The Gene-Duplication Problem: Near-Linear Time Algorithms for NNI-Based Local Searches

Mukul S. Bansal, Oliver Eulenstein, and André Wehe

Abstract—The gene-duplication problem is to infer a species supertree from a collection of gene trees that are confounded by complex histories of gene-duplication events. This problem is NP-complete and thus requires efficient and effective heuristics. Existing heuristics perform a stepwise search of the tree space, where each step is guided by an exact solution to an instance of a local search problem. A classical local search problem is the NNI *search problem*, which is based on the nearest neighbor interchange operation. In this work, we 1) provide a novel near-linear time algorithm for the NNI search problem, 2) introduce extensions that significantly enlarge the search space of the NNI search problem, and 3) present algorithms for these extended versions that are asymptotically just as efficient as our algorithm for the NNI search problem. The exceptional speedup achieved in the extended NNI search problems makes the gene-duplication problem more tractable for large-scale phylogenetic analyses. We verify the performance of our algorithms in a comparison study using sets of large randomly generated gene trees.

Index Terms—Computational phylogenetics, gene-duplication, supertrees, local search, NNI.

1 Introduction

Large-scale phylogenetic analysis is of fundamental importance to comparative genomics and ubiquitous in evolutionary biology. Most phylogenetic analyses combine genes from presumably orthologous loci, or loci whose homology is the result of speciation. These analyses largely exclude the vast amounts of sequence data from gene families, in which complex evolutionary processes such as gene duplication and loss result in gene trees that differ from species trees. One approach to utilize the data from such gene trees (gene families) is to reconcile the gene trees with species trees based on the duplication optimality criterion introduced by Goodman et al. [2]. The corresponding optimization problem is called the *gene-duplication* problem [3]. This problem can be viewed as a *supertree problem*, that is, assembling from a collection of input trees (the gene trees) a species supertree that contains all species found in at least one of the input trees. Other approaches make use of sequence similarity to reconstruct the underlying evolutionary history of genes (see, for example, [4], [5]). Probabilistic models for gene/species tree reconciliation as well as gene sequence evolution have also been developed [6], [7].

The decision version of the gene-duplication problem is NP-complete [8]. Standard heuristics aimed at solving the gene-duplication problem search the space of all possible supertrees guided by a series of exact solutions to instances of a *local search problem* [9]. The local search problem is to find an optimal phylogenetic tree under the duplication optimality criterion in the neighborhood of a given tree. The *neighborhood* of a tree is the set of all phylogenetic trees into which that tree can be transformed by applying a tree edit operation. A variety of different tree edit operations have been discussed in the literature [10], [11], and in practice, the rooted nearest neighbor interchange (NNI) tree edit operation [12], [13] can be effective for phylogenetic studies [3], [14]. However, algorithms for local search problems based on NNI operations are still in their infancy. To conduct large-scale phylogenetic analyses, there is much need for more effective NNI-based local search problems that can be solved efficiently.

In this work, we extend the NNI neighborhood to the k-NNI neighborhood. The k-NNI neighborhood contains all trees that can be obtained by performing at most k successive NNI operations on the given tree. Apart from NNI, there are two other standard (rooted) tree edit operations: the *subtree pruning and regrafting (*SPR*)* operation [15], [12], [13] and the *tree bisection and reconnection (*TBR*)* operation [15], [12], [16].[1] It can be shown [19], [20] that 2- and 3-NNI neighborhoods of a tree have very little overlap with its SPR and TBR neighborhoods. This results in novel and potentially more effective local searches. We greatly improve on the complexity of the best known (naive) solutions for 2- and 3-NNI-based local search problems. Furthermore, we show that each subsequent instance of the local search problem for 1-, 2-, and 3-NNI neighborhoods can be solved in linear time after the first instance is solved. This is especially desirable since standard local search heuristics for the gene-duplication problem can involve solving thousands of instances of the local search problem. Our novel near-linear time algorithms make it possible to perform truly large-scale phylogenetic analyses using the gene-duplication problem.

• *The authors are with the Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011. E-mail: {bansal, oeulenst, awehe}@cs.iastate.edu.*

Manuscript received 1 June 2008; revised 6 Nov. 2008; accepted 9 Jan. 2009; published online 20 Jan. 2009.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-2008-06-0100.
Digital Object Identifier no. 10.1109/TCBB.2009.7.

1. Efficient solutions are already known for local search problems based on SPR [17] and TBR [18].

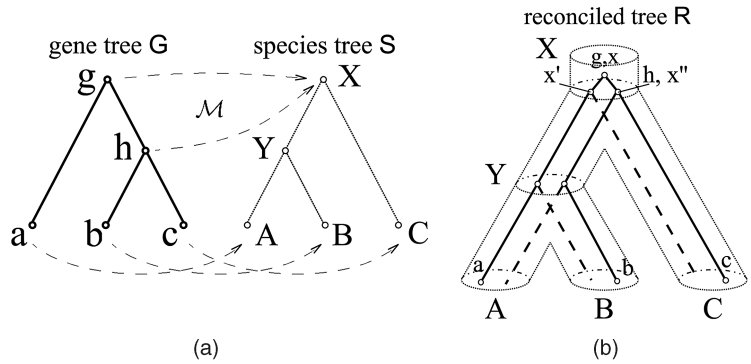

Fig. 1. (a) Gene trees G and species tree S are comparable, as the leaf-mapping from G to S indicates. $\mathcal{M}$ is the lca-mapping from G to S. (b) R is the reconciled tree for G and S. In species X of R, gene x duplicates into the genes x' and x''. The solid lines in R represent the embedding of G into R.

1.1 Previous Results

The gene-duplication problem is based on the Gene-Duplication model from Goodman et al. [2]. In the following, we 1) describe the Gene-Duplication model, 2) formulate the gene-duplication problem, and 3) describe a heuristic approach of choice [9] to solve the gene-duplication problem.

1.1.1 Gene-Duplication Model

The Gene-Duplication model is well studied [21], [3], [22], [23], [24], [25], [26], [27] and explains incompatibilities between a pair of "comparable" gene and species trees through gene duplications. A gene and a species tree are *comparable* if a *leaf-mapping* exists that provides a leaf to leaf mapping that maps every leaf node in the gene tree to a leaf node in the species tree. Biologically speaking, the leaves in the gene tree represent genes and the leaves in the species tree represent species, and the leaf-mapping essentially maps each gene to the species from which it was sampled. Consider the example shown in Fig. 1, taken from [17]. The leaf to leaf mapping from the gene tree G to the species tree S is the leaf-mapping. However, both trees describe incompatible evolutionary histories. The Gene-Duplication model explains such incompatibilities by reconciling the gene tree with postulated gene duplications. For example, in Fig. 1, a reconciled gene tree R can be theoretically inferred from the species tree S by duplicating a gene x in species X into the copies x' and x'' and letting both copies speciate according to the topology of S. In this case, the gene tree can be embedded into the reconciled tree. Thus, the gene tree can be reconciled by using the duplication of gene x to explain the incompatibility. The minimum number of gene duplications that are necessary under the Gene-Duplication model to explain the incompatibilities can be inferred from the mapping $\mathcal{M}$, which is an extension of the given leaf-mapping. $\mathcal{M}$ maps every gene in the gene tree to the most recent species in the species tree that could have contained the gene. More precisely, $\mathcal{M}$ maps each gene to the least common ancestor of the species from which the leaves (genes) of the subtree rooted at the gene were sampled (given by the leaf-mapping). A gene in the gene tree is a *(gene) duplication* if it has a child with the same $\mathcal{M}$ mapping [21], [23]. The *reconciliation cost* for a gene tree and a comparable species tree is measured in the number of gene duplications in the gene tree induced by the species tree.[2] The *reconciliation cost* for a given collection of gene trees and a species tree is the sum of the reconciliation costs for each gene tree in the collection and the species tree. The mapping function is linear time computable on a PRAM [24] through a reduction from the least common ancestor problem [28]. Hence, the reconciliation cost for a collection of gene trees and a species tree is computable in linear time.

1.1.2 Gene-Duplication Problem and Heuristics

The *gene-duplication problem* is to find, for a given collection of gene trees, a comparable species tree with minimum reconciliation cost. This approach has been successfully applied to phylogenetic inference in snakes [29], vertebrates [30], [31], *Drosophila* [32], and plants [33], among others. The decision variant of this problem and some of its characterizations are NP-complete [8], [34], while some parameterizations are fixed parameter tractable [35], [36]. Therefore, in practice, heuristics (e.g., [9]) are commonly used for the gene-duplication problem, even though they are unable to guarantee an optimal solution. These heuristics are based on local search and, consequently, involve repeatedly solving a local search problem. Such a heuristic starts with some species tree comparable with the input gene trees and finds a minimum reconciliation cost tree in its neighborhood. This constitutes one local search step. The new tree thus found then becomes the starting point for the next local search step, and so on, until a local minima is reached. Thus, at each local search step, we must solve the local search problem. The time complexity of the local search problem depends on the tree edit operation used to define the neighborhood.

Here, the edit operation of interest is the `NNI` operation [12], [13]. Rooted and unrooted `NNI` operations have been extensively studied [37]. An `NNI` operation on a species tree S (represented as an undirected graph) can be performed by "swapping" two of its node disjoint subtrees whose root nodes are connected by a simple path of length 3. If n denotes the number of leaves in S, then the neighborhood defined by the `NNI` operation on S contains $\Theta(n)$ trees.

2. Alternatively, the reconciliation cost could be defined in the number of gene duplications and losses. However, it is often problematic to accurately infer gene losses if there are missing data. Thus, for this study, we only consider gene duplications.

1.2 Contribution of This Work

We provide efficient algorithms for local search heuristics based on k-NNI neighborhoods for $k \in \{1, 2, 3\}$. In fact, we show that local searches based on 2- and 3-NNI neighborhoods are asymptotically just as efficient as those based on 1-NNI, even though they search a much larger neighborhood of trees. Assume, for convenience, that the size of the r given gene trees differs by a constant factor from the size of the resulting species tree, which we denote by n. Local searches based on k-NNI, for $k \in \{1, 2, 3\}$, induce a neighborhood of size $\Theta(n^k)$ [12], [20], and hence, best known (naive) solutions for the corresponding local search problems require $O(rn^{k+1})$ time. We provide algorithms that solve the local search problems for both 2- and 3-NNI-neighborhoods in $O(rn^2)$ time. The main idea of our algorithms is that once we compute the costs of all the $\Theta(n)$ possible NNI operations on a tree, we can reuse these values to quickly obtain the costs of most of the trees produced by the second and third NNI operations.

Furthermore, we show that each subsequent k-NNI local search, for $k \in \{1, 2, 3\}$, can be solved in $O(rn)$ time. In summary, for all three neighborhoods, the total complexity of a heuristic search involving p local search steps is $O(rn(n + p))$. Thus, if $p \geq n$, which largely holds true in practice, then the amortized time complexity per local search step is linear in the input size. Consequently, our algorithms provide a total speedup of $\Theta(\min\{n, p\})$, $\Theta(n \times \min\{n, p\})$, and $\Theta(n^2 \times \min\{n, p\})$ for heuristics that are based on 1-, 2-, and 3-NNI local searches, respectively. Note that for 2- and 3-NNI, the complexity of our algorithms is, in fact, sublinear in the size of the corresponding neighborhoods. Altogether, the substantially enlarged neighborhoods and the remarkable efficiency of our algorithms make the gene-duplication problem much more amenable to large-scale phylogenetic analyses.

We implemented our algorithms and demonstrate the improvement they offer over current solutions by applying them to several large simulated data sets. We also discuss the fixed parameter tractability of the k-NNI problem (along with its generalizations to certain other edit operations and objective functions) for arbitrary k.

2 Basic Notation and Preliminaries

In this section, we first introduce basic definitions and notation, and then the necessary preliminaries required for this work. Some of the notation and definitions are taken from [18].

2.1 Basic Definitions and Notation

A *tree* T is a connected graph with no cycles, consisting of a node set $V(T)$ and an edge set $E(T)$. T is *rooted* if it has exactly one distinguished node called the *root*, which we denote by $rt(T)$. Let T be a rooted tree. We define $\leq_T$ to be the partial order on $V(T)$, where $x \leq_T y$ if y is a node on the path between $rt(T)$ and x. The set of minima under $\leq_T$ is denoted by $Le(T)$ and its elements are called *leaves*. If $\{x, y\} \in E(T)$ and $x \leq_T y$, then we call y the *parent* of x denoted by $pa_T(x)$ and we call x a *child* of y. The set of all children of y is denoted by $Ch_T(y)$. If two nodes in T have the same parent, they are called *siblings*. The *least common ancestor* of a nonempty subset $L \subseteq \mathcal{V}(T)$, denoted as $lca(L)$, is the unique smallest upper bound of L under $\leq_T$. A *subtree* of T rooted at node $y \in V(T)$, denoted by T_y, is the tree induced by $\{x \in V(T) : x \leq_T y\}$. T is *fully binary* if every node has either zero or two children. Throughout this paper, the term *tree* refers to a rooted fully binary tree.

2.2 The Gene-Duplication Problem

We now introduce necessary definitions to state the gene-duplication problem. A *species tree* is a tree that depicts the evolutionary relationships of a set of species. Given a gene family for a set of species, a *gene tree* is a tree that depicts the evolutionary relationships among the sequences encoding only that gene family in the given species.[3] Thus, the nodes in a gene tree represent genes from some species. In this work, we shall assume that each leaf of the gene trees is labeled with the species from which that gene was sampled. Thus, unlike species trees, a gene tree might have several leaves with the same label.

In order to compare a gene tree G with a species tree S, a mapping from each gene $g \in V(G)$ to the most recent species in S that could have contained g is required.

Definition 2.1 (Mapping). *The leaf-mapping $\mathcal{L}_{G,S} : Le(G) \to Le(S)$ maps a leaf node $g \in Le(G)$ to that unique leaf node $s \in Le(S)$ which has the same label as g. The extension $\mathcal{M}_{G,S} : V(G) \to V(S)$ of $\mathcal{L}_{G,S}$ is the mapping defined by $\mathcal{M}_{G,S}(g) = lca(\mathcal{L}_{G,S}(Le(G_g)))$.*

Note: For any node $s \in V(S)$, $\mathcal{M}^{-1}_{G,S}(s)$ denotes the set of nodes in G that map to node $s \in V(S)$ under the mapping $\mathcal{M}_{G,S}$.

Definition 2.2 (Comparability). *Given trees G and S, we say that G is comparable to S if, for each $g \in Le(G)$, the leaf-mapping $\mathcal{L}_{G,S}(g)$ is well defined. A set of gene trees $\mathcal{G}$ is comparable to S if each gene tree in $\mathcal{G}$ is comparable to S.*

Throughout this paper, we use the following terminology: $\mathcal{G}$ is a set of gene trees that is comparable to a species tree S, and $G \in \mathcal{G}$.

Definition 2.3 (Duplication). *A node $v \in V(G)$ is a (gene) duplication if $\mathcal{M}_{G,S}(v) \in \mathcal{M}_{G,S}(Ch(v))$ and we define* $\texttt{Dup}(G, S) = \{g \in V(G) : g \text{ is a duplication}\}$.

Definition 2.4 (Reconciliation cost). *We define reconciliation costs for gene and species trees as follows:*

1. *$\Delta(G, S) = |\texttt{Dup}(G, S)|$ is the reconciliation cost from G to S.*
2. *$\Delta(\mathcal{G}, S) = \sum_{G \in \mathcal{G}} \Delta(G, S)$ is the reconciliation cost from $\mathcal{G}$ to S.*
3. *Let $\mathcal{T}$ be the set of species trees to which $\mathcal{G}$ is comparable. We define $\Delta(\mathcal{G}) = \min_{S \in \mathcal{T}} \Delta(\mathcal{G}, S)$ to be the reconciliation cost of $\mathcal{G}$.*

Problem 1 (Duplication).

Instance: *A set $\mathcal{G}$ of gene trees.*

Find: *A species tree S^* to which $\mathcal{G}$ is comparable, such that $\Delta(\mathcal{G}, S^*) = \Delta(\mathcal{G})$.*

3. A *gene family* is a set of homologous genes assumed to have shared ancestry.

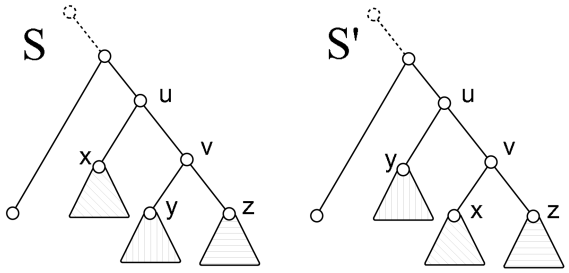

Fig. 2. The tree $S' = \texttt{NNI}_S(y)$ is obtained by swapping the subtrees S_x and S_y.

2.3 Local Search Problems

Here, we first provide the definition of the NNI edit operation [12], [13], and then formulate the related local search problems which were motivated in Section 1.

Definition 2.5 (NNI operation). *Let T be a tree. For technical reasons, we first define the set $valid(T) = V(T) \setminus (\{rt(T)\} \cup Ch(rt(T)))$ and call its elements valid nodes* in *T. Now, for $y \in valid(T)$, we denote by $\texttt{NNI}_T(y)$ the tree that is obtained from T by swapping the subtrees T_x and T_y, where x is the sibling of $pa(y)$. We say that the tree $\texttt{NNI}_T(y)$ is obtained from T by an* NNI *operation* on *y (an example is depicted in Fig. 2).*

In the remainder of this paper, whenever we write $\texttt{NNI}_T(y)$ we assume that $y \in valid(T)$.

Definition 2.6 (k-NNI neighborhood). *The k-NNI neighborhood of a tree T is defined to be the set of all trees that can be obtained by performing at most k successive* NNI *operations on T. The k-NNI neighborhood of T is denoted by k-$\texttt{NNI}_T$.*

Thus, for instance, 1-$\texttt{NNI}_T$ (or simply $\texttt{NNI}_T$) is the set $\{\texttt{NNI}_T(y) : y \in valid(T)\}$.

Problem 2 (k-NNI-Search).

Instance: *A set $\mathcal{G}$ of gene trees, and a species tree S such that $\bigcup_{G\in\mathcal{G}} \bigcup_{g\in Le(G)} \mathcal{L}_{G,S}(g) = Le(S)$.*

Find: *A tree $T^* \in k$-$\texttt{NNI}_S$ such that $\Delta(\mathcal{G}, T^*) = \min_{T\in k\text{-}\texttt{NNI}_S} \Delta(\mathcal{G}, T)$.*

In the next section, we study structural properties of 1-, 2-, and 3-NNI Search problems. In Section 4, we develop our algorithm for 2-NNI-Search Problem. Our algorithm for further speedup of subsequent local search steps for 1- and 2-NNI heuristic searches appears in Section 5. A description of our algorithm for the 3-NNI-Search problem, and its further speedup appears in Section 6. In Section 7, we discuss the fixed parameter tractability of the k-NNI problem and show how the techniques developed in this paper can be applied to other edit operations and objective functions. Experimental results are discussed in Section 8 and concluding remarks appear in Section 9.

3 Structural Properties

In this section, we study the effects of an NNI operation on the mapping $\mathcal{M}_{G,S}$ and on the gene-duplication status of nodes from G. Throughout this section, we assume that the tree S' is defined to be $\texttt{NNI}_S(y)$. Fig. 2 depicts this situation; $v = pa_S(y)$, $u = pa_S(v)$, and x and z denote the siblings of v and y in S, respectively. As depicted in Fig. 2, our naming convention preserves the name of each node in the species tree before and after an NNI operation is performed on it.

Observe that when an NNI operation is performed on a tree, it only affects the mappings in a small, constant sized region of the tree. In particular, we have the following lemmas.

Lemma 3.1. *Let $g \in V(G)$ and $\mathcal{M}_{G,S}(g) \notin \{u, v\}$. Then, $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S}(g)$.*

Proof. Suppose $\mathcal{M}_{G,S}(g) = s$. If $s \in Le(S)$, then the lemma follows immediately.

Otherwise, let $\{a, b\} = Ch_S(s)$, and $\{a', b'\} = Ch_{S'}(s)$. If $s \notin \{u, v\}$, then we must have $\{Le(S_a), Le(S_b)\} = \{Le(S'_{a'}), Le(S'_{b'})\}$. This implies that for $s \in V(S) \setminus \{u, v\}$, $\mathcal{M}_{G,S}(g) = s$ if and only if $\mathcal{M}_{G,S'}(g) = s$. The lemma follows. □

Lemma 3.1 is central to all of our algorithms in this paper. This basic idea is developed further in Lemmas 3.2-3.4. But first, we need a definition.

Definition 3.1. *For any $s \in valid(S)$, we define $diff_S(s) = \Delta(\mathcal{G}, S) - \Delta(\mathcal{G}, \texttt{NNI}_S(s))$.*

The above definition is motivated by the fact that when a given species tree is modified by an NNI operation, we will find it easier to compute the change in the reconciliation cost, rather than the new reconciliation cost directly.

Definition 3.2 (Dependent nodes). *Given $s \in valid(S)$, let a and b be the siblings of $pa_S(s)$ and s, respectively. We define $dep_S(s) = \{a, b, s, pa_S(s)\} \cup Ch_S(s) \cup Ch_S(a) \cup Ch_S(b)$, and say that the nodes in $dep_S(s)$ are dependent on node s in S.*

Definition 3.3 (Independent nodes). *Given $s \in valid(S)$, let a and b be the siblings of $pa_S(s)$ and s, respectively. We define $ind_S(s) = valid(S) \setminus dep_S(s)$, and say that the nodes in $ind_S(s)$ are independent with respect to node s in S.*

Essentially, the nodes in $ind_S(s)$ are important because they satisfy the property in Lemma 3.2. In the remainder of this paper, whenever we write $dep_S(s)$ or $ind_S(s)$, we assume that $s \in valid(S)$. A key idea in our algorithms is that when an NNI operation is performed, much of the information computed for the original tree remains the same even for the new tree. This idea is formally captured in Lemma 3.2.

Lemma 3.2. *If $s \in valid(S') \cap ind_S(y)$, then $diff_{S'}(s) = diff_S(s)$.*

Proof. Let R denote a species tree. We define $d_{G,R}(g) = 1$ if $g \in V(G)$ is a duplication under mapping $\mathcal{M}_{G,R}$, and $d_{G,R}(g) = 0$ otherwise. Let $a = pa_S(s)$, $b = pa_S(a)$, $a' = pa_{S'}(s)$, $b' = pa_{S'}(c)$, $T = \texttt{NNI}_S(s)$, and $T' = \texttt{NNI}_{S'}(s)$. By its definition, $diff_S(s)$ can be written as $\sum_{G\in\mathcal{G}} \sum_{g\in V(G)} d_{G,S}(g) - d_{G,T}(g)$. Therefore, by Lemma 3.1, we have $diff_S(s) = \sum_{G\in\mathcal{G}} \sum_{g\in\mathcal{M}^{-1}_{G,S}(a)\cup\mathcal{M}^{-1}_{G,S}(b)} d_{G,S}(g) - d_{G,T}(g)$. Similarly, we must also have

$$diff_{S'}(s) = \sum_{G\in\mathcal{G}} \sum_{g\in\mathcal{M}^{-1}_{G,S'}(a')\cup\mathcal{M}^{-1}_{G,S'}(b')} d_{G,S'}(g) - d_{G,T'}(g).$$

Let p and t be the siblings of s and a in S, and p' and t' be the siblings of s and a' in S', respectively. If $s \in valid(S') \setminus dep_S(y)$, then it can be easily verified that we must have $Le(S_t) = Le(S'_{t'})$, $Le(S_s) = Le(S'_s)$, and $Le(S_p) = Le(S'_{p'})$. This further implies that $\mathcal{M}^{-1}_{G,S}(a) = \mathcal{M}^{-1}_{G,S'}(a')$, $\mathcal{M}^{-1}_{G,S}(b) = \mathcal{M}^{-1}_{G,S'}(b')$, and for any $g \in \mathcal{M}^{-1}_{G,S}(a) \cup \mathcal{M}^{-1}_{G,S}(b)$, we must have $d_{G,S}(g) = d_{G,S'}(g)$ and $d_{G,T}(g) = d_{G,T'}(g)$. Thus, $diff_S(s) = diff_{S'}(s)$. □

The next two lemmas follow more or less from the definition of $ind_S(s)$ and they are crucial for Lemma 4.2.

Lemma 3.3. $|dep_S(s)| = |valid(S) \setminus ind_S(s)| \leq 10$.

Proof. This follows as a direct consequence of the definition of $ind_S(s)$. □

Lemma 3.4. *If $s \in valid(S)$, then*

$$|\{t \in valid(S) : s \in dep_S(t)\}| \leq 10.$$

Proof. Let a be the sibling of s and b the sibling of $pa_S(s)$ in S, and $\{c, d\} = Ch_S(s)$. It can be easily verified that according to the definition of $dep_S(t)$ if $s \in dep_S(t)$, then $t \in \{s, pa_S(s), a, b, c, d\} \cup Ch_S(a) \cup Ch_S(b)$. Therefore, the lemma follows. □

4 Solving the 2-NNI-Search Problem

In this section, we describe our algorithm to solve the 2-NNI-Search problem. The first step in our algorithm is to compute the value $diff_S(s)$ for each $s \in valid(S)$. This already gives a solution to the 1-NNI-Search problem. Subsequently, the algorithm computes a minimum reconciliation cost tree in 2-$\text{NNI}_S \setminus \text{NNI}_S$. All trees in 2-$\text{NNI}_S \setminus \text{NNI}_S$ are obtained by performing exactly two successive NNI operations on tree S. Consider some tree $T \in$ 2-$\text{NNI}_S \setminus \text{NNI}_S$. There must exist two nodes $s, t \in V(S)$ such that $T = \text{NNI}_{T'}(t)$ and $T' = \text{NNI}_S(s)$. Now there are two possible cases: 1) $t \in ind_S(s)$ or 2) $t \notin ind_S(s)$.

Our algorithm computes a minimum reconciliation cost tree among the trees that satisfy Case 1 above, and a minimum reconciliation cost tree among the trees that satisfy Case 2. It also computes a minimum reconciliation cost tree in NNI_S. The tree with minimum reconciliation cost among these three trees must be a minimum reconciliation cost tree in 2-NNI_S.

Handling Cases 1 and 2 separately allows us to apply the lemmas seen in the previous section. In particular, for Case 1, we can make use of the fact that node t is independent with respect to node s in S, which allows us to reuse the information computed previously, and for Case 2, we can make use of the fact that for any given s, the number of possible candidates for t is limited. In the remainder of this section, we show how these ideas allow us to efficiently compute a minimum reconciliation cost tree in 2-$\text{NNI}_S \setminus \text{NNI}_S$.

We will first show how to handle Case 2. That is, we show how to find a minimum reconciliation cost tree for the case when $t \notin ind_S(s)$. The following lemma shows that if $t \notin ind_S(s)$, then $t \in dep_S(s)$ (even though $valid(T)$ need not be identical to $valid(S)$).

Lemma 4.1. *Given $s, t \in V(S)$ such that $T = \text{NNI}_{T'}(t)$ and $T' = \text{NNI}_S(s)$, if $t \notin ind_S(s)$, then $t \in dep_S(s)$.*

Proof. Node t must satisfy exactly one of the following:

1. $t \in valid(S)$: In this case, the lemma must hold true by Definition 3.3.
2. $t \notin valid(S)$: It is easy to verify that this is only possible if $t = s$. And hence, the lemma holds true in this case as well.

The lemma follows. □

Based on the above lemma, we can immediately conclude that in Case 2, for each of the possible $O(n)$ candidates for S, there are only 10 possible candidates for t.

The following lemma shows how to efficiently compute a minimum reconciliation cost tree for Case 1.

Lemma 4.2. *Let A denotes the set of the first 11 nodes valid in S arranged according to the decreasing values of $diff_S(s)$. Let $\Gamma = \{T : T = \text{NNI}_{T'}(t), T' = \text{NNI}_S(s), \text{ and } t \in ind_S(s)\}$. Let $R^* \in \Gamma$ with minimum reconciliation cost. Then, there exists a pair of nodes $a, b \in A$ such that $b \in ind_S(a)$, $R = \text{NNI}_{R'}(b)$, $R' = \text{NNI}_S(a)$, and $\Delta(\mathcal{G}, R^*) = \Delta(\mathcal{G}, R)$.*

Proof. Let $c, d \in valid(S)$ such that $R^* = \text{NNI}_{S'}(d)$, where $S' = \text{NNI}_S(c)$ and $d \in ind_S(c)$. Also observe that, for any species tree T, if $s, t \in valid(T)$ and $t \in ind_T(s)$, then $t \in valid(\text{NNI}_T(s))$. Thus, it is sufficient to show that there exist two nodes $a, b \in A$, where $b \in ind_S(a)$ such that $diff_S(a) + diff_{\text{NNI}_S(a)}(b) \geq diff_S(c) + diff_{\text{NNI}_S(c)}(d)$. By Lemma 3.2, we know that $diff_{\text{NNI}_S(a)}(b) = diff_S(b)$ and $diff_{\text{NNI}_S(c)}(d) = diff_S(d)$. Therefore, to complete this proof, we will show that $diff_S(a) + diff_S(b) \geq diff_S(c) + diff_S(d)$.

Set a, b to be the pair of nodes in A such that $b \in ind_S(a)$, for which $diff_S(a) + diff_S(b)$ is maximized. By Lemma 3.3, we know that such a pair must exist. There are now four possible cases:

1. $c, d \in A$. In this case, by our choice of a and b, we must have $diff_S(a) + diff_S(b) \geq diff_S(c) + diff_S(d)$.
2. $c \in A$ and $d \notin A$. In this case, there must exist (by Lemma 3.3 and the definition of the set A) a node $e \in A$ such that $diff_S(e) \geq diff_S(d)$ and $e \in ind_S(c)$. Therefore, we must have $diff_S(a) + diff_S(b) \geq diff_S(c) + diff_S(d)$.
3. $c \notin A$ and $d \in A$. In this case, there must exist (by Lemma 3.4 and the definition of the set A) a node $e \in A$ such that $diff_S(e) \geq diff_S(c)$ and $d \in ind_S(e)$. Therefore, we must have $diff_S(a) + diff_S(b) \geq diff_S(c) + diff_S(d)$.
4. $c, d \notin A$. In this case, by the definition of set A and our choice of a and b, $diff_S(a) + diff_S(b) \geq diff_S(c) + diff_S(d)$.

The lemma follows. □

We can now present our algorithm to solve the 2-NNI-Search problem. We call this algorithm ALG-2-NNI, and a description of this algorithm appears as Algorithm 1.

Algorithm 1. ALG-2-NNI
Input: A set of gene trees $\mathcal{G}$, and, a species tree S
Output: A tree $T^* \in k$-NNI_S such that $\Delta(\mathcal{G}, T^*) = \min_{T \in k-\text{NNI}_S} \Delta(\mathcal{G}, T)$
1: **for** each $s \in valid(S)$ **do**
2: Compute the value $diff_S(s)$.

3: Let $\alpha = \arg\max_{a \in valid(S)} diff_S(a)$, and set $T_1 = \texttt{NNI}_S(\alpha)$.
4: Let A denote the set of the first 11 nodes valid in S arranged according to decreasing values of $diff_S(s)$.
5: $(\alpha, \beta) = \arg\max_{(a,b):a,b \in A,\ b \in ind_S(a)} diff_S(a) + diff_S(b)$.
6: Set $T = \texttt{NNI}_S(\alpha)$ and $T_2 = \texttt{NNI}_T(\beta)$.
7: Set $T_3 = T_2$.
8: **for** each $s \in valid(S)$ **do**
9: Set $T = \texttt{NNI}_S(s)$.
10: **for** $t \in valid(T) \cap dep_S(s)$ **do**
11: $R = \texttt{NNI}_T(t)$.
12: **if** $\Delta(\mathcal{G}, T_3) > \Delta(\mathcal{G}, R)$ **than**
13: Set $T_3 = R$.
14: **return** $\arg\min_{T \in \{T_1, T_2, T_3\}} \Delta(\mathcal{G}, T)$.

The input for Algorithm 1 is a set of gene trees $\mathcal{G}$ and a species tree S. Let $n = |Le(S)|$ and $r = |\mathcal{G}|$. To simplify the complexity analysis, we shall assume that all input gene trees have almost the same size. Thus, let $m = |Le(S)| + |Le(G)|$ for some $G \in \mathcal{G}$. Note: The speedup obtained by our algorithm does not depend on this simplifying assumption.

Theorem 4.1. *Algorithm 1 solves the 2-NNI-Search problem in $O(rmn)$ time.*

Proof (Correctness). Each tree $T \in \text{2-}\texttt{NNI}_S$ belongs to one of the following cases:

1. $T \in \texttt{NNI}_S$: The tree T_1 computed in Algorithm 1 (line 3) is a tree with minimum reconciliation cost among all trees in $\texttt{NNI}_S$.
2. $T \in \text{2-}\texttt{NNI}_S \setminus \texttt{NNI}_S$: There exist two nodes $s, t \in V(S)$ such that $T = \texttt{NNI}_{T'}(t)$ and $T' = \texttt{NNI}_S(s)$. We now have two possible cases:
   a. $t \in ind_S(s)$: According to Lemma 4.2, the tree T_2 computed by Algorithm 1 (lines 4-6) must be a minimum reconciliation cost tree among all trees in this case.
   b. $t \notin ind_S(s)$: According to Lemma 4.1, the tree T_3 computed by Algorithm 1 (lines 8-13) must be a minimum reconciliation cost tree among all trees in this case.

Therefore, a minimum reconciliation cost tree among T_1, T_2, T_3 must be a solution to the 2-NNI-Search problem. (Complexity). Computing the tree T_1 involves computing the $diff_S(s)$ value for each $s \in valid(S)$, and identifying the node a for which $diff_S(a)$ is maximum. Computing the reconciliation cost for a given species tree takes $O(rm)$ time. Therefore, computing T_1 takes $O(rmn)$ time.

After T_1 has been computed, computing the tree T_2 involves creating the set A (which takes $O(n)$ time), and then evaluating every possible two-element ordered pair from A. Each evaluation takes $O(1)$ time, and the number of possible ordered pairs is $O(|A|^2)$, i.e., $O(1)$. Therefore, computing T_2 (after having computed T_1) requires $O(n)$ time.

Computing T_3 involves evaluating the reconciliation costs of at most $10 \times n$, i.e., $O(n)$ trees, and then picking the best tree among these. Therefore, computing T_3 requires $O(rmn)$ time.

In conclusion, the time complexity of Algorithm 1 is $O(rmn)$. □

The time complexity of the naive solution for the 2-NNI-Search problem is $\Theta(rn^2m)$. Thus, our algorithm improves on the current solution by a factor of $\Theta(n)$.

5 Further Speedup for 1- and 2-NNI Heuristics

As mentioned earlier, standard local search heuristics for the Duplication problem involve solving many instances of these local search problems. Consider a heuristic search involving p instances of the local search problem, then using our faster algorithm for the 2-NNI-Search problem allows both 1- and 2-NNI-based heuristics to run in $O(prmn)$ time. We will now show that the 1-, 2-NNI-based heuristics can, in fact, both be executed in $O(rm(n+p))$ time.

5.1 Heuristics Based on 1-NNI

Existing algorithms for the 1-NNI-Search (or simply NNI-Search) problem have a time complexity of $O(rmn)$, and hence, they solve the NNI-based heuristic problem in $O(rpmn)$ time. Our algorithm to solve the NNI-Search problem involves computing the value $diff_S(s)$ for each $s \in valid(S)$, and then picking a tree T such that $T = \texttt{NNI}_S(\alpha)$, where $\alpha = \arg\max_{a \in valid(S)} diff_S(a)$. This also requires $O(rmn)$ time. However, this approach allows us to reuse most of the previously computed information in subsequent iterations of the local search.

Let T denote a minimum reconciliation cost tree in $\texttt{NNI}_S$. Then, there exists a node a such that $T = \texttt{NNI}_S(a)$. For the next iteration, we must compute a minimum reconciliation cost tree in $\texttt{NNI}_T$. As seen earlier, this involves computing the value $diff_T(s)$ for each $s \in valid\,(T)$. By Lemma 3.2, we know that $diff_T(s) = diff_S(s)$ for all $s \in ind_S(a)$. Therefore, for all $s \in ind_S(a)$, we can reuse the values from the previous iteration. In other words, we must only compute the value $diff_T(s)$ for all $s \in valid\,(T) \setminus ind_S(a)$. It follows directly from Lemma 4.1 that if $s \in valid(T) \setminus ind_S(a)$, then $s \in dep_S(a)$, that is, the number of candidates for s is at most 10.

This means that for each subsequent iteration of the NNI local search, we must compute the reconciliation costs for at most 10 trees. Thus, once the first NNI local search problem has been solved in $O(rmn)$ time, each subsequent local search instance can be solved in $O(rm)$ time. This gives a total time complexity of $O(rm(n+p))$, which gives a speedup by a factor of $\Theta(\min\{n, p\})$ over existing solutions.

5.2 Heuristics Based on 2-NNI

Let T denote a minimum reconciliation cost tree in $\text{2-}\texttt{NNI}_S$. For the next iteration of this local search, we wish to find a tree with minimum reconciliation cost in $\text{2-}\texttt{NNI}_T$. According to our algorithm (see Algorithm 1), computing such a tree involves computing the trees $T_1, T_2, T_3 \in \text{2-}\texttt{NNI}_T$. We now show how to compute each of these three special trees in $O(rm)$ time by reusing previously computed information.

There exist two nodes a, b such that $T' = \texttt{NNI}_S(a)$ and $T = \texttt{NNI}_{T'}(b)$. Computing the tree T_1 involves computing the value $diff_T(s)$ for all nodes $s \in valid(T)$. Since a and b are known (from the previous iteration of the local search),

the method used for 1-NNI above can be used to obtain the values $diff_{T'}(s)$ for all $s \in valid(T')$ in $O(rm)$ time. Once this is done, the same algorithm is reapplied to compute the values $diff_T(s)$ for all $s \in valid(T)$. This step also takes $O(rm)$ time. Hence, the tree T_1 can be computed in $O(rm) + O(rm)$, i.e., $O(rm)$ time.

Once all the $diff_T(s)$ values have been obtained for all $s \in valid(T)$, computing the tree T_2 takes $O(n)$ time (see the complexity analysis in the proof of Theorem 4.1).

In order to compute the tree T_3, we first compute the tree $\texttt{NNI}_T(s)$ and then compute the costs for at most 10 trees derived from $\texttt{NNI}_T(s)$, for each $s \in valid(T)$ (see Algorithm 1). In particular, let $s, t \in V(T)$ such that $R' = \texttt{NNI}_T(s)$ and $R = \texttt{NNI}_{R'}(t)$, and $t \notin ind_T(s)$. By Lemma 3.3, there are $O(10n)$ possible candidates for the pair (s, t). In order to compute T_3, it is sufficient to show how to compute these $O(10n)$ reconciliation costs. In particular, we must compute the value $diff_T(s) + diff_{R'}(t)$ for all possible pairs (s, t). The main idea is to show that for all but a constant number of the candidate pairs (s, t), the value $diff_T(s) + diff_{R'}(t) = diff_S(s) + diff_{\texttt{NNI}_S(s)}(t)$. Thus, most of the values computed in the previous step can be directly reused in the current step.

Observe that the equality $diff_T(s) + diff_{R'}(t) = diff_S(s) + diff_{\texttt{NNI}_S(s)}(t)$ is violated only if 1) $s \notin valid(S)$ or $t \notin valid(\texttt{NNI}_S(s))$ or 2) $diff_T(s) \neq diff_S(s)$ or $diff_{R'}(t) \neq diff_{\texttt{NNI}_S(s)}(t)$. For Case 1, there are at most two candidates each for s and t (since T is obtained from S by no more than two NNI operations). For Case 2, it can be easily shown, based on Lemmas 3.2-3.4 that there are only $O(1)$ such candidate pairs for (s, t). This implies that only $O(1)$ of the possible $O(10n)$ values need to be recomputed.

Thus, T_3 can be computed in $O(rm)$ time as well, which, in turn, implies that a minimum reconciliation cost tree in $2\text{-}\texttt{NNI}_T$ can be computed in $O(rm)$ time. This gives a total time complexity of $O(rm(n + p))$ for 2-NNI-based heuristics, which gives a speedup by a factor of $\Theta(n \times \min\{n, p\})$ over the naive solution.

6 Optimizing the 3-NNI-Search Problem

The main idea behind our algorithms for the 1- and 2-NNI-Search problems, as well as their speedup, is that when an NNI operation is performed on a tree, it only affects the mapping in a small, constant sized region of the tree. Since the reconciliation cost depends only on the mapping from the gene trees, in the new species tree thus obtained, much of the information computed for the original tree remains valid. This idea applies equally well for solving the k-NNI-Search problem, for $k > 2$, but the number of cases to be considered increases exponentially as k increases. However, for the special case of $k = 3$, the algorithm for 2-NNI-Search extends in a rather straightforward manner.

The trees in $3\text{-}\texttt{NNI}_S$ must be in at least one of $2\text{-}\texttt{NNI}_S$, or $3\text{-}\texttt{NNI}_S \setminus 2\text{-}\texttt{NNI}_S$. We have already seen how to obtain a minimum reconciliation cost tree in $2\text{-}\texttt{NNI}_S$. Therefore, the problem is to find a minimum reconciliation cost tree in $3\text{-}\texttt{NNI}_S \setminus 2\text{-}\texttt{NNI}_S$. All the trees in $3\text{-}\texttt{NNI}_S \setminus 2\text{-}\texttt{NNI}_S$ are obtained by performing exactly three successive NNI operations on tree S. Consider some tree $T \in 3\text{-}\texttt{NNI}_S \setminus 2\text{-}\texttt{NNI}_S$. Then, there must exist three nodes $s, t, u \in V(S)$ such that $T = \texttt{NNI}_{T'}(u)$, $T' = \texttt{NNI}_{T''}(t)$, and $T'' = \texttt{NNI}_S(s)$. We can divide our analysis of the possible interrelationships between s, t, and u into six cases such that if we can calculate a minimum reconciliation cost tree separately for each of these six cases, then the tree with minimum reconciliation cost among these six trees will be a minimum reconciliation cost tree in $3\text{-}\texttt{NNI}_S \setminus 2\text{-}\texttt{NNI}_S$. Again, the main idea for efficient computation is that if an NNI operation is independent of the other two, then we can reuse the already computed $diff$ values; otherwise, we make use of the fact that if two or more of the NNI operations are not independent, then they must be in close proximity to each other and this limits the number of such cases. The six cases are as follows:

1. $t \in ind_S(s), u \in ind_{T''}(t) \cap ind_S(s)$: This case can be solved in $O(rmn)$ time by a simple extension of the technique used to obtain the tree T_2 in Algorithm 1.
2. $t \in ind_S(s), u \in ind_S(s) \setminus ind_{T''}(t)$: Since $u, t \in ind_S(s)$ and $u \notin ind_{T''}(t)$, there are $O(n)$ candidate pairs (t, u). We first evaluate the change in the cost of S when the two NNI operations defined by t and u are performed successively on S. This takes $O(rmn)$ time. Since this change in the cost must be independent of the change in the cost when S is changed into $\texttt{NNI}_S(s)$, a technique similar to the one used for Case 1 can now be used to obtain a best tree for this case.
3. $t \in ind_S(s), u \in ind_{T''}(t) \setminus ind_S(s)$: It can be shown that there are at most $O(1)$ candidates for the ordered pair (t, u), such that the tree produced from S by performing successively the three NNI operations defined by (s, t, u) is different from the tree produced from S by performing successively the three NNI operations defined by (s, u, t). For such candidates, the cost can be computed naively in $O(rmn)$ time. All the remaining costs can be obtained exactly as in Case 2. (Since swapping t and u in Case 3 produces Case 2.)
4. $t \in ind_S(s), u \notin ind_{T''}(t) \cup ind_S(s)$: For any given s, there are only $O(1)$ candidates for u, and hence, by Lemma 3.4, only $O(1)$ candidates for t. This case is therefore solvable in $O(rmn)$ time.
5. $t \notin ind_S(s), u \in ind_{T''}(t) \cap ind_S(s)$: For any given s, there are only $O(1)$ candidates for t, and since $u \in ind_{T''}(t) \cap ind_S(s)$, the values of $diff_S(a)$ computed for each $a \in valid(S)$ can be reused to quickly obtain the best candidate for u, for any given s, t.
6. $t \notin ind_S(s), u \notin ind_{T''}(t) \cap ind_S(s)$: For any given s, there are $O(1)$ candidates each for both t and u. Therefore, these can be naively handled in $O(rnm)$ time.

Thus, a minimum reconciliation cost tree can be obtained for each of the six cases in $O(rmn)$ time, which, in turn, gives us an $O(rmn)$ time algorithm for the 3-NNI-Search problem.

The algorithm used to obtain further speedup for 2-NNI-based heuristics also extends in a similar fashion to 3-NNI-based heuristics. This gives a total time complexity of $O(rm(n + p))$ for the 3-NNI-based heuristic.

7 A Generalized Framework for Local Search Problems

Any local search problem on trees is defined by 1) the objective function used to evaluate the trees, 2) the tree edit operation used, and 3) the number k of edit operations to be

performed per local search step. The techniques developed in this paper can be used to efficiently solve any local search problem for which the objective function and tree edit operation are such that the number of edit operations on a tree S that are "dependent" (defined analogously) on any given edit operation on S is bounded by a constant c_1 (e.g., see Lemma 3.3), and the number of edit operations which have any given edit operation as a dependent is bounded by a constant c_2 (e.g., see Lemma 3.4). Let $c = \max\{c_1, c_2\}$. We will show that if this property is satisfied, then the local search problem is fixed parameter tractable in k. In particular, we show that such local search problems can be solved in $O^*(2^{k^2} \cdot (c^k k^2)^k)$ time.[4]

Consider a sequence of k edit operations. Then each of these edit operations can be dependent or independent with regards to each of the edit operations performed before it. Thus, given the first operation, there are two choices (dependent or independent with regards to the first one), 2^2 choices for the third one (dependent or independent with regards to the first and second ones), and so on. For the k total edit operations, this gives us $2^{\sum_{i=1}^{k-1} i}$, i.e., $\Theta(2^{k^2})$, possible cases. We will show how to compute a minimum reconciliation cost tree for any one of these $\Theta(2^{k^2})$ possible cases within $O^*(c^{k^2}k^{2k})$ time. A minimum reconciliation cost tree overall will then be the final solution.

Consider any one of the $\Theta(2^{k^2})$ possible cases: We can now, in polynomial time, partition the given sequence of k operations into disjoint maximal subsequences such that each edit operation in a subsequence is dependent on at least one other operation that occurs earlier in that subsequence. Thus, given any edit operation, say a, in a subsequence, it must be independent with regards to all edit operations that occur in a different subsequence and which also occur before a in the original unpartitioned sequence. Let us call this property the "separation" property. We will compute a minimum reconciliation cost tree for each subsequence separately. To compute a minimum reconciliation cost tree for any given subsequence, we make use of the fact that each edit operation in the subsequence (except the very first one) is dependent on at least some other edit operation in that subsequence. Thus, there are at most c candidates for each operation, with the exception of the first one. Thus, if j denotes the length of the given subsequence, then the subsequence actually represents $O^*(c^j)$ possible assignments of edit operations. Since $j < k$, all possible costs for each subsequence can thus be independently evaluated within $O^*(c^k)$ time.

We must now pick an assignment of edit operations for each subsequence such that the separation property of the subsequences is maintained, and such that the sum of the costs associated with the assigned edit operations is minimum. To do this, we use a technique similar to the one used to obtain tree T_2 in Algorithm 1. First, we sort all the $O^*(c^k)$ possible assignments, for each subsequence, by increasing costs. This requires $O^*(c^k)$ time. Let x denote the total number of subsequences. Consider the first $O(c^k kx)$ assignments for each subsequence. It can be shown (using an argument similar to the one used in the proof of Lemma 4.2) that it is possible to pick one of these $O(c^k kx)$ assignments for each subsequence such that the separation property is maintained and the sum of the costs associated with the chosen assignments is minimum overall.[5] Such a selection of assignments can be naively found by trying out all combinations in $O^*((c^k kx)^x)$, i.e., $O^*(c^{kx}(kx)^x)$, time.

Thus, since $x = O(k)$, we can conclude that the total time complexity of solving the local search problem is $O^*(2^{k^2} \cdot c^{k^2}k^{2k})$. In practice, for big k, the complexity of this approach would be much too high to be of any real use, however, our analysis does demonstrate how the ideas developed in this paper can be applied to other edit operations and objective functions.

8 Experimental Results

We evaluated the efficacy and efficiency of our novel algorithms by applying them to simulated data sets, and through comparative studies. For this evaluation, we implemented our algorithms for 1- and 2-NNI in the program DupTreeNNI.[6] In our experiments, we first compared our faster algorithm for 1-NNI to the program GeneTree [9], which implements the same standard NNI local search heuristic. Our results show that DupTreeNNI greatly outperforms GeneTree. And second, we demonstrate through our experiments that 2-NNI-based local searches are significantly better than those based on 1-NNI, while being just as efficient asymptotically.

Our experiments were performed on a workstation with a 2.40-GHz Intel Core 2 Quad Processor. All starting species trees for the heuristic searches were generated by using a standard random sequence addition algorithm (implemented in [38]).

8.1 Performance and Scalability

In order to verify the exceptional performance of our algorithms, we compared DupTreeNNI (using 1-NNI) against the program GeneTree which implements the currently best known (naive) algorithm for the 1-NNI local search problem. We measured the runtime of each program to compute its final species supertree for the same set of input gene trees and the same starting species tree. The input gene trees for each run consists of a set of 20 randomly generated gene trees, all with the same set of taxa. We used input instances having 100, 200, 400, 1,000, and 10,000 taxa in the input trees, and for each instance, we performed 10 runs. DupTreeNNI shows a vast improvement in runtime and scalability compared to GeneTree (see Table 1).

Note that both programs implement exactly the same standard 1-NNI-based local search heuristic. However, the final reconciliation costs obtained by the two programs on the same input may still be different because when multiple equally optimal trees are encountered during a local search step, ties are broken arbitrarily. In practice, we observed little or no difference in the final reconciliation costs.

4. The O^* notation ignores the polynomial component of the fixed parameter algorithm.

5. It is also possible that such an assignment for the x independent sequences does not exist. If this happens, then we can simply ignore this particular case.

6. DupTreeNNI was implemented on top of base code from the program DupTree [38], which implements standard local search heuristics for the gene-duplication problem.

TABLE 1
Time Comparison of GeneTree and DupTreeNNI

Taxa	GeneTree			DupTreeNNI
	min	max	avg	max
100	4s	13s	8s	$<$ 1s
200	43s	1:46s	1:09s	1s
300	2:03s	4:51s	3:23s	1s
400	5:12s	11:16s	7:42s	2s
1,000	–	–	–	8s
10,000	–	–	–	29:09s

8.2 Evaluating the Applicability and Performance of 2-NNI

Local searches based on 2-NNI are more desirable than those based on 1-NNI because of the following two reasons: 1) the 2-NNI neighborhood is much larger than the 1-NNI neighborhood and therefore local search heuristics based on 2-NNI could be expected to yield better results and 2) we could show that the 2-NNI local search problem could be solved in the same time complexity as the 1-NNI problem. Our experiments strongly validate both of these points. Table 2 shows the runtime comparison between local search heuristics based on 1-NNI and 2-NNI. We performed 50 runs for each taxa set size (except for 5,000 and 10,000 taxa, for which only 10 runs were performed). Observe that irrespective of the size of the input data set, local search heuristics based on 2-NNI are, on average, only about 10 times slower than those based on 1-NNI. This is partly because of the larger constant factor in the time complexity for 2-NNI local search and partly because of larger number of local search steps in the heuristic search, a consequence of the larger neighborhood size.

Table 3 compares the performance (in terms of final reconciliation cost) of 1-NNI and 2-NNI. As before, we performed 50 runs for each taxa set size (except for 5,000 and 10,000 taxa, for which only 10 runs were performed). It can be clearly seen that heuristics based on 2-NNI outperform heuristics based on 1-NNI in terms of the reconciliation costs of the inferred species trees. In fact, in all our experiments, we observed that heuristics based on 2-NNI were approximately twice as effective at reducing the reconciliation cost as heuristics based on 1-NNI. For example, for the input data set with 1,000 taxa, the average reconciliation cost of the starting species trees was 10,443; this implies that the average difference between the reconciliation costs for the starting species tree and the final species trees was 12 for the heuristic based on 1-NNI, and 26 for the heuristic based on 2-NNI.

It should also be noted that the impact of the iterative speedup increases as the number of gene trees increases. For example, for 20 gene trees on 200 taxa, the number of iterations, on average, performed by 1-NNI- and 2-NNI-based heuristics was 7 and 12, respectively; however, for 1,000 gene trees over the same taxa, the number of iterations were 38 and 56, respectively. Recall that every subsequent iteration of 1-, 2-, and 3-NNI local search (after the first local search step) requires only linear time. Typical gene tree parsimony analyses involve large numbers of gene trees, and heuristics that make use of our algorithms are therefore extremely efficient in practice.

9 Outlook and Conclusion

We introduced algorithms that significantly speed up NNI-based local search heuristics for the duplication problem. These algorithms extend naturally to local search problems based on the *Edge Contraction and Refinement (ECR)* edit operation [19], [20]. Thus, heuristic searches involving p instances of the 1-, 2-, or 3-ECR-Search problems can all be completed in $O(rm(n+p))$ time as well.

Our algorithms form the basis for extremely efficient local search heuristics. In particular, our 2- and 3-NNI local search algorithms can greatly improve on the performance of classical 1-NNI local search heuristics, without sacrificing efficiency. It is known, however, that local search heuristics based on SPR or TBR tend to be more effective than those based on NNI. Traditionally, local search heuristics based on 1-NNI have been used to perform gene tree parsimony analyses on trees that were too large to be analyzed using SPR- or TBR-based heuristics [3], [9]. Even though recent results have greatly reduced the time complexity of SPR and TBR local searches [17], [18], our new algorithms for 1-, 2-, and 3-NNI imply that local searches based on NNI are still the fastest by far. Therefore, the real power of our algorithms can be best exploited as part of a heuristic that mixes 1-, 2-, and 3-NNI local searches with SPR and TBR local searches (see [14]).

TABLE 2
Time Comparison for 1-NNI and 2-NNI

Taxa	1–NNI				2–NNI			
	min	max	avg	st dev	min	max	avg	st dev
100	$<$1s	1s	$<$1s	$<$1s	$<$1s	2s	1s	1s
200	$<$1s	1s	$<$1s	$<$1s	2s	7s	4s	1s
300	$<$1s	1s	1s	$<$1s	7s	13s	9s	1s
400	1s	2s	1s	$<$1s	12s	18s	15s	1s
500	1s	2s	2s	$<$1s	17s	30s	23s	3s
1,000	7s	8s	8s	$<$1s	1:08s	1:43s	1:26s	9s
2,000	33s	36s	34s	1s	4:55s	5:41s	5:19s	12s
4,000	2:33s	3:13s	2:52s	13s	22:07s	32:02s	27:13s	2:54s
5,000	4:16s	4:27s	4:21s	4s	35:41s	47:17s	38:30s	3:13s
10,000	22:08s	29:09s	23:57s	2:27s	3:03:55s	4:10:06s	3:53:50s	18:57s

TABLE 3
Comparison of Final Reconciliation Costs for 1-NNI and 2-NNI

Taxa	1–NNI				2–NNI			
	min	max	avg	st dev	min	max	avg	st dev
100	998	1,041	1,021	10.41	982	1,029	1,012	10.17
200	2,037	2,095	2,067	15.84	2,033	2,089	2,058	16.44
300	3,077	3,146	3,116	18.36	3,064	3,139	3,105	18.51
400	4,128	4,208	4,167	18.29	4,113	4,198	4,156	18.85
500	5,155	5,281	5,213	25.56	5,149	5,277	5,200	25.44
1,000	10,377	10,504	10,431	32.44	10,356	10,485	10,417	32.53
2,000	20,726	20,960	20,847	54.27	20,710	20,939	20,830	54.15
4,000	41,501	41,840	41,712	75.92	41,480	41,820	41,691	76.27
5,000	52,114	52,268	52,186	48.19	52,084	52,246	52,159	47.19
10,000	104,147	104,367	104,259	83.35	104,112	104,329	104,224	81.82

Such a heuristic would be both fast and effective, which would enable much larger analyses to be performed within a reasonable time. The ideas developed in this paper, along with our fixed parameter algorithm for generalizations of the k-NNI problem to other edit operations and objective functions, might also find use in other settings.

ACKNOWLEDGMENTS

The authors thank Gordon Burleigh and the anonymous referees for their invaluable comments. This work was supported in part by US National Science Foundation grant no. 0334832. A preliminary version of this paper appeared at the 2008 International Symposium on Bioinformatics Research and Applications (ISBRA) [1].

REFERENCES

[1] M.S. Bansal and O. Eulenstein, "The Gene-Duplication Problem: Near-Linear Time Algorithms for NNI-Based Local Searches," *Proc. Int'l Symp. Bioinformatics Research and Applications (ISBRA),* pp. 14-25, May 2008.

[2] M. Goodman, J. Czelusniak, G.W. Moore, A.E. Romero-Herrera, and G. Matsuda, "Fitting the Gene Lineage into Its Species Lineage: A Parsimony Strategy Illustrated by Cladograms Constructed from Globin Sequences," *Systematic Zoology,* vol. 28, pp. 132-163, 1979.

[3] R. Guigó, I. Muchnik, and T.F. Smith, "Reconstruction of Ancient Molecular Phylogeny," *Molecular Phylogenetics and Evolution,* vol. 6, no. 2, pp. 189-213, 1996.

[4] I. Wapinski, A. Pfeffer, N. Friedman, and A. Regev, "Automatic Genome-Wide Reconstruction of Phylogenetic Gene Trees," *Proc. Int'l Conf. Intelligent Systems for Molecular Biology/European Conf. Computational Biology (ISMB/ECCB) (Supplement of Bioinformatics),* pp. 549-558, July 2007.

[5] I. Wapinski, A. Pfeffer, N. Friedman, and A. Regev, "Natural History and Evolutionary Principles of Gene Duplication in Fungi," *Nature,* vol. 449, pp. 54-61, 2007.

[6] L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad, "Bayesian Gene/Species Tree Reconciliation and Orthology Analysis Using MCMC," *Proc. Int'l Conf. Intelligent Systems for Molecular Biology (ISMB) (Supplement of Bioinformatics),* pp. 7-15, 2003.

[7] L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad "Gene Tree Reconstruction and Orthology Analysis Based on an Integrated Model for Duplications and Sequence Evolution," *Proc. Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB),* pp. 326-335, 2004.

[8] B. Ma, M. Li, and L. Zhang, "From Gene Trees to Species Trees," *SIAM J. Computing,* vol. 30, no. 3, pp. 729-752, 2000.

[9] R.D.M. Page, "GeneTree: Comparing Gene and Species Phylogenies Using Reconciled Trees," *Bioinformatics,* vol. 14, no. 9, pp. 819-820, 1998.

[10] R.D.M. Page and E.C. Holmes, *Molecular Evolution: A Phylogenetic Approach.* Blackwell Science, 1998.

[11] C. Semple and M. Steel, *Phylogenetics,* pp. 30-33. Oxford Univ. Press, 2003.

[12] B.L. Allen and M. Steel, "Subtree Transfer Operations and Their Induced Metrics on Evolutionary Trees," *Annals of Combinatorics,* vol. 5, pp. 1-13, 2001.

[13] M. Bordewich and C. Semple, "On the Computational Complexity of the Rooted Subtree Prune and Regraft Distance," *Annals of Combinatorics,* vol. 8, pp. 409-423, 2004.

[14] R.D.M. Page and M.A. Charleston, "From Gene to Organismal Phylogeny: Reconciled Trees and the Gene Tree/Species Tree Problem," *Molecular Phylogenetics and Evolution,* vol. 7, pp. 231-240, 1997.

[15] D.L. Swofford, G.J. Olsen, P.J. Waddel, and D.M. Hillis, "Phylogenetic Inference," *Molecular Systematics,* D.M. Hillis, C. Moritz, and B.K. Mable, eds., chapter 11, pp. 407-509, Sinauer Assoc., 1996.

[16] D. Chen, O. Eulenstein, D. Fernández-Baca, and J.G. Burleigh, "Improved Heuristics for Minimum-Flip Supertree Construction," *Evolutionary Bioinformatics,* vol. 2, pp. 347-356, 2006.

[17] M.S. Bansal, J.G. Burleigh, O. Eulenstein, and A. Wehe, "Heuristics for the Gene-Duplication Problem: A $\Theta(n)$Speedup for the Local Search," *Proc. Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB),* pp. 238-252, 2007.

[18] M.S. Bansal and O. Eulenstein, "An $\Omega(n^2/\log n)$ Speedup of TBR Heuristics for the Gene-Duplication Problem," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 5, no. 4, pp. 514-524, July-Sept. 2008.

[19] G. Ganapathy, V. Ramachandran, and T. Warnow, "Better Hill-Climbing Searches for Parsimony," *Proc. Workshop Algorithms in Bioinformatics (WABI),* pp. 245-258, 2003.

[20] G. Ganapathy, V. Ramachandran, and T. Warnow, "On Contract-and-Refine Transformations between Phylogenetic Trees," *Proc. Symp. Discrete Algorithms (SODA),* pp. 900-909, 2004.

[21] R.D.M. Page, "Maps between Trees and Cladistic Analysis of Historical Associations Among Genes, Organisms, and Areas," *Systematic Biology,* vol. 43, no. 1, pp. 58-77, 1994.

[22] B. Mirkin, I. Muchnik, and T.F. Smith, "A Biologically Consistent Model for Comparing Molecular Phylogenies," *J. Computational Biology,* vol. 2, no. 4, pp. 493-507, 1995.

[23] O. Eulenstein, "Predictions of Gene-Duplications and Their Phylogenetic Development," PhD dissertation, Univ. of Bonn, GMD Research Series no. 20/1998, 1998.

[24] L. Zhang, "On a Mirkin-Muchnik-Smith Conjecture for Comparing Molecular Phylogenies," *J. Computational Biology,* vol. 4, no. 2, pp. 177-187, 1997.

[25] K. Chen, D. Durand, and M. Farach-Colton, "Notung: A Program for Dating Gene Duplications and Optimizing Gene Family Trees," *J. Computational Biology,* vol. 7, pp. 429-447, 2000.

[26] P. Bonizzoni, G.D. Vedova, and R. Dondi, "Reconciling a Gene Tree to a Species Tree under the Duplication Cost Model," *Theoretical Computer Science,* vol. 347, nos. 1/2, pp. 36-53, 2005.
[27] P. Górecki and J. Tiuryn, "On the Structure of Reconciliations," *Proc. Research in Computational Molecular Biology (RECOMB) Comparative Genomics Workshop,* pp. 42-52, Oct. 2004.
[28] M.A. Bender and M. Farach-Colton, "The LCA Problem Revisited," *Proc. Latin Am. Theoretical Informatics Symp. (LATIN),* pp. 88-94, 2000.
[29] J.B. Slowinski, A. Knight, and A.P. Rooney, "Inferring Species Trees from Gene Trees: A Phylogenetic Analysis of the Elapidae (Serpentes) Based on the Amino Acid Sequences of Venom Proteins," *Molecular Phylogenetics and Evolution,* vol. 8, pp. 349-362, 1997.
[30] R.D.M. Page, "Extracting Species Trees from Complex Gene Trees: Reconciled Trees and Vertebrate Phylogeny," *Molecular Phylogenetics and Evolution,* vol. 14, pp. 89-106, 2000.
[31] R.D.M. Page and J. Cotton, "Vertebrate Phylogenomics: Reconciled Trees and Gene Duplications," *Proc. Pacific Symp. Biocomputing,* pp. 536-547, 2002.
[32] J.A. Cotton and R.D.M. Page, "Tangled Tales from Multiple Markers: Reconciling Conflict between Phylogenies to Build Molecular Supertrees," *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life,* pp. 107-125, Springer-Verlag, 2004.
[33] M.J. Sanderson and M.M. McMahon, "Inferring Angiosperm Phylogeny from EST Data with Widespread Gene Duplication," *BMC Evolutionary Biology,* vol. 7, (Suppl. 1): S3, 2007.
[34] M. Fellows, M. Hallett, C. Korostensky, and U. Stege, "Analogs & Duals of the MAST Problem for Sequences & Trees," *Proc. European Symp. Algorithms (ESA),* pp. 103-114, 1998.
[35] U. Stege, "Gene Trees and Species Trees: The Gene-Duplication Problem is Fixed-Parameter Tractable," *Proc. Algorithms and Data Structures Symp. (WADS),* pp. 288-293, 1999.
[36] M.T. Hallett and J. Lagergren, "New Algorithms for the Duplication-Loss Model," *Proc. Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB),* pp. 138-146, 2000.
[37] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang, "On Distances between Phylogenetic Trees," *Proc. Symp. Discrete Algorithms (SODA),* pp. 427-436, 1997.
[38] A. Wehe, M.S. Bansal, J.G. Burleigh, and O. Eulenstein, "DupTree: A Program for Large-Scale Phylogenetic Analyses Using Gene Tree Parsimony," *Bioinformatics,* vol. 24, no. 13, pp. 1540-1541, 2008.

Mukul S. Bansal received the BTech degree in computer science and engineering from the International Institute of Information Technology, India, in July 2004, and the MS degree in computer science from Iowa State University (ISU), Ames, in December 2006. He is currently working toward the PhD degree in the Department of Computer Science at ISU. His research interests include computational biology and phylogenetics, graph theory, combinatorial optimization, approximation algorithms, and algorithms in general.

Oliver Eulenstein received the PhD degree in computational biology from the University of Bonn, with Thomas Lengauer in 1998, and was a postdoctoral fellow with Dan Gusfield at the University of California at Davis. He is an associate professor of computer science in the Department of Computer Science, Iowa State University, Ames. His research focuses on computational biology, particularly on computational phylogenetics.

André Wehe received the German diploma in information technology from the University of Applied Science of Bielefeld, Germany, in 2004. He is pursuing the PhD degree in computer engineering and comajoring in computer science at Iowa State University. He conducts research in parallel computing and computational biology. His current research focuses on algorithms for computational phylogenetics.