exact solution, developed a branch and bound algorithm, and demonstrated that we can solve nontrivial instances to optimality in a reasonable amount of time and space. We close with some lines for future research.

Further research

With an applied problem there are two basic questions: Does the problem have a practical solution? and Is it a good model? One way to approach these questions is with a computational study. Practicality could be examined by generating a random sequence of length n, making k copies, editing each copy with $\epsilon \cdot n$ random errors, computing a minimum edit distance alignment between every pair of copies, and then measuring the mean and variance of the size of the subgraph explored by the branch and bound algorithm, as function of k, n, and ϵ. The modeling issue could then be addressed by measuring how close the optimal maximum weight trace alignment is to the "true alignment" of the generated sequences. These experiments, on a full implementation of the branch and bound algorithm, are forthcoming for the final journal paper.

While it is interesting to explore how large a problem can be solved to optimality, approximation algorithms will most likely be the choice in practice. Is there a polynomial-time approximation algorithm for maximum weight trace with a constant worst-case approximation factor? In particular, is the greedy heuristic of Section 4.2, or a simple variant, such an algorithm?

Finally, much of the effort of the branch and bound algorithm is in untangling edges where the sequences do not align well, yet we suspect users are less interested in such areas of the alignment. This suggests constructing an alignment graph from the set of all significant pairwise local similarities.[7] This would give a rigorous way to form multiple alignments from local similarities, which we envision as the main application of maximum weight trace.

Acknowledgements

We thank Dr. David Lipman for the protein sequences of Figure 6.

This research was supported by a postdoctoral fellowship from the Program in Mathematics and Molecular Biology of the University of California at Berkeley under NSF Grant DMS–8720208. Author's electronic mail address: `kece@cs.ucdavis.edu`.

References

1. Altschul, Stephen F. and David J. Lipman. Trees, stars, and multiple biological sequence alignment. *SIAM Journal on Applied Mathematics* 49:1, 197–209, 1989.
2. Carrillo, Humberto and David Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics* 48, 1073–1082, 1988.
3. Chan, S.C., A.K.C. Wong and D.K.Y. Chiu. A survey of multiple sequence comparison methods. To appear in the *Bulletin of Mathematical Biology*, 1992.
4. Feng, Da-Fei and Russell F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution* 25, 351–360, 1987.
5. Garey, Michael R. and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, New York, 1979.

[7] A *local similarity* is an alignment between a substring of one sequence and a substring of another. See [17] for a definition commonly used in practice.

6. Goldberg, Andrew V. and Robert E. Tarjan. A new approach to the maximum flow problem. *Journal of the Association for Computing Machinery* 35:4, 921–940, 1988.
7. Gotoh, Osamu. Consistency of optimal sequence alignments. *Bulletin of Mathematical Biology* 52:4, 509–525, 1990.
8. Gusfield, Dan. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology* 55:1, 141–154, 1993.
9. Hsu, W.J. and M.W. Du. Computing a longest common subsequence for a set of strings. *BIT* 24, 45–59, 1984.
10. Irving, Robert W. and Campbell B. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In Proceedings of the 3rd Symposium on *Combinatorial Pattern Matching*, 211–226, 1992.
11. Kececioglu, John. *Exact and Approximation Algorithms for DNA Sequence Reconstruction.* PhD dissertation, Technical Report 91-26, Department of Computer Science, The University of Arizona, Tucson, Arizona 85721, 1991.
12. Maier, David. The complexity of some problems on subsequences and supersequences. *Journal of the Association for Computing Machinery* 25:2, 322–336, 1978.
13. Pevzner, Pavel. Multiple alignment, communication cost, and graph matching. To appear in *SIAM Journal on Applied Mathematics.*
14. Sankoff, David. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics* 28:1, 35–42, 1975.
15. Sankoff, David and Joseph B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, Massachusetts, 1983.
16. Sleator, Daniel D. and Robert E. Tarjan. Self-adjusting binary search trees. *Journal of the Association for Computing Machinery* 32:3, 652–686, 1985.
17. Smith, Temple F. and Michael S. Waterman. Identification of common molecular sequences. *Journal of Molecular Biology* 147, 195–197, 1981.
18. Vingron, Martin and Patrick Argos. A fast and sensitive multiple sequence alignment algorithm. *Computer Applications in the Biosciences* 5:2, 115–121, 1989.
19. Waterman, M.S. and R. Jones. Consensus methods for DNA and protein sequence alignment. *Methods in Enzymology* 188, 221–237, 1990.