A Large Version of the Small Parsimony Problem

Jakob Fredslund[1], Jotun Hein[2], and Tejs Scharling[1]

[1] Bioinformatics Research Center, Department of Computer Science, University of Aarhus, Denmark,
{jakobf|,tejs}@birc.dk

[2] Department of Statistics, University of Oxford, United Kingdom,
hein@stats.ox.ac.uk

Abstract. Given a multiple alignment over k sequences, an evolutionary tree relating the sequences, and a subadditive gap penalty function (e.g. an affine function), we reconstruct the internal nodes of the tree optimally: we find the optimal explanation in terms of indels of the observed gaps and find the most parsimonious assignment of nucleotides. The gaps of the alignment are represented in a so-called gap graph, and through theoretically sound preprocessing the graph is reduced to pave the way for a running time which in all but the most pathological examples is far better than the exponential worst case time. E.g. for a tree with nine leaves and a random alignment of length 10.000 with 60% gaps, the running time is on average around 45 seconds. For a real alignment of length 9868 of nine HIV-1 sequences, the running time is less than one second.

1 Introduction

The relationship of biological sequences such as proteins and DNA can be described by a rooted tree, with a series of bifurcations corresponding to duplications in the past. Because of fluctuations in rates of evolution of such molecules it is not possible to determine the position of the root and the relative order of ancient duplications. When a rooted bifurcating tree is stripped of this information it becomes an unrooted tree, where all internal vertices have three incident edges. Protein and DNA sequencing techniques allow the determination of the sequences at the leaves of such a tree (the present). The sequences that represent ancestral molecules are unobservable and must be reconstructed according to some principle.

Usually the principle of parsimony is invoked: given a function that measures the distance between two sequences, choose ancestral sequences that minimize the total distance between neighbors in the tree, i.e. the overall amount of evolution. The distance function defines the penalty, or cost, of basic events such as substitutions and insertions/deletions (indels), and thus the distance between two sequences s_1 and s_2 is the "cheapest" process of evolution in terms of substitutions and indels which transforms s_1 into s_2. Application of the parsimony principle falls in two categories, depending on whether the phylogeny is known or not. If the phylogeny is known, it is the small parsimony problem; otherwise it is the large parsimony problem. This article will only address the small parsimony problem, but since it is a hard version involving complete sequences, it is called a large version of the small parsimony problem.

The simplest case where only substitutions have occurred and they all have cost 1 was first addressed by Fitch [1] and independently by Hartigan [3]. This was generalized to allow substitutions with arbitrary costs by Sankoff [5]. The computing time used by these algorithms was of the order $O(n \cdot k)$, where n is the length of the sequences and k the number of sequences. The situation with two sequences subject to substitutions and indels of length 1 was solved by Sellers [4] and Sankoff [2]. This is called the *approximate string matching problem* and uses $O(n^2)$ computing time. Sankoff [5] combined the string matching and Fitch-Hartigan-Sankoff algorithms to devise an algorithm that reconstructs the history of a set of sequences if their phylogenetic tree is known. Its running time was $O(n^k \cdot 2^k \cdot k)$, which is prohibitive.

Waterman et al. [6] introduced sequence comparisons of two sequences with indels longer than one. Their algorithm uses $O(n^3)$ computing time. Gotoh [7] gave an algorithm that uses time $O(n^2)$ if the gap penalty function is of the form $g(l) = \alpha + \beta \cdot l$, where l is the length of the gap. Fredman [8] generalized Gotoh's algorithm to three sequences if the gap penalty function was of the form $g(l) = \alpha$, i.e. the cost of an indel is independent of its length.

Wang et al. [10] studied the *tree alignment* problem where a multiple alignment is found by reconstructing the sequences at the internal nodes of a phylogeny minimizing the sum of all pairwise alignments between neighbors in the tree. In their approach gap costs are linear and the overall goal is to find a multiple alignment of the given sequences. The algorithm was improved in [11] where a polynomium time approximation scheme is given which yields a solution with a cost of at most 1.583 times the optimum, for sequences of length up to 200.

Finally, much work exists on finding the best multiple alignment of k sequences with no phylogeny given, e.g. [12,13]. Hein [9] devised an algorithm which finds an approximation to the optimal multiple alignment of k sequences and simultaneously reconstructs their ancestors, given the tree. However, multiple alignment is not our aim in this article: we give an algorithm which *optimally* reconstructs the sequences at the internal nodes of a phylogeny, given an alignment of the sequences at the leaves. The algorithm allows indels of any length and any subadditive gap penalty function (e.g. an affine function on the form $g(l) = \alpha + \beta \cdot l$ with $\alpha, \beta \geq 0$). Its worst-case time complexity is exponential, but in practice it is very fast and has been succesfully tested for nine HIV sequences of length 9868.

2 Indels of Length Greater than One

Given a multiple alignment the problem arises of how to interpret gap symbols as indels. When only indels of length one are allowed each column can be treated separately. When longer indels are allowed the problem becomes non-trivial as illustrated in Figure 1.

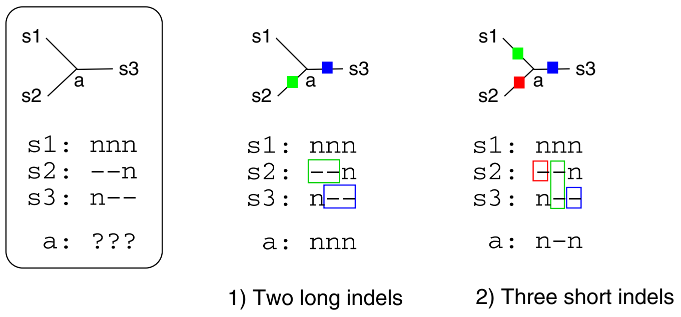


Fig. 1. Two possible explanations for the gaps in an alignment with three sequences (in rounded rectangle). Small boxes represent indels. For clarity we focus on indels and ignore substitutions, thus denoting all nucleotides simply by 'n'.

There are gap symbols in sequences s_2 and s_3. The gaps partly overlap and they can be explained in two ways:

(1) Two long indels have occurred: one on the evolutionary path between a and s_2 causing s_2's gaps, and one on the path between a and s_3 causing s_3's gaps. Consequently, a, the closest common relative of s_2 and s_3, has no gaps.
(2) Three short indels have occurred: one on the path between s_1 and a causing gaps in the middle interval in both a, s_2 and s_3 in one go; one between a and s_2 accounting for s_2's gap in the first column, and one between a and s_3 accounting for s_3's gap in the last column. With this explanation, a has a gap in the middle.

If we solve the problem the same way as when indels have length one, we would place the gaps as high (i.e., as far from the leaves) as possible in the tree, and so we would choose explanation 2) in Figure 1 — but that means that three indels have occurred whereas choosing explanation 1 means only two. On the other hand, since now both the *number* and *length* of the indels may vary between possible explanations, it is no longer obvious that the most parsimonious explanation is simply the one with the fewest indels. Whether two long indels are cheaper than three short ones depends on the chosen gap penalty function. With the function $g(l) = \alpha + \beta l$, where l is the length of the gap, the costs of the two explanations are $2(\alpha + 2\beta)$ versus $3(\alpha + \beta)$, respectively.

Not only is it non-trivial to exhaustively list all possible explanations; it may also be non-trivial to find the cost of a given explanation. Consider the example in Figure 2: in explanation 1), the gaps are caused by three indels; one of length 2 on the edge between s_2 and a causing s_2's gaps, one of length 1 on the edge

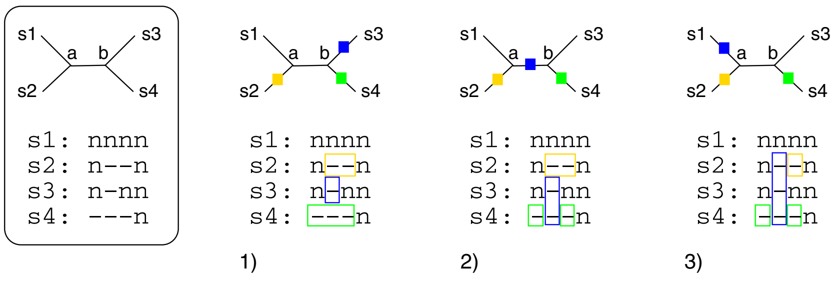


Fig. 2. Three possible explanations for the gaps in an alignment with four sequences. Obviously explanation 1) involves three indels, but in fact explanations 2) and 3) also only involve three and not four as it would seem. See text.

between s_3 and b causing s_3's gaps, and one of length 3 on the edge between s_4 and b causing s_4's gaps.

In explanation 2), a single indel occurring between a and b causes gaps in both s_3 and s_4 (and b) in the second column of the alignment. In explanation 3) this indel is pushed further back to the edge between s_1 and a now causing gaps in s_2, s_3 and s_4 (and a and b) all at once. Depending on the direction of evolution, this indel is either an insertion happening *after*, or a deletion happening *before* the emergence of b, s_3, and s_4. In both cases, the first and third alignment columns are in fact consecutive in b, s_3, and s_4. Therefore, though it might seem we need two indels to account for the two remaining gaps in s_4, they may be the result of only one. Thus, in this particular case, one indel can explain gaps in two non-adjacent alignment columns because the intermediate symbols have been deleted earlier or inserted later in the evolution. Concluding the example, we end up with the following costs for the three explanations: 1) $3\alpha + 6\beta$, 2) $3\alpha+5\beta$, and 3) $3\alpha+4\beta$. In passing, we observe that 3) is the optimal explanation for any (positive) choices of α and β.

These examples hopefully illustrate the complexity of handling indels of length more than 1. In general two questions arise: “What are the consistent explanations of a given multiple alignment in terms of indels?” and second: “Given some gap penalty function, which explanation is optimal?” In the following we give an algorithm that solves the problem.

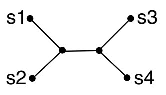
3 Our Algorithm

We assume we are given a multiple alignment $\mathcal{M}$ of $k > 2$ sequences $S_1 \dots S_k$, and a phylogeny in the form of a binary evolutionary tree $\mathcal{T}$ relating these sequences. Thus each S_i is a leaf in $\mathcal{T}$, and all other nodes in $\mathcal{T}$ have three neighbors. We do not take branch lengths into account. The length of $\mathcal{M}$ is n with the columns numbered 1 to n. Each sequence is a string over some alphabet Σ extended with a gap symbol '−'. Further, we are given a gap penalty function $g(l)$ defining the cost of a gap of length l. We assume g is subadditive: $g(l_1+l_2) \leq g(l_1)+g(l_2)$. In the following, for brevity we will let Σ denote the set of nucleotides $\{A, C, G, T\}$, and we will use the affine gap cost function $g(l) = \alpha + \beta l$, although any alphabet applies as well as, e.g., logarithmic or root gap cost functions. We will write $[a : b]$ to denote the sequence of integers from a to b, including the endpoints a and b.

No sequence has all gaps, and neither does any column. We assume that aligned nucleotides are related (call it the R assumption): if sequences s_i and s_j both have a nucleotide in some column, the closest common ancestor of s_i and s_j also has a nucleotide in this position. It follows that *all* sequences on the path from s_i to s_j in $\mathcal{T}$ have nucleotides in this position. We make no assumption about the direction of time in the tree $\mathcal{T}$. In other words, $\mathcal{T}$ is not *rooted.* For a leaf, we can talk about the unique *closest relative* since it has only one edge; for an internal node, the closest relative is not well-defined since internal nodes have three edges.

If an edge connecting two nodes a and b is removed from $\mathcal{T}$, the result is two disjoint *rooted* subtrees $\mathcal{T}_a$ and $\mathcal{T}_b$ rooted at a and b, respectively. An insertion that occurs on this edge inserts nucleotides in all nodes in $\mathcal{T}_a$ and gaps in all nodes in $\mathcal{T}_b$, or vice versa. Similarly, a deletion converts nucleotides to gaps in all nodes in $\mathcal{T}_a$ and leaves the nucleotides in all nodes in $\mathcal{T}_b$, or vice versa. In other words, *an indel affects a whole subtree* of $\mathcal{T}$. Thus we can and will identify an indel by the subtree in which it inserts gaps. Note that the closest relative of a subtree $\mathcal{T}' \subset \mathcal{T}$ is well-defined: it is the unique node $n \notin \mathcal{T}'$ which has an edge to the root of $\mathcal{T}'$.

It is easily shown that a tree with $k > 2$ leaves has $4k - 5$ different non-empty subtrees. We can represent a subtree $\mathcal{T}'$ of $\mathcal{T}$ unambiguously by its leaves; of course not all leaf subsets represent subtrees. Consider Figure 3: this tree with $k = 4$ leaves has $4k - 5 = 11$ different subtrees (counting $\mathcal{T}$ itself); e.g., $\{s_1\}$, $\{s_3, s_4\}$, and $\{s_1, s_2, s_4\}$. Note that, for example, $\{s_2, s_3\}$ is *not* a subtree: one cannot cut one edge and end up with a subtree containing exactly the leaves $\{s_2, s_3\}$.



```
s1:nn-----
s2:nn--nnn
s3:----nnn
s4:n-nn---
```

Fig. 3.

3.1 Construction of a Gap Graph

An alignment column has a *gap configuration* given by the sequences that have gap symbols in this column. A maximal interval $I \subseteq [1 : n]$ where alle columns have the same gap configuration is called a *gap interval.* E.g., the alignment in Figure 3 has four gap intervals: $I_1 = [1 : 1]$, $I_2 = [2 : 2]$, $I_3 = [3 : 4]$, and $I_4 = [5 : 7]$. Gap intervals are important since the indels used to explain the gaps in one column of a gap interval can be extended to explain the remaining columns without the cost of introducing extra indels (by subadditivity of the gap cost function this is cheaper, so indels are never broken off in the middle of a gap interval). Recall that each leaf in the evolutionary tree $\mathcal{T}$ corresponds to a sequence. The *tree covering* of a gap interval I is the minimal forest $\mathcal{F}$ of subtrees of $\mathcal{T}$ where

(C1) All leaves of $\mathcal{T}$ with gap symbols in I belong to a subtree in $\mathcal{F}$
(C2) All leaves in each subtree in $\mathcal{F}$ have gap symbols in I

Thus, a tree covering of I "covers" exactly the sequences that have gaps in I and not the ones that have nucleotides. Note that a tree covering is minimal in the *number* of subtrees that satisfy the above conditions. In other words, each subtree is maximal: it can not be extended without including leaves with nucleotides in I. In the following, we will represent a gap interval by a pair $(\mathcal{F}, I)$, where I is the gap interval itself and $\mathcal{F}$ is its associated tree covering.

Look again at Figure 3. In I_1, to satisfy condition C1 we need to cover the sequence s_3 only. Therefore, the tree covering $\mathcal{F}_1$ associated with I_1 is $\{\{s_3\}\}$, a forest of only one subtree. I_2 and I_3 are also covered by single subtree forests: $\mathcal{F}_2 = \{\{s_3, s_4\}\}$, and $\mathcal{F}_3 = \{\{s_1, s_2, s_3\}\}$. Finally, to cover I_4 we need two subtrees: $\mathcal{F}_4 = \{\{s_1\}, \{s_4\}\}$. This is a minimal forest: any subtree with more than one leaf would include s_2 or s_3 which have nucleotides in I_4, and thus the subtree would violate condition C2.

For now we focus on the gaps, and so we simply represent the alignment by its constituent gap intervals, i.e. the set $\{(\mathcal{F}_i, I_i)\}$, where, for all i, interval I_i precedes interval I_{i+1} in the alignment and $\bigcup I_i$ is a disjoint partition of $[1:n]$. We call such a set a *gap division*.

A gap division $\mathcal{D}$ now induces a *gap graph* in the following manner. Each gap interval $(\mathcal{F}_i, I_i) \in \mathcal{D}$ gives rise to a set of vertices in the graph: for each subtree $\mathcal{T}'$ in the tree covering $\mathcal{F}_i$, a vertex $(\mathcal{T}')_{I_i}$ is created. We say that the vertex $(\mathcal{T}')$ *lies in* the interval I_i. We observe that since no alignment columns have all gaps, the subtree $\mathcal{T}'$ of any vertex is a true subtree of the whole evolutionary tree $\mathcal{T}$ and so $\mathcal{T}'$ has a closest relative. As we shall see, a vertex $(\mathcal{T}')_{I_i}$ represents a *potential* indel: an indel occurring on the edge between the root of $\mathcal{T}'$ and its closest relative *could* be the cause of all the gaps in the sequences in $\mathcal{T}'$ in the gap interval I_i.

Consider a gap graph with two vertices $(\mathcal{T}_1)_{I_1}$ and $(\mathcal{T}_2)_{I_2}$ that lie in consecutive intervals I_1 and I_2. Then the subtree $\mathcal{T}_1$ has one of five possible relationships with $\mathcal{T}_2$:

- (*twin*) $\mathcal{T}_1 = \mathcal{T}_2$
- (*son*) $\mathcal{T}_1 \subset \mathcal{T}_2$
- (*father*) $\mathcal{T}_1 \supset \mathcal{T}_2$
- (*cousin*) $\mathcal{T}_1 \cap \mathcal{T}_2 \neq \emptyset$, but $\mathcal{T}_1 \not\subseteq \mathcal{T}_2$ and $\mathcal{T}_1 \not\supseteq \mathcal{T}_2$
- (*unrelated*) $\mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset$

These five relationships are manifested in the gap graph as follows:

- If $\mathcal{T}_1$ and $\mathcal{T}_2$ are twins the two vertices are *merged* into a vertex $(\mathcal{T}_1)_{I_1 \cup I_2}$ that lies in the interval $I_1 \cup I_2$ (note that this new interval is not a gap interval since it does not have the same gap configuration everywhere).
- If $\mathcal{T}_1$ is the son of $\mathcal{T}_2$ a *directed edge* is created from the vertex $(\mathcal{T}_1)_{I_1}$ to the vertex $(\mathcal{T}_2)_{I_2}$.

- If $\mathcal{T}_1$ is the father of $\mathcal{T}_2$ a *directed edge* is created from the vertex $(\mathcal{T}_2)_{I_2}$ to the vertex $(\mathcal{T}_1)_{I_1}$.
- If $\mathcal{T}_1$ and $\mathcal{T}_2$ are cousins, an *undirected zigzag edge* is created between the vertices $(\mathcal{T}_1)_{I_1}$ and $(\mathcal{T}_2)_{I_2}$.
- If $\mathcal{T}_1$ and $\mathcal{T}_2$ are unrelated, no edges are created.

Thus, to construct a gap graph from a gap division $\mathcal{D} = \{(\mathcal{F}_i, I_i)\}$, we first create a vertex for each subtree in each tree covering $\mathcal{F}_i$. Then we traverse the gap intervals I_i looking at two consecutive intervals at a time and merge any twin vertices, create directed edges between fathers and sons, and create undirected zigzag edges between cousins. Note that twin vertices may be extended to more than two gap intervals. We next proceed to a full example.

Figure 4 shows an alignment of five sequences. It has four gap intervals $I_1, \ldots I_4$, each with its own tree covering $\mathcal{F}_1, \ldots \mathcal{F}_4$. Each subtree in each tree covering induces a vertex in the gap graph, so initially we get the six vertices $\{2\}_{I_1}$, $\{2\}_{I_2}$, $\{4,5\}_{I_2}$, $\{1,2,3,4\}_{I_3}$, $\{1,2\}_{I_4}$, and $\{4\}_{I_4}$. Traversing the gap intervals, first we merge the two $\{2\}$-vertices since they lie in consecutive intervals, and then we create a directed edge from this new vertex to $\{1,2,3,4\}_{I_3}$ since $\{2\} \subset \{1,2,3,4\}$. Next we create an undirected zigzag edge between $\{4,5\}_{I_2}$ and $\{1,2,3,4\}_{I_3}$ since neither is contained in the other while they still share the leaf 4. Finally, we create directed edges going from $\{1,2\}_{I_4}$ and $\{4\}_{I_4}$ to $\{1,2,3,4\}_{I_3}$. For clarity we write the subtree of a gap graph vertex as either a variable in parentheses, like in $(\mathcal{T}')_I$, or a specific list of leaves with no parentheses, like in $\{1,2\}_I$.

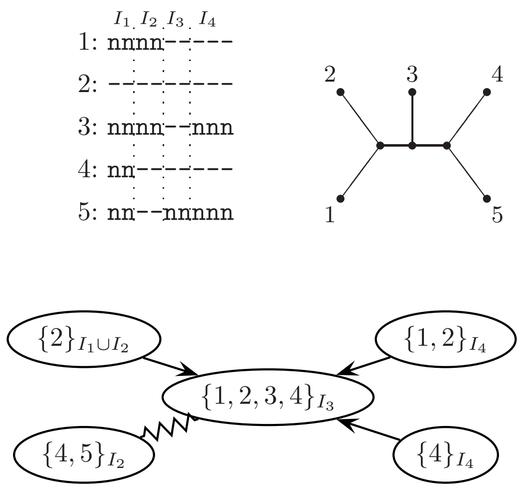


Fig. 4. Alignment with five sequences 1–5 and four gap intervals I_1–I_4, evolutionary tree, and induced gap graph. The tree coverings for the four gap intervals are $\mathcal{F}_1 = \{\{2\}\}$, $\mathcal{F}_2 = \{\{2\},\{4,5\}\}$, $\mathcal{F}_3 = \{\{1,2,3,4\}\}$, and $\mathcal{F}_4 = \{\{1,2\},\{4\}\}$.

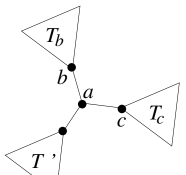
Our algorithm now works with the gap graph to find the most parsimonious set of indels that explains the gaps in the alignment, given a gap penalty function $g(l) = \alpha + \beta l$. We first go through a preprocessing phase in which we reduce the potentially very large and complex gap graph. In the second phase, we resolve the reduced graph to find the optimal solution.

3.2 Preprocessing the Gap Graph

As already said, an indel $\theta_{I,\mathcal{T}}$ results in gaps in all nodes of a subtree $\mathcal{T}$ in some (set of) interval(s) I of the alignment. The following theorem establishes a strong connection between indels and gap graph vertices. Before we can prove the theorem, we need a small lemma.

Lemma 1. *Given a subtree $\mathcal{T}'$ and a sequence S of gap graph vertices $(\mathcal{T}_k)_{I_k}$ such that $\forall k$, (1) $\mathcal{T}' \subset \mathcal{T}_k$, and (2) $\mathcal{T}_k \subseteq \mathcal{T}_{k+1}$ or $\mathcal{T}_k \supseteq \mathcal{T}_{k+1}$ (i.e., no neighbors are cousins). Then there exists a subtree $\mathcal{T} \supset \mathcal{T}'$ such that $\mathcal{T} \subseteq \mathcal{T}_k$ for all k.*

Proof. Since $\mathcal{T}'$ is a real subtree of the $\mathcal{T}_k$'s, $\mathcal{T}'$ has a closest relative a, and each $\mathcal{T}_k$ consists of at least $\mathcal{T} \cup a$. Now a cannot be a leaf since the $\mathcal{T}_k$'s are real subtrees of the whole evolutionary tree (if some $\mathcal{T}_w$ were the whole tree, all sequences would have gaps in the associated interval I_w). Thus, a has relatives b and c with associated (possibly empty) subtrees $\mathcal{T}_b$ and $\mathcal{T}_c$, and the evolutionary tree looks like the figure below. Consider $\mathcal{T}_1$, the first subtree in the sequence S.



Since it contains $\mathcal{T}'$ and a, it must also contain either $\mathcal{T}_b$ or $\mathcal{T}_c$, but not both. Assume without loss of generality that it contains $\mathcal{T}_b$. Then its neighbor in the sequence, $\mathcal{T}_2$ contains $\mathcal{T}_b$ as well. If not, $\mathcal{T}_1$ and $\mathcal{T}_2$ would be cousins: they have $\mathcal{T}'$ in common, $\mathcal{T}_1$ contains $\mathcal{T}_b$ (and not all of $\mathcal{T}_c$), but $\mathcal{T}_2$ would contain $\mathcal{T}_c$. This argument is easily extended by induction to show that *all* subtrees $\mathcal{T}_k$ contain $\mathcal{T}_b$. Let $\mathcal{T} = \mathcal{T}' \cup a \cup \mathcal{T}_b$. Then $\mathcal{T} \supset \mathcal{T}'$ and $\mathcal{T} \subseteq \mathcal{T}_k$ for all $\mathcal{T}_k \in S$. □

Theorem 1. *Let E be an optimal set of indels explaining an alignment $\mathcal{M}$ with respect to a gap cost function $g(l) = \alpha + \beta l$, and let $\mathcal{G}$ be the gap graph associated with $\mathcal{M}$. Let $\theta_{I,\mathcal{T}} \in E$ be an indel that creates gaps in the nodes of the subtree $\mathcal{T}$ in the interval I. Then*

1) $\exists$ a vertex $(\mathcal{T}')_{I'}$ in $\mathcal{G}$ with $I' \subseteq I$ such that $\mathcal{T}' = \mathcal{T}$, or
2) $\exists$ two cousins in I, $(\mathcal{T}_1)_{I_1}$ and $(\mathcal{T}_2)_{I_2}$ such that $\mathcal{T} \subseteq \mathcal{T}_1$ and $\mathcal{T} \subseteq \mathcal{T}_2$.

Thus, either the indel corresponds to (the subtree of) an exisiting vertex that lies somewhere in the indel's interval I, or it "crosses" a cousin edge in I.

Proof. Let $I = \bigcup I_k$ where the I_k are are disjoint gap intervals in the gap graph $\mathcal{G}$. All leaves of $\mathcal{T}$ have gaps in all of I, and so these leaves are covered in each gap interval I_k in $\mathcal{G}$. Thus, in each I_k there exists a vertex $(\mathcal{T}_k)_{I_k}$ such that $\mathcal{T} \subseteq \mathcal{T}_k$. Let S be the set of subtrees of these vertices. If two neighbor subtrees in S, $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$, are cousins, we are done. Otherwise, we have that $\mathcal{T}_k \subseteq \mathcal{T}_{k+1}$ or $\mathcal{T}_k \supseteq \mathcal{T}_{k+1}$ for all k. Assume that $\forall \mathcal{T}_k : \mathcal{T} \subset \mathcal{T}_k$.

Lemma 1 ensures the existence of a subtree $\mathcal{T}^*$ such that $\mathcal{T} \subset \mathcal{T}^* \subseteq \mathcal{T}_k$ for all $\mathcal{T}_k \in S$. Recalling that each $\mathcal{T}_k$ comes from a vertex $(\mathcal{T}_k)_{I_k}$ in the gap graph, all nodes/sequences in $\mathcal{T}^*$ must have gaps in I. By assumption, the optimal explanation E contains one indel $\theta_{I,\mathcal{T}}$ that accounts for the gaps in

$\mathcal{T}$ in interval I. The cost of this indel is $\alpha + \beta\ell$, if we let ℓ be the length of I. Further, some number of indels (say, $m > 0$) must account for the gaps in the nodes in $\mathcal{T}^* \setminus \mathcal{T}$ in interval I. Note that these m indels may stretch beyond I, so their total cost is $m\alpha + \beta(m\ell + d)$, where we sum the extra lengths beyond I in d. Thus, we can write the total cost g of the optimal explanation E as

$$g = (\alpha + \beta\ell) + m\alpha + \beta(m\ell + d) + q$$

where q is the cost of all indels accounting for other gaps. Consider the explanation E^* obtained from E by replacing $\theta_{I,\mathcal{T}}$ with the indel $\theta_{I,\mathcal{T}^*}$ which creates gaps in interval I in all nodes in the subtree $\mathcal{T}^*$. The cost of this new indel is still $\alpha + \beta\ell$, while we no longer have to let the m extended indels affect interval I. Thus we save $\beta m\ell$ on the cost, and the total cost g^* of the explanation E^* is

$$g^* = (\alpha + \beta\ell) + m\alpha + \beta d + q < g$$

But this contradicts the optimality of E. Therefore, $\mathcal{T}$ must be equal to at least one of the $\mathcal{T}_k$'s, and so the gap graph $\mathcal{G}$ must have a vertex $(\mathcal{T}_k)_{I_k}$ that corresponds to the indel $\theta_{I,\mathcal{T}}$. □

In general, given a vertex $(\mathcal{T}')_I$, the question is whether it corresponds directly to an indel happening on the edge between the root of $\mathcal{T}'$ and its closest relative, or whether the gaps of $\mathcal{T}'$ resulted from several indels happening inside $\mathcal{T}'$. In the latter case we say that the vertex is *decomposed*, otherwise it is *confirmed*. Theorem 1 tells us where to look when searching for the indels in the optimal explanation: Candidates are found among the (subtrees of the) gap graph vertices, and the *intersection between cousins*. In a (part of a) gap graph with no cousin edges all indels in the optimal explanation correspond directly to subtrees of existing vertices. If a cousin edge is present, however, the optimal explanation may contain indels not represented directly as vertices; such an indel must then correspond to a subtree which is contained in both of the cousins' subtrees and thus lies in their intersection.

Thus, Theorem 1 serves two causes. First it pins out cousin vertices as special cases, and second it justifies the perhaps intuitively reasonable notion that one should not place indels "lower" than necessary in the tree: if a vertex $\{1,2\}_I$ is not connected to leaf vertices $\{1\}_{I_j}$ and $\{2\}_{I_k}$ anywhere (and no cousins either), the corresponding gaps in sequences s_1 and s_2 do not both extend beyond I, and so there is no reason not to make the gaps in s_1 and s_2 in I in one go, i.e. on the edge between the root of the subtree $\{1,2\}$ and its closest relative. In fact we can say that if a vertex is decomposed, *all* of the decomposing indels continue and help explain other vertices in the gap graph. We formalize this in a lemma (which follows easily from Theorem 1 so we omit the proof) and elaborate it in a theorem.

Lemma 2. *If a gap graph vertex $(\mathcal{T})_I$ is decomposed in the optimal explanation of the corresponding alignment, then all of the indels explaining gaps in $\mathcal{T}$ extend beyond interval I.*

Theorem 2. *If a gap graph vertex $(\mathcal{T}')_I$ is decomposed in the optimal explanation of the corresponding alignment, the decomposing indels do not all extend from I in the same direction.*

Proof. Let $(\mathcal{T}')_I$ be decomposed into m indels. By Lemma 2, all decomposing indels extend beyond interval I. Assume they all extend in the same direction.

Let r be the root of $\mathcal{T}'$ and let a be the closest relative of $\mathcal{T}'$. Let ℓ be the length of I. Let $g(l) = \alpha + \beta l$ be the cost of an indel of length l, and let g^* be the cost of the optimal explanation E^* of all gaps in the alignment, (i.e. the explanation that minimizes the total cost over all indels). In this explanation, the gaps in the leaves of $\mathcal{T}'$ are explained by the m indels that extend beyond I. The total cost of these indels can be written as $m\alpha + \beta(m\ell + d)$ (each such indel has some length greater than ℓ and we sum these extra lengths in the constant d). We can then write

$$g^* = m\alpha + \beta(m\ell + d) + q$$

where q is the total cost of all other indels. Since $\mathcal{T}'$ is decomposed into at least two indels, $m \geq 2$.

These m indels all start in interval I and by assumption they extend in the same direction. If they all end in exactly the same column c_e in the alignment, we replace them with only one indel occurring between r and a and extending from I to c_e. Such an explanation would trivially be cheaper than g^*, contradicting the optimality of E^*. Thus, the m indels do not have equal lengths. Let c be the first alignment column where one of the m indels ends (call this indel θ_1). The column c is outside of I since θ_1 extends beyond I. Define I_x to be the extended interval that starts with and includes I and ends in column c; let its length be ℓ_x.

Consider the explanation E_1 obtained from E^* by replacing θ_1 with the indel $\theta_{I_x,\mathcal{T}'}$ occurring between r and a in the evolutionary tree and extending all through the interval I_x. This new indel explains the gaps in all $\mathcal{T}'$ in all of I_x, including I, while $m-1$ of the original indels (all except θ_1) still have to be made since they go beyond I_x; however, as we have covered all gaps in I_x with $\theta_{I_x,\mathcal{T}'}$ we save $\beta m\ell_x$ on the cost. Therefore, the cost of explanation E_1 is

$$\begin{aligned} g_1 &= (\alpha + \beta\ell_x) + (m-1)\alpha + \beta(m\ell + d) + q - \beta m\ell_x = \\ &\quad m\alpha + \beta(m\ell + d) + q - \beta(m\ell_x - \ell_x) \qquad = g^* - \beta\ell_x(m-1) \end{aligned}$$

Since $m \geq 2$ we have that $\beta\ell_x(m-1) > 0$ and thus $g_1 < g^*$. But this contradicts the optimality of E^* and so our assumption must be wrong. Therefore, the decomposing indels do not all extend in the same direction. □

Recall that a gap graph vertex represents gaps in a subtree of sequences in some region of an alignment. The vertex may have edges to other vertices, and these edges can have three different types (in-, out-, and cousin edges). It turns out that vertices with certain configurations of edges can immediately be confirmed to correspond to optimal indels. In other words certain types of vertices can be "decided" immediately and locally. Using the two strong Theorems 1 and 2 we can characterize formally exactly which types of vertices we can immediately decide. We will not go into the details here but simply give the results.[3]

A vertex is called a *leaf vertex* if its subtree consists of one leaf node only. Such a vertex can be confirmed right away and thus corresponds to an optimal indel occurring between this node and its closest relative in the tree. Likewise, *orphans* and *end vertices* can be confirmed: they are vertices with no edges and vertices with edges on one side only, respectively. All sequences in the subtree of an orphan have nucleotides on either side of its interval.

Consider a vertex $(\mathcal{T}')_I$ where *some but not all* of the leaves of its subtree $\mathcal{T}'$ have gaps in an adjacent interval. Such a vertex is called a *patriarch*. A patriarch has at least one edge (zigzag or in-going), but its subtree also has at least one leaf which is not included in a subtree in any vertex in an adjacent interval. We say that such a leaf has *solo gaps*: it has gaps in interval I but nucleotides on both sides. Patriarchs may also be confirmed immediately (using Lemma 2); intuitively, since it is necessary to spend an indel on the solo gaps, this indel may as well create all other gaps of the vertex' subtree also. A patriarch has another important property: after its confirmation, its edges are removed and new edges are created directly between its former neighbors on either side (if any are needed). A patriarch has no out-edges, and its subtree contains the subtrees of its in-edge neighbors; thus, by the same argument explained in connection with Figure 2, the intervals associated with the patriarch's neighbors are in fact consecutive and so the neighbors may be connected directly if they share leaves. A special case is when the patriarch has two cousins; in this situation the edges cannot be removed since we have to keep the information that no indel with a larger subtree than that of the patriarch can "pass" the patriarch and cause gaps on both of its sides in one go.

A few other reduction rules help reduce the gap graph further. The preprocessing phase makes use of these theoretical results by going over the gap graph in a series of passes, each pass reducing the graph by local applications of the reduction rules. Since the removal of edges following a patriarch confirmation may turn previously undecidable vertices decidable, several passes are performed.

3.3 Resolving the Reduced Gap Graph

Once the preprocessing is done, the gap graph has been reduced significantly (as illustrated by Figure 6 below). In other words, most of the gaps have been (optimally) explained. We now turn to the *chains* of the gap graph: two vertices belong to the same chain if and only if there exists a path connecting the two which does not cross a leaf vertex. Thus, chains often end in leaf vertices, some of which may belong to two chains (see Figure 5). The chains are important

[3] Proofs are found at `http://www.daimi.au.dk/~chili/BiRC/gapgraphAppendix.pdf`.

since they can be dealt with independently: indels causing gaps in the vertices of one chain could not have caused gaps in vertices of another chain.

Say that after preprocessing, (a part of) the gap graph looks like Figure 5. In the second phase of our algorithm, we analyze each chain in turn. Looking first at chain a, we see that only the vertex (123) is still undecided (the others being leaf vertices). Referring to Theorem 1, we note that since there are no cousin edges in this chain, the only indels we need to consider for the optimal explanation are the ones already represented by vertices in the chain. That means there are only two possible explanations for the gaps represented by the vertex (123): 1) Either they are the result of one indel occurring between the root of the subtree $\{1, 2, 3\}$ and its closest relative (in which case we confirm the vertex), or 2) they are the result of three indels occurring between 2 and its closest relative, 1 and its closest relative, and 3 and its closest relative, respectively, extending the already confirmed neighboring indels/vertices (in which case we decompose the vertex). If we let l_{123} be the length of its associated alignment interval, the extra cost for resolving the vertex (123) in each of these two explanations is $\alpha + \beta l$ versus $3\beta l$. For the optimal explanation we simply choose the cheaper one (in general, if several options are optimal, we choose one arbitrarily).

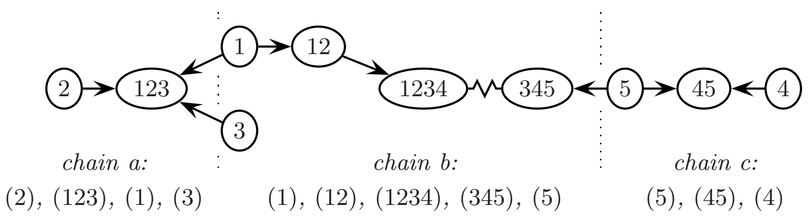


Fig. 5. Gap graph with three chains (same tree as in Figure 4). We omit interval labels on the vertices, and we write, e.g., (2345) to denote the subtree $\{2, 3, 4, 5\}$.

Chain b has three undecided vertices: (12), (1234), and (345). Moreover, the latter two are cousins and so by Theorem 1 we need to consider indels whose subtrees lie in their intersection. That gives two additional potential indels not already represented in the chain by vertices, namely $\{3\}$ and $\{4\}$. Interestingly, the vertex (12) can therefore be confirmed right away: its only possible decomposition is in subtrees $\{1\}$ and $\{2\}$, but since $\{2\}$ is neither present as a vertex nor lies in the cousin intersection, this indel could never be part of an optimal explanation. For the same reason, the vertex (1234) may either be confirmed or decomposed in one way only, namely with three indels with the subtrees $\{1, 2\}$, $\{3\}$, and $\{4\}$. And finally the vertex (345) may either be confirmed or decomposed in three indels with the subtrees $\{3\}$, $\{4\}$, and $\{5\}$ (this time, Theorem 1 dictates that the indel $\{4, 5\}$ is not an option in chain b). In total therefore we have four combinations which we need to check in this chain. Lastly, in chain c we may either confirm the vertex (45) or decompose it in $\{4\}$ and $\{5\}$.

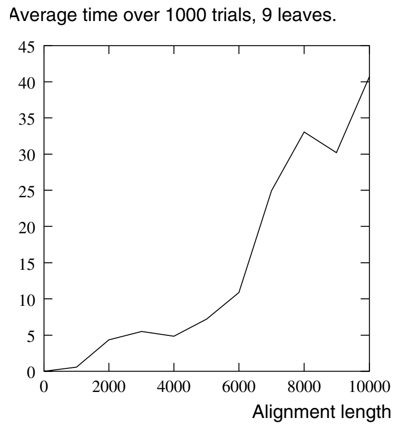
Checking all combinations we find the optimal indels. Each causes gaps in some set of columns in a full subtree, and by the R assumption (see Section 3) we place anonymous nucleotides in the internal nodes in the rest of the tree in the same columns. It remains to optimally name these nucleotides. For each column we ignore the subtrees that have all gaps and get a still connected tree with all nucleotides, and on this tree we now perform the Finch-Hartigan-Sankoff algorithm as a final step to optimally assign nucleotides to the internal nodes.

4 Performance and Future Work

Checking the cost of each combination of confirmation/decomposition of all the vertices in a chain cannot be done in constant time. E.g., looking again at Figure 5 and the combination in chain b of decomposing both (1234) and (345) we have to look at both decompositions to see that the decomposing indels $\{3\}$ and $\{4\}$ can extend to both vertices and do not have to be opened anew. Note also that since we have to check "all with all", the number of combinations to check is exponential in the number of undecided vertices in the chain. I.e., it is critical that as many edges as possible are removed in the preprocessing phase. If the tree is large, some vertices may have many possible decompositions; if the alignment has many "crisscrossing" gaps, the chains may become very long. Encountering, e.g., a chain with 11 undecided vertices each with five possible decompositions (including not decomposing) except two which have 10 decompositions, we have in total $5^9 \cdot 10^2 = 195312500$, close to 200 million, combinations to check (each taking some hard to characterize but more than constant amount of time) for this chain alone (see Figure 6). Thus, the worstcase time complexity is exponential in the length of the alignment.

On the other hand, if the gaps mostly fall in straight blocks delimited by full columns of nucleotides, chains will be short and the running time will be very fast. Thus, the running time of our algorithm is extremely dependent on the particular problem instance. To demonstrate this, we have tested the algorithm on a huge number of different alignments and phylogenies. The main type of data was randomly generated alignments with some ratio of gaps scattered individually across all sequences. We wanted to challenge our algorithm with very hard data, and a long such alignment with a high gap ratio of 60% is indeed a very hard problem to solve. Real alignments do not look like this (see Figure 7 for a screenshot of our program running a 60% gap alignment): first they do not have that many gaps unless the aligned sequences are only very distantly related. And second, since gaps appear as the result of indels, they do appear in connected blocks and not as much as independent single gaps. For this reason we also did some experiments on alignments with only 40% and 50% gaps, and finally we ran the algorithm on an alignment of nine full HIV-1 subtype genomes.

In the left plot of Figure 6 we show the average performance for random alignments with 60% gaps over different alignment lengths with 1000 trials for each, on a tree with nine leaves. For an alignment of length 10000 the average time was around 40 seconds. For lengths below 6000, the time is less than 10



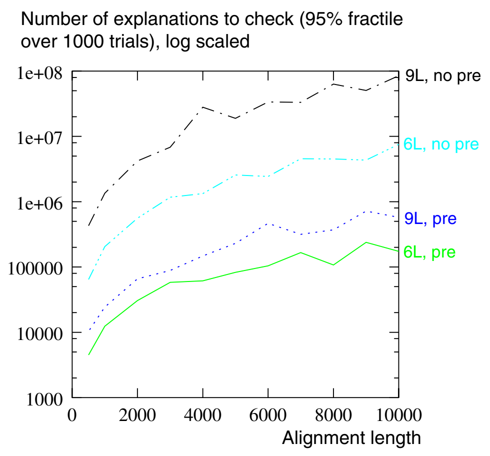


Fig. 6. Performance on random alignments with 60% gaps. Left plot: average times for different lengths, on tree with nine leaves. Right plot: 95% fractiles of the number of combinations to check for different alignment lengths, on trees with six and nine leaves (6L/9L) with and without preprocessing (pre/no pre). The curve '9L, pre' comes from an experiment similar to the one underlying the left plot.

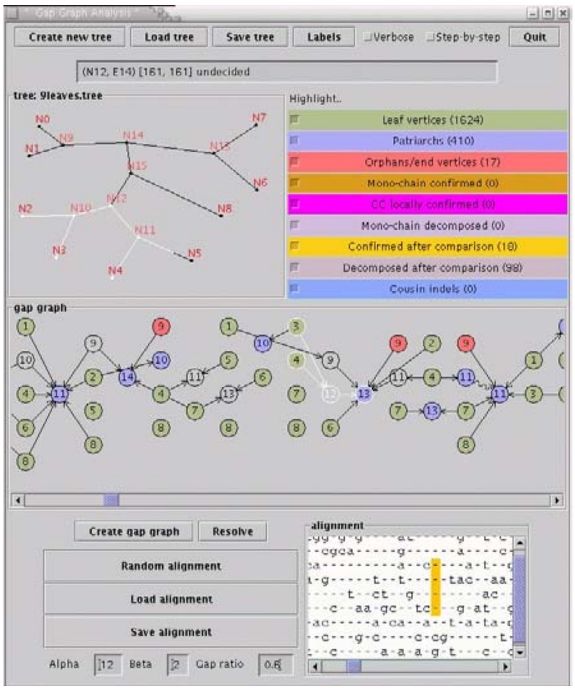
seconds.[4] The curve does not appear smooth; this is due to the abovementioned dependency on the particular problem instance resulting in a huge variance.
The right plot of Figure 6 is an attempt to show the effect of the preprocessing phase. Since it would have been too time consuming to actually find the optimal explanation in 1000 trials with no preprocessing, we instead report the number of combinations it would have been necessary to check. Again, the variance is huge so we report the 95% fractiles (i.e., a value of 10000 means that in 95% of the trials, the number of explanations to check was at most 10000). We did experiments with and without preprocessing, and to also get an idea of the performance for different tree sizes we used both a tree with nine leaves and one with six leaves. For example, for alignment length 10000 and a tree with nine leaves, the numbers are about 500.000 with preprocessing versus about 100 million without preprocessing; i.e. preprocessing reduces the number of combinations to check by a factor of about 200 in this case. We also observe that for the smaller tree, the number of combinations to check is substantially smaller.

This performance is contrasted by our results for a tree with nine leaves and alignments with only 40% or 50% gaps: for all lengths up to 8000, the average running time was less than one second. We also analyzed a full genome alignment of B.ES.89.S61K15, B.FR.83.HXB2, B.GA.88.OYI, B.GB.83.CAM1, B.NL.86.3202A21, B.TW.94.TWCYS, B.US.86.AD87, B.US.84.NY5CG, and B.US.83.SF2, nine HIV-1 subtypes retrieved from the Los Alamos HIV database (`http://hiv-web.lanl.gov/content/index`). After constructing a phylogeny using the neighbor-joining Sanger Institute Quicktree software (`http://www.sanger.ac.uk/Software/analysis/quicktree/`) we had a tree with nine leaves

[4] All experiments were done on a 2.4 GHz Pentium 4 machine with 512 MB RAM running Linux.

and an alignment of length 9868 (this example only serves to demonstrate our algorithm, the methods chosen to obtain phylogeny and alignment were arbitrary). Solving this problem again took less than one second. Because of the great similarity of the sequences, the resulting gap graph was small and simple.

We are interested in running our program on more, real data sets with alignments and trees of various sizes. Much larger trees with shorter alignments (100–500 leaves, length < 500) is an interesting application which we have not yet looked at. Also, we intend to combine the algorithm with some heuristic search method so as to *find* an approximately optimal alignment which fits the tree, rather than assume it is given (a variant of the tree alignment problem). One might consider our algorithm a way of ranking a given combination of tree and alignment using a gap cost function and a substitution matrix, and so trying different alignments, or even trees, might actually improve the data.



(N12, E14) [161, 161] undecided

Highlight..
- Leaf vertices (1624)
- Patriarchs (410)
- Orphans/end vertices (17)
- Mono-chain confirmed (0)
- CC locally confirmed (0)
- Mono-chain decomposed (0)
- Confirmed after comparison (18)
- Decomposed after comparison (98)
- Cousin indels (0)

Alpha 12 Beta 2 Gap ratio 0.6

Fig. 7. Screenshot with phylogeny, alignment, and induced gap graph. Note what a random alignment with 60% gap looks like.

References

1. W. M. Fitch: Towards defining the course of evolution: minimum change for a specific tree topology. Systematic Zoology 20.406–416, 1971.
2. D. Sankoff: Matching sequences under deletion/insertion constraints. Proc. Natl. Acad. Sci. USA 69.4–6, 1972.
3. J. A. Hartigan: Miminum mutation fits to a given tree. Biometrics 20.53–65, 1973.
4. P. Sellers: An algorithm for the distance between two finite sequences. J. Comb. Theory 16.253–258, 1974.
5. D. Sankoff: Minimal mutation trees of sequences. SIAM J. appl. Math 78.35–42, 1975.
6. M. S. Waterman, T. F. Smith, and W. A. Beyer: Some biological sequence metrics. Advances in Mathematics 20.367–387, 1976.
7. O. Gotoh: An improved algorithm for matching biological sequences. J. Mol. Biol. 162.705–708, 1981.

8. M. L. Fredman: Algorithms for computing evolutionary similarity measures with length independent gap penalties. Bull. Math. Biol. 46.4.545–563, 1984.
9. J. J. Hein: A method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. Mol.Biol.Evol. 6.6.649–668, 1989.
10. L. Wang, T. Jiang, and E. L. Lawler: Approximation Algorithms for Tree Alignment with a Given Phylogeny. Algorithmica 16:302–315, 1996.
11. L. Wang and D. Gusfield: Improved Approximation Algorithms for Tree Alignment. CPM 1996: 220–233.
12. J. Stoye: Multiple sequence alignment with the divide-and-conquer method. Gene 211 (2), GC45–GC56, 1998.
13. E. Althaus, A. Caprara, H.-P. Lenhof, and K.Reinert: Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. ECCB 2002: 4–16.