A Linear-Time Algorithm for the Multiple Gene Duplication Problem

Vacharapat Mettanant Jittat Fakcharoenphol

Department of Computer Engineering
Kasetsart University, Bangkok, 10900 Thailand.
E-mails: kidkus28@hotmail.com, jittat@gmail.com

Abstract

Reconstructing the Tree of Life is one of the most important research goals in evolutionary biology. In this paper, we consider one of the issues regarding inconsistency among the trees reconstructed using sequences from different genes, called gene trees. This inconsistency is often a result of gene duplication. Given a species tree and a set of gene trees, the Multiple Gene Duplication Problem as formulated by Guigo, Muchnik, and Smith asks for an assignment of gene duplication events on the species tree node to minimize the number of duplication episodes. Recently, Bansal and Eulenstein give the first efficient algorithm for solving that problem. Their algorithm runs in quadratic time. In this paper we give the first linear-time algorithm for the problem. Our algorithm can be seen as a faster implementation of Bansal and Eulenstein's method. We also present a more practical algorithm that runs in near-linear time. Experiments on synthetic data show a very good improvement on the running time (at least twice faster).

1 Introduction

In evolutionary biology, reconstructing the evolutionary history of a set of species is one of the most important problem. Usually the evolutionary process is modeled as a phylogenetic tree, a tree whose leaves represent species and internal nodes represent ancestral species.

All methods for reconstructing a phylogenetic tree of a given set of species employ some measure of similarity based on their genomic data, which are either sequence data or, not until recently, gene-order data (see survey in [12]). A vast number of research work has been conducted regarding reconstruction based on sequence data. Under certain evolutionary models the reconstruction can be done very efficiently in terms of sequence length [4, 11]. Also, a large number of tools for infering a phylogenetic tree from sequence data have been developed [16, 15, 6].

The problem arises when there are many sets of sequence data. It is well observed that phylogenetic trees constructed from sequences of different genes are not compatible (see, e.g., in [5, 9]). The main reason for this inconsistency is an important evolutionary event called *gene duplication* (see review on biological background in Section 2). Guigo, Muchnik, and Smith [7] are among the first who address this problem and formulate the tree reconstruction problem from a set of inconsistent trees. Their goal is to find the "best" species tree from a set of inconsistent trees obtained from different sets of gene sequence data, called gene trees. The best species tree is the tree which requires the smallest number of gene duplication events needed to explain all the gene trees.

There are many attempts to capture the notion of smallest number of events (see discussion in [1]). Recently Bansal and Eulenstein [1] formally define the notion of *multiple gene duplication episodes* and define the *Multiple Gene Duplication Problem* that aims to find the assignment of duplication events on gene tree nodes to nodes in the species tree with the minimum number of multiple gene duplication episodes. Bansal and Eulenstein also give the first efficient algorithm that solves the multiple gene duplication problem.

While the algorithm of Bansal and Eulenstein is efficient and runs very quickly on small inputs, its running time is still quadratic (see Section 3, for the actual running time). The need for faster algorithm does not come only from the theoretical point of view. The problem of finding the minimum number of duplication episodes between a given species tree and gene trees in itself is not very important, but it is used (see [7] and mentioned also in [1]) as a subroutine in many local search algorithms for finding the best species tree that best explains the set of gene trees. This later problem is called Gene-Tree Reconciliation, which is very important (see, e.g., [10, 13]).

1.1 Our contribution

In this paper we give a linear-time algorithm for the Multiple Gene Duplication Problem. Our algorithm is essentially a faster implementation of Bansal and Eulenstein's idea.

While our first algorithm runs in linear time, it is quite difficult to implement. Therefore, we also provide another way to implement our algorithm which

AAAA GACTGACAGC ATTGAGAG
duplication event ⇓
AAAA GACTGACAGC GACTGACAGC ATTGAGAG

Figure 1: An example showing gene duplication.

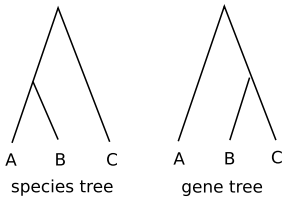

Figure 2: An inconsistent gene tree.

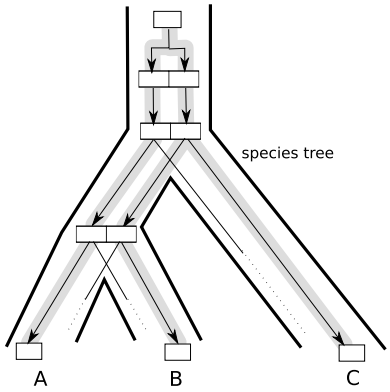

Figure 3: A duplication event on a species tree.

is easier to program and runs in near-linear time. We have conducted some experiments. The results shows that our near-linear-time algorithm runs twice faster than the original algorithm of Bansal and Eulenstein even on small inputs. For larger inputs, the speed ups also increase.

2 Preliminaries

In this section we review some biological background and later provide basic notations and definitions.

2.1 Biological background

We briefly sketch the background on gene duplication and its effect on gene-tree species-tree inconsistency. The paper of Guigo *et.al.* [7] serves well as a good starting point for further references.

Gene duplication is the event that some region of DNA that contains a gene gets duplicated (see Figure 1).

To see how gene duplication causes gene-tree-species-tree inconsistency, we reproduce here a classic example taken from [7]. Figure 2 shows a species tree and an inconsistent gene tree, which shows that under that particular gene, species B is closer to species C than to species A.

Figure 3 illustrates how gene duplication explains this inconsistency. Consider gene g, shown as rectangles. First, at the root of the species tree gene duplication occurs; now the top-most species has two copies of gene g. Both copies evolve independently. Next, species C splits off from the AB evolutionary path; along the way to become species C, the first copy of the g is lost. On the AB path later species A and B split; while species A loses the second copy of the g and B loses the first copy. Now, since species B and C have the same copy of the gene after the duplication, they look closer to each other than to species A. The final reconstructed tree using this gene sequence is then shown as thick shaded lines.

2.2 Definitions

We work mainly on full binary trees, binary trees whose nodes are either leaf nodes or internal nodes with exactly two children. Let T be a full binary tree. We use $V(T), E(T), L(T)$, and $r(T)$ to denote the node set, the edge set, the leaf set, and the root node of T, respectively. For a node u in T, we denote by T_u the subtree of T rooted at u. For any tree T, we denote by $h(T)$ the height of T, i.e., the number of nodes on the longest path from its root to some leaf.

There are two types of trees in this paper. A *species tree* is a binary tree of which each leaf is labeled with a distinct species. A species tree represents evolutionary relationship among species. A *gene tree* is also a binary tree representing, for a given gene family, the evolutionary relationship among genes in the given set of species; thus, each leaf in a gene tree represents a gene in a particular species. We therefore label each leaf on a gene tree with an associated species of that gene. Since we use binary trees to represent evolutionary history, internal nodes represents ancient species (for species trees) or ancestral gene (for gene trees).

Given a gene tree G and a species tree S, to compare these two trees we shall map each node u in G to some node in S. If u is a leaf node, we map u to the corresponding species in S. When u is an internal node, recall that u is an ancestral gene for all leaves of subtree G_u, i.e., genes in $L(G_u)$; thus a species v that contains u must also be an ancestor of all species that contain all these genes. This motivates the following definition of the least-common-ancestor (LCA) mapping.

Given a gene tree G and a species tree S, an LCA mapping $LCA : V(G) \rightarrow V(S)$ can be defined as follows.

Definition 1 (LCA mapping) *For a node $u \in V(G)$, $LCA(u)$ is a node $v \in V(S)$ which has the most distance from $r(S)$ such that $L(G_u) \subseteq L(S_v)$.*

From the definition, for a gene node u, $LCA(u)$ is either $LCA(P(u))$ or a descendant of $LCA(P(u))$, where $P(u)$ is the parent of u. If $LCA(u)$is a descendant of $LCA(P(u))$, the mutation from $P(u)$ to

u can explain a path of speciations from $LCA(P(u))$ to $LCA(u)$. If $LCA(u) = LCA(P(u))$, it means that there is a gene mutation which has no corresponding speciation. This leads us to get the concept of gene duplications to explain the event.

Given an LCA mapping LCA, a node u in a gene tree G is called a *duplication node* if $LCA(u) = LCA(u')$ for some u' having u as the parent.

It is not difficult to find the set $Dup(G, S)$ of all duplication nodes when a gene tree G and a species tree S are given. Nevertheless, for a duplication node u, the duplication event may not happen in the species of $LCA(u)$. Hence, we will consider where each duplication event appears in S.

For nodes $u, v \in V(S)$ where u is a descendant of v, we denote $Mid(u, v)$ to be the set of all nodes that are ancestors of u and are descendants of v. In the other words, $Mid(u, v)$ is the set of all nodes between u and v in S.

Definition 2 *For a duplication node $u \in Dup(G, S)$, we define the set of possible locations in S of the event at u (called the interval $I(u)$) to be*

- $\{LCA(u)\} \cup \{v \in V(S) | v$ *is an ancestor of* $LCA(u)\}$ *if u is a root node,*
- $\{LCA(u)\} \cup Mid(LCA(u), LCA(P(u)))\}$ *otherwise.*

For a node $u \notin Dup(G, S)$, we define $I(u) = \{LCA(u)\}$.

We say that a mapping M from $V(G)$ to $V(S)$ is *valid* if for each $u \in V(G)$, $M(u) \in I(u)$

In our problem, we are given k gene trees: $G_1, G_2, \ldots, G_k$. Given a mapping from each gene tree to a species tree S, we shall let a mapping M from the union of all gene trees to a species tree be a union of all such mappings.

For a node v in S, denote by $M^{-1}(v)$ the set $\{u | M(u) = v\}$, i.e., the set of nodes in gene trees having v as an image from M. We also let $F_{M^{-1}}(v)$ be the forest in $\cup_i G_i$ induced by nodes in $M^{-1}(v)$. For a given forest F, let the height $h(F)$ be the maximum height of any trees in F.

Given k gene trees $G_1, G_2, \ldots, G_k$, and a species tree S, the *Multiple Gene Duplication Problem* is to find a valid mapping M such that

$$\sum_{v \in V(S)} h(F_{M^{-1}}(v))$$

is minimized.

This problem can be solved in polynomial time. The first efficient algorithm was proposed by Bansal and Eulenstein [1].

3 The Bansal-Eulenstein algorithm

In this section we describe the Bansal-Eulenstein algorithm and review its running time analysis. We first introduce more notations.

The Bansal-Eulenstein algorithm
input: gene trees $G_1, ..., G_k$ and a species tree S

1: Initialize M to be the LCA mapping from each G_i to S.
2: Compute $I(u)$ for each node u in the gene trees.
3: **for** each node $v \in S$ in a post-order traversal **do**
4: **if** $|M^{-1}(v)| > 0$ **then**
5: **if** all leading nodes in $M^{-1}(v)$ are free **then**
6: Update M by moving the maps of all these leading nodes to $P(v)$.
7: **end if**
8: **end if**
9: **end for**
10: Return M.

Figure 4: the Bansal-Eulenstein algorithm.

For a mapping M and a node $v \in V(S)$, we call a node $u \in M^{-1}(v)$ a *leading node* if u is a root node of one of the highest trees in $F_{M^{-1}}(v)$. For a node $u \in M^{-1}(v)$, if the parent of v is in $I(u)$, we call u a *free node*.

The Bansal-Eulenstein algorithm is shown in Figure 4.

3.1 Algorithm description

The algorithm maintain a valid mapping M from nodes in gene trees to nodes in the species tree. Initially, it set M to be the LCA mapping. The algorithm proceeds by considering each node v in the species tree in the same order as in the post-order tree traversal. It then modifies the mapping M for all leading nodes u in $M^{-1}(v)$ if they are all free. The new mapping M' maps u to $P(v)$.

Bansal and Eulenstein [1] proved that this algorithm produces the minimum cost valid mapping. We refer to their paper for proofs of correctness.

3.2 Running time

In order to analyse the running time of the Bansal-Eulenstein algorithm, we denote by n the number of species, and m_i the number of genes in G_i. We let $m = \sum_{i=1}^{k} m_i$.

Bansal and Eulenstein give a very crude analysis of the running time. During the preprocessing, the algorithm computes the LCA mapping and, for each node u in any gene trees, the interval $I(u)$. The interval $I(u)$ is needed so that the algorithm can verify if u is free in $O(1)$ time.

The running time of the Bansal-Eulenstein is dominated by Steps 4-8. For each node v in Step 3 of the algorithm, it looks at all inverse image $M^{-1}(v)$. Since the upper bound on the size of $M^{-1}(v)$ is $\sum_{i=1}^{k} m_i = m$, the algorithm spends $O(m)$ time for each node v; thus, this leads to the $O(mn)$-time bound.

4 A faster algorithm

In this section we present a faster algorithm for the Multiple Gene Duplication Problem.

As in the Bansal-Eulenstein algorithm, our algorithm starts by computing the LCA mapping from every gene tree nodes to nodes in the species tree. Given that mapping $I(u)$ for each node u in a gene trees can be computed very easily. These steps can be implemented to run very efficiently; Section 5 discusses some of the implementation.

We give a faster implementation for the map improvement steps (i.e., Steps 3-9). Instead of recomputing and checking every node in $M^{-1}(v)$ at every step, we only look at nodes in gene trees when we really have to.

The following properties regarding how many steps a leading node can move up the tree are very crucial to our analysis.

Lemma 1 *Let u be a free leading node in $M^{-1}(v)$ for $v \in V(S)$ and a valid mapping M such that all leading nodes in $M^{-1}(v)$ are free, and let M' be M after moving the maps of all the leading nodes from v to $P(v)$. u is a one-node tree in $F_{M'^{-1}}(P(v))$.*

Proof: Let c be a child of u in the original gene tree. There are two cases of c: $c \in M^{-1}(v)$ or $c \notin M^{-1}(v)$. If $c \in M^{-1}(v)$, it means that c is also a child of u in $F_{M^{-1}}(v)$ and cannot be a leading node. The map of c in M cannot move to $P(v)$ for the step. If $c \notin M^{-1}(v)$, from the definition of a valid mapping, $M(c)$ is a descendant of v, and so is each node in $I(c)$. Therefore, $P(v)$ is not an element in $I(c)$ so the map of c cannot move to $P(v)$, or else, it leads to a non-valid mapping.

From both cases, there is no child of u appearing in $F_{M'^{-1}}(P(v))$, then u is a one-node tree. ∎

Lemma 2 *Let $LN(v)$ be the set of all leading nodes in $M^{-1}(v)$, the number of steps in which each node in $LN(v)$ can move its map is less than or equal the minimum size of $I(w)$ minus one for all w in $LN(v)$. In the other words, each node u in $LN(v)$ cannot move its map more than $\min_{w \in LN(v)}(|I(w)| - 1)$ times.*

Proof: In the Bansal-Eulenstein algorithm, the maps of nodes in $LN(v)$ are moved only if all of them are free. After moving, either they all are still leading nodes or none of them are. It means that the moving of the maps of all nodes in $LN(v)$ will stop at the same time. Therefore, the number of moving steps is the least of all, which can be at most $\min_{w \in LN(v)}(|I(w)| - 1)$ (the number of possible locations leftover). ∎

From Lemma 1, We can see that after a leading node moves its map up from v to $P(v)$, it will be a leading node of $P(v)$ if and only if all the leading nodes of $P(v)$ are roots of one-node trees, or there is no node mapping to $P(v)$. If it is still a leading node after moving, lemma 2 will helps us to decide if we can move its map again. This leads us to a new algorithm presented in Figure 5.

Algorithm 2.
input: gene trees $G_1, ..., G_k$ and a species tree S

1: Initialize M to be the LCA mapping from each G_i to S.
2: Compute $|I(u)|$ for each node u in the gene trees.
3: Compute $LN(v)$ for each $v \in S$, and Let $H(v) = h(F_{M^{-1}}(v))$.
4: Compute the number of steps which all nodes in $LN(v)$ can move in, denoted by $Step(v) = \min_{u \in LN(v)}(|I(u)| - 1)$ for each $v \in S$.
5: **for** each node $v \in S$ in a post-order traversal **do**
6: **if** $LN(v) \neq \emptyset$ and $Step(v) > 0$ **then**
7: Update M by moving the maps of all nodes in $LN(v)$ from v to $P(v)$.
8: $Step(v) \leftarrow Step(v) - 1$
9: **if** $H(P(v)) = 1$ **then**
10: $Step(P(v)) \leftarrow$
11: $\min(Step(P(v)), Step(v))$
12: $LN(P(v)) \leftarrow LN(P(v)) \cup LN(v)$
13: **else if** $H(P(v)) = 0$ **then**
14: $Step(P(v)) \leftarrow Step(v)$
15: $LN(P(v)) \leftarrow LN(v)$
16: $H(P(v)) \leftarrow 1$
17: **end if**
18: **end if**
19: **end for**
20: Return M.

Figure 5: a faster algorithm

In the initial part in Algorithm 2, we add the computing of leading nodes and the numbers of steps. For each map improvement step, we check whether there are leading nodes and whether they can move their maps by looking the number of steps they can move (the value of $Step$). If the leading nodes can move their map up, we instantly check whether they will be leading nodes again. If they will, we then update the set of leading nodes of the new species node and update the number of steps for the new leading node set. We will show that for each species node, we can check all these conditions in constant time.

Theorem 1 *Algorithm 2 always produces an optimal valid mapping for the Multiple Gene Duplication Problem.*

Proof: Since Algorithm 2 looks and updates leading nodes in the same way as the Bansal-Eulenstein algorithm, we can see that Algorithm 2 produces the same output as the Bansal-Eulenstein algorithm. Then, because the Bansal-Eulenstein algorithm solves this problem, Algorithm 2 does too. ∎

Now, we consider the running time of Algorithm 2. Let n be the number of species and m be the number of all genes.

Theorem 2 *The running time of Algorithm 2 is $O(m + n)$.*

Proof: Consider Step 1 in the algorithm. The LCA mapping can be constructed in $O(m + n)$ by using

$O(n)$ for preprocessing and $O(m)$ to map each node in gene trees to a node in the species tree. In Step 2, we do not need to find the set $I(u)$ because we use only its size in the algorithm. Computing the size of each $I(u)$ can be done in $O(m)$ if we know the level of each node in the species tree. Steps 3-4 can also be computed in $O(m+n)$. Now we consider each loop from Step 5 to 18. In line 7, we can update M in $O(1)$ time using a list data structure (see Section 5). So each loop takes $O(1)$ time. Because there are $O(n)$ loops, line 5 to 18 take $O(n)$ time. Therefore, Algorithm 2 takes the running time of $O(m+n)$. ∎

5 Practical implementation

Algorithm 2 looks simple to implement if we can find the LCA mapping and can update maps efficiently. We describe in this section how to compute the LCA mapping, and then how to update maps in Step 7 of Algorithm 2 in constant time.

The problem of finding lowest common ancestors has been studied for long time. Harel and Tarjan were the first to develop a data structure for this problem, based on union-find [8]. Nowadays, there are many algorithms that take $O(n)$ time for preprocessing a tree of n nodes, and answer an LCA query in constant time [2, 3, 14]. However, those algorithms are so complicated. In our implementation, we choose a simpler algorithm which takes $O(n \log n)$ time for preprocessing and answer each query in constant time. This algorithm was shown in [2]. For the LCA mapping in our problem, we run the preprocessing step and find the map from each node in gene tree. From the fact that $LCA(u)$ is the lowest common ancestor of $LCA(u_l)$ and $LCA(u_r)$ where $u = P(u_l) = P(u_r)$, we can use $O(1)$ time to find the map for each gene node using a post-order traversal. Therefore, computing LCA mapping can be done in $O(m + n \log n)$ time.

For the map improvement steps, we use two linked lists for each species node v, called *moving list* and *stopped list*. The moving list contains all leading nodes of v that still can move their maps over. The stopped list contains the other nodes mapping to v. When we want to update the maps of the leading nodes of v, we get the head of the moving list to point to the tail of the moving list of $P(v)$ if the leading nodes still can move up their maps, and to the stopped list of $P(v)$ otherwise. These steps can be done in constant time.

When the map improvement steps are finished, we shall update the real map from each gene node u to the species node having u in the lists. Each species node v is considered and all nodes in both lists of v will map to v. All these steps can be done in $O(m+n)$ time. So, our implementation takes $O(m+n \log n)$ time for computing the LCA mapping and takes $O(m+n)$ time for the other steps.

Table 1: Running Time of BansalOpt and LinearOpt (in seconds)

Number of species	BansalOpt	LinearOpt
500	1.32	0.71
1000	3.02	1.51
1500	4.63	2.29
2000	7.18	3.40
3000	10.99	4.73
4000	61.43	9.59

6 Experimental results

In our experiment, we implemented Algorithm 2 and compared its running time with the Bansal-Eulenstein algorithm. The LCA mapping was implemented as described in Section 5. The input datasets were generated randomly when the number of species and gene trees had been given.

For a species tree, after the number of species (n) had been given, we created a set of n elements to be the leaves. We then randomly picked up two elements in this set and create a new element of the set to be their parent. When we did it until the set had one element, the species tree were constructed. To generate a gene tree, we randomized a number for each species. This number was used to be how many genes in the gene tree are found in this species. When we had got a set of genes, we could randomly build a gene tree the same way as the species tree.

We tested on six sizes of datasets. Each of the datasets had 200 gene trees and a species tree. For each gene tree we allowed a species to have from 0 to 10 genes. It means that the number of genes can be 2000 times as many as the number of species. The number of species was varied in six values. For a number of species, we randomly generated five datasets and compute the average time of each algorithm. Table 1 shows the running time of the program of the Bansal-Eulenstein algorithm, denoted by BansalOpt, and the program of Algorithm 2, denoted by LinearOpt. For all experiment, we used Windows XP operating system on a 3.40GHz Intel(R) Pentium(R) D CPU with 1 GB of RAM.

7 Conclusions and open problems

In this paper, we present a linear-time algorithm for the Multiple Gene Duplication Problem. We also show a practical implementation which runs in near-linear time. Experiments show that our algorithm runs faster than the previous algorithm by Bansal and Eulenstein [1].

While our algorithm can be used to speed up various heuristics for finding good reconciled trees, only few special cases for the tree reconciliation problems have provably-good methods [10]. This is one of many important open problems in this area.

References

[1] Mukul S. Bansal and Oliver Eulenstein. The multiple gene duplication problem revisited. *Bioinformatics (Oxford, England)*, 24(13), July 2008.

[2] Michael A. Bender and Martin Farach-Colton. The lca problem revisited. In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, London, UK, 2000. Springer-Verlag.

[3] Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, and Jeffery R. Westbrook. Linear-time pointer-machine algorithms for least common ancestors, mst verification, and dominators. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 279–288, New York, NY, USA, 1998. ACM.

[4] Constantinos Daskalakis, Elchanan Mossel, and Sébastien Roch. Optimal phylogenetic reconstruction. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 159–168, New York, NY, USA, 2006. ACM Press.

[5] K. G. Field, G. J. Olsen, D. L. Lane, Giovanni, Ghiselin, Raff, Pace, and Raff. Molecular phylogeny of the animal kingdom. *Science*, 239:748–753, 1988.

[6] P. Goloboff, S. Farris, and K. Nixon. Tnt (tree analysis using new technology), 2000. (BETA) Published by the authors, Tucum.

[7] Roderic Guigo, Ilya Muchnik, and Temple F. Smith. Reconstruction of ancient molecular phylogeny. *Molecular Phylogenetics and Evolution*, 6:189–213(25), October 1996.

[8] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[9] J. A. Lake. Origin of the metazoa. *Proc. Natl. Acad. Sci.*, 1987.

[10] Bin Ma, Ming Li, and Louxin Zhang. From gene trees to species trees. *SIAM Journal on Computing*, 30(3):729–752, 2001.

[11] R. Mihaescu, D. Adkins, C. Hill, A. Jaffe, and S. Rao. A simple polynomial time algorithm for reconstructing all phylogenies from log-sized sequences. Manuscript, 2005.

[12] B. M. E. Moret, J. Tang, and T. Warnow. Reconstructing phylogenies from gene-content and gene-order data. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, pages 321–352. Oxford University Press, 2005.

[13] R. Page. Maps between trees and cladistic analysis of historical associations among genes, 1994.

[14] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

[15] Alexandros Stamatakis. Raxml-vi-hpc: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed. To be published in Bioinformatics, 2006.

[16] D. Swofford. Paup*: Phylogenetic analysis using parsimony* (and other methods). 4.0b7 beta version.