Journal of Bioinformatics and Computational Biology

ON THE INFERENCE OF PARSIMONIOUS INDEL EVOLUTIONARY SCENARIOS

LEONID CHINDELEVITCH, ZHENTAO LI

School of Computer Science, McGill University,
3480 University Street, Montreal, Quebec, H3A 2A7, Canada
lchind@po-box.mcgill.ca, zhentao.li@mail.mcgill.ca

ERIC BLAIS, MATHIEU BLANCHETTE*

McGill Centre for Bioinformatics and School of Computer Science, McGill University
3775 University Street, Montreal, Quebec, H3A 2B4, Canada
eblais@mcb.mcgill.ca, blanchem@mcb.mcgill.ca

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Given a multiple alignment of orthologous DNA sequences and a phylogenetic tree for these sequences, we investigate the problem of reconstructing a most parsimonious scenario of insertions and deletions capable of explaining the gaps observed in the alignment. This problem, called the Indel Parsimony Problem, is a crucial component of the problem of ancestral genome reconstruction, and its solution provides valuable information to many genome functional annotation approaches. We first show that the problem is NP-complete. Second, we provide an algorithm, based on the fractional relaxation of an integer linear programming formulation. The algorithm is fast in practice, and the solutions it produces are, in most cases, provably optimal. We describe a divide-and-conquer approach that makes it possible to solve very large instances on a simple desktop machine, while retaining guaranteed optimality. Our algorithms are tested and shown efficient and accurate on a set of 1.8 Mb mammalian orthologous sequences in the *CFTR* region.

Keywords: Insertions and deletions; Ancestral DNA sequences; Sequence evolution; Integer Linear Programming.

1. Background and Motivation

The large number of genomes sequenced or in the process of being sequenced has shown to be an extremely valuable source of information for studying the evolution of various species. An exciting prospect was raised recently by Blanchette et al. [3]: given the genomes of sufficiently many extant species, it may be possible to reconstruct to a high degree of accuracy the genomes of some ancestral species having lived tens of millions of years ago. The ancestral genome reconstruction procedure

*Corresponding author

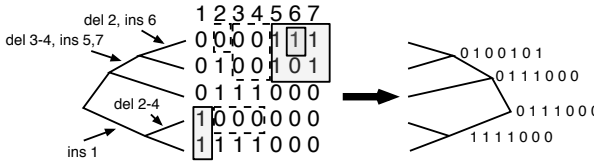

Fig. 1. Example of an input to the Indel Parsimony Problem, together with one optimal solution (left) and the induced ancestral sequences (right). A 1 indicates that a nucleotide was present at a given position of a given species, while a 0 denotes the absence of a nucleotide (i.e. a gap). The effect of deletions is shown by dashed boxes, while the effect of insertions is shown by shaded boxes. The optimal scenario involves 3 deletions and 3 insertions. Notice that the insertion of positions 5 and 7 should be counted as a single event, because the base at position 6 does not yet exist at the time the insertion happens.

involves several difficult steps, including the identification of orthologous regions in different extant species[20,22], ordering of the syntenic blocks[5], multiple alignment of the orthologous sequences within each syntenic block[4], and reconstruction of the ancestral sequences for each aligned block[3]. In this paper, we focus on one crucial and poorly understood aspect of the ancestral sequence reconstruction that we call the Indel Parsimony Problem. It consists of inferring the most parsimonious set of insertions and deletions, performed along the branches of a *given* phylogenetic tree, that may have led to the gaps observed in a *given* multiple alignment. Figure 1 provides an example of a possible input to the Indel Parsimony Problem, together with an optimal (most parsimonious) solution. Notice that since indels often affect several consecutive nucleotides, alignment columns cannot be treated independently, as opposed to the parsimony problem for substitutions[9]. A solution to the Indel Parsimony Problem directly translates into a prediction of the presence or absence of a base at a given position of the ancestral sequence at an internal node of the tree (see Figure 1). The standard substitution-based maximum-likelihood approach[8] could then be used to decide which of the four possible nucleotides (A, C, G, or T) is most likely at a given position (for more information on this process, see Blanchette et al.[3]). Notice, however, that indel evolutionary scenarios are not only useful for predicting ancestral sequences, but also for annotating functional regions of extant genomes, including protein-coding regions[21], RNA genes[18], and other functional regions[14].

In this paper, we start by giving a formal definition of the Indel Parsimony Problem. We then prove the NP-completeness of the problem. After giving polynomial-time algorithms for two special cases of the problem, we show how to encode a general instance of the Indel Parsimony Problem as a 0-1 Integer Linear Programming (ILP) problem. Although 0-1 ILP is also NP-complete[11], we describe a fractional relaxation approach that provides fast and provably optimal solutions in almost all the cases, based on our empirical results. We then provide a divide-and-conquer algorithm for breaking the original ILP problem into a set of independent subprob-

lems, whose solution can be combined to obtain an optimal solution to the original problem. This is of practical importance, as it allows very large problems to be solved on a simple desktop computer and allows easy parallelization. Finally, our algorithms are tested on two sets of actual biological sequences, consisting respectively of 9 and 20 orthologous mammalian sequences for a 1.8Mb region around the human *CFTR* gene[23].

2. Related work

The Indel Parsimony Problem has received surprisingly little attention, and its complexity remained unknown until now. The only authors to attack the problem head on are Fredslund *et al.*[10], who provide an elegant graph-based algorithm to resolve most cases using a set of relatively simple rules. However, some cases cannot be resolved based on these rules and the algorithm then relies on an exhaustive enumeration to solve the difficult parts of the problem, which results in a worst case running time that is exponential in the length of the alignment. Still, it performs well in practice, provided that the number and length of the sequences in the alignment are not too large.

The problem of inferring ancestral sequences has in fact more often been considered as a part of the multiple alignment problem. For example, Hein[13] described an algorithm for simultaneous alignment and ancestral sequence inference, which, though quite accurate, remains much too slow for large scale applications. The other line of work that is relevant to ancestral genome inference is the recent development of statistical alignment procedures (see for example Lunter et al.[15]), which have the advantage of providing confidence estimates for the predictions made. Unfortunately, here again, the current methods are too slow for reconstructing more than a few kilobases.

3. Problem definition

We start by giving a precise definition of the problem under study. Consider a rooted phylogenetic tree $T = (V_T, E_T)$ with its n leaves labeled with DNA sequences. Consider a multiple alignment $\mathcal{A}$ of these n orthologous sequences, and let L be the number of columns in $\mathcal{A}$. Since the only evolutionary events of interest here are insertions and deletions, $\mathcal{A}$ can be reduced to a binary matrix, where gaps are replaced by 0's and nucleotides by 1's. Let A_u be the row of the binarized alignment corresponding to the sequence at leaf u of T, and let $A_u[i]$ be the binary character at the i-th position of A_u. For convenience, we add two extra columns, $A[0]$ and $A[L+1]$, made exclusively of 1's.

3.1. *Basic definitions*

Definition 1 (Phylogenetic correctness). An alignment $\mathcal{A}$ is *phylogenetically correct* if it aligns together exactly the nucleotides that come from a common

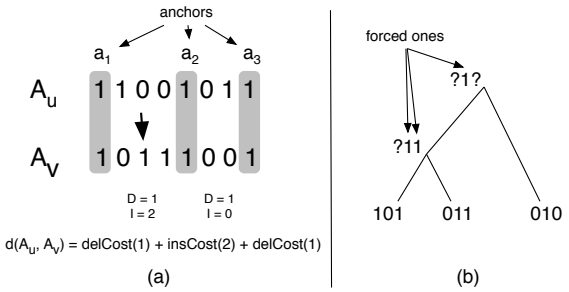

Fig. 2. Examples of anchors and forced 1's.

ancestor (i.e. all aligned nucleotides are derived from a common ancestral nucleotide through direct inheritance or substitution).

Note that without extensive fossil record, it is impossible to guarantee that an alignment is phylogenetically correct. Nonetheless, good heuristics have been developed to compute accurate multiple alignments for large genomic regions[4,7,6], and, for the purposes of this paper, we will assume that we are working on correct alignments. The effect of alignment errors on the accuracy of the ancestral reconstruction has been studied by Blanchette et al. [3].

A phylogenetically correct alignment tells us a lot about the ancestral states at certain positions of the alignment. More precisely, for any $i = 1...L$ and any $u, v \in$ leaves(T), if $A_u[i] = 1$ and $A_v[i] = 1$, then for any w on the path from u to v in T, $A_w[i] = 1$.

We define the indel distance $d(A_u, A_v)$ between two alignment rows A_u and A_v, where u is the parent of v in T, as a function of the number and lengths of insertions and deletions required to transform A_u into A_v. The definition below turns out to be equivalent but it is more amenable to the algorithms. Refer to Figure 2 (a) for an illustration of these concepts.

Let x and y be two binary strings of equal length. We need the following definitions:

Definition 2 (Anchors, deletions, and insertions).

- The set of *anchors* for x and y is the set of positions where both strings have a 1:
  $anchors(x, y) = \{i | x[i] = y[i] = 1\}$.
- The number of characters deleted from x to y is denoted by
  $D(x, y) = |\{k|\ x[k] = 1, y[k] = 0\}|$.
- The number of characters inserted from x to y is denoted by
  $I(x, y) = |\{k|\ x[k] = 0, y[k] = 1\}|$.

Definition 3 (Indel distance). Let A_u and A_v be two alignment rows with

$anchors(A_u, A_v) = \{a_1, ..., a_q\}$. The *indel distance from* A_u *to* A_v is given by

$$d(A_u, A_v) = \sum_{i=0}^{q-1} (insCost(I(A_u[a_i...a_{i+1}], A_v[a_i...a_{i+1}])) + delCost(D(A_u[a_i...a_{i+1}], A_v[a_i...a_{i+1}])))$$

The costs of insertions and deletions can be defined under different models. The most general model we consider here is the edge-dependent, asymmetric affine-gap penalty, given by $insCost(l) = \alpha_e \cdot l + \beta_e$ and $delCost(l) = \delta_e \cdot l + \gamma_e$, for non-negative $\alpha_e, \beta_e, \delta_e, \gamma_e$, where the parameters depend on the branch $e \in E_T$ along which the indels take place. We also consider the simpler *unit-cost* model, where $\alpha_e = \delta_e = 0, \beta_e = \gamma_e = 1 \, \forall e \in E_T$ (i.e. all indels have cost 1, no matter their length or the branch along which they occur). The algorithms described in this paper apply to the general edge-dependent, asymmetric affine-gap model, unless stated otherwise. It should be noted that $I(A_u[a_i...a_{i+1}]]$ and $D(A_u[a_i...a_{i+1}]]$ depend only on the number of inserted and deleted positions between anchors a_i and a_{i+1}, and not on their arrangement. Indeed, the arrangement of the columns between anchors is arbitrary, since they contain no orthologous positions. Thus, it is always possible to go from $A_u[a_i...a_{i+1}]$ to $A_v[a_i...a_{i+1}]$ using at most one (multi-base) deletion and one (multi-base) insertion.

Definition 4 (Admissible ancestors). Given an internal node u of the tree T, we say that a predicted ancestral alignment row A_u is *admissible* if, for any $i \in \{1...L\}$ and for any leaves v, w such that $A_v[i] = A_w[i] = 1$ and such that u is on the path between v and w in T, we have $A_u[i] = 1$. This an immediate consequence of the phylogenetic correctess of $\mathcal{A}$. We say that $A_u[i]$ is *forced to one* (see Figure 2 (b) for an example).

3.2. *Metric properties of the indel distance*

Clearly, the indel distance function d is not a metric on the set of all strings of length L, because it fails to satisfy the triangle inequality (indeed, under the unit-cost model, the distance from any string to the string 0^L is one, but the distance between two non-zero strings can be up to $\lceil 2L/3 \rceil$). However, it satisfies the first two axioms of a metric: the Hausdorff principle and symmetry, and under the conditions below, it also satisfies the triangle inequality.

Theorem 3.1. *If we consider only triplets of tree nodes* $u, v, w \in V_T$ *where* v *is located on the path between* u *and* w*, then for any phylogenetically correct triplets of sequences* A_u, A_v, A_w*, we have that* $d(A_u, A_v) + d(A_v, A_w) \geq d(A_u, A_w)$.

Proof. The phylogenetic correctness of the alignment guarantees that $anchors(A_u, A_w) \subseteq anchors(A_u, A_v) \cap anchors(A_v, A_w)$. It follows that $insCost(A_u, A_v) + insCost(A_v, A_w) \geq insCost(A_u, A_w)$ and $delCost(A_u, A_v) + delCost(A_v, A_w) \geq delCost(A_u, A_w)$ because any mismatch between A_u and A_w is necessarily present as a mismatch of the *same* type between either A_u and A_v or between A_v and A_w. Hence, $d(A_u, A_v) + d(A_v, A_w) \geq d(A_u, A_w)$. □

3.3. *The Indel Parsimony Problem*

We are now ready to define the problem discussed in this paper.

INDEL PARSIMONY PROBLEM (IPP)

Given: An alignment $\mathcal{A}$ and a rooted phylogenetic tree T with the leaves of T labeled with the rows of $\mathcal{A}$.

Find: An admissible extension of $\mathcal{A}$ to the internal nodes of T such that $cost(A) = \sum_{(u,v)\in E_T} d(A_u, A_v)$ is minimized.

Notice that if we did not enforce this admissibility rule, most instances would have for optimal solution the trivial solution where all ancestral positions are set to zero, which usually would yield a score of n (under the unit-cost model), no matter the size of L.

4. Computational complexity of the Indel Parsimony Problem

We first settle the complexity of the IPP by proving that it is NP-Complete on phylogenetic trees of unbounded degree (we use a star tree with one ancestor and n leaves). We first reduce the 3SAT problem to an Independent Set Problem on a special type of interval graphs. We then show how to reduce this Independent Set Problem to the Indel Parsimony Problem.

Theorem 4.1. *The INDEL PARSIMONY PROBLEM on trees with unbounded degree is NP-complete, under any of the cost models discussed in this paper.*

Proof. See Appendix A. □

The complexity of the problem on phylogenetic trees with bounded degree and unit-cost indels remains unknown. However, if the cost of indels is allowed to depend on the branch along which they occur, the problem is NP-complete even on binary trees, as it is trivial to set the cost of indels on internal branches to be so high that the tree effectively becomes a star tree.

5. Polynomial-time solutions to special cases

Interestingly, while minimizing the total number of insertions and deletions is an NP-complete problem, the special cases where only deletions are allowed (at unit cost), and the special case where only insertions are allowed (with any cost function), are both solvable in polynomial-time, as described in the rest of this section.

5.1. *Deletion-only case*

We first describe a simple greedy algorithm that is guaranteed to find the optimal solution to a simplified version of the IPP in which *only deletions* are allowed (i.e. all gaps observed in the alignment are assumed to be due to deletions), at unit cost. Although this is not a biologically realistic situation, it does provide some insights

into the structure of the problem. Deletions performed on internal branches of the tree result in gaps in all the leaves of the affected subtree, so this problem is a variant of rectangle covering: one wants to find the minimal number of rectangles required to cover the gaps observed in the alignment. However, here, only the rectangles corresponding to the leaves of a subtree of T are allowed. Contrary to many similar 0-1 rectangle covering problems [17], this version has a polynomial-time solution.

Definition 5.

- A *deletion* is a pair $(v, [j, k])$, where v is a node of T, $[j, k]$ is an interval with $1 \leq j \leq k \leq L$, such that $A_w[l] = 0 \ \forall w \in leaves(subtree(v))$ and $j \leq l \leq k$.
- A deletion $(v, [j, k])$ *covers* position i of node u if u is a descendant of v (or v itself), and $j \leq i \leq k$.
- A deletion $(v, [j, k])$ *contains* a deletion $(w, [l, m])$ if w is a descendant of v (or v itself) and $[l, m] \subseteq [j, k]$.
- A *maximal* deletion is one that is not contained in any other deletion.

The problem to be solved is thus to find a smallest set $\mathcal{D}$ of deletions such that every gap in the alignment is covered by at least one deletion. Notice that a given gap at a leaf may be covered by more than one deletion. Since, in our current formulation, all deletions have unit-cost, there is no harm in always making deletions as large as possible. More precisely, if an optimal solution contains deletion $(v, [j, k])$, then replacing this deletion with a larger deletion that contains it will also provide an optimal solution. Thus, there exists an optimal solution that only employs maximal deletions. However, since a given gap may in general be covered by more than one maximal deletion, it is not obvious which one should be chosen. The following theorem resolves this uncertainty.

Theorem 5.1. *For every leaf u and position i, if $A_u[i] = 0$, then there exists a leaf v and a position j such that $A_v[j] = 0$ and such that $A_v[j]$ is covered by a unique maximal deletion that also covers $A_u[i]$.*

Proof. Let S be the set of all maximal deletions covering the gap at $A_u[i]$. Let $M = (w, [k, l])$ be an element of S such that $l-k$ is maximized (i.e. M has maximum possible length). We claim that M is the unique maximal deletion for some gap $A_v[j]$ covered by it. Notice that the choice of M implies that $A_u[k-1] = 1 = A_u[l+1]$, since otherwise, we would have been able to extend M either to the left or to the right. If w is the root of T, the claim above is trivially true. Otherwise, let p be the parent of w in T. By maximality of M, we know that $(p, [k, l])$ is not a valid deletion. Hence, there exists a position m, with $k \leq m \leq l$, and a $q \in leaves(subtree(p))$, such that $A_q[m] = 1$. Consider the position $A_u[m]$; by assumption, $A_u[m] = 0$. By the choice of M, any maximal deletion containing $A_u[m]$ is contained in M, because it cannot be extended to the left or to the right, and it cannot be extended

to the parent p since $A_q[m] = 1$. Thus, M is the unique maximal deletion containing $A_u[m]$, proving the claim and the theorem. □

The proof above actually shows that we can create our minimal set of deletions by taking, for each maximal set of consecutive gaps at a node, the deletion containing it and rooted at the node of largest possible height (where the root of the deletion $(v, [i, j])$ is v). This motivates a bottom-up algorithm for finding one such minimal set of deletions:

Algorithm: DeletionParsimony(Alignment A, tree T)
Initialization:
For every leaf u, set $S_u = \{(i, j) | A_u(k) = 0 \ \forall i \leq k \leq j, A_u(i-1) = 1 = A_u(j+1)\}$
Recursion: For each internal node $u \in V_T$, in a post-order traversal, do
Let v and w be the two children of u
Set $R_v = \{(i, j) \in S_v \mid \exists (k, l) \in S_w \text{ with } (i, j) \subseteq (k, l)\}$
Set $R_w = \{(i, j) \in S_w \mid \exists (k, l) \in S_v \text{ with } (i, j) \subseteq (k, l)\}$
Set $S_u = (S_v - R_v) \cup (S_w - R_w)$
Termination:
Return $\mathcal{D} = \bigcup_{u \in E_T} \bigcup_{(i,j) \in S_u - R_u} \{(u, [i, j])\}$.

5.2. *Insertion-only case*

The flip-side of the deletion-only case, called the insertion-only case, can also easily be solved in polynomial time. Notice first that not all phylogenetically correct alignments can be explained by an insertion-only scenario, but only those for which, for any column i, the set of leaves where $A_u[i] = 1$ forms an exact subtree of T. In that case, by our assumption of phylogenetic correctness, for any column i and any node u in the subtree rooted at the least common ancestor of all 1's in column i, we will have $A_u[i] = 1$, and all others nodes will have value zero. Thus, there is a unique admissible solution, which can be trivially identified, and whose score is easily computed in time $\Theta(n \cdot L)$.

6. Integer Linear Programming Formulation

Linear programming (LP) is the problem of optimizing a linear function of a set of variables, subject to a set of linear constraints on these variables. When the variables are free to take arbitrary real values, the problem can be solved in polynomial time using the so-called ellipsoid method [12]. When the variables are constrained to take only integer values (ILP), the problem is in general NP-hard [11]. The special case where the variables are restricted to take only values in $\{0, 1\}$, known as 0-1 ILP, is also NP-hard [11]. However, ILP is a very active area of research and efficient heuristics have been developed [19]. Here, we describe an encoding of an instance of the IPP as an instance of the 0-1 ILP problem. Our formulation is suitable for the

most general cost model studied here, where insertions and deletions are considered simultaneously, under the affine cost, edge-dependent, asymmetric model.

We start by defining the set of variables we are going to use, then describe the constraints on these variables, and conclude with the objective function. All the variables have binary 0-1 values.

- For every internal node u and for every $i \in \{0...L+1\}$, define $X_u[i]$ to be the character (0 or 1) at position i of the ancestor at node u. A truth assignment to the variables X constitutes a possible solution to the ancestral reconstruction problem. For simplicity of notation, we also define the X variables at the leaves of the tree, in which case $X_u[i] = A_u[i]$.
- For every edge $e = (u, v) \in E_T$ and for every $i \in \{0...L+1\}$, define, for $a, b \in \{0, 1\}$, $U_e^{ab}[i]$ to be 1 if and only if $X_u[i] = a$ and $X_v[i] = b$ (in which case we say that the edge (u, v) at column i has type ab). The U variables are used to keep track of changes between neighboring nodes.
- For every edge $e = (u, v) \in E_T$ and for every $i \in \{0...L+1\}$, define $V_e^{01}[i]$ to be 1 if and only if there exists $j \leq i$ such that $X_u[j] = 0$, $X_v[j] = 1$ and for all $j < k \leq i$, $U_e^{11}[k] = 0$. The variables V^{01} are used to keep track of insertions.
- Similarly, for every edge $e = (u, v) \in E_T$ and for every $i \in \{0...L+1\}$, define $V_e^{10}[i]$ to be 1 if and only if there exists $j \leq i$ such that $X_u[j] = 1$ and $X_v[j] = 0$ and for all $j < k \leq i$, $U_e^{11}[k] = 0$. The variables V^{10} are used to keep track of deletions.
- For every edge $e = (u, v) \in E_T$ and for every $i \in \{0...L+1\}$, define $W_e^{01}[i]$ as follows. Let $j \leq i$ be the maximal index for which $U_e^{11}[j] = 1$. Then, $W_e^{01}[i] = 1$ if and only if $U_e^{01}[i] = 1$ and for all $j < k < i$, $U_e^{01}[k] = 0$. Informally, $W^{01}[i] = 1$ means that position i is the first column of type (01) encountered since the last column of type (11). Define $W_e^{10}[i]$ analogously.

Thus, the complete set of variables contains: $(n-1) \cdot (L+2)$ X variables (excluding those at the leaves, whose values are fixed), $3(2n-2) \cdot (L+2)$ U variables (the U^{00} variables are never used), and $2(2n-2) \cdot (L+2)$ V and W variables, for a total of $15 \cdot (n-1) \cdot (L+2)$ variables, all of which are binary. The number of variables in the ILP is thus linear in the size of the alignment.

Then, the linear constraints in Table 1 enforce the consistency of the set of variables. A total of 23 constraints are required for each edge and each column of the alignment, so we get $23 \cdot (2n-2) \cdot (L+2)$ constraints in total. Notice that all the constraints in Table 1 are in fact in 3-CNF form, a fact that can be exploited by certain 0-1 ILP solvers like PBS[1]. The constraints of admissibility of the solution are not listed in Table 1, but the variables $X_u[i]$ corresponding to forced 1's are constrained to take the value 1, thus considerably reducing the size of the ILP.

The objective function to minimize is defined based on an indel cost function that may depend on the edge along which the indel takes place and on the length of the indel. Assuming that the cost of an insertion of length l along edge e is the

Logic form	**0-1 ILP form**
$\forall ab \in \{01,10,11\}, i=0...L+1, e=(u,v) \in T$: $U_e^{ab}[i] \leftrightarrow (X_u[i]=a) \wedge (X_v[i]=b)$.	$\forall i = 0...L+1, e=(u,v) \in T$**:** $(1-U_e^{11}[i]) + X_u[i] \geq 1$ $(1-U_e^{11}[i]) + X_v[i] \geq 1$ $U_e^{11}[i] + (1-X_u[i]) + (1-X_v[i]) \geq 1$ $(1-U_e^{10}[i]) + X_u[i] \geq 1$ $(1-U_e^{10}[i]) + (1-X_v[i]) \geq 1$ $U_e^{10}[i] + (1-X_u[i]) + X_v[i]) \geq 1$ $(1-U_e^{01}[i]) + (1-X_u[i]) \geq 1$ $(1-U_e^{01}[i]) + X_v[i] \geq 1$ $U_e^{01}[i] + X_u[i] + (1-X_v[i]) \geq 1$
$\forall ab \in \{01,10\}$, $i=1...L+1, e=(u,v) \in T$: $V_e^{ab}[i] \leftrightarrow (U_e^{ab}[i] \vee (\neg U_e^{11}[i] \wedge V_e^{ab}[i-1])$	$\forall ab \in \{01,10\}, i=1...L+1, e=(u,v) \in T$: $V_e^{ab}[i] + (1-U_e^{ab}[i]) \geq 1$ $V_e^{ab}[i] + (1-V_e^{ab}[i-1]) + U_e^{11}[i] \geq 1$ $(1-V_e^{ab}[i]) + U_e^{ab}[i] + V_e^{ab}[i-1] \geq 1$ $(1-V_e^{ab}[i]) + U_e^{ab}[i] + U_e^{11}[i] \geq 1$
$\forall ab \in \{01,10\}$, $i=1...L+1, e=(u,v) \in T$: $W_e^{ab}[i] \leftrightarrow (V_e^{ab}[i] \wedge \neg V_e^{ab}[i-1])$	$\forall ab \in \{01,10\}$, $i=1...L+1, e=(u,v) \in T$: $(1-W_e^{ab}[i]) + V_e^{ab}[i-1] \geq 1$ $(1-W_e^{ab}[i]) + (1-V_e^{ab}[i]) \geq 1$ $W_e^{ab}[i] + (1-V_e^{ab}[i]) + V_e^{ab}[i-1] \geq 1$

Table 1. Integer Linear Programming constraints for the Indel Parsimony Problem.

affine function $\alpha_e l + \beta_e$ and the cost of a deletion of length l is the affine function $\delta_e l + \gamma_e$, the objective function is:

$$minimize \sum_{e=(u,v) \in E_T} \sum_{i=1...L} (\alpha_e \cdot U_e^{01}[i] + \beta_e \cdot W_e^{01}[i] + \delta_e \cdot U_e^{10}[i] + \gamma_e \cdot W_e^{10}[i])$$

Indeed, each insertion along edge e contributes α_e, and the insertions are in bijection with the set $\{i | W_e^{01}[i] = 1\}$. Furthermore, each column of type (01) contributes β_e and such columns are in bijection with the set $\{i | U_e^{01}[i] = 1\}$. Hence, the cost of all insertions along edge e is given by $\alpha_e \cdot |\{i | W_e^{01}[i] = 1\}| + \beta_e \cdot |\{i | U_e^{01}[i] = 1\}| = \sum_i (\alpha_e \cdot W_e^{01}[i] + \beta_e \cdot U_e^{01}[i])$. The same reasoning applies to deletions.

7. Fractional relaxation heuristics

Integer Linear Programming is an NP-hard problem, and the best ILP solvers available to date (e.g. PBS [1]) can only be used to solve optimally relatively small systems of integer linear equations. Fractional relaxation is a technique whereby the linear program is solved in the space of real numbers, which can be done in polynomial time using the ellipsoid method[12], and the non-integer variable assignments are somehow interpreted in terms of the original problem[16]. This approach has produced efficient exact or approximate algorithms for several important algorithmic

problems[16]. We use fractional relaxation as part of a simple but efficient heuristic that produces good (and usually provably optimal) results in practice, although we currently do not have any approximation bounds for it. Our scheme is summarized as follows:

(1) Solve the fractional relaxation of the ILP, to obtain a fractional solution $X^{(f)}$ for the X variables, with an optimal value $O^{(f)}$ of the objective function when no integrality constraints are enforced. Notice that $O^{(f)}$ is a lower bound on the optimal value of the objective function with integrality constraints.
(2) In a typical example, the large majority of the variable assignments in $X^{(f)}$ are already integers. Let $X^{(r)}$ be the solution obtained by setting to zero all variables such that $X_u^{(f)}[i] < 1$, and let $score(X^{(r)})$ be the score of that solution. If $score(X^{(r)}) = O^{(f)}$, then $X^{(r)}$ is an optimal solution and we are done.
(3) If, in $X^{(f)}$, there are variables that are assigned values exactly 0 or 1, then do step 3a, otherwise do step 3b.
3a. For each such variable, substitute, in the ILP constraints, the variable by its value, thus resulting in an ILP with a smaller number of variables. Return to step (1) to solve the reduced ILP.
3b. Solve the remaining ILP exactly (in exponential time) using programs like lpsolve[2] (with integrality constraints), or PBS[1]. If the ILP is too large to be solved in reasonable time, simply round down the fractional solution.

This scheme is particularly efficient when the lpsolve program is used as the fractional LP solver, and in almost all cases (see Section 9), few iterations suffice to find a provably optimal solution without having to perform the expensive step 3b.

8. Divide-and-conquer approaches

Since the running time of any exact or approximation algorithm for IPP is likely to be $\Omega((n \cdot L)^{1+\epsilon})$, we would obtain gains in time complexity if we could divide the given IPP instance into a set of smaller IPPs, whose optimal solutions would then be combined to produce an optimal solution to the original problem. Breaking the original problem into subproblems is thus of great practical importance, especially given than we eventually want to apply these algorithms to genomes that are several billion nucleotides long. Here, we describe a procedure for doing so which guarantees that the optimal solutions of the subproblems can be combined into an optimal solution to the original problem. This is achieved by building a dependency graph G whose connected components correspond to these independent subproblems.

We start by defining an undirected graph $H = (V_H, E_H)$ which simply consists of a set of copies of T, one for each alignment position: $V_H = \{v_{(u,i)} | u \in V_T, 0 \leq i \leq L+1\}$, and $E_H = \{(v_{(a,i)}, v_{(b,i)}) | (a,b) \in E_T, 0 \leq i \leq L+1\}$ (see Figure 3). We can view solutions to the IPP as an assignment to the vertices of H. Now, define a function f on the nodes of H: $f(v_{(a,i)}) = 1$ if $A_v[i]$ is forced to one, and $f(v_{(a,i)}) = 0$ otherwise.

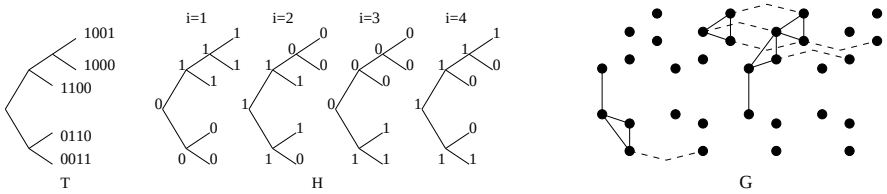

Fig. 3. Example of graphs H and G constructed from a given IPP instance. The numbers on the vertices v of H are $f(v)$ values. In G, dashed lines connect vertices from consecutive columns while full lines connect vertices with same column.

The dependency graph $G = (V_G, E_G)$ is defined as a variant of the line-graph of H (see Figure 3). We have $V_G = \{w_{(a,b,i)} | (v_{(a,i)}, v_{(b,i)}) \in E_H\}$. The edges E_G are of two types: those connecting vertices corresponding to the same alignment column (shown with full lines in Figure 3) and those connecting vertices from consecutive columns:

$$E_G^1 = \{(w_{(a_1,b_1,i)}, w_{(a_2,b_2,i)}) | 0 \leq i \leq L+1, b_1 = a_2, f(v_{(b_1,i)}) = 0\}$$
$$E_G^2 = \{(w_{(a,b,i)}, w_{(a,b,i+1)}) | 0 \leq i \leq L, \text{NAND}(f(v_{(a,i)}), f(v_{(b,i)})) \wedge \text{NAND}(f(v_{(a,i+1)}), f(v_{(b,i+1)}))\}$$
$$E_G = E_G^1 \cup E_G^2$$

Intituitively, two vertices $w_1 = (a, b, i)$ and $w_2 = (c, d, j)$ of V_G are connected if an assignment to $v_{(a,i)}$ or $v_{(b,i)}$ in V_H could affect the decision to assign $v_{(c,j)}$ or $v_{(d,j)}$. If $i = j$ and $b = c$ but w_1 and w_2 are disconnected, it is because their common endpoint b is forced to 1, so that assignments on each side will not directly affect the other. If $j = i + 1$ and $a = c, b = d$ but w_1 and w_2 are not connected, it is because both $v_{(a,i)}$ and $v_{(b,i)}$ are forced to one, or both $v_{(c,j)}$ and $v_{(d,j)}$ are forced to one, in which case the assignments do not affect each other since one of them is fixed. Note finally that if, for some $(a, b) \in E_T$, we have $f(v_{(a,i)}) = f(v_{(b,i)}) = 1$, then $w_{(a,b,i)}$ is incident to no edges. It is therefore in a connected component of its own.

The following theorem, whose proof is given in Appendix B, shows that it is possible to solve each connected component of G independently, while still guaranteeing the global optimality of the solution.

Theorem 8.1. *Let $C_1, \ldots, C_c$ be the connected components of G. An optimal solution to the IPP can be obtained by combining optimal solutions obtained independently for each C_i, for $i = 1...c$.*

8.1. *Near-optimal divide-and-conquer*

In some cases, the ILP associated to some of the connected components of the graph G will remain too large to be solved exactly by existing algorithms. In this case, educated guesses about the state of certain ancestral characters have to be made.

While these guesses may result in a non-optimal solution, the following theorem proves that each such guess cannot worsen the solution by more than 3 (under the unit-cost model).

Theorem 8.2. *Suppose* $a \in V_T$ *and* $f(v_{(a,i)}) = 0 \, \forall i = \alpha, \ldots, \beta$. *Let* S' *be an optimal solution. Then, there is a solution* S *such that* $cost(S) \leq cost(S') + 3$ *and* $S_a[i] = 0 \ \forall i = \alpha, \ldots, \beta$. *Thus, guessing that* $S_a[i] = 0 \ \forall i = \alpha, \ldots, \beta$ *cannot worsen the solution by more than 3.*

Proof. Let $S_a[i] = 0 \ \forall i = \alpha, \ldots, \beta$ and $S_v[i] = S'_v[i]$ everywhere else. S is an admissible solution since $f(v_{(a,i)}) = 0 \ \forall i = \alpha, \ldots, \beta$. Let b be a neighbour of a in T. Let j be the first anchor for (a, b) to the left of α. Let k be the first anchor for (a, b) to the right of β. Then, $d(S_a[j \ldots k], S_b[j \ldots k]) \leq d(S'_a[j \ldots k], S'_b[j \ldots k]) + 1$. This is true since, for the left hand side, the deletion cost is at most 1 and there are no insertions. Since a has at most three neighbours in T, the result follows. □

In pratice, if a connected component is too large to be solved, we identify the node a with largest eligible interval $\alpha...\beta$ and set the corresponding ancestral values to zero. For the same reason that we do not have an edge in G when the common endpoint that two edges from H share is forced to 1, we no longer have an edge between them in G when the common endpoint is set to 0. This is because the edges no longer share a common vertex whose assignment needs to be optimized.

9. Experimental results

We have implemented the algorithms described in Sections 7 and 8 and used the lpsolve [2] program to solve the resulting ILPs. Two biological datasets of mammalian sequences were analyzed and the speed and accuracy are reported on both. The first set consists of an 1.8 Mb region of the human genome containing the *CFTR* gene[23], together with orthologous sequences from chimp, macaque, mouse, rat, rabbit, cow, dog, and armadillo. The second set is a superset of the first one, using the 20 species used in Blanchette et al. [3].

Our algorithm first breaks the problem into independent subproblems using the exact algorithm of section 8. As can be seen in Table 2, the original problems are broken into a large number of components, most of which are small enough to be solved by our fractional relaxation procedure. In practice, lpsolve can solve our fractional LPs for connected components of up to 3000 vertices. In our 9-species dataset, only 20 components exceed this size (with the largest containing 9361 vertices), while this number rises to 25 for 20 species (where the largest component contains 37391 vertices).

When the resulting subproblems remain too large to be solved with lpsolve, we apply the procedure described in Section 8.1, in which educated guesses are made to resolve large blocks of consecutive unforced positions. This procedure usually quickly breaks large components into suitably small ones. As can be seen in Table

Subproblem size	0	1	2-9	10-99	100-999	1000-9999	≥ 10000
9 species	1401	2744	2281	1565	77	40	0
20 species	2731	2736	1963	1818	352	45	10

Table 2. Number of connected components of G with a given number of vertices, after the divide-and-conquer approach has been applied. A size of 0 means that the component is a forced on.

	# components (exact) (1)	# guesses(2)	# guessed components (3)
9 species	8108	31	10400
20 species	9655	81	11995

Table 3. (1):Number of connected components after the application of the exact divide-and-conquer procedure. (2): Total number of guesses that had to be made to break all components to a size of at most 3000 vertices. (3): Number of connected components after the guesses have been made.

	# components	=LB	=LB+1	=LB+2	=LB+3
9 species	10400	10396	4	0	0
20 species	11995	11968	17	7	3

Table 4. The number of guessed components that differ by 0,1, 2 or 3 from the lower bound which is the value of the optimal solution of the fractional relaxation of the ILP (see Section 8 step 1). No solution was off by more than 3 from the lower bound.

3, a very small number of these guesses is sufficient to obtain solvable components. Furthermore, a visual inspection of the guesses made revealed that almost all of them were very likely to be correct, given the context of the alignment. Unfortunately, it is difficult to quantify the effect of the guesses made on the accuracy of the solution, as these problems are to large to be solved by any other method.

Our fractional relaxation method not only gives us an approximate solution to the IPP, but it also gives us a lower bound on the total indel score of a connected component. When the IPP solution produced has a score equal to the lower bound, we know that the solution is optimal. As can be seen from table 4, the solution obtained is provably optimal for more than 99.7% of the connected components solved.

The complete 9-species data set took 395 minutes to solve on a desktop computer, while the 20-species dataset took 794 minutes. As seen in Table 5, most of the time is spent on a few large components, with the worst 9-species component requiring about one hour to solve and the worst 20-species component requiring nearly two hours. Note that an interesting aspect of our approach is that we can make as many heuristic guesses as desired (at the cost of sometimes making wrong choices),

Time(sec)	0-1	1-2	2-10	10-100	100-1000	> 1000
9 species	10205	78	44	39	28	6
20 species	7924	0	3933	82	49	7

Table 5. Distribution of the running time required to solve the connected components. Note that components with less than 5 vertices are solved by enumerating all 32 possible solutions, which is faster than calling the ILP solver.

thus reducing the size of the connected components to be solved to whatever size is suitable given one's computing resources. Also of interest is the fact that our algorithm is easily paralellizable (by solving each connected component on a different CPU), an important factor for large-scale, genome-wide ancestral reconstructions.

10. Conclusion and future work

Reconstructing indel scenarios is a crucial part of the more general problem of inferring ancestral genomes[3] and is equally important for the annotation of functional elements[14]. In this paper we have shown that reconstructing the most parsimonious set of insertions and deletions required to explain a given multiple alignment is an NP-complete problem. We formulated the problem as an Integer Linear Programming problem and described a divide-and-conquer method for breaking up the ILP into smaller components. We showed that in practice, a polynomial-time procedure based on fractional relaxation of the ILP almost always yields provably optimal results. Our algorithm is sufficiently fast and parallelizable to comtemplate whole-genome reconstructions.

Several questions remain open:

- What is the time complexity of the Indel Parsimony Problem on binary trees under the unit-cost model?
- Does the solution to the fractional relaxation of the 0-1 ILP provide guaranteed approximation bounds for the original problem?
- Can we elaborate more sophisticated rules for dividing the problem into subcomponents?
- Is it possible to take advantage of the particularly regular structure of the 0-1 ILP to solve it faster?
- Can the techniques described in this paper be used to solve the Indel Maximum Likelihood Problem?

11. Acknowledgments

We thank David Haussler for initiating the ancestral genome reconstruction project and formulating the first deletion-only polynomial-time algorithm; Ashish Sabarwhal for his suggestions regarding solving ILPs; Adam Siepel and Gill Bejerano for useful early discussions on this project; and Bruce Reed for his invaluable sug-

gestions on the NP-completeness proofs. We thank the referees for their useful comments.

References

1. F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A backtrack search pseudo-boolean solver. In *Proceedings of the 5th International Symposium on the Theory and Applications of Satisfiability Testing (SAT)*, pages 346–353, 2002.
2. M. Berkelaar. Lpsolve program. www.cs.sunysb.edu/ algorith/implement/lpsolve/implement.shtml.
3. M. Blanchette, E. D. Green, W. Miller, and D. Haussler. Reconstructing large regions of an ancestral mammalian genome in silico. *Genome Res*, 14(12):2412–2423, Dec 2004.
4. M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. A. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, D. Haussler, and W. Miller. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708–715, Apr 2004.
5. G. Bourque, P. Pevzner, and G. Tesler. Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse, and rat genomes. *Genome Research*, 14(4):507–16, 2004.
6. N. Bray and L. Pachter. MAVID: constrained ancestral alignment of multiple sequences. *Genome Research*, 14(4):693–699, Apr 2004.
7. M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, E. D. Green, A. Sidow, and S. Batzoglou. LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Research*, 13(4):721–731, Apr 2003.
8. J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.
9. W. M. Fitch. Toward defining the course of evolution: Minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1971.
10. J. Fredslund, J. Hein, and T. Scharling. A large version of the small parsimony problem. In *Proceedings of the 4th Workshop on Algorithms in Bioinformatics (WABI)*, 2004.
11. M. Garey and D. Johnson. *Computers and intractability*. W.H. Freeman and company, 1979.
12. M. Grotschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1993.
13. J. Hein. A method that simultaneously aligns, finds the phylogeny and reconstructs ancestral sequences for any number of ancestral sequences. *Molecular Biology and Evolution*, 6(6):649–668, 1989.
14. D. Kolbe, J. Taylor, L. Elnitski, P. Eswara, J. Li, W. Miller, R. Hardison, and F. Chiaromonte. Regulatory potential scores from genome-wide three-way alignments of human, mouse, and rat. *Genome Research*, 14(4):700–7, 2004.
15. G. Lunter, I. Miklos, Y. Song, and J. Hein. An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. *J Computational Biology*, 10(6):869–89, 2003.
16. G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, New York., 1988.
17. S. Porschen. On the Time Complexity of Rectangular Covering Problems in the Discrete Plane . In *Proceedings of the International Conference on Computational Science and its Applications*, volume 3045, pages 137–146, 2004.

18. E. Rivas. Evolutionary models for insertions and deletions in a probabilistic modeling framework. *BMC Bioinformatics*, 6(1):63, 2005.
19. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.
20. S. Schwartz, W. Kent, A. Smit, Z. Zhang, R. Baertsch, R. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with blastz. *Genome Research*, 13(1):103–7, 2003.
21. A. Siepel and D. Haussler. Combining phylogenetic and hidden markov models in biosequence analysis. *J Comput Biology*, 11(2-3):413–28, 2004.
22. The International Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–62, 2002.
23. J. W. Thomas, J. W. Touchman, and et al. Comparative analyses of multi-species sequences from targeted genomic regions. *Nature*, 424(6950):788–793, Aug 2003.

Appendix A. NP-Completeness proof for the IPP

We prove the NP-completeness of the decision problem version of the INDEL PARSIMONY PROBLEM (IPP):

INDEL PARSIMONY DECISION PROBLEM (IPDP)
Given: An alignment $\mathcal{A}$, a rooted phylogenetic tree T with the leaves of T labeled with the rows of $\mathcal{A}$, and a positive integer K.
Question: Does there exist an admissible extension of $\mathcal{A}$ to the internal nodes of T such that $\sum_{(u,v)\in E_T} d(A_u, A_v) \leq K$?

The proof will be separated into two stages. First, we will define a related problem, the INDEPENDENT SET ON PAIRED INTERVALS problem (ISPI) and show that the ISPI problem is NP-complete. We will then use this result to show that the decision version of IPP is also NP-complete.

A.1. *Definitions*

A *paired interval* consists of a pair of non-overlapping closed intervals $[x_1, x_2]$ and $[x_3, x_4]$ on the real line, with $x_1 < x_2 < x_3 < x_4$. A paired interval *contains* a point p if $x_1 \leq p \leq x_2$ or $x_3 \leq p \leq x_4$. The points x_1 and x_4 form the *external boundary* of the paired interval, and the points x_2 and x_3 define the *internal boundary* of the paired interval. The open interval (x_2, x_3) is called the *internal hole* of the paired interval. Two paired intervals *overlap* if they both contain a common point on the real line. An *independent set* of paired intervals is a set of paired intervals such that no two paired intervals in the set overlap. The *overlap graph* of a set of paired intervals is the graph obtained by constructing a vertex for every paired interval in the set and adding edges between vertices corresponding to paired intervals that overlap.

A.2. *NP-Completeness of the INDEPENDENT SET ON PAIRED INTERVALS*

Let us now define an auxiliary problem that will be used in the proof that the decision problem version of the IPP is NP-complete:

INDEPENDENT SET ON PAIRED INTERVALS (ISPI)
Given: A set S of paired intervals and a positive integer K
Question: Does there exist a subset $S' \subseteq S$ of K paired intervals such that no two paired intervals in S' overlap?

Theorem A1. *The ISPI problem is NP-complete.*

Proof. The ISPI problem is clearly in NP, since a nondeterministic algorithm can just guess a set of K paired intervals and verify in polynomial time that none of them overlap.

To show that the problem is NP-complete, we show that the 3SAT problem[11] can be reduced to the ISPI in polynomial time. Let an instance of the 3SAT problem be defined by the set $U = \{u_1, u_2, ...u_m\}$ of literals and the set $C = \{c_1, c_2, ...c_n\}$ of 3-clauses. Let us now construct the overlap graph G for a set of paired intervals and choose an integer value K such that G contains an independent set of size at least K if and only if C is satisfiable.

Our construction resembles the one presented in Garey and Johnson for the proof of NP-completeness of the VERTEX COVER problem[11]. For every clause c_i, we create a gadget with vertices $c_{i,1}$, $c_{i,2}$, and $c_{i,3}$ all connected to each other in a triangle. We then create a gadget for each literal. Let $N(u_i)$ represent the number of times u_i or $\bar{u}_i$ is present in the clauses (whichever is greater). Then, if $N(u_i) = 1$, we simply create a vertex for u_i and another one for $\bar{u}_i$ and connect the two vertices together. However, if $N(u_i) > 1$, we will create a cycle U_i with the vertices $V(U_i) = \{u_{i,1}, \bar{u}_{i,1}, u_{i,2}, \bar{u}_{i,2}, \ldots, u_{i,N(u_i)+1}, \bar{u}_{i,N(u_i)+1}\}$ and the edges $E(U_i) = \{\{u_{i,1}, \bar{u}_{i,1}\}, \{\bar{u}_{i,1}, u_{i,2}\}, \ldots, \{u_{i,N(u_i)+1}, \bar{u}_{i,N(u_i)+1}\}, \{\bar{u}_{i,N(u_i)+1}, u_{i,1}\}\}$.

Once all clause and literal gadgets are complete, we connect the clauses to their assigned literals. For every clause vertex $c_{i,j}$, we connect $c_{i,j}$ to the literal $u_{k,l}$, where the j-th literal in clause i is u_k and l is the smallest integer such that $u_{k,l}$ is not already attached to another clause vertex. An example of the resulting graph is shown in Figure 4.

Let V_U represent the total number of vertices in the gadgets for the literals and V_C represent the total number of vertices in the gadgets for the clauses. Let $K = \frac{V_U}{2} + \frac{V_C}{3}$. Then we can choose at most K independent vertices from the graph since any independent set can contain at most one vertex from each clause gadget and half of the vertices from the each literal gadgets, corresponding to all the vertices u_i or the vertices of its complement $\bar{u}_i$.

We can now show that the maximum independent set on our overlap graph is of size K if and only if the corresponding clauses are satisfiable. First, note that

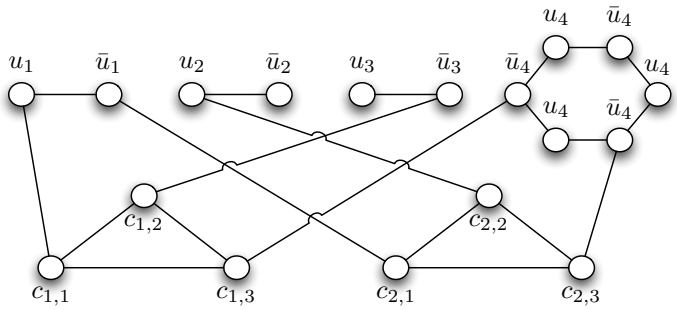

Fig. 4. Overlap graph for the 3SAT problem with literals $U = \{u_1, u_2, u_3, u_4\}$ and clauses $C = \{\{u_1, \bar{u}_3, \bar{u}_4\}, \{\bar{u}_1, u_2, \bar{u}_4\}\}$.

when the clauses are satisfiable by some truth assignment of the literals, we can select the vertices corresponding to the literals not included in the truth assignment along with the vertices for the satisfied clause elements in each clause. This will give us a set of K independent vertices. Conversely, if we have a set of K independent vertices from our graph, we must have $\frac{V_U}{2}$ literal vertices corresponding to either the literal u_i or its complement $\bar{u}_i$ for every $u_i \in U$. By choosing the truth assignment corresponding to the literal vertices not included in our independent set, we will find that it satisfies all the clauses. Therefore, the two problems are equivalent.

The final step of our proof consists of showing that the overlap graph created in our construction indeed represents the overlap graph of a set of paired intervals (refer to Figure 5 for an example). To do this, we create one paired interval for each vertex. We can start by connecting the first two vertices for each literal gadget by aligning the left intervals for each pair of u_i and $\bar{u}_i$ vertex. Similarly, we create the triangle edges for each clause gadget by aligning the left interval of each clause vertex in a triangle together. We are now free to align the right interval of each clause element to its corresponding literal. Whenever we have a literal occurring more than once, the right interval for the matched literal extends further than the clause interval so that it may overlap with the next vertex in the literal cycle. Finally, once all clause gadgets are fully connected to their associated literals, we close all cycles with the remaining right intervals and position any unmatched literal right interval to the right of all other intervals. This construction is clearly polynomial in the size of the input to the original 3SAT problem, and requires a constant amount of work per gadget. Therefore, our reduction is completed in polynomial time and we have shown the NP-completeness of the ISPI problem. □

For the purpose of the second part of the proof, we define a *set of indel intervals* as a set of paired intervals with a few restrictions:

(1) All boundaries of the paired intervals in the set must be in $\mathbb{Z}$.

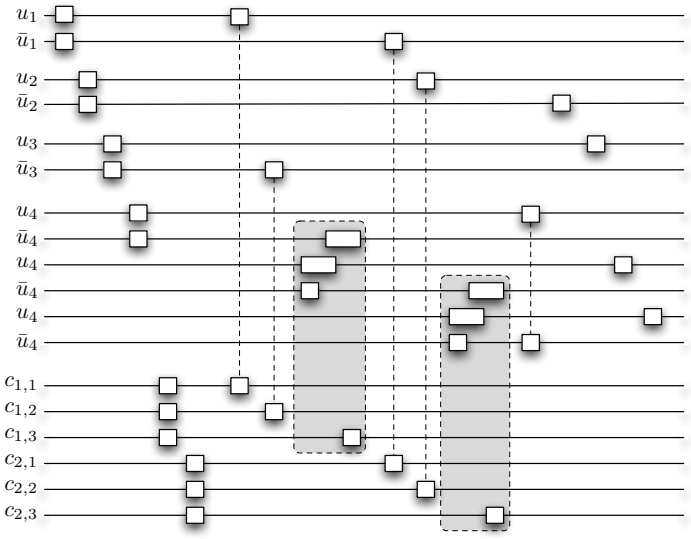

Fig. 5. Set of paired intervals corresponding to the overlap graph of Figure 4. Simple alignments of literals with clauses are illustrated with dashed lines. The more complicated assignments requiring the extension of a cycle for the literal gadgets are highlighted in the shaded boxes.

(2) The left (right) external boundaries of each paired intervals are considered hard boundaries. No paired interval may cross hard bounderies: a paired interval may not contain the left (right) external boundary of another paired interval unless it is also its left (right) external boundary.
(3) At least one hard boundary must be present in the internal hole of each the paired interval.
(4) No two paired intervals in the set may share an internal boundary.

Corollary A2. *The INDEPENDENT SET ON INDEL INTERVALS problem is NP-complete.*

Proof. This follows directly from our proof of Theorem A1. In order to show this, we only need to modify the construction of the set of paired intervals slightly to ensure that all paired intervals are created on the integer line, and that no two of them share any internal boundaries. As we can easily see, this can be done without affecting the overlap graph. The other two conditions - that no paired interval crosses any hard boundary and that each paired interval contains at least one hard boundary in its internal hole - are already satisfied by the construction. □

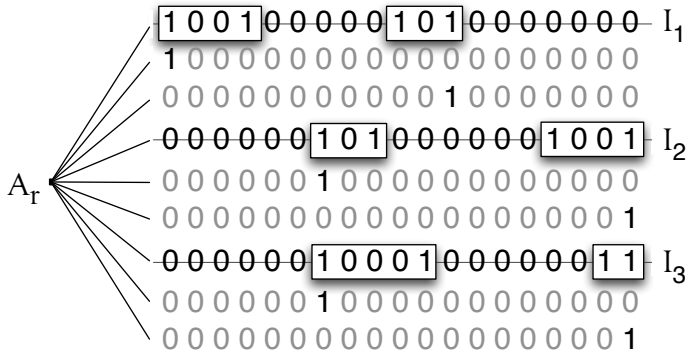

Fig. 6. A set of three indel intervals and their corresponding sequences. In this example, the optimal parsimony score is obtained with $A_{\{I_1,I_2\}}$.

A.3. *NP-Completeness of the IPDP*

Theorem A3. *The INDEL PARSIMONY DECISION PROBLEM (IPDP) is NP-complete.*

Proof. The IPDP problem is clearly in NP since a nondeterministic algorithm can pick a set of insertions and deletions explaining the evolution of observed sequences. We can then check in polynomial time whether the K insertions and deletions are valid and explain the evolution of the sequences.

To show that the problem is NP-complete, let us reduce the INDEPENDENT SET ON INDEL INTERVALS problem to the IPDP. We will do this by creating a tree with a single ancestor and 3 aligned sequences for each indel interval (refer to Figure 6 for an example of the construction). For the first sequence, we add a 1 at the four boundary positions of the indel intervals, and 0s elsewhere. We refer to this sequence as the interval sequence. The other two sequences are support sequences that contain a single 1, at the left and right external boundary of the indel interval respectively. The support sequences are present to ensure that the ancestor sequence also has a 1 present at the external boundaries of the indel interval, and are at a constant distance from any admissible ancestor root.

Let the *default ancestor* sequence A_r^ϕ represent the sequence obtained at the root node r by setting $A_r^\phi[i] = 1$ if and only if at least two of the leaf sequences have a 1 at position i. For a set of indel intervals I, we set A_r^I to be the ancestor obtained by taking the bit-wise OR operation of A_r^ϕ and the interval sequences corresponding to the intervals in I.

A_r^ϕ contains the minimum total number of 1's in the ancestor sequence. Since we cannot reduce the number of necessary deletions by adding more 1's in the ancestor, A_r^ϕ also sets a lower bound on the minimum number of deletions required to get from

any ancestor to the leaf sequences. Also, all interval sequences can be reached from A_r^ϕ with a few deletions followed by a single insertion. Therefore, only ancestors that remove the need for at least one of these insertions need to be considered to find the ancestor generating the optimal parsimony score. Since the only ancestors that can possibly remove insertions are those from A_r^I, these are also the only ancestors that we need to consider.

Let S_x represent the number of indel operations needed to go from the ancestor A_r^x to the leaf sequences in our one-level tree. We will show that we can obtain the indel parsimony score $S_\phi - K$ if and only if the original indel interval set contains an independent set of size at least K.

First, assume that the set of indel intervals contains an independent set I of size K. Since none of the sequences of I overlap, we can set the ancestor to be A_r^I and expand deletion operations that were necessary with the ancestor A_r^ϕ to include all extra 1's in A_r^I. Also, we now do not need to have insertions in the K sequences of I to get to the leaf sequence. Therefore, we now need $S_\phi - K$ operations to get to all the leaf sequences.

Let us now consider the case where the set of indel intervals S does not contain an independent set of size K. Then any subset $U \subseteq S$ of size $|U| \geq K$ must contain at least one pair of overlapping intervals s and t such that s contains one of the internal boundaries of t. Denote that contained point as p_t. By the definition of indel intervals, we also know that there is a hard boundary p_h in the internal hole of s. In s, p_t and p_h are separated by one of the internal boundaries of s, denoted by p_s. When going from A_r^U to the s in the corresponding IPDP, we wish to conserve the point p_s so we must delete p_t and p_h with two separate deletion operations. Let $U' = U \setminus s$. With the ancestor $A_r^{U'}$, p_s is not present in $A_r^{U'}$ so we can delete p_t and p_h in a single deletion operation before we insert p_s. Therefore, going from A_r^U to s requires as many indel operations as going from $A_r^{U'}$ to s. We can also see that changing the ancestor from A_r^U to $A_r^{U'}$ cannot improve the number of indel operations needed to get from the ancestor to any of the other leaf sequences. Therefore, we have obtained a set U' of size $|U'| = |U| - 1$ such that $S_{U'} = S_U$. If the size of U' is still greater or equal to K, we can apply this logic recursively until we get a set U'' of size less than K. At this point, the score of U'' must be $S_{U''} \geq S_\phi - |U''| > S_\phi - K$.

Thus, the indel parsimony score of our set of sequences is $S_\phi - K$ if and only if the original indel interval set contains an independent set of size at least K. The reduction described is clearly done in polynomial time, so we have shown that IPDP is NP-complete. □

Appendix B. Divide-and-conquer theorem

This section gives the proof for Theorem 8.1 presented in Section 8: An optimal solution to the IPP can be obtained by combining optimal solutions obtained independently for each C_α, $\alpha = 1 \ldots k$. B

Lemma B1. *If $w_{(a,b,i)} \in C_\alpha$ and $w_{(a,b,i+1)} \in C_\alpha$, then $(w_{(a,b,i)}, w_{(a,b,i+1)}) \in E_G$.*

Proof. Suppose $(w_{(a,b,i)}, w_{(a,b,i+1)}) \notin E_G$. Then either $f(v_{(a,i)}) = f(v_{(b,i)}) = 1$ or $f(v_{(a,i+1)}) = f(v_{(b,i+1)}) = 1$ (or both). Then, either $w_{(a,b,i)}$ or $w_{(a,b,i+1)}$ (or both) would be incident to no edges, which contradicts the assumption that they belong to the same connected component. □

Lemma B2. *Suppose C_α contains at least two vertices and suppose $w_{(a,b,i)} \in C_\alpha$ but $w_{(a,b,i-1)} \notin C_\alpha$. Then, $f(v_{(a,i-1)}) = f(v_{(b,i-1)}) = 1$. Similarly, if $w_{(a,b,i)} \in C_\alpha$ but $w_{(a,b,i+1)} \notin C_\alpha$, then, $f(v_{(a,i+1)}) = f(v_{(b,i+1)}) = 1$.*

Proof. Assume that C_α contains at least two vertices and $w_{(a,b,i)} \in C_\alpha$ but $w_{(a,b,i-1)} \notin C_\alpha$.

Suppose $f(v_{(a,i-1)}) = 0$ or $f(v_{(b,i-1)}) = 0$. Since $w_{(a,b,i)} \in C_\alpha$ and C_α contains at least two vertices, we do not have $f(v_{(a,i)}) = f(v_{(b,i)}) = 1$. Therefore, there is an edge $(w_{(a,b,i-1)}, w_{(a,b,i)})$, which contradicts $w_{(a,b,i-1)} \notin C_\alpha$.

A similar argument applies for the second half of the lemma. □

Lemma B3. *If C_α contains only a single vertex $w_{(a,b,i)}$ then either $f(v_{(a,i)}) = f(v_{(b,i)}) = 1$ or $f(v_{(a,i-1)}) = f(v_{(b,i-1)}) = f(v_{(a,i+1)}) = f(v_{(b,i+1)}) = 1$*

Proof. If we do not have $f(v_{(a,i)}) = f(v_{(b,i)}) = 1$, then the argument in the previous lemma applies. □

Lemma B4. *Suppose that $f(v_{(a,i)}) = 0$, p is the parent of a in T and c is a child of a in T. Then $w_{(p,a,i)}$ and $w_{(a,c,i)}$ are in the same connected component.*

Proof. Follows directly from the definition of G. □

Definition 1 (Block). Suppose $(a, b) \in E_T$ and $\forall i = \alpha, \ldots, \beta$, $f(v_{(a,i)}) = 0$ or $f(v_{(b,i)}) = 0$. Suppose further that $f(v_{(a,\alpha-1)}) = f(v_{(b,\alpha-1)}) = f(v_{(a,\beta+1)}) = f(v_{(b,\beta+1)}) = 1$. Then, $\{(v_{(a,i)}, v_{(b,i)}) |\ i = \alpha, \ldots, \beta\} \subseteq E_H$ constitutes a block.

Definition 2. The score of $E_H(C_\alpha)$ is the sum of the costs of all the blocks it contains.

Theorem B5. *The score of an optimal solution to the IPP on tree T and alignment $\mathcal{A}$ is the sum of the scores of each connected component in G.*

Proof.

By lemmas B2 and B3, we have that all the connected components are bounded by forced anchors on each side. Since the union of all connected components contains all the vertices in G, it contains all the edges in H. Thus, the sum of the costs of all the $E_H(C_\alpha)$ is the total cost of the tree, by definition of the indel distance d. □

Theorem B6. *Changing the assignment to $v_{(a,i)} \in V_H(C_{k_1})$ will not change the cost of $E_H(C_{k_2})$ for $k_1 \neq k_2$.*

Proof. By lemmas B2, B3 and B4, the blocks containing all the edges $(v_{(a,i)}, v_{(b,i)})$ are in the same connected component, namely C_{k_1} (since $v_{(a,i)} \in V_H(C_{k_1})$ so one of the edges $(v_{(a,i)}, v_{(b,i)}) \in E_H(C_{k_1})$). Since all the edges are in only one of the $E_H(C_\alpha)$, the cost of $E_H(C_{k_1})$ is the only one that is affected. □

Corollary B7. *An optimal solution to the IPP can be obtained by combining optimal solutions obtained independently for each C_α, $k = 1 \ldots n$.*