Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree*

R. Ravi† John D. Kececioglu‡

September 13, 1996; revised March 2, 1998

Abstract We consider the problem of *multiple sequence alignment under a fixed evolutionary tree*: given a tree whose leaves are labeled by sequences, find ancestral sequences to label its internal nodes so as to minimize the total length of the tree, where the length of an edge is the edit distance between the sequences labeling its endpoints. We present a new polynomial-time approximation algorithm for this problem, and analyze its performance on regular d-ary trees with d a constant. On such a tree, the algorithm finds a solution within a factor $\frac{d+1}{d-1}$ of the minimum in $O(k^d T(d,n) + k^{2d} n^2)$ time, where k is the number of leaves in the tree, n is the length of the longest sequence labeling a leaf, and $T(d,n)$ is the time to compute a Steiner point for d sequences of length at most n. (A *Steiner point* for a set $\mathcal{S}$ of sequences is a sequence P that minimizes the sum of the edit distances from P to each sequence in $\mathcal{S}$. The time $T(d,n)$ is $O(d2^d n^d)$, given $O(ds^{d+1})$-time preprocessing for an alphabet of size s.) The approximation algorithm is conceptually simple and easy to implement, and actually applies to any metric space in which a Steiner point for any fixed-sized set can be computed in polynomial time.

We also introduce a new problem, *bottleneck tree-alignment*, in which the objective is to label the internal nodes of the tree so as to minimize the length of the *longest* edge. We describe an exponential-time exact algorithm for the case of unit-cost edit operations, and show there is a simple linear-time approximation algorithm for the general case that finds a solution within a factor $O(\log k)$ of the minimum.

Keywords Computational biology, approximation algorithms, multiple sequence alignment, evolutionary trees.

1 Introduction

Multiple sequence alignment is a ubiquitous problem in computational biology for which many objective functions have been studied, including the sum-of-pairs objective [3, 6, 13, 2, 5], the maximum-trace objective [12, 15], and objectives defined in terms of an evolutionary tree [16,

*Dedicated to the memory of Eugene Lawler. An earlier version of this paper was presented at the 5th Symposium on Combinatorial Pattern Matching [14].

†Graduate School of Industrial Administration, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. Email: ravi+@cmu.edu.

‡Department of Computer Science, University of Georgia, Athens, GA 30602. Email: kece@cs.uga.edu.
Research supported by a DOE Human Genome Distinguished Postdoctoral Fellowship.

21, 9, 10, 1, 6, 13, 11, 20, 19]. We study this latter form of the objective, in which the task of aligning a set of biological sequences is guided by a tree that represents the evolutionary relationship of the species from which the sequences have been obtained. The input is the tree with leaves labeled by sequences of existing species. The problem is to find ancestral sequences to label the internal nodes so as to optimize some function of the tree edge-lengths. If the criterion of maximum parsimony is used, the length of an edge is taken to be the edit distance between the sequences labeling its endpoints, and the objective is to minimize the sum of the edge lengths. For reasons that will be explained shortly, this problem is usually called *multiple sequence alignment under a fixed evolutionary tree*, and is often abbreviated to simply *tree alignment*.

Sankoff [16] initiated the formal study of tree alignment and gave an exact algorithm for the problem. By observing that for every solution there is a pairwise alignment on each tree edge that achieves the edge length and that any tree of pairwise alignments induces a multiple alignment of the leaf sequences, he showed that the problem is equivalent to a form of multiple alignment and can be solved by dynamic programming. For a tree of k leaves, each labeled by a sequence of length at most n from an alphabet of size s, the algorithm runs in $O(ks^2 2^k n^k)$ time using $O(n^k)$ space. Sankoff, Cedergren and Laplame [18] suggested a local-search heuristic for tree alignment that starts with a labeling of internal nodes by leaf sequences, and repeatedly improves the solution by choosing an internal node and replacing its label with the optimal sequence for the star problem defined on the immediate neighbors of the node in the tree. Wang and Jiang [20] proved that the problem is NP-complete for binary trees, even when the edit cost-function is a metric, and MAX SNP-hard for star-trees under a non-metric cost function.

Jiang, Lawler and Wang [11] gave the first approximation algorithm for tree alignment. As in the heuristic of Sankoff et al., their algorithm labels internal nodes with leaf sequences. By an elegant argument, Jiang et al. showed that a particular labeling by leaves gives an alignment that is within a factor 2 of the minimum. For trees of degree bounded by a constant d, they also provided the first polynomial-time approximation-scheme for the problem. Given any $t > 0$, their approximation scheme finds an alignment within a factor $1 + \frac{3}{t}$ of the minimum in time $O(k^{2+d^{t-1}} n^{(d^{t-1}-1)/(d-1)})$. The running time of both the approximation algorithm and the approximation scheme has been recently improved by Wang and Gusfield [19] (see also [7]).

Minimizing the total length

In this paper we propose a new, natural heuristic for tree alignment. While the algorithm applies to a tree of any shape, the class of trees for which we can prove a performance guarantee are *regular d-ary trees* where d is a constant. Such a tree is rooted (though the location of the root is unimportant) and every internal node has exactly d children.

Theorem 1 *There is an approximation algorithm for multiple sequence alignment under a fixed evolutionary tree that finds a solution of cost at most $\frac{d+1}{d-1}$ times the minimum in $O(d2^d k^d n^d + k^{2d} n^2 + ds^{d+1})$ time and $O(n^d + k^{d+1} + s^d)$ space, given a regular d-ary tree of k leaves, with each leaf labeled by a sequence of length at most n over an alphabet of size s.*

As is often the case, this worst-case performance guarantee is exceedingly pessimistic, and though our proof of the guarantee relies heavily on the symmetry of the tree and does not extend to arbitrary bounded-degree trees, the result suggests that the algorithm (which is conceptually simple and easy to implement) may perform well in practice. Moreover, the running time compares favorably with the prohibitive running time of the approximation scheme of [11], while providing guarantees better than two for trees of degree greater than four. The algorithm actually applies to any metric space in which a Steiner point for any fixed-sized set can be computed in polynomial time.[1]

Minimizing the bottleneck length

Finding alignments that minimize the total length of the tree has some disadvantages: the alignment can be biased by over-represented sequences, and to ensure that most of the edges in the tree are short, a few can be made comparatively long.

We propose the investigation of tree alignment under an alternate objective: the length of the *longest* edge in the tree. We call this *bottleneck alignment under a fixed evolutionary tree*, or *bottleneck tree-alignment* for short. On a star-tree, its solution may produce a consensus sequence that more faithfully reflects the variation in leaf sequences, as the consensus is less likely to be influenced by an over-sampled sequence. On a general tree, its solution may produce an alignment with more uniform edge lengths, which may be more consistent with an assumption of a uniform molecular clock.

Define the *radius* of an unrooted tree to be the maximum over all internal nodes of the number of edges from the node to a nearest leaf.

Theorem 2 *There is a simple linear-time approximation algorithm for bottleneck tree-alignment that finds a solution within a factor $2\rho + 1$ of the minimum, where ρ is the radius of the tree.*

As in the heuristic of Sankoff et al. [18] and Jiang et al. [11], the algorithm labels internal nodes with leaves. For trees on m nodes having k leaves and no internal nodes of degree two, $\rho \leq \lg m \leq \lg k + 1$, so the above is an $O(\log k)$-approximation. For star-trees, the proof yields a bound that is somewhat tighter than given above: labeling the center with any leaf is within a factor 2 of the minimum.

We also describe an exact algorithm for bottleneck tree-alignment for the case of unit-cost edit operations that takes time and space polynomial in n but exponential in k using dynamic programming.

Plan of the paper

In the next section, we present our algorithm for classical tree alignment and its analysis on d-ary trees. Section 3 considers bottleneck alignment, and Section 4 closes with some open questions.

2 Classical tree alignment

The basic subproblem that we use repeatedly in our approximation algorithm for classical tree alignment is the *Steiner problem* on a multiset of sequences: given sequences $S_1, \ldots, S_k$, find a

[1] A *Steiner point* for a subset $\mathcal{S}$ of a metric space is a point P in the space that minimizes the sum of the distances from P to each element of $\mathcal{S}$. In our application, the metric space is the set of all sequences under edit distance. In this context, a Steiner point for a set of k sequences of length at most n from an alphabet of size s can be computed in $O(k2^k n^k)$ time, given $O(ks^{k+1})$-time preprocessing [16].

sequence P that minimizes the sum $\sum_{1\le i\le k} D(P,S_i)$, where $D(x,y)$ denotes the edit distance between sequences x and y. Sequence P is called a *Steiner sequence* for the multiset.

Approximation algorithm

Our algorithm is parameterized by p, the maximum size of a subset of leaves on which it solves the Steiner problem. Let T be the leaf-labeled input tree. The algorithm has two phases.

- The first phase computes a Steiner sequence for every q-subset of leaves of T for all $q \le p$. (For $q < 3$ the Steiner sequence is a leaf.) Let the set of these Steiner sequences be $\mathcal{S}^{(p)}$.
- In the second phase, the algorithm finds an optimal labeling of the internal nodes of T, using labels drawn from $\mathcal{S}^{(p)}$.

Both the set $\mathcal{S}^{(p)}$ of Steiner sequences, and the minimum-cost labeling of T from $\mathcal{S}^{(p)}$, can be computed efficiently using dynamic programming, as we show next.

Time and space

Though the total number of labelings of a tree from a set $\mathcal{S}$ is exponentially large in the size of the tree, we can efficiently find the best among these by dynamic programming as discovered independently by Fitch [4], Hartigan [8], and Sankoff [16] (see also [17]). For an internal node v and a label $x \in \mathcal{S}$, let $C(v,x)$ denote the minimum cost of all labelings of the subtree rooted at v where v is labeled by x. We can compute $C(v,x)$ from the C-values of v's children $w_1,\ldots,w_d$ by the recurrence

$$C(v,x) = \sum_{1\le i\le d} \min_{y\in\mathcal{S}} \Big\{ D(x,y) + C(w_i,y) \Big\},$$

where for a leaf l we take $C(l,x)$ to be zero if l is labeled by x in the input tree, and infinite otherwise. Thus the C-values for every node can be evaluated by the recurrence and tabulated in one bottom-up pass over the tree. Once the cost of the best labeling of the root is known, the best labeling of the tree can be recovered from the tabulated C-values in a second top-down pass. The total time then to optimally label a tree of k leaves from a set $\mathcal{S}$ of size m, assuming no internal node has degree 2 and that distances between points in $\mathcal{S}$ can be determined in constant time, is $O(km^2)$, using $O(km)$ space.

We can now bound the time for the approximation algorithm on the input tree T. In phase 1, the number of subsets over which we compute a Steiner sequence is $\sum_{1\le q\le p}\binom{k}{q}$, which is $O(k^p)$ for p a constant, and these subsets can be enumerated in time $O(pk^p)$. Computing the Steiner sequence for any subset by applying Sankoff's multiple alignment algorithm [16] to a star-tree takes time $O(p2^p n^p)$ and space $O(n^p)$, given $O(ps^{p+1})$ time and $O(s^p)$ space to precompute the alignment-column cost-function for an alphabet of size s. Thus the time to determine all Steiner sequences in phase 1 is $O(p2^p k^p n^p + ps^{p+1})$.

For phase 2, the time to precompute the edit distance between any two labels is $O((k^p)^2 n^2)$. After this preprocessing, the time to optimally label T by the Fitch-Hartigan-Sankoff algorithm is $O(k(k^p)^2)$. Thus the time to find the best labeling of T from $\mathcal{S}^{(p)}$ in phase 2 is $O(k^{2p+1} + k^{2p}n^2)$. This dominates the time to recover the multiple alignment from the labeling, and gives a total time of $O(p2^p k^p n^p + k^{2p}n^2 + ps^{p+1})$ and $O(n^p + k^{p+1} + s^p)$ total space. Taking $p = d$ for a d-ary tree yields the time and space of Theorem 1.

Performance guarantee

In this section we show that the cost of the best labeling of T from $\mathcal{S}^{(d)}$ is within a factor of $\frac{d+1}{d-1}$ of the minimum when T is a regular d-ary tree. We do this by demonstrating a labeling of T from $\mathcal{S}^(d)$ of a special form, which we call a *consistent labeling*, that has cost at most $\frac{d+1}{d-1}$ times the minimum. (This labeling uses Steiner sequences on d-subsets of leaves alone, and requires that the subsets meet a consistency criterion.) Since the approximation algorithm with $p = d$ computes the minimum over all labelings of T from $\mathcal{S}^{(d)}$, which includes consistent labelings, it will follow that the algorithm achieves the claimed performance ratio. The bound on the cost of the best consistent labeling is obtained using averaging: we compute an explicit upper bound on the total cost of all consistent labelings, then divide this bound by the number of such labelings, and use the fact that the best labeling has cost at most this average value.

For a Steiner sequence S^* on a multiset of sequences $\{S_1, \ldots, S_p\}$, we refer to each S_i as a *component* of S^*. We refer to the sum $\sum_{1\leq i\leq p} D(S^*, S_i)$ as the *Steiner cost* of S^*, which we denote by $c(S^*)$.

We define a consistent labeling in two steps as follows. For a node v of T, let $L(v)$ denote the multiset of sequences labeling the leaves of the subtree of T rooted at v. A *valid* label for an internal node v with children $w_1, \ldots, w_d$ is any Steiner sequence for multiset $\{S_1, \ldots, S_d\}$ where $S_i \in L(w_i)$ for $1 \leq i \leq d$. In other words, a valid label for a node v is a Steiner sequence S^* where each child of v has a leaf sequence that is a component of S^*. A *valid labeling* of tree T is one in which every internal node has a valid label. To identify a good ancestral labeling, we not only require that the labels at nodes be valid, but also that they are consistent in the following way up the tree. Given a valid labeling of the children $w_1, \ldots, w_d$ of node v, a *consistent* label for v is any Steiner sequence for $\{S_1, \ldots, S_d\}$ where for $1 \leq i \leq d$, (1) $S_i \in L(w_i)$, so that the sequence labeling v is valid, and (2) the Steiner sequence labeling w_i has S_i as one of its components. In other words, a consistent label for a node v is a valid lable that shares a component with the Steiner sequence labeling each child of v. A *consistent labeling* of the tree T is a valid labeling in which the label of every internal node is consistent.

We can simplify the analysis by observing that the alignment problem on a rooted regular d-ary tree can always be treated as one on a complete d-ary tree. Any incomplete portion of the tree can be padded by a complete d-ary subtree of the appropriate size, where the subtree that replaces a leaf labeled by sequence S has all its leaves labeled by S. It is straightforward to check that there is a minimum cost alignment over the complete d-ary tree where every node in such a padded subtree is labeled by S. Thus in our analysis we consider the problem on a complete d-ary tree, and from now on, T denotes the padded complete tree.

At a high level, our argument proceeds as follows. To relate the cost of a consistent labeling to the cost of an optimal solution, we use the triangle inequality on edit distance. Any edge in a consistent labeling of T has its endpoints labeled by Steiner sequences that share a component. We upper bound the cost of the edge between these two sequences by adding together the distance from each sequence to the component leaf sequence they share in common. The symmetry in a complete d-ary tree allows us to account in this way for the costs of all edges in all consistent labelings of T by a weighted sum of the Steiner costs $c(S^*)$ taken over all Steiner sequences S^* labeling the nodes of T. By a balancing argument, we can compute the multiplier for each of the costs in the weighted sum and argue that it is the same for all nodes at a given level. With this estimate of the total cost of all consistent labelings in hand, we complete the proof by averaging over the number of such labelings. We elaborate on this below.

Recursively define the *level* of any node in the complete tree T to be zero if the node is a leaf, and one plus the level of any of its children if it is an internal node.

Lemma 1 *The number of valid labels for a node at level i in the complete d-ary tree T is $d^{(i-1)d}$.*

Proof Immediate from the definition of a valid label and the fact that a node at level i in T has d children, each of which has d^{i-1} leaves in its subtree. $\square$

We introduce a few more definitions. Let T^* denote T with an optimal labeling, and let $c(T^*)$ denote the total cost of T with this labeling. For any internal node v of T, let $\mathrm{star}(v)$ denote the sum of the lengths of the edges $(v, w_1), \ldots, (v, w_d)$ in T^*. For any level i, let $\mathrm{star}(i) = \sum_{v \text{ at level } i} \mathrm{star}(v)$.

Lemma 2 *For any level i in T,*

$$\sum_{v \text{ at level } i} \; \sum_{\text{valid labels } S^*} c(S^*) \leq \sum_{1 \leq j \leq i} \frac{\mathrm{star}(j)}{d^{i-j}}.$$

Proof Consider a Steiner sequence S^* that is a valid label for an internal node v at level i in T. Let the components of this sequence be the leaves $S_1, \ldots, S_d$ in T. The key observation is that the cost $c(S^*) = \sum_{1 \leq i \leq d} D(S^*, S_i)$ is at most the sum of the lengths of the edges in T^* on the paths between v and the leaves labeled $S_1, \ldots, S_d$. We can thus use these edges in T^* to upper bound the cost $c(S^*)$.

The number of valid labels for v is exactly $d^{(i-1)d}$ by Lemma 1. We can now sum over all valid labels for v, accounting for the costs of these labels by paths in T^*. In this summation, every edge of the form (v, w_i) is used once for every valid label of v, for a total of $d^{(i-1)d}$ times. However, all the edges one level below v, such as an edge from a child w_i to a grandchild of v, are used in only $1/d$ of these accounting paths, i.e. $d^{(i-1)d}/d$ times. In general any edge from a node ℓ levels below v to a node $\ell + 1$ levels below v is used exactly $d^{(i-1)d}/d^{\ell}$ times in the accounting. The cost of all edges from level i to level $i-1$ is $\mathrm{star}(i)$ by definition. Thus, in the accounting, $\mathrm{star}(i-\ell)$ is used exactly $d^{(i-1)d}/d^{\ell}$ times. Summing over all $0 \leq \ell < i$ gives the lemma. $\square$

Let h denote the level of the root of T. For any fixed valid label for the root, let $\mathcal{N}$ denote the number of consistent labelings of T that contain this label at the root. Let $\mathcal{T}$ denote the number of consistent labelings of T. Since there are $d^{(h-1)d}$ valid labels for the root by Lemma 1, by symmetry

$$\mathcal{T} \;=\; d^{(h-1)d}\mathcal{N}. \tag{1}$$

Consider any internal node v of T at level i where $i < h$. Applying Lemma 1 again, the number of valid labels for v is $d^{(i-1)d}$. Note that each of these labels appears an equal number of times in all consistent labelings of T. Thus the *usage* of any of these labels in all consistent labelings of T, i.e. the total number of consistent labelings of T in which v has one of these labels, is by (1)

$$\frac{\mathcal{T}}{d^{(i-1)d}} \;=\; d^{(h-i)d}\mathcal{N}. \tag{2}$$

We are now ready to prove the performance guarantee for the approximation algorithm.

Lemma 3 *The best consistent tree has cost at most $\frac{d+1}{d-1}$ times the minimum.*

Proof We use the strategy outlined at the beginning of this section. We first derive an upper bound on the cost of all consistent labelings of T. To accomplish this, we first use a simple upper bound on the cost of an edge in any consistent tree. We upper bound the cost of such an edge by adding the two distances from the Steiner sequences labeling its endpoints to the unique leaf component they have in common. Summing such pairs of distances over all edges in all consistent trees, we determine how many times each distance from a Steiner sequence S^* labeling a node v at level i to a component leaf labeled with say S_j, i.e. $D(S^*, S_j)$, is used in this upper bound. The usage of a Steiner sequence, computed in (2), comes in handy here. We claim that every such cost $D(S^*, S_j)$ is used exactly $\frac{d+1}{d} d^{(h-i)d} \mathcal{N}$ times. To see this, note that the distance $D(S^*, S_j)$ is used $d^{(h-i)d}\mathcal{N}$ times over all consistent valid trees when accounting for edges from node v to the child containing leaf S_j, while it is used an extra $\frac{1}{d} d^{(h-i)d}\mathcal{N}$ times over all consistent valid trees when accounting for the edge from v to its parent in T.

We can now write an upper bound for the sum of the costs of all consistent trees.

$$
\begin{aligned}
\sum_{\substack{\text{consistent labelings}\\ \mathcal{L} \text{ of } T}} c(T \text{ labeled with } \mathcal{L}) &\le \sum_{1\le i\le h} \sum_{v \text{ at level } i} \sum_{\substack{\text{valid labels}\\ S^* \text{ of } v}} \frac{d+1}{d} d^{(h-i)d} \mathcal{N} c(S^*) \\
&\le \sum_{1\le i\le h} \frac{d+1}{d} d^{(h-i)d} \mathcal{N} d^{(i-1)d} \sum_{1\le j\le i} \frac{\mathrm{star}(j)}{d^{i-j}} \\
&= \frac{d+1}{d} d^{(h-1)d} \mathcal{N} \sum_{1\le i\le h} \sum_{1\le j\le i} \frac{\mathrm{star}(j)}{d^{i-j}} \\
&= \frac{d+1}{d} \mathcal{T} \sum_{1\le j\le h} \mathrm{star}(j) \sum_{0\le i\le h-j} \frac{1}{d^i} \\
&\le \frac{d+1}{d} \mathcal{T} \sum_{1\le j\le h} \mathrm{star}(j) \frac{d}{d-1} \\
&= \frac{d+1}{d-1} \mathcal{T} c(T^*).
\end{aligned}
$$

Since the total number of consistent trees is $\mathcal{T}$, by averaging, the minimum-cost consistent tree has cost at most $\frac{d+1}{d-1} c(T^*)$. □

By the argument at the start of this section, this proves Theorem 1.

3 Bottleneck tree-alignment

In this section we study the bottleneck problem: given an arbitrary unrooted tree with leaves labeled by sequences, find a labeling of internal nodes by sequences that minimizes the length of the longest edge in the tree.

Approximation algorithm

We use the following simple lower bound to derive an approximation algorithm.

Lemma 4 *Suppose x and y are sequences labeling two leaves that are l edges apart in tree T. Then $D(x,y)/l$ is a lower bound on the bottleneck cost of any labeling of T.*

Proof Consider the path in T between the leaves labeled x and y. By the triangle inequality, the sum of the lengths of these edges in an optimal labeling must be at least $D(x,y)$. By averaging, at least one edge on the path has length at least $D(x,y)/l$, which lower bounds the bottleneck cost. □

The approximation algorithm motivated by this bound simply labels each internal node with the sequence of its nearest leaf. Such a labeling may be thought of as a particular way to lift leaf sequences to internal nodes, where the criterion for lifting a leaf-label to an internal node is that the leaf is closest. Note that closest in this context is with respect to the number of edges between the internal node and the leaf; a different lifting criterion is used for classical tree alignment in [11].

This algorithm can be implemented in time proportional to the size of the tree, independent of the lengths of the sequences labeling its leaves, since the lifting criterion is solely in terms of path lengths. A simple conceptual way to see how each node can be assigned its closest leaf in linear time is as follows: add a new node s and an edge from s to every leaf in the tree T. A breadth-first search over this augmented graph with s as the source will identify the leaf nearest each internal node in linear time.

To bound the performance ratio, consider the longest edge of $\widetilde{T}$, the input tree with labels assigned by the approximation algorithm. Let the sequences labeling the endpoints of the longest edge in $\widetilde{T}$ be x and y, and let the number of edges on the path in T between the leaves labeled x and y be l. Then the bottleneck cost $c(\widetilde{T})$ is $D(x,y)$, while l is at most $2\rho+1$ where ρ is the radius of T. (Recall that the *radius* of a tree is the maximum over all nodes of the minimum number of edges from the node to a nearest leaf.) By Lemma 4, a lower bound on the bottleneck cost of the optimally-labeled tree T^* is

$$c(T^*) \;\geq\; \frac{D(x,y)}{l} \;\geq\; \frac{D(x,y)}{2\rho+1} \;=\; \frac{c(\widetilde{T})}{2\rho+1},$$

which proves Theorem 2.

Note that the argument applied to a star-tree gives a slightly better ratio of $\delta+1=2$.

Exact algorithm

While an exact algorithm for tree alignment to minimize the *total* edge-length has been designed using dynamic programming [16], it is not at all obvious how to design such an algorithm to minimize the *maximum* edge-length. Since the cost of a solution described by an alignment is no longer the independent sum over columns of a fixed column cost-function, the principle of optimality cannot be applied in the traditional way.

We first sketch an algorithm in the simpler setting of a star-tree without insertions and deletions. More precisely, we are given a star-tree on k sequences of length n, and we wish to find a sequence S of length n such that the maximum number of mismatches between S and any of the k sequences (the length of the longest edge) is minimized. If the objective was simply the sum of the edge lengths, taking the majority character at each column would give an optimal sequence S. For the bottleneck objective, however, this does not hold. Consider for example the input

aa,
aa,
bb.

Taking the majority character at both columns gives sequence aa, for which the tree has edge lengths $(0, 0, 2)$ and bottleneck cost 2. Sequences ab and ba, however, both result in a tree with edge lengths $(1, 1, 1)$ and bottleneck cost 1, and are both optimal solutions. As this simple example illustrates, taking the majority character independently at each column ignores the correlation of characters within sequences, and is biased by over-represented sequences.

To solve the problem by dynamic programming, for each prefix of length l for $1 \leq l \leq n$, we maintain a bit-vector of length $O(n^k)$. Each position in the vector corresponds to a point in k-dimensional space with integer coordinates in the range 0 to l. Setting the bit for point $(p_1, \ldots, p_k)$ means there is a sequence of length l whose distance (number of mismatches) to the ith sequence is p_i. The vector for length $l+1$ may be obtained from the vector for length l by considering each character in the alphabet at position $(l+1)$ in candidate sequence S. The final answer is obtained by looking at the vector for length n and finding the point that is set and has minimum maximum-coordinate. The overall time is $O(skn^{k+1})$ for an alphabet of size s. Space is $O(n^{k+1})$ bits.

It is not hard to extend this approach to allow unit-cost insertions and deletions by considering all frontiers of the k sequences and all their extensions and maintaining a bit-vector for each of the frontiers. (A *frontier* corresponds to a subproblem over a prefix of the sequences, and an extension corresponds to appending a column to an optimal alignment of these prefixes.) Examining $O(n^k)$ frontiers, each with a bit-vector of length $O((2n)^k)$, and for each of the 2^k extensions, taking $O(sk(2n)^k)$ time to set the bits of the vector for the extended frontier, gives a time of $O(sk4^k n^{2k})$, using $O(2^k n^{2k})$ bits.

The approach may be further extended to arbitrary trees by assigning sequences to internal nodes as we progress through the frontiers of the alignment of the leaf sequences and maintaining a bit-vector for each frontier, where the number of dimensions of the points encoded by the vectors is equal to the number of edges in the tree. For a tree with no internal nodes of degree 2, this gives a time of $O(k(8s)^k n^{3k-2})$ using $O(4^k n^{3k-2})$ bits.

4 Open problems

We have proposed a new approximation algorithm for aligning sequences via an evolutionary tree that labels internal nodes with Steiner sequences from subsets of p leaves, and have analyzed its worst-case performance-ratio on regular d-ary trees for $p = d$. The algorithm improves on the performance ratio of the approximation algorithm of Jiang et al. [11] for d higher than three, while using less time than the full approximation scheme. It remains an interesting open problem to analyze the performance of our algorithm on arbitrary trees. Since the algorithm is practical for $p = 3$, as a starting point for such an investigation we ask the question that provided the original motivation for our research, namely, is there an approximation algorithm for arbitrary trees that labels internal nodes with Steiner sequences from leaf sets of size 3 and achieves a performance ratio better than 2?

We also introduced the bottleneck tree-alignment problem and presented a simple approximation algorithm for arbitrary trees, as well as a dynamic-programming exact algorithm for unit-cost edit operations. On star trees the problem provides a way to find a consensus for a set of strings that is robust to unequal representation in the set. The time and space are exhorbitant for our exact algorithm, however, and we ask, is there a method to solve the problem, perhaps avoiding dynamic programming, that uses considerably less time and space?

We hope these open some interesting avenues for future research.

Acknowledgments

This work was performed in large part while the authors were postdoctoral fellows at the Department of Computer Science, University of California, Davis, and was invigorated by discussions with the late Eugene Lawler.

References

[1] Altschul, S.F. and D. Lipman. "Trees, stars, and multiple biological sequence alignment." *SIAM Journal on Applied Mathematics* 49:1, 197–209, 1989.

[2] Bafna, V., E.L. Lawler and P.A. Pevzner. "Approximation algorithms for multiple sequence alignment." Proceedings of the 5th *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 807, 43–53, 1994.

[3] Carrillo, H. and D. Lipman. "The multiple sequence alignment problem in biology." *SIAM Journal on Applied Mathematics* 48, 1073–1082, 1988.

[4] Fitch, W.M. "Towards defining the course of evolution: minimum change for a specific tree topology." *Systematic Zoology* 20, 406–416, 1971.

[5] Gupta, S.K., J.D. Kececioglu, and A.A. Schäffer. "Improving the practical space- and time-efficiency of the shortest-paths approach to sum-of-pairs multiple-sequence alignment." *Journal of Computational Biology* 2:3, 459–472, 1995.

[6] Gusfield, D. "Efficient methods for multiple sequence alignment with guaranteed error bounds." *Bulletin of Mathematical Biology* 55, 141–154, 1993.

[7] Gusfield, D. and L. Wang. "New uses for uniform lifted alignments." Technical Report 96-4, Department of Computer Science, University of California, Davis, January 1996.

[8] Hartigan, J.A. "Minimum mutation fits to a given tree." *Biometrics* 29, 53–65, 1973.

[9] Hein, J. "A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given." *Molecular Biology and Evolution* 6:6, 649–668, 1989.

[10] Hein, J. "Unified approach to alignment and phylogenies." *Methods in Enzymology* 188, 626–645, 1990.

[11] Jiang, T., E.L. Lawler, and L. Wang. "Aligning sequences via an evolutionary tree: complexity and approximation." Proceedings of the 26th ACM *Symposium on the Theory of Computing*, 760–769, 1994. To appear in *Algorithmica*.

[12] Kececioglu, J. "The maximum trace problem in multiple sequence alignment." Proceedings of the 4th *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 684, 106–119, 1993.

[13] Pevzner, P.A. "Multiple alignment, communication cost, and graph matching." *SIAM Journal on Applied Mathematics* 52:6, 1763–1779, 1992.

[14] Ravi, R. and J.D. Kececioglu. "Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree." Proceedings of the 5th *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 937, 330–339, 1995.

[15] Reinert, K., P. Mutzel, H.-P. Lenhof, K. Mehlhorn and J. Kececioglu. "A branch-and-cut algorithm for multiple sequence alignment." Proceedings of the 1st ACM Conference on *Computational Molecular Biology*, 241–249, 1997.

[16] Sankoff, D. "Minimal mutation trees of sequences." *SIAM Journal on Applied Mathematics* 28:1, 35–42, 1975.

[17] Sankoff, D. and R. Cedergren. "Simultaneous comparison of three or more sequences related by a tree." In D. Sankoff and J. Kruskal, editors, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison Wesley, Reading, MA, 253–264, 1983.

[18] Sankoff, D., R. Cedergren and G. Laplame. "Frequency of insertion-deletion, transversion, and transition in the evolution of 5S ribosomal RNA." *Journal of Molecular Evolution* 7, 133–149, 1976.

[19] Wang, L. and D. Gusfield. "Improved approximation algorithms for tree alignment." Technical Report 96-2, Department of Computer Science, University of California, Davis, January 1996.

[20] Wang, L. and T. Jiang. "On the complexity of multiple sequence alignment." *Journal of Computational Biology* 1, 337–348, 1994.

[21] Waterman, M.S. and M.D. Perlwitz. "Line geometries for sequence comparisons." *Bulletin of Mathematical Biology* 46:4, 567–577, 1984.