Better Hill-Climbing Searches for Parsimony

Ganeshkumar Ganapathy, Vijaya Ramachandran*, and Tandy Warnow**

Department of Computer Sciences, University of Texas, Austin, TX 78712;
{gsgk, vlr, tandy}@cs.utexas.edu

Abstract. The reconstruction of evolutionary trees is a major problem in biology, and many evolutionary trees are estimated using heuristics for the NP-hard optimization problem Maximum Parsimony. The current heuristics for searching through tree space use a particular technique, called "tree-bisection and reconnection", or TBR, to transform one tree into another tree; other less-frequently used transformations, such as SPR and NNI, are special cases of TBR. In this paper, we describe a new tree-rearrangement operation which we call the *p*-ECR move, for *p*-Edge-Contract-and-Refine. Our results include an efficient algorithm for computing the best 2-ECR neighbors of a given tree, based upon a simple data structure which also allows us to efficiently calculate the best neighbors under NNI, SPR, and TBR operations (as well as efficiently running the greedy sequence addition technique for maximum parsimony). More significantly, we show that the 2-ECR neighborhood of a given tree is incomparable to the neighborhood defined by TBR, and properly contains all trees within two NNI moves. Hence, the use of the 2-ECR move, in conjunction with TBR and/or NNI moves, may be a more effective technique for exploring tree space than TBR alone.

1 Introduction

The Maximum Parsimony Problem, also called the Hamming Distance Steiner Tree Problem, is one of the main optimization problems in phylogenetic analysis. Because it is NP-hard [5], heuristics are used to analyze datasets. Most of the favored heuristics operate by hill-climbing through tree space where each move changes a tree using some specific transformation, and then scores the new tree, and the search terminates when no allowed move improves the score. Transformations that are used in standard hill-climbing procedures are NNI, SPR, and TBR, with NNI being a special case of SPR, and SPR being a special case of TBR; thus, TBR searches are the most exhaustive, and also the most preferred [15]. Even TBR searches, however, can get caught in local optima (that is, trees that have no neighbors under TBR moves which are better and yet are not globally optimal)

The main result in this paper is a mathematical analysis of a new transformation, which we call *p*-edge-contract-and-refine, or *p*-ECR. This transformation is similar to other techniques described in other papers [2,9], and has a similar motivation; what is new here is the mathematical analysis. We provide a fast algorithm for computing the

* Supported by NSF grant CCR-9988160

** Supported by NSF grant EIA 01-21680 and by a fellowship from the David and Lucile Packard Foundation

G. Benson and R. Page (Eds.): WABI 2003, LNBI 2812, pp. 245–258, 2003.

optimal 2-ECR neighbors of a given tree, and show that the number of 2-ECR neighbors that are also TBR neighbors is small, namely $O(n)$, where n is the number of leaves. In contrast, we show that the size of the 2-ECR neighborhood is itself $\Theta(n^2)$, and it has been shown that the TBR neighborhood could be $\Theta(n^3)$. Our other main result is a simple algorithm, called *Three-Way-Labels*, which can be used to speed-up exhaustive search for optimal neighbors under these and other transformations on trees. See Section 7 for pointers to related work on these problems.

The rest of the paper is organized as follows. In Section 2 we describe the Maximum Parsimony problem, and describe an algorithm to compute the parsimony score of a given tree. In Section 3, we define the NNI, SPR and TBR moves and present some known properties about the neighborhoods induced by these moves. In Section 4, we describe the p-ECR move, and compare the neighborhoods defined by the 2-ECR, TBR, and NNI operations. In Section 5, we formally define the problem of finding the best neighbor under the p-ECR move and describe a general algorithmic technique that we use to obtain a fast algorithm for the optimal neighbor problem under the 2-ECR move. In Section 6 we describe how our general technique can be used to obtain fast algorithms for the optimal neighbor problem under NNI, SPR and TBR moves, and also for computing the Greedy Sequence Addition algorithm for maximum parsimony. We conclude with Section 7 where discuss related work.

2 Basics

2.1 The Maximum Parsimony Problem

The input to the Maximum Parsimony problem is a collection S of n strings of the same length k over a given alphabet, Σ; these are the "given" nodes. The Steiner nodes (i.e., the nodes which can be used to connect the given nodes together) are drawn from Σ^k, i.e., all strings of length k over Σ. The objective is a tree T, with the given nodes at the leaves, and internal nodes from Σ^k, which minimizes the sum of the Hamming distances on the edges, where the Hamming distance on an edge $e = (x,y)$, denoted $H(x,y)$, is the number of positions in which x and y differ. Informally, this quantity is the minimum number of changes (via point mutations) needed to explain the evolution of the dataset from a common ancestor. We formalize this as follows.

Definition 1. *Parsimony score of a tree*

Let S be a set of sequences of length k over the alphabet Σ. Let T be a binary tree with leaf set S, and let f be an assignment of sequences to the internal nodes of T. The score of T under the assignment f, denoted $score(T,f)$ equals $\sum_{(u,v)\in E(T)} H(f(u), f(v))$. The parsimony score of T, denoted by $pscore(T)$, is the minimum $score(T,f)$ over all possible assignments f.

We now define the Maximum Parsimony (MP) problem.

Definition 2. *The Maximum Parsimony Problem*

Input: *Set S of sequences of length k over an alphabet Σ.*

Output: *A binary tree T whose leaves are bijectively labeled with sequences in S, such that the parsimony score of T, $pscore(T)$, is minimum.*

2.2 Computing the Maximum Parsimony Score of a Fixed Tree

Although finding the most parsimonious tree is NP-hard, we can find the optimal labeling of the internal nodes of a given tree in polynomial time. The standard algorithm for this problem, by Fitch [6], is the basis of our *Three-Way-Labels* algorithm, and so is included here.

The input to the fixed-tree maximum parsimony problem is a set S of n strings over a fixed alphabet Σ; for the typical cases, Σ is either the set of four nucleotides, or the set of amino-acid sequences, and thus is quite small. The elements of Σ are called the "states". We make the typical assumption that the sequences are already aligned, so that all sequences have the same length k. The positions within the sequences are sometimes called "sites."

The algorithm operates as follows. First, the tree is rooted (arbitrarily), either at a leaf, or by subdividing an edge e and rooting the tree at the newly introduced node. The cost of the tree (also called its "length") is then computed using dynamic programming. The algorithm is usually described as having two phases, where the first phase computes the length of the tree as well as a representation of candidate labels (strings over Σ^k that would produce optimal scores) for the root of each subtree of the tree; the second phase then actually produces a specific labeling for each node achieving the optimal score. We are primarily interested in the first phase, which we modify for use in our *Three-Way Labels* algorithm. However, the whole algorithm is of general interest, and so we provide it here.

Note that each position within the strings can be handled separately, so it suffices to describe the fixed-tree maximum parsimony algorithm as though there were only one position to consider. Since we have rooted T (arbitrarily), for every internal node v in T, we can define the rooted subtree T_v, and also the children of v. We let $States_v$ denote the set of state assignments for the node v (i.e., elements from Σ) which are part of an optimal assignment of states to all nodes in T_v so as to minimize the total parsimony score in T_v. We assume that T is binary, and that v's children are x and y (the algorithm can be applied more generally, however), and we similarly define $States_x$ and $States_y$. Then, the following equality holds (see [6]):

$$\begin{array}{ll} v \text{ is a leaf}: & States_v = \{\text{state of } v\} \\ v \text{ has two children } x,\ y: & States_v = \begin{cases} States_x \cap States_y \text{ if } States_x \cap States_y \neq \emptyset \\ States_x \cup States_y \text{ otherwise} \end{cases} \end{array}$$

This allows us to compute $States_v$ for every node v in T, from the bottom up. The optimal cost, i.e. the parsimony score, of T can also be calculated from the bottom-up at the same time: every time $States_x \cap States_y = \emptyset$ we increment the parsimony score of the tree by one. (Since we perform this computation for each site – i.e., position – independently, the sum of these values over all the sites is the parsimony score of the tree.)

In the second phase, we obtain the labeling on the internal nodes using a pre-order traversal. Once again, we can handle the positions (sites) independently. For the root r arbitrarily assign the state for r to be any element of $States_r$. Then visit the remaining nodes in turn, every time assigning a state to the node v from its set $States_v$. When we visit a node v we will have already set the state of its parent, u. If the selected state for u is an element of $States_v$, then we use the same state; otherwise we pick a state arbitrarily from $States_v$.

This algorithm takes $O(nrk)$ time to compute the labeling of every node in T and the optimal length (i.e., maximum parsimony cost) of T, where $r = |\Sigma|$, $n = |S|$, and k is the sequence length.

3 Hill-Climbing Heuristics for MP Analysis

The general structure of a heuristic search is as follows:

- First, an initial tree (or set of trees) is obtained, typically using the Greedy Sequence Addition method (see Section 6.2).
- Then, for each tree in the initial set, a search is initiated in which the given tree is modified (using a transformation that modifies trees), and the new tree is then scored. This process is repeated until a local optimum is found – that is, a tree which has no neighbor that has a better score.
- Finally, of all the local optima found, the set of trees that have the best MP score, or a "consensus" of these trees is returned; sometimes, sub-optimal trees are also returned.

Note that since MP is NP-hard, a local optimum need not be globally optimal, and in general this scheme will not return optimal trees in polynomial time.

We now describe three currently used tree-rearrangement operations and present some properties of the neighborhood induced around a tree by each of the three operations. Our definitions closely follow those in [1].

Nearest Neighbor Interchange (NNI) The NNI move swaps one rooted subtree on one side of an internal edge e with another on the other side; note that this is equivalent to contracting the edge e, and then resolving the resultant tree into a new binary tree. See Figure 1 for an example of this procedure.

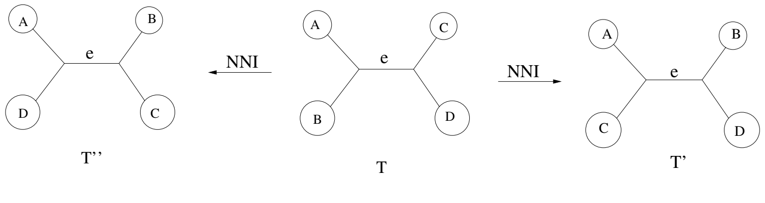

Fig. 1. Tree T can be transformed into either T' or T'' with one NNI move

Subtree Prune and Regraft (SPR) An SPR move on a tree T is defined as cutting any edge and thereby pruning a subtree, t, and then regrafting the subtree by the same cut edge to a new vertex obtained by subdividing a pre-existing edge in $T - t$. Any internal node that might arise that has degree two is suppressed in the resulting tree.

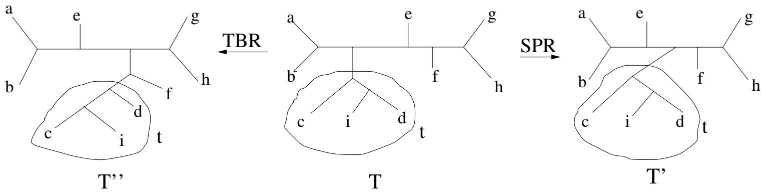

Fig. 2. Tree T' is one SPR move away from T, while T'' is one TBR move away.

Tree Bisection and Reconnection (TBR) In a TBR move an edge is removed from T, creating subtrees t and $T-t$, and then a new edge is added between the midpoints of any two edges in t and $T-t$, creating a new tree. Again, throughout the operation any internal node of degree two is suppressed. The last two operations are illustrated in Figure 2.

Each of the tree rearrangement operations described above naturally induces a distance metric in the space of trees. For instance, the NNI distance between two trees is defined as the minimum number of NNI moves required to transform one tree to another. The metrics induced by NNI, SPR and TBR moves have been discussed in [1]. We will denote the NNI metric by δ_{NNI}, the SPR metric by δ_{SPR}, and the TBR metric by δ_{TBR}. Note that every NNI move is an SPR move, and that every SPR move is a TBR move. Hence we have the following result:

Observation 1 *(From [12]) For any two unrooted leaf-labeled binary trees T and T' on the same set of leaves,*

$$\delta_{TBR}(T,T') \le \delta_{SPR}(T,T') \le \delta_{NNI}(T,T').$$

It is known that all of these distances are finite (Robinson showed this for the NNI distance in [14]).

Note that TBR searches explore a *superset* of trees, compared to both SPR and NNI, which is desirable. However, TBR searches are also more expensive, since there are more trees that are TBR neighbors of a given tree.

Induced neighborhoods We define the neighborhood of an unrooted binary leaf-labeled tree T under a tree-rearrangement move to be the set of all trees that can be obtained from T by one move. The following theorem, about the neighborhoods induced by NNI, SPR and TBR moves, is from [1].

Theorem 1. *[1] The size of the neighborhood for T is:*

1. $2n-6$ *for the NNI operation,*
2. $2(n-3)(2n-7)$ *for the SPR operation,*
3. *at most* $(2n-3)(n-3)^2$*, and dependent on the topology of T for the TBR operation.*

See [10,1,4] for results related to computing the distance between trees under these metrics, and [11,1] for results related to the maximum pairwise distance between trees under these metrics.

4 The p-ECR Operation

In this section we describe the p-edge-contract-and-refine (p-ECR) move, and we compare neighborhoods defined by the 2-ECR, TBR, and NNI operations. Our main results are Lemma 3 and Theorem 2, which show that for any tree T, (1) the size of the 2-ECR neighborhood is $\Theta(n^2)$, but that (2) there are at most $O(n)$ trees that are in both the 2-ECR neighborhood and the TBR neighborhood of T. We also show that the 2-ECR neighborhood strictly contains all trees within two NNI moves from T.

The p-ECR move is a generalization of the NNI move in the following sense: Since an NNI move can also be viewed as an edge contraction followed by a refinement at the newly created unresolved node, we can generalize NNI by contracting p edges all at once, creating unresolved nodes in the process, and then refining these unresolved nodes give back a binary tree. Note that this process is *not*, in general, equivalent to contracting and refining each of the p edges in succession. Indeed, in Figure 3 we give an example of two trees T and T' such that $\delta_{p-ECR}(T,T') = 1$, but $\delta_{NNI}(T,T') > p$.

Since NNI is the same as 1-ECR, it follows from [14] that $\delta_{p-ECR}(T,T')$ is finite for all pairs of binary trees on the same leaf set (i.e, we can go from any tree to any other tree through a sequence of p-ECR moves).

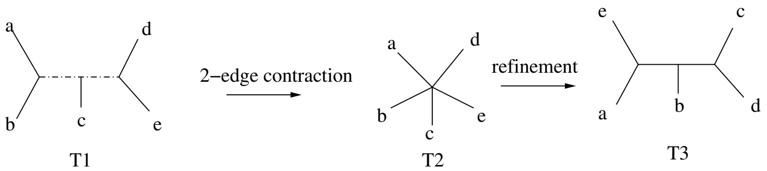

Fig. 3. A 2-ECR move. The dashed edges in T1 are contracted to give T2, and then T2 is fully refined to give T3. Note that $\delta_{NNI}(T1,T3) = 3$, although $\delta_{2-ECR}(T1,T3) = 1$.

4.1 Comparing p-ECR with TBR

In this section we show that the number of 2-ECR neighbors of a tree on n leaves is $\Omega(n^2)$, and that the number of 2-ECR neighbors that are also TBR neighbors is only $O(n)$.

We now define some concepts that will be necessary for our analyses. Every edge in a binary leaf-labeled tree T induces a *bipartition* of the set of leaves. Let the bipartition induced by an edge e be π_e. Then the set $C(T) = \{\pi_e | e \in E(T)\}$ uniquely defines the tree T (see [3,17]). In the subsequent discussion all trees will be assumed to be binary, leaf-labelled trees.

Definition 3. *Robinson-Foulds distance [13].*

The Robinson-Foulds *(RF) distance between two binary leaf-labeled trees T and T' is defined to be $|(C(T)\Delta C(T')|$, i.e, $|C(T) - C(T')|$ + $|C(T') - C(T)|$.*

Neither contraction nor refinement of a set of edges alters bipartitions induced by other edges in a tree, and hence we have the following:

Observation 2 *Let T and T' be two binary leaf-labeled trees. Then, for any $1 \leq p \leq n-3$, $\delta_{p-ECR}(T,T') = 1 \implies RF(T,T') \leq 2p$.*

We now define the *Maximum Agreement Forest* [10] between two binary leaf-labeled trees.

Let $F = \{t_1, t_2, \ldots, t_m\}$ be a forest of m trees that results from deleting $m-1$ edges from a tree T. Let F' be a forest of m trees obtained similarly from T'. F (or F') is said to be an *agreement forest* for T and T' iff $F = F'$. A *maximum agreement forest* (MAF) for T and T' is an agreement forest with the *minimum* number of trees.

Lemma 1. *(From [1]) Let T and T' be two binary leaf-labeled trees. Let F be a maximum agreement forest for T and T'. Then $\delta_{TBR}(T,T') = |F| - 1$.*

We now show that for every and n and every $1 < p < n-3$ there are trees whose p-ECR distance is less than their TBR distance, and vice versa.

Lemma 2. *The following is true for every natural number n and every natural number $p < n-3$:*

1. *$\exists T,\ T'$ s.t $\delta_{TBR}(T,T') < \delta_{p-ECR}(T,T')$.*
2. *$\exists T,\ T'$ s.t $\delta_{p-ECR}(T,T') = 1$ and $\delta_{TBR}(T,T') \in \Omega(p)$.*

Due to space requirements, we omit the proof; however, see Figure 4 for a pair of trees satisfying the first condition, and Figure 5 for a pair of trees satisfying the second condition.

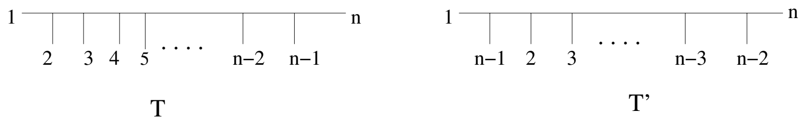

Fig. 4. $\delta_{TBR}(T,T') = 1$ but $RF(T,T') = 2n-6$.

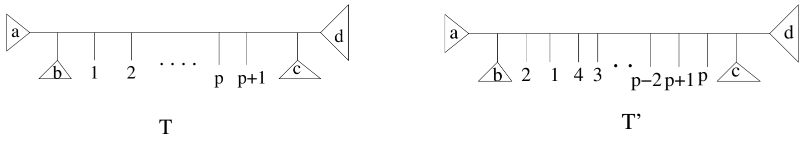

Fig. 5. $|MAF(T,T')| \in \Omega(p)$ but $\delta_{p-ECR}(T,T') = 1$.

We now prove some results about the neighborhood induced around a tree by the 2-ECR operation, and in particular we show that the neighborhood of a tree induced by the

2-ECR operation is very different from the one induced by the TBR operation. We will denote the 2-ECR neighborhood of a tree T as $N_{2-ECR}(T)$, and the TBR neighborhood as $N_{TBR}(T)$.

Lemma 3. *For any binary leaf-labeled tree T, $|N_{2-ECR}(T)| \in \Theta(n^2)$.*

Proof. Omitted due to space constraints.

We now prove that most of the trees in $N_{2-ECR}(T)$ are not in $N_{TBR}(T)$.

Theorem 2. *For a binary leaf-labeled tree T, $|N_{2-ECR}(T) \cap N_{TBR}(T)| \in O(n)$.*

Proof. Let $X(T) = N_{2-ECR}(T) - N_{NNI}(T)$. Note then that each tree $T' \in X(T)$ can be obtained by contracting two edges e_1 and e_2 in T, and then refining the resultant tree. Consider the set S of all trees T' in $X(T)$ such that the corresponding contracted edges e_1 and e_2 are separated in T by at least two edges. Note that there are only $\Theta(n)$ pairs of edges in T that are either adjacent, or separated by exactly one edge. Consequently, it follows that $|N_{2-ECR} - S| \in O(n)$.

We now show that $S \cap N_{TBR}(T) = \emptyset$. Suppose to the contrary that there is a tree $T' \in S \cap N_{TBR}(T)$. Note that $RF(T,T') = 4$, since $C(T) - C(T') = \{\pi_{e_1}, \pi_{e_2}\}$, where $C(T)$ is the set of bipartitions in T, and e_1 and e_2 are those two edges through whose contraction (and subsequent refinement) T' was obtained from T.

However, T and T' are one TBR move apart. Hence, it can be shown that if $\{\pi_{e_1}, \pi_{e_2}\} \subseteq C(T) - C(T')$, then the bipartitions induced by all edges (except, possibly, the edge that was broken in the TBR move) in the path in T between e_1 and e_2 are in $C(T) - C(T')$. Now, since e_1 and e_2 are separated by at least two edges, the set $C(T) - C(T')$ must contain at least one more bipartition, which is a contradiction. This completes our proof.

5 Computing Optimal 2-ECR Neighbors

In this section we consider the problem of finding an optimal neighbor under the p-ECR tree-rearrangement operation, and present a fast algorithm for solving the problem. The technique that we use to obtain the fast algorithm is general and can be used to obtain fast algorithms for the optimal neighbor problem under the NNI, SPR and TBR moves as well. We now define the Optimal 2-ECR Neighbor problem.

Definition 4. *Optimal 2-ECR Neighbors*

Input *An unrooted binary tree T on n leaves, each bijectively leaf-labeled by a set S of sequences of length k over an alphabet of size r.*

Output*: An unrooted binary tree T' on n leaves, each bijectively labelled by the same set S, such that T' has the minimum MP score among all such trees t for which $\delta_{2-ECR}(T,t) = 1$.*

Henceforth, we will call such a tree an *optimal* 2*-ECR-neighbor* of T. The *Optimal TBR-neighbor*, *Optimal SPR-neighbor* and *Optimal NNI-neighbor* problems are defined similarly. Note also that there can be more than one optimal neighbor, and that in general the objective is to find all optimal solutions.

At the outset we observe that a brute-force algorithm for the above problem would take $\Theta(n^3rk)$ time, since there are $\Theta(n^2)$ 2-ECR neighbors for any tree, and computing the parsimony score of each tree using Fitch's algorithm would take $\Theta(nrk)$ time. We will obtain a $\Theta(n^2rk)$ time algorithm for the Optimal 2-ECR problem which will return all the optimal neighbors.

As was observed in Section 4, an *NNI* move can be thought of as a 1-ECR move. So, not surprisingly, we use a fast algorithm for computing optimal NNI-neighbors in our algorithm for computing optimal 2-ECR-neighbors. A brute-force optimal NNI neighbors algorithm would run in $\Theta(n^2rk)$ time, but our algorithm runs in $\Theta(nrk)$ time.

5.1 An $O(nrk)$ Algorithm for the Optimal NNI Neighbor Problem

The way we obtain a speed-up over the brute-force techniques for each of the problems we address is by performing a preprocessing step in which we assign three labels to each node in the tree.

In order to understand why we do this preprocessing step, consider an NNI move across an edge, say (u,v), in a given tree T. Let W and X be the rooted subtrees below u, and let Y and Z be the rooted subtrees below v. The NNI move will, e.g, involve swapping W with Y. Let the resulting tree be T'. Supposing that we have the parsimony scores and optimal state assignments for the rooted subtrees W, X, Y and Z, the parsimony score of T' can be computed in $\Theta(rk)$ time, thus: we can subdivide edge (u,v) and root T' at the newly created node, which we shall call x. This produces a binary tree rooted at x, with subtrees off x rooted at u and v. The parsimony score of the subtree of T' rooted at u depends just on the parsimony scores and optimal state assignments of X and Y, and can be computed from them in $\Theta(rk)$ time, as in Fitch's algorithm. Similarly, the parsimony score of the subtree of T' rooted at v can be computed in $\Theta(rk)$ time. Finally, the parsimony score of T' (rooted at x) can be computed in $O(rk)$ time from the parsimony scores and optimal state assignments of the subtrees rooted at u and v.

The above observations suggest that a preprocessing stage that computes the parsimony score and the optimal state assignments for every rooted subtree will let us compute the parsimony score of each NNI neighbor in $\Theta(rk)$ time. The brute-force way of performing this preprocessing step would take $\Theta(n^2rk)$ time, but we will next see how to perform this preprocessing stage in $\Theta(nrk)$ time. We will call the preprocessing step the Three-Way Labels algorithm since it would assign three optimal state-assignment labels to each internal node.

5.2 Three-Way Labels: The Dynamic Programming Algorithm

In a tree T, consider an internal node v with three neighbors a, b and c, as in Figure 6. The node v is the root of three rooted subtrees, one where a and b are its children ($tree(v,a,b)$ in the figure), one where a and c are its children ($tree(v,a,c)$) and one where b and c are its children ($tree(v,b,c)$). The preprocessing step would involve assigning three labels (optimal state assignments) for each such internal node v - namely the optimal state assignments at the roots of $tree(v,a,b)$, $tree(v,b,c)$ and $tree(v,a,c)$.

The parsimony score and the optimal state assignment of, for example, $tree(v,a,b)$ can be computed from the parsimony score and optimal state assignments of the subtrees rooted at a and b. Note that the subtrees of $tree(v,a,b)$ rooted at a and b have fewer leaves than $tree(v,a,b)$. This suggests the following dynamic programming algorithm:

Bucket sort the rooted subtrees in T by the number of leaves in the subtree in $O(n)$ time. For subtrees that contain just a single leaf, the label is just the sequence at the leaf. For subtrees such as $tree(v,a,b)$, the label is computed by in the usual way: for a given site, if the corresponding sets at a and b are disjoint we take the union of the sets, and otherwise we take the intersection. Note the when we compute the label at v corresponding to $tree(v,a,b)$, the necessary labels at a and b are already available since the subtrees rooted a and b are smaller. There are $O(n)$ rooted trees like $tree(v,a,b)$, and for each of them the optimal state assignment at the root can be computed in $\Theta(rk)$ time using the dynamic programming technique. Also, the parsimony score of the rooted subtrees can be computed along side their optimal state assignments.

Therefore, we have the following:

Lemma 4. *The Three-Way Labels algorithm takes $O(nrk)$ time, where n is the number of leaves in the tree T, and each leaf is labeled by a sequence of length k over an alphabet of size r.*

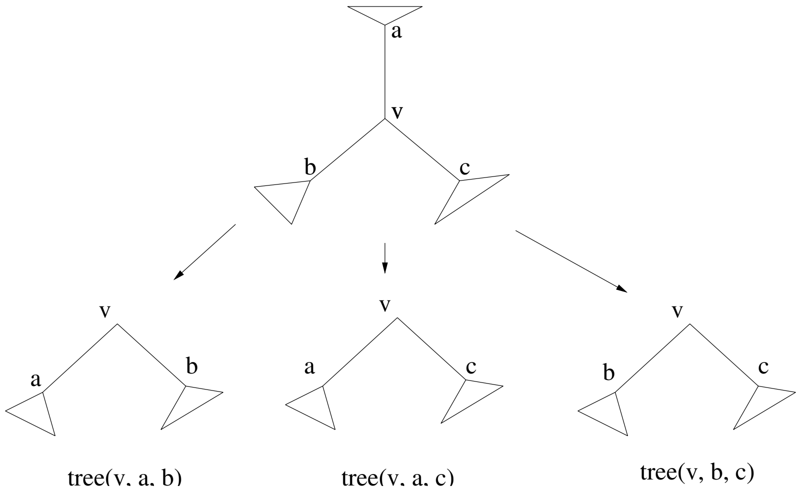

Fig. 6. Internal node v and the three subtrees associated with it.

As we saw in the previous section, the preprocessing stage would let us compute the parsimony score of each NNI-neighbor in $\Theta(rk)$ time. Since there are $2n-6$ NNI-neighbors, the optimal NNI neighbors can be identified in $\Theta(nrk)$ time after the preprocessing step. To summarize,

Theorem 3. *We can solve the Optimal NNI-Neighbors Problem in* $\Theta(nrk)$ *time.*

5.3 Computing an Optimal 2-ECR Neighbor

We now show how to compute an optimal 2-ECR neighbor of an unrooted binary tree on n labeled leaves, each labeled by a sequence of length k over an alphabet of size r, in $\Theta(n^2rk)$ time, thus spending only $\Theta(rk)$ time per neighbor.

A 2-ECR move on a tree T is specified by the two edges e_1 and e_2 to be contracted, and the refinement of the resulting contracted tree into an unrooted binary tree that differs from T.

Our algorithm will handle the following two cases separately.

1. The edges e_1 and e_2 are not adjacent to each other.
2. The edges e_1 and e_2 are adjacent to each other.

We now show how to handle case (1). We first state a lemma that in this case any 2-ECR move can be "simulated" by two successive NNI moves. We omit the proof because of space limitations.

Lemma 5. *Let T be an unrooted leaf-labeled tree and let T' be a 2-ECR neighbor of T such that the 2-ECR move involves the contraction and refinement of two non-adjacent edges in T. Then T' can be reached from T through two NNI moves.*

We now continue with the discussion of the Optimal 2-ECR neighbors algorithm.
Case 1: the edges are not adjacent By Lemma 5, in this case the optimal 2-ECR neighbors can be obtained by two sequential NNI moves. To compute the optimal 2-ECR neighbors of T in this case, we compute the optimal NNI neighbors of every NNI neighbor of T. There are $\Theta(n)$ NNI neighbors of T, and the optimal NNI neighbors of a given tree can be found in $\Theta(nrk)$ time by Theorem 3. Hence, the set of optimal 2-ECR neighbors can be computed in $\Theta(n^2rk)$ time for this case.
Case 2: the edges are adjacent Note that on a tree with n leaves, there are only $O(n)$ pairs of adjacent edges. For each possible way of contracting a pair of adjacent edges, we create a tree with a single unresolved node (that is, a node of degree more than three), and the unresolved node has degree 5. Hence, there are 15 possible binary trees that resolve each such tree. Furthermore, each of the 15 refinements involves only a rearrangement of the 5 rooted subtrees off the unresolved node around the two new edges that result from the refinement. Therefore, the algorithm operates as follows. First, we compute the optimal labels at the root of all such subtrees in $\Theta(nrk)$ time in a preprocessing step as in Section 5.2. Then, for each of the $O(n)$ pairs of adjacent edges, in $O(rk)$ additional time we can compute the optimal neighbors obtainable by contracting and refining those edges. Hence, for the case of adjacent edges, can compute the set of optimal 2-ECR neighbors in $\Theta(nrk)$ time.

The overall optimal 2-ECR neighbors will be those with the best score, and hence we have the following:

Theorem 4. *Let T be an unrooted binary tree on n leaves, each labeled by a sequence of length k over an alphabet of size r. Then the optimal 2-ECR neighbors of T can be computed in $\Theta(n^2rk)$ time.*

6 Application of Three-Way Labeling to Other Problems

In this section we describe how our Three-Way labeling algorithm can be used to compute the set of optimal SPR and TBR neighbors in $O(n^2rk)$ time and $O(n^3rk)$ time respectively. We then describe how the technique can be used to compute the Greedy Sequence Addition parsimony algorithm in $O(n^2rk)$ time.

6.1 Optimal SPR and TBR Neighbors

An SPR move on a tree T to create a tree T' involves the following three steps:

- Delete an edge $e = (x_1, x_2)$ from T, thus producing two trees T_1 and T_2 with $x_1 \in V(T_1)$ and $x_2 \in V(T_2)$.
- Pick one of the two subtrees (say, T_1). Then, pick an edge e_2 in T_2, and subdivide e_2, thus creating a new node v_2.
- Add the edge (x_1, v_2).

To compute the parsimony score of T', we can root T' at the edge (x_1, v_2). We will then need the parsimony score and optimal state assignments of T_2 rooted at v_2 and those of T_1 rooted at x_1. We perform a Three-Way labeling of the nodes in T_1 and T_2. This takes $O(nrk)$ time, and would allow us to compute the parsimony score and optimal state assignments of T_2 rooted at v_2 and those of T_1 rooted at x_1 in $O(rk)$ time. Once this information is available, the parsimony score of T' can be computed in $O(rk)$ time. Thus, for a fixed way of deleting an edge (x_1, x_2), all SPR neighbors can be evaluated in $O(nrk)$ time. There are $O(n)$ ways of deleting an edge, and thus we can evaluate all SPR neighbors and identify the optimal ones in $O(n^2rk)$ time.

As for TBR, the only difference here is that for every way of deleting an edge (x_1, x_2) in T_1, there can be $O(n^2)$ TBR neighbors. Evaluating all these neighbors can be done in $O(n^2rk)$ time if we Three-Label the two trees (T_1 and T_2) that result from the deletion of (x_1, x_2) from T. There are $O(n)$ ways of deleting an edge from T, and thus we can evaluate all TBR neighbors and identify the optimal ones in $O(n^3rk)$ time.

6.2 A Faster Algorithm for Greedy Sequence Addition Parsimony

We begin by describing a brute-force algorithm for the Greedy-MP algorithm.

Brute-Force Greedy MP Greedy-MP constructs a tree for a set S of sequences based upon a specified (usually random) ordering on S; suppose that ordering is $s_1, s_2, \ldots, s_n$. It begins with the star tree on the first three sequences, s_1, s_2, and s_3, and then sequentially adds each of the remaining sequences into the tree it has constructed so far. Before it attempts to insert the i^{th} sequence, s_i, it has a tree t_{i-1} on the first $i-1$ sequences. In order to insert s_i into t_{i-1}, it computes the length of each possible extension of t_{i-1}, in the obvious way: for each way of adding s_i (by subdividing an edge in t_{i-1}, and making the newly created node the parent of s_i), it uses Fitch's algorithm (see Section 2.2) to score the resultant tree. If there is a tie (more than one way of adding s_i gives a minimal score), then a best tree is selected arbitrarily. When all sequences have been added, the resultant tree is returned.

The running time of this brute-force algorithm follows from the analysis of the cost of adding s_i to the tree t_{i-1}. First, note that there are $O(i)$ ways to add s_i to t_{i-1}, and that scoring the resultant trees costs $O(irk)$ per tree, where r is the alphabet size, and k is the sequence length. Hence, computing t_i, given t_{i-1}, costs $O(i^2rk)$. Since we do this for $i = 4, 5, \ldots, n$, the total cost is $O(n^3rk)$.

Faster Greedy-MP If we do a Three-Way Labeling of t_{i-1}, then we can compute the optimal placement of s_i into t_i in only $O(irk)$ time, so that Greedy-MP can be completed in $O(n^2rk)$ time.

7 Related Work

In the "sectorial search" technique used in the parsimony software TNT, developed by Goloboff *et. al* ([9]), repeatedly a set of edges is identified (using some specific technique) to be contracted and then refined. This can be viewed as a p-ECR based search, where the value for p is determined indirectly. The general approach of contracting edges and then finding an optimal resolution has also been suggested in [2]. Empirical comparisons in [9] of sectorial search to other search strategies suggested that this kind of approach would be potentially useful. Our contribution here is theoretical rather than empirical, and our findings are consistent with those positive observations reported in [9].

Our other main contribution, namely the 3-Way-Labels algorithm, and its use in finding optimal neighbors under various tree transformations, is new, but similar techniques have been presented before (see [16,7,8]).

8 Acknowledgments

We thank an anonymous referee for pointing out earlier papers related to this work.

References

1. B. Allen and M. Steel. Subtree Transfer Operations and their Induced Metrics on Evolutionary Trees. *Annals of Combinatorics*, 5:1–15, 2001.
2. M. Bonet, M. Steel, T. Warnow, and S. Yooseph. Better Methods for Solving Parsimony and Compatibility. *Journal of Computational Biology*, 5(3):409–422, 1998.
3. P. Buneman. The Recovery of Trees from Measures of Dissimilarity. *Mathematics in the Archaelogical and Historical Sciences*, pages 387–395, 1971.
4. B. Dasgupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On the Distances Between Phylogenetic Trees. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 427–436. ACM-SIAM, 1997.

5. L. R. Foulds and R. L. Graham. The Steiner problem in Phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.
6. W. Fitch. Toward Defining Course of Evolution: Minimum Change for a Specified Tree Topology. *Systematic Zoology*, 20:406–416, 1971.
7. P. A. Goloboff. Character Optimization and Calculation of Tree Lengths. *Cladistics*, 9:433–436, 1994.
8. P. A. Goloboff. Methods for Faster Parsimony Analysis. *Cladistics*, 12:199–220, 1996.
9. P. A. Goloboff. Analyzing Large Datasets in Reasonable Times: Solutions for Composite Optima. *Cladistics*, 15:415–428, 1999.
10. J. Hein, T. Jiang, L. Wang, and K. Zhang. On the Complexity of Comparing Evolutionary trees. *Discrete Applied Mathematics*, 71:153–169, 1996.
11. M. Li, J. Tromp, and L. Zhang. On the Nearest Neighbour Interchange Distance Between Evolutionary Trees. *Journal of Theoretical Biology*, 182:463–467, 1996.
12. D. R. Maddison. The Discovery and Importance of Multiple Islands of Most Parsimonious Trees. *Systematic Zoology*, 43(3):315–328, 1991.
13. D. F. Robinson and L. R. Foulds. Comparison of Phylogenetic Trees. *Mathematical Biosciences*, 53:131–147, 1981.
14. D. F. Robinson. Comparison of Labeled Trees with Valency Three. *Journal of Combinatorial Theory*, 11:105–119, 1971.
15. D. Swofford, G. J. Olson, P. J. Waddell, and D. M. Hillis. *Molecular Systematics*, chapter Phylogenetic Inference, pages 407–425. Sinauer Associates, Sunderland, Massachusetts, second edition, 1996.
16. D. L. Swofford. *Studies in Numerical Cladistics: Phylogentic Inference Under the Principle of Maximum Parsimony*. PhD thesis, University of Illinois at Urbana-Champaign, 1986.
17. T. Warnow. Tree Compatibility and Inferring Evolutionary History. *Journal of Algorithms*, 16:388–407, 1994.