The tree alignment problem

Andrés Varón and Ward C Wheeler*

Abstract

Background: The inference of homologies among DNA sequences, that is, positions in multiple genomes that share a common evolutionary origin, is a crucial, yet difficult task facing biologists. Its computational counterpart is known as the multiple sequence alignment problem. There are various criteria and methods available to perform multiple sequence alignments, and among these, the minimization of the overall cost of the alignment on a phylogenetic tree is known in combinatorial optimization as the Tree Alignment Problem. This problem typically occurs as a subproblem of the Generalized Tree Alignment Problem, which looks for the tree with the lowest alignment cost among all possible trees. This is equivalent to the Maximum Parsimony problem when the input sequences are not aligned, that is, when phylogeny and alignments are simultaneously inferred.

Results: For large data sets, a popular heuristic is Direct Optimization (DO). DO provides a good tradeoff between speed, scalability, and competitive scores, and is implemented in the computer program POY. All other (competitive) algorithms have greater time complexities compared to DO. Here, we introduce and present experiments a new algorithm Affine-DO to accommodate the indel (alignment gap) models commonly used in phylogenetic analysis of molecular sequence data. Affine-DO has the same time complexity as DO, but is correctly suited for the affine gap edit distance. We demonstrate its performance with more than 330,000 experimental tests. These experiments show that the solutions of Affine-DO are close to the lower bound inferred from a linear programming solution. Moreover, iterating over a solution produced using Affine-DO shows little improvement.

Conclusions: Our results show that Affine-DO is likely producing near-optimal solutions, with approximations within 10% for sequences with small divergence, and within 30% for random sequences, for which Affine-DO produced the worst solutions. The Affine-DO algorithm has the necessary scalability and optimality to be a significant improvement in the real-world phylogenetic analysis of sequence data.

Keywords: Tree alignment, Tree search, Phylogeny, Sequence alignment, Direct optimization

Background

The inference of homologies among DNA sequences, that is, positions in multiple genomes that share a common evolutionary origin, is a crucial, yet difficult task facing biologists. Its computational counterpart is known as the multiple sequence alignment problem. There are various criteria and methods available to perform multiple sequence alignments (e.g. [1-9]). Among these, given a distance function, to minimize the overall cost of the alignment on a phylogenetic tree is known in combinatorial optimization as the Tree Alignment Problem (TAP) [10-15]. The TAP typically occurs as a subproblem of the Generalized Tree Alignment Problem (GTAP) which looks for the tree with the lowest alignment cost among all possible trees [10]. The GTAP is equivalent to the Maximum Parsimony problem when the input sequences are not aligned, that is, when phylogeny and alignments are simultaneously inferred.

An important element in sequence alignment and phylogenetic inference is the selection of the edit function, and in particular, the cost $G(k)$ of a sequence of $k$ consecutive insertions or deletions, generically called indels (e.g. an insertion of 3 consecutive $T$ ($k = 3$) in the sequence $AA$ could create the sequence $ATTTA$). The same operation in the opposite direction would be a deletion. The sequence alignment implied would be $A- - - A/ATTTA$, where $-$ represents an indel. $G(k)$ can have a significant impact in the overall analysis [16,17]. There are four plausible indel cost functions described in the literature: $G(k) = bk$ (non-affine) [18], $G(k) = a + bk$ (affine)
ever, it was not defined for affine edit distance. POY finds a solution of cost 298.75, close to the best GESTALT [25] and SALSA [26]. Using an approximate example, the alignment for the Sankoff produced by DO has cost 302.25, matching that of experimental tests. These experiments show that the solutions of Affine-DO are close to the lower bound inferred from an Linear Programming (LP) solution. Moreover, iterating over a solution produced using Affine-DO has very little impact in the overall solution, a positive sign of the algorithm's performance.

Although we build Affine-DO on top of the successful aspects of DO, DO has never been formally described, nor have its basic properties been demonstrated. To describe Affine-DO, we first formally define DO and demonstrate some of its properties.

Related Work

The TAP is known to be NP-Hard [31]. Due to its difficulty, a number of heuristic methods are applied to produce reasonable (but most likely suboptimal) solutions. The first heuristic techniques [11,12] consist of iteratively improving the assignment of each interior vertex as a median between the sequences assigned to its three neighbors. This method can be applied to any initial assignment of sequences and adjust them to improve the overall tree cost. In recent work, Yue et al. [32] used this algorithm in their computer program MSAM for the tree alignment problem, using as initial assignment the median computed between the 3 closest leaves to the interior vertex (ties arbitrarily resolved).

Hein [13,14], designed a second heuristic solution which is implemented in the TreeAlign program. In TreeAlign, sets of sequences are represented by alignment graphs, which hold all possible alignments between a pair of sequences. The complete assignment can be performed in a post-order traversal of a rooted tree, where each vertex is assigned an alignment graph of the two closest sequences in the alignment graph of its two children vertices. The final assignment can be performed during a backtrack on the tree. Although this method is powerful, it is not scalable (e.g. using TreeAlign to evaluate one of the simulations used in this study did not finish within 48 hours). Moreover, the TreeAlign program does not allow the user to fully specify the distance function. This algorithm was later improved by Schwikowski and Vingron, producing A comparison of the tree scores of various methods was recently performed in [30] and is implicit in some of the conclusions of [17]. The authors concluded that when using a heuristic fit for the GTAP, the hypotheses have scores better than those produced by other methods. Therefore, without hindsight (i.e., when accuracy cannot be measured), biologists would prefer the hypotheses generated under the GTAP.

In this paper, we introduce and present experiments for a new algorithm Affine-DO. Affine-DO has the same time complexity of DO, but is correctly suited for the affine gap edit distance. We show its performance experimentally, as implemented in POY version 4, with more than 330,000 experimental tests. These experiments show that the solutions of Affine-DO are close to the lower bound inferred from an Linear Programming (LP) solution. Moreover, iterating over a solution produced using Affine-DO has very little impact in the overall solution, a positive sign of the algorithm's performance.

Although we build Affine-DO on top of the successful aspects of DO, DO has never been formally described, nor have its basic properties been demonstrated. To describe Affine-DO, we first formally define DO and demonstrate some of its properties.

\[G(k) = a + b \log k \text{ (logarithmic)} \] [16,19-22], and \[G(k) = a + bk + c \log k \text{ (affine-logarithmic)} \] [16]. Simulations and theoretical work have found evidence that affine-logarithmic yields the most satisfactory results, but provides marginal benefits over the affine function, while its time complexity is much greater [16]. For this reason, many biologists adopt the affine indel cost function. (This topic is still a subject of controversy.)

For large data sets, a popular heuristic is Direct Optimization (DO) [15]. DO provides a good tradeoff between speed, scalability, and competitive scores, and is implemented in the computer program POY [23,24]. For example, the alignment for the Sankoff data set produced by DO has cost 302.25, matching that of GESTALT [25] and SALSA [26]. Using an approximate version of DO that has the same time complexity, POY finds a solution of cost 298.75, close to the best known cost of PRODAL1 (295.25) [27]. All other (competitive) algorithms have greater time complexities compared to DO (e.g [25-27]). An important limitation of DO, however, is that it was not defined for affine edit distance functions.

The properties of DO and the GTAP (DO+GTAP) for phylogenetic analysis were experimentally evaluated in [28]. The main conclusion of that study is that DO+GTAP could lead to phylogenies and alignments less accurate than those of the traditional methods (e.g. CLUSTALW + PAUP*). The initial work of Ogden and Rosenberg [28] raised a number of important questions: Do the conclusions hold if a better fit heuristic is used for the tree search in the GTAP? What would be the effect of using an affine edit distance function? How do the hypotheses scores compare among the different methods? These questions have since been answered in various followup papers.

In [29], the author found that the opposite conclusion can be drawn when a better fit heuristic for the GTAP is used. That is, when the resulting tree is closer to the optimal solution, DO+GTAP is a superior method. Moreover, a good fit heuristic is a fundamental aspect in phylogenetic analysis that cannot be overlooked.

Although [28] performed simulations under affine gap costs, the study used the non-affine distance functions described for DO at the time of publication. Whether or not a different distance function could yield different conclusions was tackled in [17]. The authors found that when using the GTAP as phylogenetic analysis criterion under the affine gap cost function, the resulting phylogenies are competitive with the most accurate method for simulated studies (i.e. Procons using a ML analysis under RaxML) [17]. It is important to note that [17] used an early implementation of the algorithms presented in this paper (available in POY version 4 beta).
the best tree alignment known for the Sankoff data set [33].

The most important theoretical results for the TAP are several 2 approximation algorithms, and a Polynomial Time Approximation Scheme (PTAS) [34-37]. These algorithms solve the TAP from a theoretical perspective, but the execution time of the PTAS renders it of no practical use. On the other hand, the 2-approximation algorithms have shown very poor performance when compared to heuristic methods such as that of TreeAlign.

Direct Optimization (DO) [15] is a heuristic implemented in the computer program POY [23,24,38], which yields good execution times and competitive alignment costs. Given that DNA sequences have a small alphabet (4 elements extended with an indel to represent insertions and deletions), DO represents a large number of sequences in a compact way by using an extended alphabet (potentially exponential in the sequence length). In the spirit of the TreeAlign method, DO heuristically assigns to each vertex, during a post order traversal, a set of sequences in an edit path connecting two of the closest vertices. Subsequently, in a pre-order backtrack, a unique sequence is assigned to the interior vertices to produce the solution.

DO can be implemented with a time complexity of $O(n^3|V|)$, where $n$ is the length of the longest sequences, and $V$ is the vertex set of the tree. For larger alphabets the total time complexity is $O(n^3|V||\Sigma|)$, where $|\Sigma| \ll n$ is the alphabet.

Schwikowski and Vignon [27] published the best heuristic algorithm for the TAP, as implemented in the PRODALI software. Although powerful, this algorithm has a potentially exponential time and memory complexity, which in turn makes it non-scalable and difficult to use for the GTAP. Moreover, PRODALI is not publicly available. It is for these reasons that DO is the algorithm of choice for the GTAP, yielding slightly weaker tree cost approximations when compared to those of PRODALI, but suitable for better performance on larger data sets.

**Results and discussion**

**Direct Optimization**

Direct Optimization (DO) has only been described informally in the literature [15,38], and to build on it, we must first fill this gap. At the core of the algorithm is the use of an extended alphabet to represent sets of sequences. We begin by exploring the connection of this method with those using a tree alignment graph.

In TreeAlign and PRODALI, the set of optimal alignments between sequences are represented in an *alignment graph*. These graphs are aligned at each vertex in the tree to find their closest sequences. An alignment graph is then computed between these sequences, and assigned to a vertex of the tree. The alignment between a pair of such graphs, however, is an expensive computation, both algorithmically ($O(n^4)$), and in its implementation. PRODALI is more expensive in practice, as it not only stores the set of optimal, but also suboptimal alignments.

In DO, not all the possible alignments are stored, but only one. However, it stores all the possible sequences that can be produced from this alignment. We will call such set of sequences a reduced alignment graph (RAG). Thanks to their simplicity, DO use a more compact representation of a RAG, to permit greater scalability than that of TreeAlign or PRODALI. DO represents them as sequences in an extended alphabet by which we can then represent a complete RAG with an array.

It is then possible to align RAG’s, find the closest sequences contained in them, and compute their RAG with time complexity $O(n^2)$. The following section formalizes these ideas.

**Sets of Sequences, Edition Distance, and Medians**

The first goal is to find a compact representation of sets of sequences produced in a pairwise alignment. For example, the alignment ATTG A--C is represented in an alignment graph shown in Figure 1. Such graph can then be extended to include intermediate sequences such as ATG or ATC (Figure 1).

The same information can be efficiently stored by using an extended alphabet $\Sigma_P = \mathcal{P}(\Sigma) \setminus \{\emptyset\}$ that includes all the subsets of $\Sigma$ with the exception the empty set, as follows

$$\{[A]|T, \text{indel}|[T, \text{indel}],[G, C]\}.
$$

We call such representation a Reduced Alignment Graph (RAG). Notice that all the intermediate sequences can be produced by selecting an element from each set in the RAG, and removing all the indels from the resulting sequence. If a sequence can be generated by following this procedure, then we say that the sequence is included in the RAG. The example then includes the sequences ATTG, ATTC, ATC, ATG, AC, and AG.

**Observation 1.** Let $A \in \Sigma^*_P$ be a RAG. Then there are $\prod_{x \in A} |X|$ sequences contained in $A$.

In the original problem definition, we are given a distance $d$ between the elements in $\Sigma$. Let $d_P(A, B) = \min_{a \in A, b \in B} d(a, b)$, be the distance between elements in $\Sigma_P$. The following observation is by definition:

**Observation 2.** For all $A, B \in \Sigma_P$, there exists an $a \in A$ and $b \in B$ such that $d_P(A, B) = d(a, b)$.

Define the RAG edit distance by setting $d = d_P$ in Equation 1.
The sequence edit distance can be computed using dynamic programming [39], following the recursive function:

$$e(A_{1..i}, B_{1..j}) = \begin{cases} 
  e(A_{1..i-1}, B_{1..j-1}) + d(A_i, B_j) \\
  e(A_{1..i-1}, B_{1..j}) + d(A_i, \text{indel}) \\
  e(A_{1..i}, B_{1..j-1}) + d(B_j, \text{indel}) 
\end{cases}$$

(1)

with base cases $e(\langle \rangle, \langle \rangle) = 0$, and $e(\langle \rangle, A) = e(A, \langle \rangle) = \sum_{1 \leq i \leq |A|} d(A_i, \text{indel})$. The affine case is more elaborate but possesses the same spirit and time complexity [40].

We will show that we can find efficiently the closest sequences in a pair of RAGs, as well as their edit distance. Thanks to these properties, a RAG is used instead of an alignment graph, to bound the cost of a tree with lower time complexity.

**Lemma 1.** For all RAGs $A$, $B$, there exists sequences $U$, $V$ such that $U$ is contained in $A$, $V$ is contained in $B$, and $e(A, B) = e(U, V)$.

**Proof.** We define a procedure to produce $U$ and $V$. Start with an empty $U$ and $V$, and follow the backtrack of Equation 1. For each case, prepend the following to $U$ and $V$:

- **case 1** Select an element $x \in X_i$ that holds Observation 2 and prepend it to $U$. Then find an element $y \in Y_j$ that is closest to $x$ and prepend it to $V$. From Observation 2 we know that $d(x, y) = d_p(x_i, y_j)$.
- **case 2** Select an element $x \in X_i$ closest to indel and prepend $x$ to $U$ and indel to $V$. Again from Observation 2 we know that $d(x, \text{indel}) = d_p(x_i, \text{[indel]})$.
- **case 3** Symmetric to case 2.

Observe that the overall time complexity remains $O(n^2)$ as in the original Needleman-Wunsch algorithm [39].

**The DO Algorithm**

**DO(T, χ), Direct Optimization**

DO (Algorithm 1) estimates the cost of a tree by proceeding in a post-order traversal on a rooted tree, starting at the root $ρ$, and assigning a RAG to each interior vertex.

**Data:** A binary tree $T$ with root $ρ$

**Data:** An assignment $χ : L(T) → Σ$ of sequences to the leaves $L$ of $T$

**Data:** $S(v)$ holds a set of sequences for vertex $v$, initially empty for every vertex

**Result:** cost holds an upper bound of the cost of $T$, $χ$

begin
  cost ← 0
  foreach level of $T$, with the bottom level first do
    foreach node $v$ at the level do
      if $v$ is a leaf (has no children) then
        $S(v) ← \langle a_i, a_i = \{χ(v_i)\} \rangle$
      else
        Data: $v$ has children $u$ and $w$
        cost ← cost + $e_p(S(u), S(w))$
        $U$, $W$ ← the alignment of $S(u)$ and $S(w)$ respectively;
        $S(v) ← m_p(U, W)$;
      end
    end
  end
return cost
end

We have not defined yet $m_p(U, W)$ to compute each RAG. Let $m(X, Y)$ be the set of elements in $X$ and
Lemma 2. Let \( C = m_p(A, B) \). Then for all \( X \) included in \( C \), there exists \( Y \) and \( Z \) included in \( A \) and \( B \) respectively, such that \( e_p(A, B) = e(Y, Z) = e(X, Y) + e(X, Z) \). Moreover, \( Y \) and \( Z \) are (some of) the closest sequences to \( C \) that are contained in \( A \) and \( B \) respectively.

Proof. Follows directly from the median definition and Lemma 1.

Lemma 2 is important for the correctness of the DO algorithm. It shows that for every sequence contained in \( C \), there are corresponding sequences in \( A \) and \( B \) of edit distance equal to \( e_p(A, B) \). This lemma can then be used in the DO algorithm to delay the selection of a sequence from each RAG, and use \( e_p \) directly to calculate the overall cost of the tree. Without it, \( e_p \) cannot be used for this purpose directly.

Definition 1. Compatible assignments Two assignments \( \chi : V \rightarrow \Sigma^* \) and \( \chi' : V \rightarrow \Sigma^* \) are compatible if both assign the same sequences to corresponding leaves, that is, for all \( v \in L \), \( \chi(v) = \chi'(v) \).

The following Theorem shows that the tree cost computed by DO is feasible:

Theorem 1. There exists an assignment of sequences \( \chi' \) compatible with \( \chi \) such that

\[
\text{DO}(T, \chi) = \sum_{(u,v) \in E} e(\chi'(u), \chi'(v)).
\]

Proof. Let \( T \) have root vertex \( \rho \). Call \( \chi' \) the final assignment of sequences to the vertices of \( T \). Select any \( X \) included in \( S(\rho) \) and set \( \chi'(\rho) \leftarrow X \). Then for each other vertex \( v \) with parent \( p \), following a pre-order traversal starting at \( \rho \), let \( \chi(v) \leftarrow X \) where \( X \in \Sigma^* \) is included in \( S(v) \) and is closest to \( \chi'(p) \). From Lemma 2, we know that for any selection at \( p \) there exists a selection in its children that would yield the additional cost computed at \( p \) during the DO algorithm. Moreover, at each pre-order traversal step, we assign to each vertex \( v \) the closest sequence to \( \chi'(p) \) included in \( S(v) \). Again from Lemma 2, we know that the total cost of the two edges connecting \( p \) with its children must be greater than or equal to the additional cost computed for vertex \( p \) in the DO algorithm. Therefore, \( \text{DO}(T, \chi) \geq \sum_{(u,v) \in E(T)} e(\chi'(u), \chi'(v)) \).
$A_i$ and $B_j$ align elements other than an indel. $d[i, j]$ is the cost of an alignment using indel elements in $A_i$ and $B_j$. $v[i, j]$ is the cost of an alignment where we use a “vertical” indel block by aligning $B_j$ with an indel. Finally, $h[i, j]$ is the cost of an alignment where we use a “horizontal” indel block by aligning $A_j$ with an indel.

To compute these values, we define a number of accessory functions. The cost of a pure substitution $\text{subst}(X, Y) = d_{p}(X \setminus \{ \text{indel} \}, Y \setminus \{ \text{indel} \})$. Symmetric to the substitution cost, we need the cost of extending a gap when $\text{indel} \in A, B \subseteq \Sigma$:

$$\text{diag}(X, Y) = \begin{cases} 
0 & \text{if indel} \in X \text{ and indel} \in Y \\
\infty & \text{otherwise}.
\end{cases}$$

There are three remaining accessory functions required to compute the matrices $g, h, v,$ and $d$. Each function handles various cases where $a$ or $b$ needs to be added. The first function, $\text{go}(A, i)$, evaluates whether or not it is necessary to add a gap opening value when aligning $A_i$ with a gap:

$$\text{go}(A, i) = \begin{cases} 
0 & \text{if } i = 1 \text{ and indel} \in A_i \\
0 & \text{if } i > 1 \text{ and indel} \notin A_{i-1} \text{ and indel} \in A_i \\
a & \text{otherwise}.
\end{cases}$$

The second function $\text{go}'(X, Y)$ calculates the extra cost incurred when not selecting an indel in one of the sequences means splitting an indel block:

$$\text{go}'(X, Y) = \text{subst}(X, Y) + \begin{cases} 
0 & \text{if indel} \notin X \\
a & \text{otherwise}.
\end{cases}$$

The third, and final accessory function, computes what would be the extra cost of extending an indel, that is:

$$\text{ge}(X) = \begin{cases} 
0 & \text{if indel} \in X \\
b & \text{otherwise}.
\end{cases}$$

Finally, the recursive functions for the cost matrices is defined as:

$$
g[i, j] = \begin{cases} 
g[i - 1, j - 1] + \text{subst}(A_i, B_j) \\
d[i - 1, j - 1] + \text{subst}(A_i, B_j) + \text{go}(A_i) + \text{go}(B_j) \\
v[i - 1, j - 1] + \text{go}'(B_j, A_i) \\
h[i - 1, j - 1] + \text{go}'(A_i, B_j),
\end{cases}$$

$$
h[i, j] = \begin{cases} 
h[i - 1, j - 1] + \text{ge}(B_j) \\
d[i - 1, j - 1] + \text{ge}(B_j) + \text{go}(B_j) \\
v[i - 1, j] + \text{ge}(A_i) \\
d[i - 1, j] + \text{ge}(A_i) + \text{go}(A_i),
\end{cases}$$

$$
v[i, j] = \begin{cases} 
v[i - 1, j] + \text{ge}(A_i) \\
d[i - 1, j] + \text{ge}(A_i) + \text{go}(A_i),
\end{cases}$$

$$
d[i, j] = \text{diag}(A_i, B_j) + \begin{cases} 
\text{min} \{d[i - 1, j - 1] \\
g[i - 1, j - 1] + \text{go}(A_i) + \text{go}(B_j),
\end{cases}$$

with base cases $g[0, 0] = 0, d[0, 0] = \infty, v[0, 0] = \text{go}(A, 1), h[0, 0] = \text{go}(B, 1), g = [0, 0] = d[0, 0] = v[0, 0] = \infty, h[0, 0] = h[0, 1 - 1] + \text{ge}(B_j), 1 \leq i \leq |B|, v[j, 0] = v[j - 1, 0] + \text{ge}(A_i),$ and $g[j, 0] = d[j, 0] = h[j, 0] = \infty, 1 \leq j \leq |A|.$

The following theorem shows that if we align a pair of sequences in $A, B$, then we can bound the cost of the closest pair of sequences included in them.

**Theorem 2.** There exists a sequence $X$ contained in $A$ and a sequence $Y$ contained in $B$ such that $e_{\text{aff}}(A, B) \geq e_{\text{aff}}(X, Y)$.

**Proof.** We are going to create a pair of sequences $X$ and $Y$ contained in $A$ and $B$ respectively that have edit cost at most $e_{\text{aff}}(A, B).$ To do so, follow the backtrack that yields $e_{\text{aff}}(A, B)$, and at each position $i$ and $j$ in the aligned $A$ and $B$ assign $X_k$ and $Y_k$, where $k$ is the alignment position corresponding to the aligned $X_i$ and $Y_j$ as follows:

1. $g[i, j]$ is the cost of aligning $A_{1-1}$ and $B_{1-1}$ when a non-indel element of $A_i$ and $B_j$ is aligned. If the backtrack uses $g[i, j]$ then assign to $X_k$ and $Y_k$ the closest elements in $A_i \setminus \{ \text{indel} \}$ and $B_j \setminus \{ \text{indel} \}$. Observe that all the cases in Equation 2 align a non-indel element from $A_i$ and $B_j$, and add a cost that is always greater than or equal to $\text{subst}(A_i, B_j) = d(X_i, Y_j)$.

2. $h[i, j]$ is the cost of extending an indel in the horizontal direction. Therefore, select $X_k = \text{indel}$, and

$$Y_k = \begin{cases} 
\text{indel} & \text{if indel} \in B_j \\
y, y \in B_j & \text{otherwise}.
\end{cases}$$

If $Y_k = \text{indel}$, then the alignment of $X_k$ and $Y_k$ causes no additional cost in the particular alignment being built between $X$ and $Y$. Otherwise, then there is an extra cost, of at least the $b$ parameter, which both cases of Equation 3 account for. Additionally, if the previous pair of aligned elements are a pair of indels (second case in 3, see below for the treatment of this option), then an extra indel opening cost is added.

3. $v[i, j]$ is the cost of extending an indel block in the vertical direction. The treatment is symmetric to that of $h$, with $Y_k = \text{indel}$ and

$$X_k = \begin{cases} 
\text{indel} & \text{if indel} \in A_i \\
x, x \in A_i & \text{otherwise}.
\end{cases}$$

4. $d[i, j]$ is the cost of extending an indel in the diagonal direction, that is, when both $A$ and $B$ hold indels, and those indels are being selected during the backtrack. Equation 6 ensures that this choice is only possible by assigning $\infty$ whenever at least one of $A_i$ or $B_j$ does not contain an indel. Otherwise, if this option is
selected, then simply assign *indel* to both *X*ₖ and *Y*ₖ with no extra cost for the alignment of *X* and *Y*.

**The Main Algorithm: Affine-DO**

We will now use \( e_{aff}(A, B) \) to bound the cost of a tree using a post-order traversal, in the same way we did with DO (Algorithm 1). In order to do so a RAG to be assigned on each step must be defined (i.e. the function \( m_P \) in Algorithm 1). To create the RAG \( M \) (initially empty), do as follows in each of the 4 items described in the proof of Theorem 2:

1. If we selected two indels in *X*ₖ and *Y*ₖ, don’t change \( M \).
2. If \( X_k = \text{indel} \) and \( Y_k \neq \text{indel} \), then prepend \{indel\} \( \cup B_j \) to \( M \).
3. If \( X_k \neq \text{indel} \) and \( Y_k = \text{indel} \), then prepend \{indel\} \( \cup A_i \) to \( M \).
4. If \( X_k \neq \text{indel} \) and \( Y_k \neq \text{indel} \), then prepend \{ \( x \in A_i \), for some \( y \in B_j \), \( d(x, y) = d(X_k, Y_k) \) \} + \{ \( y \in B_j \), for some \( x \in A_i \), \( d(x, y) = d(X_k, Y_k) \) \} to \( M \).
5. Once the complete \( M' \) is created, remove all the elements \( M_i = \{ \text{indel} \} \) to create the indel-less RAG \( M \). We call \( M \) the RAG produced by \( m_{aff}(A, B) \).

**Definition 2.** Affine-DO is Algorithm 1, modified by replacing \( m_P \) with \( m_{aff} \) and \( e_P \) with \( e_{aff} \).

It is now possible to use the Affine-DO algorithm to bound heuristically the cost of an instance of the TAP.

**Theorem 3.** Given a rooted tree \( T \) with root \( \rho \), and an Affine-DO assignment \( S \) \( : V(T) \to \Sigma^* \) there exists an
Table 1 Simulation parameters

| Parameter          | Values Evaluated |
|--------------------|------------------|
| Substitution Rate  | 1.5              |
| Average Branch Length | 0.05, 0.1, 0.2, 0.3, ∞ |
| Max. Gap           | 1, 2, 5, 10, 15  |
| Root Sequence Length | 70, 100, 150, 200, 300, 400, 500, 1000 |

All combinations of parameters were employed to generate the test data sets. The branch length variation equals the average branch length.

assignment $\chi' : V(T) \rightarrow \Sigma^*$ such that $X = \chi'(\rho)$ and the cost computed by Affine-DO equals that implied by $\chi'$.

**Proof.** If there are no indels involved in the tree alignment, then the arguments of Theorem 1 would suffice. Hence, we now concentrate on the cases that involve indels.

To prove those remaining cases, we will use induction on the vertices of the tree. To do so, we will count the credits that each vertex adds to the subtree it roots as added by the Affine-DO algorithm. The credits represent the maximum total cost of the indels involved in a particular subtree; we will compare them with the debits incurred by a set of indels, and verify that the credits are always greater than or equal to the debits. To simplify the description, we will call type A subsequences of maximal size holding only indels, and type B subsequences of maximal size holding sets that include, but are not limited to, indels, and type C maximal subsequences holding sets with no indel.

We will count without loss of generality the credits and debits within those subsequences. In Figures 3 and 4, Type A is represented as a line, type B as a box with a center line, and type C as an empty box.

For the inductive step, consider the leaves of the tree. By definition, for all $v \in L$, $S(v)$ can contain subsequences of neither type A nor B, as there are no indels allowed. Therefore, the theorem holds true, with a credits = debits = 0.

Consider now the interior vertex $v$, with children $u$ and $v$. In Figure 3, all the simple cases where the limits of the subsequences in $S(u)$ and $S(v)$ match those of $S(p)$. It is straightforward to see that in all those cases credits = debits.

Consider now the more difficult case when the blocks do not have exact limits. Assume without loss of generality that $S(u)$ and $S(v)$ have a segment of type B, and $S(p)$ has in the corresponding segment a series of blocks of type A and C (Figure 4). (There can be no subsequences of type B in $S(p)$ aligned with those of type B in $S(u)$ and $S(v)$ as $m_{aff}$ does not allow it.)

The total credit granted by Equation 2 is $c \geq 2ma + 2b\sum_{i=1}^{m} s_i$. We can transfer $c/2$ to $u (v)$, so that in one edge rooted by $u (v)$, a series of insertions corresponding to the subsequences $s_1, s_2, \ldots, s_m$ can occur (Figure 4), while the other branch supports a single deletion of length $l - \sum_{i=1}^{m} s_i$ (Figure 4 lower, upper dashed box). The total debit of these events now rooted in $u$ would be

$$a(m+1) + b\sum_{i=1}^{m} s_i + b(l - \sum_{i=1}^{m} s_i) \leq c/2 + a + bl. \quad (7)$$

By the inductive hypothesis, the subtree rooted by $u (v)$ has credits $\geq$ debits, and from Equation 7 we also have that credits $>$ debits in $p$, therefore the theorem holds, and the overall tree rooted by $p$ has a sequence assignment of cost at most that computed by the Affine-DO algorithm.

**Theorem 4.** If $\Sigma$ is small, then Affine-DO has time complexity $O(n^2 |V|)$ time, otherwise the time complexity is $O(n^2 |V||\Sigma|)$.

**Proof.** If the alphabet is small, then $m_{aff}$ and $d_p$ can be pre-computed in a lookup table for constant time comparison of the sets. For large alphabets the maximum size of the sets contained in $\Sigma_p$ can be made constant. Otherwise, a binary tree representation of the sets would be necessary, adding a $|\Sigma|$ factor to the set comparison. Each heuristic alignment can be performed using dynamic programming, with time complexity $O(n^2)$ where $n$ is the maximum sequence length (Ukkonen’s [41] algorithm

![Figure 5 Approximate DO. An iteration of the approximated iterative improvement. To improve $x_1$, Affine-DO is used to produce $x_1$, $x_2$, and $x_3$ in the three possible rooted trees with leaves $u$, $v$, and $w$. If the best assignment $x_1$ yields better cost than the original $x$, then it is replaced, otherwise no change is made.](image-url)
makes no obvious improvement as insertions and deletions could have cost 0 when aligning sequences in \( \Sigma_p^* \).
Each alignment must be repeated for \(|V|\) vertices during the post-order traversal, yielding the claimed time complexity.

**Experimental Evaluation**

In this section, we describe the methods used to generate the instance problems, assess the solutions generated by each algorithm, and compare the algorithms. This allows the assessment of the performance of each algorithm, Affine-DO in greater detail, and an evaluation of Affine-DO using exact solutions for trees with only 3 leaves.

**Data Sets**

To generate the instance problems, we simulated a number of sequences using DAWG 1.1.1 [42] with insertions and deletions following a power law distribution. The simulations followed random binary trees of 50 leaves comprising all the combinations of the parameters listed in Table 1. These produced a total of 96,000 independent simulations. For each data set, we collected the true sequence assignment. This information allows the comparison of the cost calculated by Affine-DO with the cost implied by the true sequence of events. Our expectation was to produce costs lower than those using the true sequence assignment.

**Solution Assessment**

The sequences assigned by the simulation can be far from the optimal solution. To evaluate Affine-DO, we used two algorithms: the standard Fixed States algorithm, which is known to be a 2-approximation, and the cost calculated by the solution of an LP instance of the problem. A good heuristic solution should always be located between these two bounds. As a comparison measure for each solution, the ratio between the solution cost and the LP bound was computed. The closer the ratio to 1.0, the better is the solution.

This form of evaluation has the main advantage (but also disadvantage), of being overly pessimistic. Most likely, the LP solution is unachievable, and therefore, the approximation ratio inferred for the solution produced by Affine-DO will most likely be an overestimate. To assess how over-negative the LP bound is, we produced 2100 random sequences divided in triplets of lengths between 70 and 1000. For each triplet, the Affine-DO, the LP bound, and the exact solution were computed. These three solutions were compared to provide an experimental overview of the potential performance of our algorithm. We selected random sequences because...
preliminary experiments showed evidence that these produce the most difficult instances for Affine-DO.

**Algorithms compared**

We implemented a number of algorithms to approximate the tree alignment problem. Our implementation can be divided into two groups: initial assignment, and iterative improvement.

**Initial Assignment** includes the Fixed States (a stronger version of the Lifted Assignment [34,43]), Direct Optimization [15], and Affine-DO. Each of these algorithms starts with a function $\chi$ and creates a $\chi'$ compatible with $\chi$ which is an instance solution. DO and Affine-DO have already been described. The Fixed States [43] is a simple algorithm where the interior vertices are optimally assigned one of the leaf sequences of the input tree, yielding a 2-approximation solution [34].

**Iterative Improvement** modifies an existing $\chi'$ by readjusting each interior vertex using its three neighbors. This procedure is repeated iteratively, until a (user provided) maximum number of iterations is reached, or no further tree cost improvements can be achieved. The adjustment itself can be done using an approximated or an exact three dimensional alignment, which we call the Approximate Iterative and Exact Iterative algorithms. Approximate Iterative (Figure 5), uses DO or Affine-DO (the selection depends on which kind of edit distance function is used) to solve the TAP on the three possible rooted trees formed by the three neighbors of the vertex used as leaves. The assignment yielding the best cost is selected as the new center. The exact three dimensional alignment has time complexity $O(n^2)$ [44]. Our implementation uses the low memory algorithms implemented by Powell [44], though they can be improved to $O(n^2)$ memory consumption [45].

We compared MSAM [32], Affine-DO, Approximate Iterative, Exact Iterative, and Fixed States, using a lower bound computed with an LP solution. We do not include DO in the comparisons because it could not solve this problem [46]. It is therefore impossible to compare it directly with our algorithm. GESTALT, SALSA, and PRODALI were unavailable, and so, could not be used in our comparative evaluation. TreeAlign did not produce a solution for the simulations within 48 hours of execution time, and therefore, was not included in the comparisons.

In total, more than 330,000 solutions were evaluated. We only present those results that show significant differences, and represent the overall patterns detected. The Exact Iterative algorithm was only evaluated for the short sequences (70 to 100 bases), due to the tremendous execution time it requires. Fixed States followed by iterative improvement is not included because its execution time is prohibitive for this number of tests (POY version 4 supports this type of analysis). Nevertheless, preliminary analyses showed that this combination of algorithms produce results in between Fixed States and Affine-DO, but not competitive with Affine-DO.

| Subst. | Gap Op. | Branch Len. | Algorithm          | Min.   | Median  | Max   |
|--------|---------|-------------|-------------------|--------|---------|-------|
|        |         |             | Simulated         | 1.088  | 1.218   | 1.337 |
| 1      | 0       | 0.05        | Fixed States      | 1.275  | 1.534   | 1.755 |
|        |         |             | ADO               | 1.044  | 1.148   | 1.215 |
|        |         |             | ADO + Iter.       | 1.044  | 1.123   | 1.202 |
|        |         |             | Simulated         | 1.731  | 2.022   | 2.396 |
| 1      | 0       | 0.3         | Fixed States      | 1.621  | 1.725   | 1.816 |
|        |         |             | ADO               | 1.314  | 1.398   | 1.453 |
|        |         |             | ADO + Iter.       | 1.300  | 1.377   | 1.393 |
|        |         |             | Simulated         | 1.108  | 1.262   | 1.415 |
| 4      | 3       | 0.05        | Fixed States      | 1.302  | 1.557   | 1.766 |
|        |         |             | ADO               | 1.084  | 1.208   | 1.312 |
|        |         |             | ADO + Iter.       | 1.067  | 1.171   | 1.283 |
|        |         |             | Simulated         | 2.012  | 2.284   | 2.611 |
| 4      | 3       | 0.3         | Fixed States      | 1.688  | 1.795   | 1.868 |
|        |         |             | ADO               | 1.433  | 1.500   | 1.542 |
|        |         |             | ADO + Iter.       | 1.388  | 1.442   | 1.453 |

Each individual indel has cost 1.
Algorithm Comparison

The most important patterns observed between the evaluated algorithms are presented in Figure 6 and Table 2. In general, Affine-DO yields a better approximation than Fixed States. According to the density histograms (data not shown), the expected approximation ratio of 1.1 (versus 1.5 for Fixed States) in the best parameter combination, and 1.5 (versus 1.7) for the worst. Iterative improvement (both in exact and approximated forms) has a small overall impact in the approximation ratio (with a maximal decrease of 0.05 when compared with the solution inferred by Affine-DO alone). In all cases, Affine-DO found better solutions than the simulations (Table 2).

Although the combination of Affine-DO and Iterative improvement produces better solutions, its execution time is dramatically higher. In the current implementation, running on a 3.0 Ghz, 64 bit Intel Xeon 5160 CPU with 32 GB of RAM, Affine-DO evaluates each tree in less than 1 second in the worst case, while Affine-DO + Iterative improvement may take more than 1 hour per tree. For this reason, Affine-DO is well suited for heuristics that require a very large number of tree evaluations such as the GTAP, where millions of trees are evaluated during a heuristic search.

Approximation of Affine-DO

Figure 7 shows the density histogram of the guaranteed approximation of the Affine-DO algorithm when compared with the LP theoretical solution for a representative

![Figure 7 Affine-DO vs. Theoretical LP bound. Guaranteed approximation ratio of Affine-DO compared with the theoretical LP bound, for different cost and sequence generation parameters. a. substitutions = 1, a = 0, b = 1. b. substitutions = 2, a = 1, b = 1. c. substitutions = 4, a = 1, b = 3.]

![Figure 8 Affine-DO vs. Theoretical LP bound with random sequences. Guaranteed approximation of Affine-DO for random sequences. In the left substitutions=1, a = 0, b = 2, in the center substitutions=1, a = 0, b = 1, and in the right substitutions=2, a = 1, b = 1. These are representative of the distributions observed in the experiments.]

...
set of parameters. The results show that Affine-DO has a guaranteed approximation of less than 60% in every case.

Typically, the larger the sequence divergence, the larger is the approximation degree of Affine-DO. The same pattern is observed for larger $a$. To test an extreme case, were the branch length is maximal, we evaluated the behavior of random sequences in the same set of trees. Figure 8 shows the results of this experiment.

The worst case is observed with an average approximation slightly over 1.5. This variation, however, could have been caused by a more relaxed LP bound, which could be producing an overly pessimistic evaluation of the algorithm. To assess the importance of this factor, we evaluated its tightness experimentally.

**Comparison with an exact solution**

To assess Affine-DO and the tightness of the LP bound, we computed the exact solution for 700 unrooted trees consisting of 3 leaves with random sequences assigned to their leaves, under all the parameter sets tested. Figure 9 shows the density histograms for the results obtained.

Note that the LP-inferred bound is overly negative even for these small test data sets, with the inferred approximation expected at around 1.15, while in reality Affine-DO finds solutions that are expected to approximate within 1.05 of the optimal solution, a 10% difference for trees consisting of only 3 sequences.

**Conclusions**

We have presented a novel algorithm that we have called Affine-DO for the TAP under affine gap costs. Our experimental evaluation, the largest performed for this kind of problem, shows that Affine-DO performs better than Fixed States. However, we observed that the LP bound is too pessimistic, producing unfeasible solutions 10% worse, even for the smallest non-trivial tree consisting of 3 leaves. Based on these observations, we believe that Affine-DO is producing near-optimal solutions, with approximations within 10% for sequences with small divergence, and within 30% for random sequences, for which Affine-DO produced the worst solutions.

Affine-DO is well suited for the GTAP under affine sequence edit distances, and yields significantly better results when augmented with iterative methods. The main open question is whether or not there exists a guaranteed bound for DO or Affine-DO. Then, if the answer is positive, whether or not it is possible to improve the PTAS using these ideas. Additionally, many of these ideas can be applied for true simultaneous tree and alignment estimation under other optimality criteria such as ML and MAP. Their use under these different optimality criteria remains to be explored.
Competing interests
The authors declare that they have no competing interests.

Authors’ contributions
WW defined the Fixed States and DO algorithms. AV developed Affine-DO and performed all analyses under supervision of WW. AW and WW wrote and revised the manuscript. Both authors read and approved the final manuscript for publication.

Acknowledgements
This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under grant number W911NF-05-J-0271.

Received: 17 May 2012  Accepted: 22 October 2012  Published: 9 November 2012

References
1. Thompson JD, Higgins DG, Gibson TJ. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions–specific gap penalties and weight matrix choice. Nucl Acids Res 1994, 22:4673–4680.
2. Morgenstern B. CLUSTALW: Improving the accuracy of multiple sequence alignment in constraint-based approaches. Bioinformatics 1999, 15:211–218.
3. Katoh K, Misawa K, Kuma K, Miyata T. MAFFT: a novel method for rapid multiple sequence alignment. Nucl Acids Res 2002, 30:3059–3066.
4. Edgar RC. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics 2004, 5:113.
5. Fleissner R, Metzler D, von Haeseler A. Simultaneous Statistical Multiple Alignment and Phylogeny Reconstruction. Syst Biol 2005, 54(4):548–561.
6. Redelings BD, Suchard MA. Joint Bayesian Estimation of Alignment and Phylogeny. Syst Biol 2005, 54:401–418.
7. Do CB, Mahabhashyam MSP, Brudno M, Batzoglou S. ProbCons: Probabilistic consistency-based multiple sequence alignment. Genome Res 2005, 15:330–340.
8. Wheeler WC. Dynamic Homology and the Likelihood Criterion. Cladistics 2006, 22:157–170.
9. Nelesen S, Liu K, Zhao D, Linder CR, Warnow T. The effect of the guide tree on multiple sequence alignments and subsequent phylogenetic analyses. PLoS Comput Biol 2008, 4:253–36.
10. Sankoff D. Minimal Mutation Trees of Sequences. SIAM J Appl Mathematics 1975, 28:35–42.
11. Sankoff D, Cedergren RJ, Lapalme G. Frequency of Insertion-Deletion, Transversion, and Transition in the Evolution of 5S Ribosomal RNA. J Mol Evol 1976, 10:33–149.
12. Sankoff D, Cedergren RJ. Simultaneous Comparison of Three or More Sequences Related by a Tree. Discret Appl Math 1990, 183:626–645.
13. Hein J. A New Method That Simultaneously Aligns and Reconstructs Ancestral Sequences for Any Number of Homologous Sequences, When The Phylogeny is Given. Mol Biol Evol 1989, 6:649–668.
14. Hein J. Unified approach to alignment and phylogenies. Methods in Enymology 1999, 305:262–645.
15. Wheeler WC. Optimization Alignment: The End of Multiple Sequence Alignment in Phylogenetics? Cladistics 1996, 12:1–9.
16. Cartwright RA. Logarithmic gap costs decrease alignment accuracy. BMC Bioinformatics 2006, 7:527–539.
17. Liu K, Nelesen S, Raghavan S, Linder CR, Warnow T. Barking up the wrong treelength: the impact of gap penalty on alignment and tree accuracy. IEEE Trans Comput Biol Bioinf 2009, 6:7–21.
18. Waterman MS, Smith TF, Beyer WA. Some biological sequence metrics. Advances in Mathematics 1976, 20(3):367–387. [http://www.sciencedirect.com/science/article/B6W6F-4CRY725-1TA/1/ad09f0468f8302917f17dca45d4d801]
19. Benner SA, Cohen MA. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. J Mol Evol 1993, 229:1005–1082.
20. Gu X, Li WH. The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. J Mol Evol 1995, 40(4):464–473. [http://dx.doi.org/10.1007/BF00164032]
21. Zhang Z, Gerstein M. Patterns of nucleotide substitution, insertion and deletion in the human genome inferred from pseudogenes. Nucl Acids Res 2003, 31(18):S339–S348. [http://nar.oxfordjournals.org/cgi/content/abstract/31/18/S338]
22. Chang MSS, Benner SA. Empirical Analysis of Protein Insertions and Deletions Determining Parameters for the Correct Placement of Gaps in Protein Sequence Alignments. J Mol Biol 2004, 341(2):617–631. [http://www.sciencedirect.com/science/article/B6WKJ-4CNHJDH-6-2/0/9eb74262076e6105d3e2904154328020c]
23. Wheeler WC, Gladstein D, De Laet. POY, Phylogeny Reconstruction via Optimization of DNA and other Data version 3.0.11 (May 6 of 2003). American Museum of Natural History, 2003. [ftp://ftp.amnh.org]
24. Varón A, Vinh LS, Wheeler WC. POY version 4: phylogenetic analysis using dynamic homologies. Cladistics 2009, 25:72–85.
25. Lancia G, Ravi R. GESTALT: Genomic steiner alignments. Lecture Notes in Computer Science 1999, 1645:101.
26. Lancia G, Ravi R. SALS A: Sequence alignment via Steiner Ancestors. 2008. [http://citeseer.ist.psu.edu/356333.html].
27. Schwikowski B, Vingron M. Weighted sequence graphs: boosting iterated dynamic programming using locally suboptimal solutions. Algorithmica 2003, 37:297–317.
28. Ogden TH, Rosenberg MS. Alignment and Topological Accuracy of the Direct Optimization approach via POY and Traditional Phylogenetics via ClustalW + PAUP*. Syst Biol 2007, 56(2):182–193.
29. Lehtonen S. Phylogeny Estimation and Alignment via POY versus Clustal + PAUP*: A Response to Ogden and Rosenberg (2007). Syst Biol 2008, 57:665–667.
30. Wheeler WC. Sequence Alignment, edited by M. S. Rosenberg. Berkeley, CA: USA: University of California Press, 2009. chap. Simulation Approaches to Evaluating Alignment Error and Methods for Comparing Alternate Alignments. 179–208.
31. Wang L, Jiang T. On the Complexity of Multiple Sequence Alignment. J Comput Biol 1994, 1:337–348.
32. Yue F, Shi L, Tang J. Simultaneous phylogeny reconstruction and multiple sequence alignment. BMC Bioinf 2009, 10(Suppl 1):S11.
33. Schwikowski B, Vingron M. The deferred path heuristic for the generalized treealignment problem. In RECOMB ’07: Proceedings of the first annual international conference on Computational molecular biology. New York, NY, USA: ACM Press, 1997:257–266. [http://doi.acm.org/10.1145/635251.635284].
34. Wang L, Jiang T, Lawler EL. Approximation Algorithms for Tree Alignment with a Given Phylogeny. Algorithmica 1996, 16:302–315.
35. Wang L, Gusfield D. Improved Approximation Algorithms for Tree Alignment. J Algorithms 1997, 25(2):255–273.
36. Ravi R, Recevic I. Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree. Discret Appl Math 1998, 88:355–366.
37. Wang L, Jiang T, Gusfield D. A More Efficient Approximation Scheme for Tree Alignment. SIAM J Comput 2000, 30:283–299.
38. Wheeler WC, Aagesen L, Arango CP, Faivovich J, Grant T, D’Haese C, Janies D, Smith MW, Varón A, Giribet G. Dynamic Homology and Phylogenetic Systematics: A Unified Approach using POY. American Museum of Natural History: 2006 pp. 355.
39. Needleman SB, Wunsch CD. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. J Mol Biol 1970, 48:443–453.
40. Gotch O. An extended algorithm for matching biological sequences. J Mol Biol 1972, 68:705–708.
41. Lifton E. Algorithms for approximate string matching. Inf Control 1985, 64:1(3):100–118.
42. Cartwright RA. DNA Alignment with gaps (Dawg): simulating sequence evolution. Bioinformatics 2005, 21(Suppl 3):i31–i38.
43. Wheeler WC. Fixed Character States and the Optimization of Molecular Sequence Data. Cladistics 1999, 15:379–385.
44. Powell CR, Allison J, Dix TI. Fast optimal alignment of three sequences using linear gap costs. J Theor Biol 2000, 207:325–336.
45. Yue F, Tang J. A divide-and-conquer implementation of three sequence alignment and ancestor inference with affine gap costs. In The IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2007), 143–150.

46. Varón A, Wheeler WC. Application note: on extension gap in POY version 3. Cladistics 2008; 24(6):1070–1070.

doi:10.1186/1471-2105-13-293

Cite this article as: Varón and Wheeler. The tree alignment problem. BMC Bioinformatics 2012 13:293.