Deferred path heuristic for phylogenetic trees revisited

Nielsen, Benny Kjær; Lindgren, S.; Winter, Pawel; Zachariasen, Martin

Published in:
Proceedings Of CompBioNets05: Algorithms and Computional Methods for Biochemical and Evolutionary Networks

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Nielsen, B. K., Lindgren, S., Winter, P., & Zachariasen, M. (2005). Deferred path heuristic for phylogenetic trees revisited. In Proceedings Of CompBioNets05: Algorithms and Computional Methods for Biochemical and Evolutionary Networks (pp. 75-92). King's College Publications.

Deferred Path Heuristic for Phylogenetic Trees Revisited

Benny K. Nielsen, Stinus Lindgreen, Pawel Winter, and Martin Zachariasen

ABSTRACT. We reexamine the deferred path heuristic originally developed by Schwikowski and Vingron in the late 1990s. Given a set of sequences, e.g. DNA, this heuristic simultaneously finds a multiple alignment and a phylogenetic tree. We give a unified description of the heuristic where such intermediate constructs as fork alignment graphs are made obsolete. Based on this description an efficient implementation is straightforward. We discuss two improvements of which one is intended to provide better quality solutions and one is intended to provide a substantial speed-up. This is confirmed by experiments on several real-life problem instances.

1 Introduction

One of the most important problems in biology is to explain how a group of species has evolved over time and how they are related to one another. This is usually done by means of *phylogenetic trees* where leaves represent the species while the interior nodes represent their hypothetical ancestors. The major problem that faces biologists when constructing phylogenetic trees is that the information about ancestors is very limited or indeed non-existent. So the problem is to infer the interior nodes and their relationship with each other and with the studied species.

Several models for the determination of phylogenetic trees have been considered over the years (see for example some of the monographs addressing mathematical and computational aspects of phylogenetic trees that appeared in recent years [Felsenstein, 2003; Semple and Steel, 2003]). From a computational point of view, most models are intractable as they lead to NP-hard problems. In order to obtain reasonable solutions even for relatively small problem instances, one therefore has to turn to heuristics where solutions can be obtained efficiently (but solutions are not necessarily optimal).

Many sequence based heuristics for finding phylogenetic trees are based on multiple alignments. Ideally, such an alignment should be based on the true evolutionary tree. This is not available, and instead some multiple alignment heuristics (such as ClustalW [Thompson *et al.*, 1994]) use a guide tree constructed by using pairwise alignment distances. This has the drawback of making the multiple alignment and subsequently the sought phylogenetic tree biased towards the guide tree. Alternatively, one can attempt to simultaneously find a multiple alignment and a phylogenetic tree. This is known as the problem of *generalized tree alignment*.

In this paper we reexamine the *deferred path heuristic* for generalized tree alignment. It was originally developed by Schwikowski and Vingron in the late 1990s [Schwikowski and Vingron, 1997; Schwikowski, 1998; Schwikowski and Vingron, 2003] and their work was based on the idea of using *sequence graphs* introduced by Hein [1989]. We give a unified description of the heuristic where such intermediate constructs as fork alignment graphs used in [Schwikowski and Vingron, 2003] are made obsolete. We discuss improvements of the heuristic which either result in better quality solutions or in a substantial speed-up. Finally, we test the heuristic on several real-life problem instances.

The paper is organized as follows. In Section 2 we give a general introduction to the concepts of tree alignment and generalized tree alignment. We also discuss how these concepts are related to pairwise and multiple alignment. Sequence graphs — which form the backbone of the deferred path heuristic — and the heuristic itself are described in Section 3. Some new improvements are discussed in Section 4. Computational experience with some real-life problem instances are covered in Section 5. Conclusions and ideas for further research are discussed in the closing Section 6.

2 Phylogenetic Trees and Sequence Alignment

Construction of phylogenetic trees for a set of species represented by DNA, RNA- or protein sequences is intimately related to the alignments of such sequences. In this section we briefly discuss these alignment problems and their relation to the construction of phylogenetic trees.

2.1 Tree Alignment

Consider a phylogenetic tree $T = (V, E)$ with n leaves where V denotes the set of nodes and E denotes the set of edges. Each leaf of T corresponds to a sequence S_j over a finite alphabet Σ (for example nucleotides in DNA or amino acids in a protein). In this paper we address the problem of determining *unrooted* phylogenetic trees. Identifying a root of a phylogenetic tree is non-trivial (and in fact it is a controversial issue in the "evolutionary"

community). Neither do we address evolutionary time issues which often are associated with the construction of phylogenetic trees.

The internal nodes of the phylogenetic tree T are assumed to have degree 3 (realistic assumption for most practical applications). T has therefore $n-2$ internal nodes. If sequences are assigned to all internal nodes, the cost of the tree can be defined as the sum of costs of optimal pairwise alignments (see below) between sequences in adjacent nodes. The *tree alignment* problem for a given tree T is therefore to assign sequences to the internal nodes such that the cost of the tree is minimized.

2.2 Pairwise Alignment

In *pairwise alignment* one considers two sequences S_i and S_j over a finite alphabet Σ. The *gap* symbol "$-$" is a special character not in Σ. Σ extended with the gap symbol is denoted by Σ'. An *alignment* of sequences S_i and S_j is a pair of equal-length sequences S_i' and S_j' over Σ' such that the removal of gaps would yield S_i and S_j, respectively. The sequences S_i' and S_j' can be placed in an *alignment matrix* A (with two rows) such that each column identifies a pair of symbols from Σ' aligned with each other. A column of A that consists of two gaps is called a *gap column*. Gap columns are normally irrelevant in pairwise alignment and can be removed from consideration. Alignments without gap columns are called *reduced* alignments. All alignments discussed in this paper are reduced alignments.

In this paper, we consider alignments in a minimization framework. In order to measure the quality of a given pairwise alignment, *cost* values are defined for all pairs of symbols in Σ'. Given $s_1, s_2 \in \Sigma'$, we denote the corresponding cost value $\delta(s_1, s_2)$. It is also assumed that $\delta(s_1, s_2) = 0$ if $s_1 = s_2$. In the simplest version, the cost of a pairwise alignment is the sum of costs taken over all columns of A. An *optimal pairwise alignment* is a pairwise alignment with the lowest possible cost. More than one optimal pairwise alignment can exist for the same pair of sequences.

More sophisticated measures incorporate the observation that contiguous gaps of length k are more likely to occur in sequences than k isolated gaps. So the cost of a pairwise alignment consists of two components. The cost of columns with no gaps is still the sum of their individual costs. A *maximal gap* in either S_i' or S_j' is a maximal contiguous sequence of gaps. Each maximal gap in a pairwise alignment contributes to the total cost of the pairwise alignment by the affine gap penalty $g(k) = a + b \cdot k$, where k is the length of the gap, and a and b are the gap open and gap extension penalties, respectively.

Optimal pairwise alignment is a well-understood optimization problem which can be solved in quadratic time by standard dynamic programming

techniques [Needleman and Wunsch, 1970].

2.3 Multiple Alignment

In *multiple alignment* the problem is to find an optimal alignment of n sequences, $n \geqslant 2$. In this case, the alignment matrix A has n rows. Most cost functions for multiple alignment are based on column costs and affine gap penalties. Tree alignment has also been suggested as a possible method of evaluating multiple alignments. Unfortunately, this approach is intractable unless the tree has a very simple topology (for example a star topology with one internal node adjacent to all leaf nodes). Even for trees with internal nodes of degree 3, the tree alignment problem is NP-hard. The cost scheme used in the proof of this result was a metric cost scheme with values 0, 1, and 2 and a 4-letter alphabet [Jiang *et al.*, 1994]. Hence, using tree alignment to determine the cost of a multiple alignment seems practically infeasible even for relatively short sequences.

2.4 Generalized Tree Alignment

Generalized tree alignment is an important and non-trivial extension of the tree alignment problem. Furthermore, it can be used to find good quality multiple alignments. Given n sequences $S_1, S_2, \ldots, S_n$, the problem is to find a tree T such that its tree alignment cost is smallest possible (for some appropriately defined pairwise alignment cost function). Once the tree and the sequences in internal nodes are determined, optimal pairwise alignments on adjacent nodes in the tree can be used to obtain a good quality multiple alignment. Unfortunately, the generalized tree alignment problem is not only NP-hard but in fact it is *MAX* SNP-hard [Wang and Jiang, 1994]. As a consequence, neither efficient exact algorithms nor polynomial approximation algorithms with arbitrarily small error ratio are likely to exist. Use of heuristics seems therefore to be the only feasible way to solve this problem computationally.

The generalized tree alignment problem is closely related to the *Steiner tree problem in graphs*: Given a graph $\mathcal{N} = (\mathcal{W}, \mathcal{F})$ with positive edge costs and a subset $\mathcal{S}$ of the vertices, the problem is to find a minimum-cost connected subnetwork of $\mathcal{N}$ spanning $\mathcal{S}$. It is obvious that a solution must be a tree. It may contain other vertices of $\mathcal{W}$ than those in $\mathcal{S}$. These additional vertices are usually referred to as *Steiner vertices*. It is straightforward to see that the generalized tree alignment can be formulated as the Steiner tree problem in a (very big) graph. More specifically, let $\mathcal{W}$ represent the set of all sequences over the alphabet Σ. If the lengths of sequences are not bounded, the graph will have an infinite number of vertices. But in practical applications the lengths can be bounded by a small multiple of the longest sequence among $S_1, S_2, \ldots, S_n$. Two vertices in $\mathcal{N}$ are connected by an edge

if the corresponding sequences can be obtained from one another by a substitution of one letter, or by insertion or deletion of one or several contiguous letters. The cost of such an edge is of course the cost of the corresponding substitution, insertion or deletion. If we let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be the sequences of the set of the given species, the generalized tree alignment is exactly the Steiner tree problem in the *Steiner graph* $\mathcal{N} = (\mathcal{W}, \mathcal{F})$.

3 Deferred Path Heuristic

The main purpose of this section is to give a description of the basic heuristic. The description is fairly detailed and for the sake of clarity, the initial description does not include so-called *detours* or affine gap penalties. The use of detours is instead postponed to Section 3.4 in which affine gap penalties are also briefly discussed.

3.1 Tree Construction

A *sequence graph* $G = (U, A)$ is a connected directed acyclic graph with a unique *source* vertex s (no incoming arcs) and a unique *sink* vertex t (no outgoing arcs). Each arc $a \in A$ has a label s_a taken from a finite alphabet Σ'. The set of all source-sink paths defines a set of sequences, each consisting of the consecutive labels of the arcs on a path. Our interest in sequence graphs stems from the fact that they are extremely well-suited to represent sets of related sequences in a compact manner. Sequence graphs were originally suggested by Hein [1989].

Note that the vertices in a sequence graph can be sorted topologically, i.e., if given a graph with $m + 1$ vertices we can make a numbering of the vertices using the integers from 0 to m such that if we refer to an arc a by its end-vertices, (i, j), we can safely assume that $i < j$. Furthermore, we can also assume that the source is assigned 0 and the sink is assigned m.

Suppose that n sequences $S_1, S_2, \ldots, S_n$ over a finite alphabet Σ are given. The objective is to find a phylogenetic tree $T = (V, E)$ with as low tree alignment cost as possible. The basic idea of the heuristic is to represent these n sequences by linear sequence graphs with labels on their unique paths from the source to the sink corresponding to the symbols in the sequences. These sequence graphs will correspond to leaf nodes in the phylogenetic tree $T = (V, E)$ that we want to construct. The heuristic then selects two sequence graphs $G = (U_G, A_G)$ and $H = (U_H, A_H)$ that are "closest" to each other. These two sequence graphs are replaced by another appropriately defined sequence graph $G \otimes H$. An internal node is added to T and it is connected by edges to the nodes corresponding to G and H. This process is repeated until only two sequence graphs remain. The corresponding nodes in T are connected by an edge and the construction of T is completed.

As already pointed out, during each iteration of the heuristic a pair of sequence graphs G and H is replaced by a new sequence graph $G \otimes H$. This new sequence graph can be defined in several ways. In the simplest approach, $G \otimes H$ would be a linear sequence graph representing exactly one sequence S such that the sum of costs of optimal pairwise alignments between S and the two sequences represented by G and H is minimized.

The problem with such an approach is of course that the choice of S can prove to be very bad in subsequent iterations of the heuristic. Sequence graphs are fortunately capable of representing several sequences in a compact way. The idea is therefore to let $G \otimes H$ represent several sequences and to *defer* the selection of a particular sequence. Suppose that S_G and S_H are sequences from G and H such that the cost of their optimal pairwise alignment is the smallest possible among all pairs of sequences, one from G and one from H. The cost of such an alignment is called the *distance* between G and H and we refer to the corresponding pair of sequences as an *optimal pair*. We can interpret such a pairwise alignment as a shortest path in the Steiner graph $\mathcal{N}$ where intermediate vertices correspond to sequences that all could act as S. Hence, it is natural to let $G \otimes H$ contain them all (including S_G and S_H). In fact, there could be several optimal pairs of sequences, one sequence of such a pair in G and the other sequence in H. So $G \otimes H$ can be extended to contain all sequences on the corresponding shortest paths in the Steiner graph $\mathcal{N}$. The problem of finding all such alignments can be solved, as we will see, by a straightforward dynamic programming algorithm.

Once the number of active sequence graphs has been reduced to two, a single optimal pair of sequences can be selected for the remaining pair of graphs. These sequences together with the tree T can be used to backtrack through preceding sequence graphs and to select appropriate sequences in each of them (using optimal pairs).

When sequence graphs represent several sequences and the choice of a particular sequence can be deferred as explained above, the heuristic will normally produce better solutions than the simple method using linear sequence graphs (representing one sequence only).

3.2 Distances between Sequences Graphs

To be able to select the closest pair of sequence graphs we naturally need to be able to compute the distance between two sequence graphs as defined above. The classic approach for finding the distance between two simple sequences of symbols is a dynamic programming algorithm. Such an algorithm uses the fact that the distance between two sequences can be based on the distances between the immediate prefixes of the sequences, i.e. the

problem exhibits the optimal substructure property. This is also true for sequence graphs.

Given two sequence graphs G and H of length m and n, respectively, we denote the distance between them $\delta(G, H)$. Let $G^{\triangleleft}(i)$ denote the *prefix subgraph* of G consisting of all paths from the source 0 to a given vertex i and let $H^{\triangleleft}(k)$ denote a similar prefix subgraph of H. Now $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$ is the distance between prefix graphs $G^{\triangleleft}(i)$ and $H^{\triangleleft}(k)$.

The distance between G and H can be found by iteratively filling out the values of a matrix D of size $(m+1) \times (n+1)$ as shown in Algorithm 1. Each entry in the matrix corresponds to a pair of vertices from G and H. Starting from the pair of sources, matrix values are updated by calculating the cost of aligning all outgoing arcs from the current vertices, i and k, with each other and with gaps. The calculations in the inner loops are based on the value of $D[i, k]$. Due to the topological sorting of the vertices we know that all paths to i and k have been handled earlier in the algorithm and thus $D[i, k]$ must be equal to $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$. In particular, $D[m, n] = \delta(G^{\triangleleft}(m), H^{\triangleleft}(n)) = \delta(G, H)$.

Algorithm 1 Graph variation of the classic pairwise alignment algorithm.

Given graphs G and H with vertices 0 to m and 0 to n.
Allocate matrix $D[m+1, n+1]$ with all entries set to ∞.
$D[0, 0] = 0$
for $i = 0$ to m **do**
  for $k = 0$ to n **do**
    for all arcs $a = (i, j)$ in G **do**
      for all arcs $b = (k, l)$ in H **do**
        $D[j, l] \leftarrow \min(D[j, l], D[i, k] + \delta(s_a, s_b))$
    for all arcs $a = (i, j)$ in G **do**
      $D[j, k] \leftarrow \min(D[j, k], D[i, k] + \delta(s_a, -))$
    for all arcs $b = (k, l)$ in H **do**
      $D[i, l] \leftarrow \min(D[i, l], D[i, k] + \delta(-, s_b))$
return $D[m, n]$

As previously noted, a sequence of symbols can be represented by a simple linear sequence graph. Note that in this case Algorithm 1 reduces to a standard dynamic programming algorithm for aligning two sequences. An example of the use of Algorithm 1 for two linear sequence graphs can be seen in Figure 1 where the final values of the D-matrix are given. Figure 1 also illustrates that one could easily store the origins of each entry in the D-matrix and thus generate all possible optimal alignments.

Each pair of edges is compared exactly once in Algorithm 1 thus the

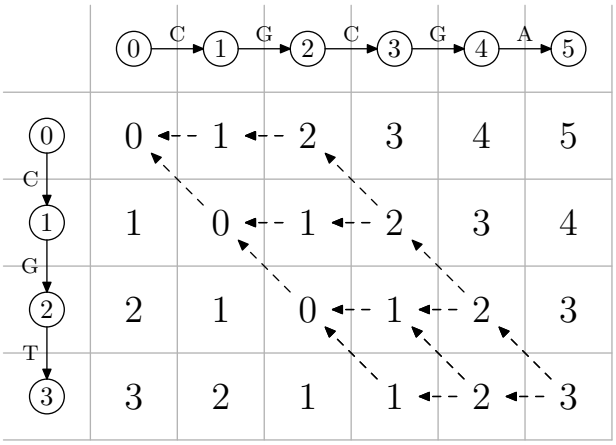


Figure 1. Example of the D-matrix of Algorithm 1 for a simple unit cost scheme i.e. $\delta(s_1, s_2) = 1$ for all $s_1 \neq s_2$. The value in the lower right corner of the matrix (3) is the cost of an optimal alignment. The arrows show all the possible ways back through the matrix corresponding to all possible optimal alignments, e.g. the rightmost path corresponds to aligning `CGCGA` with `--CGT`.

running time is $O(|A_G| \cdot |A_H|)$. The space requirement is dominated by the D-matrix and is $O(|U_G| \cdot |U_H|)$.

3.3 All Shortest Paths Sequence Graph

When a pair of sequence graphs, $G = (U_G, A_G)$ and $H = (U_H, A_H)$, has been selected the next step is to create a new sequence graph $G \otimes H$ representing all sequences which are on shortest paths between G and H in the Steiner graph $\mathcal{N}$.

As noted the D-matrix used in Algorithm 1 could easily be extended to save the paths corresponding to all possible optimal alignments (Figure 1). Any combination of the symbols aligned on these paths are then among the sequences we would like to include in the new sequence graph. In the following we describe how to do this efficiently.

Given a vertex i in G and a vertex k in H we have already defined the prefix distance $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$ and we have also seen that it is implicitly computed in Algorithm 1. Another set of values has to be computed before the combined sequence graph $G \otimes H$ can be constructed. Let $G^{\triangleright}(j)$ denote the *suffix subgraph* of G consisting of all paths from j to the sink m and let $H^{\triangleright}(l)$ denote a similar suffix subgraph of H. It should be clear that the distance values $\delta(G^{\triangleright}(j), H^{\triangleright}(l))$ for all pairs (j, l) can be computed by

Algorithm 1 simply by reversing the directions of all arcs in the graphs.

Now, the vertices of $G \otimes H$ will belong to the set $U_{G\otimes H} \subseteq U_G \times U_H$. More precisely, vertices are added to $G \otimes H$ for all vertex pairs $i \otimes k$ for which $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(G^{\triangleright}(i), H^{\triangleright}(k)) = \delta(G, H)$. In particular, the source of $G \otimes H$ will be the vertex $0 \otimes 0$ (the pair of sources in G and H) and the sink of $G \otimes H$ is the vertex $m \otimes n$ (the pair of sinks in G and H). An example of such a vertex set can be seen in Figure 2a in which the values of $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$ and $\delta(G^{\triangleright}(i), H^{\triangleright}(k))$ are included.

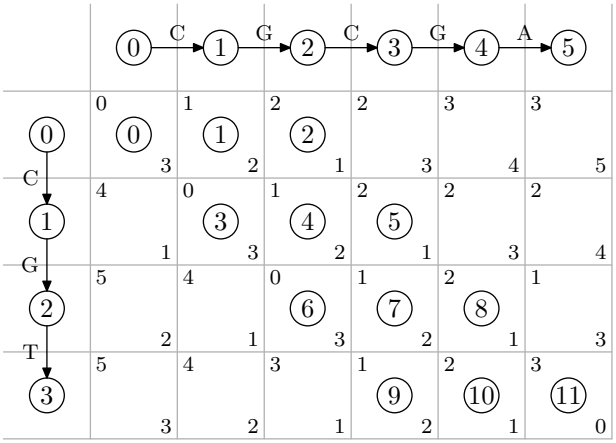
The arcs of $G \otimes H$ are constructed as follows. Given two vertices $i \otimes k$ and $j \otimes l$ in $G \otimes H$, consider an arc $a = (i, j)$ from G and an arc $b = (k, l)$ from H. Let s_a denote the label associated with a and let s_b denote the label associated with b. An arc between $i \otimes k$ and $j \otimes l$ labeled s, $s \in \Sigma'$, is added to $G \otimes H$ if there is a sequence S_G in G and a sequence S_H in H which can be aligned with each other such that labels of a and b are in the same column and the cost of the alignment is exactly $\delta(G, H)$. Any such lowest cost alignment requires that the portion of S_G in the prefix graph $G^{\triangleleft}(i)$ is (optimally) aligned with the portion of S_H in the prefix graph $H^{\triangleleft}(k)$, the label of a is aligned with the label of b and the portion of S_G in the suffix graph $G^{\triangleright}(j)$ is optimally aligned with the portion of S_H in the suffix graph $H^{\triangleright}(l)$. A necessary and sufficient requirement for such a bounded cost alignment to exist is that $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(s, s_a) + \delta(s, s_b) + \delta(G^{\triangleright}(j), H^{\triangleright}(l))$ is equal to $\delta(G, H)$.

When constructing $G \otimes H$, one also has to decide if there are optimal alignments such that the arc $a = (i, j)$ is aligned with a gap while a subsequence of the prefix graph $G^{\triangleleft}(i)$ is aligned with a subsequence of the prefix graph $H^{\triangleleft}(k)$ for some vertex k of H and a subsequence of the suffix graph $G^{\triangleright}(j)$ is aligned with a subsequence of the suffix graph $H^{\triangleright}(k)$. If $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(s, s_a) + \delta(s, -) + \delta(G^{\triangleright}(j), H^{\triangleright}(k))$ is equal to $\delta(G, H)$, then an arc between $i \otimes k$ and $j \otimes k$ with label s is added to $G \otimes H$. Note that in the case of affine gap penalties, the situation becomes more complicated as one has to take into account that several symbols can be aligned with contiguous gap symbols.

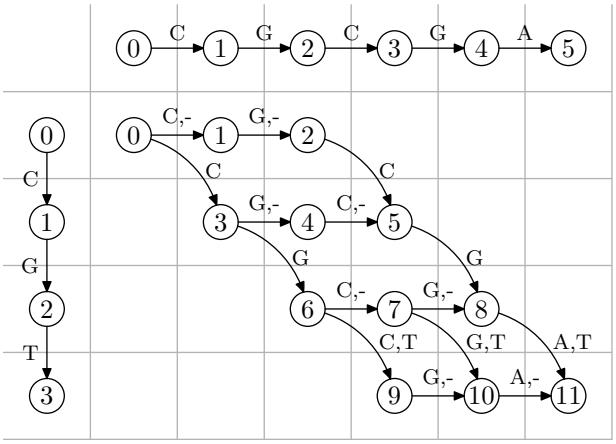
The symmetric situation where an arc from H is to be aligned with a gap symbol is dealt with in a similar manner. This and more details can be seen in Algorithm 2. The running example is completed in Figure 2b in which the vertices found in Figure 2a are supplemented by the arcs described above. Both running time and space requirement is easily seen to be $O(|A_G| \cdot |A_H|)$.

3.4 Detours

As described by Schwikowski and Vingron [2003], it is still possible to further improve the flexibility of the heuristic by letting sequence graphs in-



(a)



(b)

Figure 2. Continuation of the example in Figure 1 showing the use of Algorithm 2. (a) The 12 vertices used in a sequence graph representing all sequences on all shortest paths between the two original sequences in the Steiner graph $\mathcal{N}$. Upper left corner values are prefix distances and lower right corner values are suffix distances. For each vertex the sum of the corner values is equal to 3. (b) The final sequence graph. Note that only one arc is drawn for a connected pair of vertices, but in some cases with multiple symbols. Algorithm 2 would create an arc for each symbol.

Algorithm 2 Construction of a sequence graph containing all sequences on shortest paths between two given sequence graphs

Given graphs $G = (U_G, A_G)$ and $H = (U_H, A_H)$ with vertices 0 to m and 0 to n.
Initialize an empty graph $G \otimes H$. Vertices will be a subset of $U_G \times U_H$.
Add vertices to $G \otimes H$ for all pairs $i \otimes k$ for which $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(G^{\triangleright}(i), H^{\triangleright}(k)) = \delta(G, H)$.
for all vertex pairs $i \otimes k$ in $G \otimes H$ **do**
  $\delta^{\triangleleft} \leftarrow \delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$
  // Symbols on arcs from both graphs aligned.
  for all arcs $a = (i, j)$ in G **do**
    for all arcs $b = (k, l)$ in H **do**
      for all symbols $s \in \Sigma'$ **do**
        if $\delta^{\triangleleft} + \delta(s, s_a) + \delta(s, s_b) + \delta(G^{\triangleright}(j), H^{\triangleright}(l)) = \delta(G, H)$ **then**
          Add/update arc with symbol s from $i \otimes k$ to $j \otimes l$ in $G \otimes H$.
  // Symbols on arcs from one graph aligned with gaps.
  for all arcs $a = (i, j)$ in G **do**
    for all symbols $s \in \Sigma'$ **do**
      if $\delta^{\triangleleft} + \delta(s, s_a) + \delta(l, -) + \delta(G^{\triangleright}(j), H^{\triangleright}(k)) = \delta(G, H)$ **then**
        Add/update arc with symbol s from $i \otimes k$ to $j \otimes k$ in $G \otimes H$.
  for all arcs $b = (k, l)$ in H **do**
    for all symbols $s \in \Sigma'$ **do**
      if $\delta^{\triangleleft} + \delta(l, -) + \delta(s, s_b) + \delta(G^{\triangleright}(i), H^{\triangleright}(l)) = \delta(G, H)$ **then**
        Add/update arc with symbol s from $i \otimes k$ to $i \otimes l$ in $G \otimes H$.
return $G \otimes H$

clude suboptimal sequences. To facilitate this, *weighted* sequence graphs are introduced. A *weighted* sequence graph is a sequence graph $G = (U, A)$ where each arc a also has a weight w_a. This weight is a measure of the minimum additional cost caused by using this arc in a path from source to sink compared to an optimal path. The weight of a source-sink path in G is defined as the sum of weights of its arcs. In particular, an optimal path only contains 0-weight arcs.

The *weighted distance* between a pair of sequences, one in a weighted sequence graph G and the other in a weighted sequence graph H is defined as the distance between them supplemented by the weights of their paths. The *weighted distance* between G and H is defined as the smallest weighted distance between pairs of sequences, one in G and one in H. This is also denoted by $\delta(G, H)$.

Let Δ be a nonnegative real number. In the *deferred path heuristic with detours*, the weighted sequence graph $G \otimes H$ will contain all sequences S where the sum of the weighted distances from S to a sequence S_G in G and a sequence S_H in H is at most Δ away from the weighted distance between G and H. In other words, $G \otimes H$ will contain all sequences which are at most Δ away from the shortest paths between sequences of an optimal pair from G and H in the Steiner graph $\mathcal{N}$.

The algorithms presented so far need surprisingly few changes to be able to handle the Δ parameter. The distance algorithm simply includes the arc weights introduced when calculating the values of the matrix entries. The changes to the construction algorithm are also quite simple. Some of the changes needed are given in Algorithm 3. Especially note the calculation of the arc weight. When adding an arc a from $i \otimes k$ to $j \otimes l$, the weight is computed by adding the costs of the arc to the cost $\delta(G^{\triangleright}(j), H^{\triangleright}(l))$ of an optimal path from the sink to $j \otimes l$, and then subtracting the cost $\delta(G^{\triangleright}(i), H^{\triangleright}(k))$ of an optimal path from the sink to $i \otimes k$.

It it obvious that as Δ increases, so does the number of sequences (represented by paths) in weighted sequence graphs. So the choice of an appropriate Δ is crucial. If chosen too low, the quality of the solution can be poor. If chosen too high, the sizes of weighted sequence graphs can explode. It should also be noted that increasing Δ can also result in worse results for the deferred path heuristic with detours — even for a fixed tree topology. This is due to the fact that a Δ_1 sequence graph in an internal node of such a fixed tree is not guaranteed to be a subgraph of the corresponding Δ_2 sequence graph even if $\Delta_2 > \Delta_1$.

It is possible to extend all of the algorithms described to also work for affine gap penalties without changing the running time with more than a constant factor [Schwikowski, 1998]. The idea is basically the same as for

Algorithm 3 Some of the changes needed when constructing a sequence graph including detours.

```
...
Add vertices to G ⊗ H for all pairs i⊗k for which δ(G^◁(i), H^◁(k)) +
δ(G^▷(i), H^▷(k)) ⩽ δ(G, H) + Δ.
for all arcs a = (i, j) in G do
  for all arcs b = (k, l) in H do
    for all symbols s ∈ Σ' do
      if δ^◁ + δ(s, s_a) + δ(s, s_b) + δ(G^▷(j), H^▷(l)) ⩽ δ(G, H) + Δ then
        w ← δ(G^▷(j), H^▷(l)) + δ(s, s_a) + w_a + δ(s, s_b) + w_b − δ(G^▷(i), H^▷(k))
        Add/update arc with symbol s and weight w from i⊗k to j⊗l in
        G ⊗ H.
...
```

making basic pairwise alignments with affine gap penalties [Gotoh, 1982]. In short, one needs three values in each entry of the matrix corresponding to each of the possible gap-insertions (a gap in one of the sequences or no gap at all). When building a new sequence graph each of these values can cause a vertex to be created.

4 Improvements

In the following we suggest two improvements to the deferred path heuristic with detours. The first one focuses on decreasing the amount of space and time used by the heuristic with limited sacrifice of solution quality and the second one focuses on increasing solution quality at the cost of time.

4.1 Segmentation

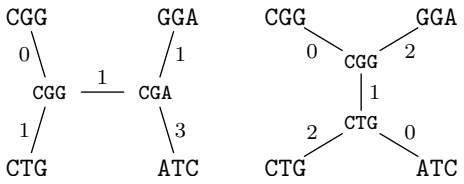
The rapidly increasing size of sequence graphs with increasing Δ can be limited by pruning edges and nodes. Schwikowski and Vingron [2003] enforces a limit of 1000 edges by randomly removing edges from any sequence graphs generated. In the following we suggest an alternative approach which we call *segmentation*. The basic idea is to heuristically split the set of input sequences into a series of sets and then work on each set in the series individually when measuring distances and creating new sequence graphs.

Consider the example given in Figure 3a. Three input sequences are given. If solving this problem without using detours ($\Delta = 0$) we get the alignment in Figure 3b. In the middle of the sequences we notice a no-cost alignment of 3 consecutive symbols. When solving the problem with detours these symbols are not likely to be aligned differently and therefore we could fix the location of this segment and instead solve the problem for three smaller sets of sequences as indicated in Figure 3c.

```
AGTGAGTCATG        A-GTGAGTCATG-      AGTG  | AGT | CATG
TGCAGTAGGT         T-G-CAGT-AGGT      TGC   | AGT | AGGT
TCTTCAGTGCC        TCTTCAGT--GCC      TCTTC | AGT | GCC
```

(a) (b) (c)

Figure 3. Segmentation example.



(a) (b)

Figure 4. Example with the two possible topologies when solving a problem with four sequences. (a) Using a unit cost scheme an initial greedy choice is to join `CGG` and `CTG` at a cost of 1. The final solution then has a total cost of 6. (b) A non-greedy choice at a cost of 2 results in a better final solution with a total cost of 5.

In the example above, a sequence of 3 consecutive perfect columns was used to fix a segment. In general this approach could be parameterized by requiring a minimum of σ consecutive perfect columns. A large value of σ would cause less segments to be found, but also decrease the risk of missing good solutions.

Using perfect columns is not the only possible strategy. Numerous other strategies could be used and one could even use some fast multiple alignment method such as ClustalW [Thompson *et al.*, 1994] to get the initial alignment and then use this to also fix segments when solving the problem for $\Delta = 0$.

4.2 Tree Construction Strategies

As previously described, the basic tree construction heuristic simply makes a greedy choice in each iteration, selecting the sequence graphs which are "closest" in terms of distance. This is also the only strategy considered by Schwikowski and Vingron [2003]. The greedy choice is not necessarily the best choice. This is illustrated with a small example in Figure 4 in which the best solution is based on a non-greedy initial choice.

	Size	Length			Origin
		Min.	Max.	Avg.	
`Sankoff`	9	118	122	120.1	[Sankoff *et al.*, 1976]
`rRNA`	602	96	122	116.6	[Szymanski *et al.*, 2002]
`SRP_euk`	71	256	317	295.3	[Rosenblad *et al.*, 2003]

Table 1. Data instances used in the experiments.

An exhaustive search of all tree topologies is not a viable alternative for most problem instances, but other approaches could be considered. Here we suggest a simple *look ahead* scheme in which we improve the greedy choice by extending the search in both width and depth. The idea is to make several choices and then continue the heuristic for each of them comparing the choices at a later stage. Three parameters can be used to describe this scheme. The number of levels to look ahead (the depth), λ_d, before comparing the choices made, the number of (best) pairs of sequence graphs to consider at each level (the width), λ_w, and the number of steps, λ_s, to actually take when the best sequence of choices has been found. The normal greedy choice would correspond to $\lambda_d = 0, \lambda_w = 1$ and $\lambda_s = 1$.

5 Computational Experiments

An implementation of the deferred path heuristic with detours has been done in C++ and the experiments have been executed on a 3GHz Pentium with 1 MB level 2 cache and 3.5 GB of main memory. The source code is available on request.

The main purpose of the computational experiments is to see if the segmentation scheme and the look ahead scheme behave as expected. Three data instances are used in the experiments (Table 1), all containing sequences with nucleotides. The classic `Sankoff` instance only contains 9 sequences and it is included to verify that we obtain the same solution as Schwikowski and Vingron [2003]. The other instances contain far too many sequences to be handled by our implementation since the sequence graphs would quickly become too big. Instead we select 10 random sets of 10 sequences from each instance and report average results. This should also ensure that we get results which to some extent indicate the general behaviour of the heuristic and the new improvements.

The cost matrix used for all experiments is the same as the one originally used for the `Sankoff` instance, i.e., $\delta(\mathtt{A},\mathtt{G}) = \delta(\mathtt{C},\mathtt{T}) = 1$, $\delta(\mathtt{A},\mathtt{C}) = \delta(\mathtt{A},\mathtt{T}) = \delta(\mathtt{C},\mathtt{G}) = \delta(\mathtt{G},\mathtt{T}) = 1.75$ and linear gap cost $g(k) = 2.25k$. All experiments are done with Δ ranging from 0 to 3 and segmentation with $\sigma \in 2, 3, 6, \infty$,

where ∞ means no segmentation. Only two tree construction schemes are tested. The standard greedy approach and a lookahead scheme with $\lambda_w = 3$, $\lambda_d = 3$ and $\lambda_s = 1$.

The results are presented in Table 2. The best known result for the `Sankoff` instance [Schwikowski and Vingron, 2003] is found in all cases when $\Delta \geqslant 2$. In general, results are not becoming worse when using segmentation, but these instances also only allow few segments. An average of 9 segments for the `rRNA` instance when $\sigma = 2$ reveals a considerable decrease in running time which is an indication of the potential of segmentation, but other segmentation strategies should be considered to increase the number of segments. The look ahead scheme works as expected finding better solutions in most cases, but the cost in running time is large compared to the results obtained using large Δ values. More experiments are needed to further analyze this time/quality tradeoff.

6 Conclusions

We have given a short and concise description of the deferred path heuristic with detours and we have shown that there is still room for improving both computation speed and quality of solutions. We have also taken the first steps to further examine the potential of using sequence graphs for generalized tree alignment and it is our intention to continue this work and to improve the implementation to make it of practical use. Most interesting is to find new ways of limiting the size of the sequence graphs without sacrificing solution quality.

More experiments are needed, especially a comparison of the quality of our phylogenetic trees with the trees constructed by the best existing solution methods.

BIBLIOGRAPHY

[Felsenstein, 2003] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2003.

[Gotoh, 1982] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[Hein, 1989] J. Hein. 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*, 6:649–668, 1989.

[Jiang *et al.*, 1994] T. Jiang, E. L. Lawler, and L. Wang. Aligning Sequences via an Evolutionary Tree: Complexity and Approximation. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 760–769. ACM Press, 1994.

[Needleman and Wunsch, 1970] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[Rosenblad *et al.*, 2003] M. A. Rosenblad, J. Gorodkin, B. Knudsen, C. Zwieb, and T. Samuelsson. SRPDB: Signal Recognition Particle Database. *Nucleic Acids Research*, 31(1):363–364, 2003.

	Greedy			Look ahead		
	$\sigma \geqslant 6$	$\sigma = 3$	$\sigma = 2$	$\sigma \geqslant 6$	$\sigma = 3$	$\sigma = 2$
`Sankoff`	Cost			Cost		
$\Delta = 0$	299.50	299.50	299.50	299.50	299.50	299.50
$\Delta = 1$	298.00	298.00	298.00	296.75	296.00	296.00
$\Delta = 2$	295.75	295.75	295.75	295.75	295.75	295.75
$\Delta = 3$	295.75	295.75	295.75	295.75	295.75	295.75
	Seconds			Seconds		
$\Delta = 0$	0.07	0.07	0.07	0.63	0.63	0.62
$\Delta = 1$	0.11	0.07	0.04	0.93	0.62	0.45
$\Delta = 2$	0.20	0.13	0.10	2.04	1.21	0.92
$\Delta = 3$	0.51	0.31	0.24	5.66	3.12	2.37
#segs	1.00	5.00	11.00	1.00	5.00	11.00
`rRNA`	Cost			Cost		
$\Delta = 0$	391.85	391.85	391.85	387.27	387.27	387.27
$\Delta = 1$	383.43	383.43	383.75	382.95	382.95	382.68
$\Delta = 2$	382.73	382.73	383.95	382.52	382.52	382.45
$\Delta = 3$	382.30	382.30	382.62	381.25	381.25	381.18
	Seconds			Seconds		
$\Delta = 0$	0.10	0.11	0.11	1.03	1.25	0.83
$\Delta = 1$	0.13	0.12	0.07	1.77	1.92	0.72
$\Delta = 2$	0.27	0.26	0.15	5.41	6.01	1.99
$\Delta = 3$	0.91	0.89	0.49	24.36	24.63	8.01
#segs	1.00	2.00	9.40	1.00	2.00	9.00
`SRP_euk`	Cost			Cost		
$\Delta = 0$	1193.08	1193.08	1193.08	1184.62	1184.62	1184.62
$\Delta = 1$	1169.55	1169.55	1169.22	1163.65	1163.40	1164.15
$\Delta = 2$	1159.47	1159.47	1160.33	1154.55	1154.33	1152.75
$\Delta = 3$	1155.60	1155.60	1156.40	1151.40	1151.42	1151.65
	Seconds			Seconds		
$\Delta = 0$	0.61	0.61	0.61	6.05	6.05	6.02
$\Delta = 1$	1.40	1.06	0.91	16.06	9.87	9.26
$\Delta = 2$	4.86	3.59	3.04	68.42	42.33	40.88
$\Delta = 3$	23.16	17.50	14.75	357.23	215.61	205.51
#segs	1.00	4.00	5.80	1.00	4.80	6.60

Table 2. Results from the computational experiments. The `Sankoff` data is the result of a single instance with 9 sequences. The rest of the data are averages of 10 runs on instances with 10 sequences. The results in the columns with $\sigma \geqslant 6$ correspond to not using segmentation at all since none of the data instances can be split into segments when requiring 6 or more consecutive no-cost columns.

[Sankoff *et al.*, 1976] D. Sankoff, R. J. Cedergren, and G. Lapalme. Frequency of insertion-deletion, transversion, and transition in evolution of 5S ribosomal RNA. *Journal of Molecular Evolution*, 7:133–149, 1976.

[Schwikowski and Vingron, 1997] B. Schwikowski and M. Vingron. The deferred path heuristic for the generalized tree alignment problem. *Journal of Computional Biology*, 4(3):415–431, 1997.

[Schwikowski and Vingron, 2003] B. Schwikowski and M. Vingron. Weighted sequence graphs: boosting iterated dynamic programming using locally suboptimal solutions. *Discrete Applied Mathematics*, 127(1):95–117, 2003.

[Schwikowski, 1998] B. Schwikowski. A new algorithmic approach to the construction of multiple alignments and evolutionary trees. Technical Report 11, German National Center for Information Technology, 1998.

[Semple and Steel, 2003] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003.

[Szymanski *et al.*, 2002] M. Szymanski, M. Z. Barciszewski, V. A. Erdmann, and J. Barciszewski. 5S Ribosomal RNA Database. *Nucleic Acids Research*, 30(1):176–178, 2002.

[Thompson *et al.*, 1994] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[Wang and Jiang, 1994] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computional Biology*, 1(4):337–348, 1994.

Benny K. Nielsen, Pawel Winter, and Martin Zachariasen

Department of Computer Science
University of Copenhagen, DK-2100 Copenhagen Ø, Denmark
{benny,pawel,martinz}@diku.dk

Stinus Lindgreen

Bioinformatics Centre
University of Copenhagen, DK-2100 Copenhagen Ø, Denmark
stinus@binf.ku.dk