BIOINFORMATICS ORIGINAL PAPER Vol. 25 no. 2 2009, pages 190–196
doi:10.1093/bioinformatics/btn606

Phylogenetics

A practical method for exact computation of subtree prune and regraft distance

Yufeng Wu

Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA

Received on September 26, 2008; revised and accepted on November 17, 2008

Advance Access publication November 19, 2008

Associate Editor: Martin Bishop

ABSTRACT

Motivation: Subtree prune and regraft (SPR) is one kind of tree rearrangements that has seen applications in solving several computational biology problems. The *minimum* number of rooted SPR ($_r$SPR) operations needed to transform one rooted binary tree to another is called the $_r$SPR distance between the two trees. Computing the $_r$SPR distance has been actively studied in recent years. Currently, there is a lack of practical software tools for computing the $_r$SPR distance for relatively large trees with large $_r$SPR distance.

Results: In this article, we present a simple and practical method that computes the *exact* $_r$SPR distance with integer linear programming. By applying this new method on several simulated and real biological datasets, we show that our new method outperforms existing software tools in term of accuracy and efficiency. Our experimental results indicate that our method can compute the exact $_r$SPR distance for many large trees with large $_r$SPR distance.

Availability: A software tool, SPRDist, is available for download from the web page: http://www.engr.uconn.edu/~ywu.

Contact: ywu@engr.uconn.edu

1 INTRODUCTION

The evolutionary history of multiple species is commonly represented as a leaf-labeled tree, which is called a phylogenetic tree. Often, a phylogenetic tree designates an ancestral species as the root of the tree. Although multifurcating trees are sometimes studied, phylogenetic trees are often (but not always) assumed to be binary. Thus, throughout this article, we restrict our attention to *rooted binary* leaf-labeled trees. See Figure 1 for an example of phylogenetic trees.

An important problem in phylogenetics is to study rearrangements of phylogenetic trees. A tree rearrangement operation modifies the tree topology by applying structural changes. There are several popular tree rearrangements. See, for example Felsenstein (2004), for an overview on various kinds of tree rearrangements. Here, we focus on one kind of rearrangement: subtree prune and regraft (SPR). An SPR operation removes a branch from the tree (and thus detaches the subtree under this branch); then the subtree is re-inserted into the remaining part of tree by attaching it to a branch within the remaining tree. See Figure 1 for an illustration of SPR operation. Since we focus on rooted trees, we denote the SPR operation on rooted tree as $_r$SPR operation. Note that the subtree's root is where the re-attachment occurs (i.e. the orientation of the subtree remains the same). Also note that the pruned subtree can be re-inserted as the sibling of the root of the remaining tree, and the tree is then re-rooted. There is a slight technical difficulty here: previously we say that the subtree should be attached to an existing branch and there is no such branch when the re-rooting occurs. To get around, we can imagine there is a dummy branch sticking out of the root, which can serve as the branch that the pruned subtree can be regrafted to.

In this article, we are focused on the $_r$SPR distance problem. Imagine that a rooted binary tree undergoes an unknown number of $_r$SPR operations. Further suppose both the resulting tree and the original tree are known. Then, a natural question is how many $_r$SPR operations are needed to transform the original tree to the resulting tree. That is, we ask what is the *minimum* number of $_r$SPR operations that are needed to transform the original tree to the resulting tree.

1.1 The $_r$SPR distance problem

The $_r$SPR distance problem is given two rooted binary trees T and T', compute the minimum number of $_r$SPR operations needed to transform T to T'.

The $_r$SPR distance is taken as an intuitive measure of topological and of evolutionary similarity between two rooted binary trees. Note that the $_r$SPR distance may not capture all biological phenomena. One important aspect that is missing from the $_r$SPR distance formulation and often brought up in literature is the timing constraint. Without further constraints, the $_r$SPR distance may not be biologically feasible in some biological contexts (such as hybridization). This is because time contradiction and 'cycles' in $_r$SPR operations may occur when a series of unconstrained $_r$SPR operations are performed. See, e.g. Bordewich *et al.* (2007) for related work on the timing constraint issue. Also see some interesting discussion on the timing violation in lateral gene transfer (LGT) in MacLeod *et al.* (2005). In this article, we do not consider timing constraint in computing the $_r$SPR distance (unless otherwise stated).

Computation of the $_r$SPR distance is motivated by several important biological problems. One example is LGT. A commonly asked question on LGT is: given two rooted binary trees, what is the minimum number of LGT events needed to transform one tree to the other tree. LGT operations can be simulated by $_r$SPR operation, so this is very closely related to the $_r$SPR distance problem. See, e.g. Beiko and Hamilton (2006) for more elaboration on the relationship between $_r$SPR and LGT. The $_r$SPR distance problem may also be useful in studying recombination for population sequences (Hein, 1990, 1993; Song and Hein, 2005).

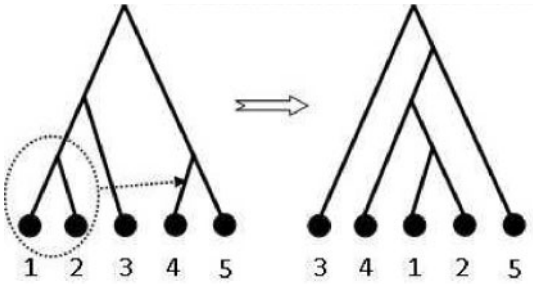

Fig. 1. An illustration of $_r$SPR operation on a tree. By pruning the subtree containing leaves 1 and 2, and regrafting to the edge out of leaf 4, we transform the left tree into the right tree.

The $_r$SPR distance problem has been actively studied in recent years. It is known that this problem is NP-hard (Bordewich and Semple, 2004; Hein *et al*., 1996) and several approximation algorithms have been developed (Bonet *et al*., 2006; Bordewich *et al*., 2008; Hein *et al*., 1996; Rodrigues *et al*., 2001). The current best polynomial-time approximation algorithm has an approximation ratio of *three* (Bordewich *et al*., 2008). Moreover, Bordewich and Semple (2004) developed fixed parameter tractable (FPT) algorithms (Bordewich *et al*., 2008). They showed that the $_r$SPR distance can be computed with $O(4^{2k}k^4+n^4)$ time, where k is the SPR distance and n is the number of leaves (i.e. taxa) in the trees. Although theoretically interesting, the performance of the FPT algorithm degrades when k increases. This algorithm seems to be impractical for $k=10$ and $n=100$, for example. Practical applicability of the FPT algorithm is still not well evaluated. Recently, Bordewich *et al*. (2007) developed a practical method that computes the minimum number of hybridization events, which is closely related to the $_r$SPR distance. Their method (implemented in a program called HybridNumber) is similar to ours in nature: it always tries to find optimal solutions but may not finish its computation within a reasonable amount of time. However, their method and our method here still compute different values. In fact, the $_r$SPR distance is sometimes equal to, but in general is a lower bound on the minimum number of hybridization events. This is because hybridization imposes more timing constraints on the $_r$SPR operations. In Section 3, we show that our method is much more efficient than the method implemented by program HybridNumber.

To summarize, currently there is no practical software tool that can compute the exact $_r$SPR distance for two relatively large rooted binary trees when the $_r$SPR distance is also relatively large. So it is desirable to find better ways to compute the $_r$SPR distance. As far as we know, all the existing methods for computing the $_r$SPR distance are heuristics. We do not know a previous implementation that can compute this distance exactly. In this article, we develop a practical method to compute the *exact* value of the $_r$SPR distance. Our experiments with both simulated and real biological data suggest that our method can handle many data of current interest (Section 3). In particular, we present a simple integer linear programming (ILP) formulation that solves the $_r$SPR distance problem for many datasets. We develop a software tool (called *SPRDist*) that only uses a publicly available ILP solver. We also show that our method outperforms several existing methods in terms of accuracy and efficiency in computing the $_r$SPR distance. The high level idea behind our approach is similar to that in Bordewich *et al*. (2008). A main contribution of this work is the development of a software tool that makes the computation of the exact $_r$SPR distance practical for many datasets.

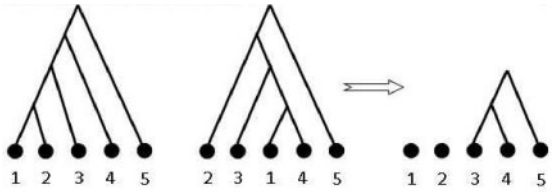

Fig. 2. An illustration of agreement forest of two trees. The three trees on the right comprise an agreement forest. It is easy to see that it is also a maximum agreement forest.

1.2 Background

Some definitions provided in this section are based on Bordewich *et al*. (2008). A rooted phylogenetic tree (or simply phylogeny) T is a binary rooted leaf-labeled tree. The set of the leaf labels is denoted as $L(T)$. The set of branches of T is denoted as $E(T)$ and the set of vertices as $V(T)$. Given a tree T, we can create a forest of trees $F(T)=\{T_1,T_2,\ldots,T_f\}$ from T by deleting a subset of $E(T)$. Note that the union of all $L(T_i)$ is equal to $L(T)$. In other words, the forest $F(T)$ induces a partition of $L(T)$. Also, any two trees T_i and T_j are node disjoint. Conversely, we say a list of trees T_i form a forest for T if (i) for any tree T_i, $L(T_i)\subseteq L(T)$; (ii) for each T_i, the (unique) *minimal subtree* tree connecting the nodes in $L(T_i)$, denoted $S(T_i)$, is identical to T_i when nodes with degree two of $S(T_i)$ are contracted; if this holds, we say T_i *appears* in T; and (iii) for any two trees T_i and T_j, $S(T_i)$ and $S(T_j)$ are node disjoint.

The concept of agreement forest has been used in many previous papers (Bonet *et al*., 2006; Bordewich *et al*., 2008; Hein *et al*., 1996; Rodrigues *et al*., 2001) and is also crucial to the current work. Let T and T' be two rooted phylogeny with the same set of leaf labels. An agreement forest $F(T,T')$ is a set of trees $T_1,T_2,\ldots,T_f$ with the following properties. First, $L(T_1),\ldots,L(T_f)$ are pairwise disjoint and form a *partition* of $L(T)$. Second, each T_i must appear in both T and T'. Third, for any T_i and T_j, $S(T_i)$ and $S(T_j)$ are node disjoint in T and in T′. Intuitively, an agreement forest gives a set of trees which can be derived by cutting the same number of branches of T and T'. See Figure 2 for an illustration of an agreement forest. It is easy to see that there always exists an agreement forest for any two rooted phylogeny with the same set of leaves: one trivial agreement forest consists of n trees, each with just one of the n leaves. But clearly, there are many agreement forests. We are interested in the agreement forest with the *smallest* number of trees (called maximum agreement forest or MAF). It is easy to see that the agreement forest in Figure 2 is also a MAF. The MAF problem is to compute the number of trees (and find these trees) in a MAF.

One of the most important observations on the $_r$SPR distance problem is its connection with the MAF problem, as demonstrated in Lemma 1.1 (Bordewich and Semple, 2004; Hein *et al*., 1996). Note that there is a subtle issue with this result. It was shown in Bordewich and Semple (2004) that the argument in Hein *et al*. (1996) has a flaw, but could be fixed by a simple trick: add a dummy leaf (with a new leaf label) out of the root of each of the

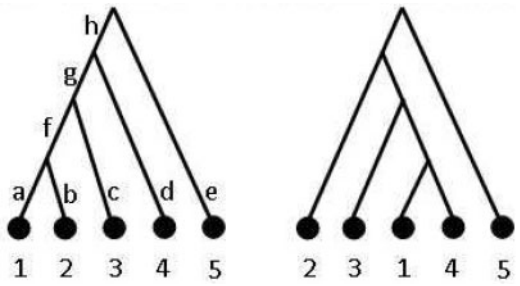

Fig. 3. An illustration of triples of leaves of two trees T (left) and T' (right). For leaves 1, 2 and 3, $TriPath(\{1,2,3\})=\{a,b,c,f\}$. Note triple {1,2,3} is topologically incompatible in T and T'. For leaves 1 and 3, $Path(1,3)=\{a,c,f\}$.

two trees. Without this dummy leaf, Lemma 1.1 may not hold (in fact, could be off by one). The dummy leaf can be treated as a new taxa, and does not cause further complexity in the following algorithmic development. Therefore, in the following, we assume that the two trees have been preprocessed to add the new leaf in the two trees. For example, in Figure 3, the leaf 5 may be considered the new leaf added by preprocessing, in addition to the original leaves 1,2,3 and 4.

LEMMA 1.1. *After adding a dummy leaf out of the root of each tree, the $_r$SPR distance of the two original trees is equal to the number of trees in a MAF (of the two resulting trees) minus one.*

2 APPROACH

Like most of the previous work on the $_r$SPR distance problem, our method is based on Lemma 1.1. That is, we will find the MAF in order to compute the $_r$SPR distance. A main reason why the MAF formulation is useful to solve the $_r$SPR distance problem is due to the static nature of the MAF definition. Without the MAF formulation, naively one has to try all possible sequences of $_r$SPR operations to calculate the $_r$SPR distance, which looks quite challenging. The static MAF formulation is especially important when one wants to apply existing combinatorial optimization tool such as ILP.

Now we describe how to cast the MAF problem to a simple optimization problem, which we will solve using ILP. Our approach uses the similar high-level idea as Bordewich *et al.* (2008). Obviously, we need to place the smallest number of *branch cuts* in the two trees so that the two resulting forests contain the same set of trees. Suppose we cut branches of T and get a set of trees T_i. The *key* properties are: each resulting T_i also appears in T', and for any two T_i and T_j in $F(T)$, the subtrees $S(T_i)$ and $S(T_j)$ are node disjoint in T'. We now address these two requirements separately. First, to ensure the resulting tree T_i appears in T', we rely on triples of leaves in T and T'. A triple of leaves i,j and k is a rooted minimal subtree with only i,j and k as leaves. For example, in Figure 2, we consider triples for leaves 1, 2 and 3 in T and T'. The triple in T is topologically different from that in T': in T leaves 1 and 2 are siblings while in T' leaves 1 and 3 are siblings. In this case, we say the triple of 1, 2 and 3 is topologically *incompatible* in T and T'. On the other hand, the triple of 3, 4 and 5 appears in both T and T'. We say triple of 3, 4 and 5 is topologically equivalent in T and T'. The following well-known lemma (Semple and Steel, 2003) tells how one can use triples of leaves to test whether two trees are equivalent.[1]

LEMMA 2.1. *Two rooted binary trees are equivalent iff every triple of leaves in the two trees is topologically equivalent.*

Intuitively, we choose the same number of branches in T and T' to cut and get an agreement forest $F(T,T')$. The branch cuts must be under some *constraints* so as to ensure that each resulting tree $T_i \in F(T,T')$ must appear also in both T and T'. Further, although the problem involves choosing branches to cut from both T and T', we can actually focus on branch cuts in T and use T' as a *reference*, as we will explain next. From this point of view, the problem is to choose the minimum number of edges to cut in T, creating forest $F(T)$, so that $F(T)$ is also a forest for T'. Lemma 2.1 implies that we need to place enough branch cuts in T so that at least one branch of each topologically incompatible triple is cut. Otherwise, this triple will appear in $F(T)$ and since $F(T)$ is also a forest of T', the triple also appears in T'. But this contradicts the assumption that the triple is incompatible in T and T'. Now, for a triple i, j and k, we denote the set of branches in T that connect leaves i, j and k as $TriPath(\{i,j,k\})$. Therefore, at least one of the branches in $TriPath(t)$ is cut for each topologically incompatible triple t. We call this type of constraints on branch cut *type one*. See Figure 3 for an illustration of triples. Once type one constraints are satisfied, any trees with three or more leaves in the resulting forest formed by cuts in T must also appear in T' due to Lemma 2.1.

However, type one constraints by themselves do not guarantee the creation of an agreement forest. For example, in Figure 3, if edges e and f are cut, the type one constraints are satisfied but the resulting forest is not an agreement forest. This is because the tree containing leaves 1 and 2 and the tree containing leaves 3 and 4 intersect in T'. To remove any such intersection, we need to impose additional constraints for all *pairs* of leaves i and j so that if i and j are not separated by branch cuts in T, then i and j are not separated in T'.

To impose this additional property, we consider leaves i,j,k and l in T and T'. For two leaves i and j, we denote the edges in T connecting i and j as $Path(i,j)$. See Figure 3 for an illustration of paths. Suppose the path between i and j intersects with the path between k and l in T'. Then if $Path(i,j)\cap Path(k,l)=\emptyset$ (i.e. the path between i and j and the path between k and l do not intersect in T), we need to ensure that there is at least one of the edges in $Path(i,j)\cup Path(k,l)$ is cut (and we call these two paths *incompatible*). We call this kind of constraints *type two*. Otherwise, we need to enforce *no* extra constraints for these four leaves. Lemma 2.2 states that a set of branch cuts in T creates an agreement forest for T and T' iff the chosen branch cuts satisfy *both* type one and type two constraints.[2]

LEMMA 2.2. *For two rooted binary trees T and T', a set of branch cuts in T decomposes T into an agreement forest for T and T' iff the branch cuts satisfy both type one and type two constraints.*

PROOF: first suppose the branch cuts decompose T into an agreement forest. Then each tree in the forest must also appear in T' and the set of corresponding trees in T' must not intersect.

[1]This lemma is also important to the method in Bordewich *et al.* (2007). See Semple and Steel (2003) for more definitions regarding to triples and rooted trees.

[2]One should note Lemma 2.2 and Lemma 3.1 in Bordewich *et al.* (2008) that are closely related.

Therefore, type one constraints are satisfied because any incompatible triple cannot appear in both T and T' and thus such incompatible triple must be cut in T. Now suppose the two paths between leaves i and j and between k and l intersect in T' but are disjoint in T, and there is no branch cut in either of the two paths in T. Then i and j belong to a same tree and k and l also belongs to a same tree in the forest (formed from T). But since path between i and j intersects the path between k and l in T', due to the property of agreement forest, i,j and k,l belong to the same tree in the forest. But this violates the triple condition, because there exists a triple of some three leaves out of i,j,k and l that appears only in one of T and T'. This is a contradiction. Thus, in any agreement forest, type two constraints are also satisfied. Therefore, both type one and type two constraints by branch cuts form an agreement forest.

Conversely, consider a set of branch cuts that satisfy both type one and two constraints. We will show that these cuts create an agreement forest. We first consider the forest created by cuts in T. For any tree T_f in the forest, it is clear that T_f also appears in T' as an induced subtree due to the type one constraint. So we only need to show that any two such induced trees in T' are disjoint. For contradiction, suppose T_f and $T_{f'}$ (induced by the forest of T) intersect in T'. Note that neither of T_f and $T_{f'}$ can have only one node (otherwise they cannot intersect). So there must exist a pair of leaves $i,j \in T_f$ and $k,l \in T_{f'}$ so that the path between i and j and the path between k and l intersect. Since i,j and k,l belong to different trees in forest of T, the path between i and j and the path between k and l are disjoint in T. However, type two constraints specify that there must exist branch cuts in either the path between i and j or the path between k and l. Then either i,j or k,l are not in a same tree. This contradicts our previous assumption. ■

Lemma 2.2 implies that the MAF problem can be solved by finding the *fewest* branch cuts that satisfy both type one and two constraints. Since the $_r$SPR distance problem is NP-hard, it is unlikely that there exists a polynomial-time algorithm to find such set of minimum branch cuts. To develop a practical method, we formulate the $_r$SPR distance problem as an optimization problem using ILP. Although solving ILP formulation is a well-known NP-hard problem, great efforts have been spent on developing practical software that can solve many ILP instances and there are also powerful commercial ILP solvers (such as CPLEX) available. In recent years, ILP has been used to solve problems in computational biology with many successes. For example, Gusfield (2003) used ILP to solve the parsimonious haplotyping problem. Although the size of his ILP formulation is exponential in terms of the input data size, many problem instances can be solved surprisingly fast in practice. One of the main goals of this article is to demonstrate that as a powerful and available optimization tool, ILP might be useful for more phylogenetics problems.

We have the following simple ILP formulation for computing the $_r$SPR distance between two binary trees. This formulation resembles the classic hitting set problem. Intuitively, we use the smallest number of branch cuts to *hit* the set of branches from the system of the triples and the pairs of paths. In particular, we define a binary variable C_i for each edge e_i in T, where $C_i = 1$ if edge e_i is cut. Then the ILP formulation is:

(1) Goal: Minimize $\sum_{i=1}^{m} C_i$

(2) Subject to

(3) $\sum_{e_i \in TriPath(t_j)} C_i \geq 1$, for each incompatible triple t_j.

(4) $\sum_{e_i \in Path(p_{j_1}) \cup Path(p_{j_2})} C_i \geq 1$, for each incompatible pair of paths (p_{j_1}, p_{j_2}).

(5) For each tree edge e_i in T, there is a binary variable C_i

As the example, we consider the incompatible triple with leaves 1, 2 and 3 in Figure 3. There is the following constraint in the ILP formulation: $a+b+c+f \geq 1$. Similarly, for the incompatible pair of paths (1,2) and (3,4), we have this constraint: $a+b+c+d+g \geq 1$. Here, for simplicity, we let the variables for branches carry the same name as the branches themselves.

2.1 Speedup

The above ILP formulation is very simple and small. There are in total $2n-2$ binary variables and $O(n^4)$ constraints in this ILP formulation. And it works well for relatively small trees. However, we found an equivalent ILP formulation that is faster for larger input when using GLPK solver. This formulation uses more variables and constraints, but each constraint is simpler than that in the previous formulation.

For any two nodes i and j (leaf or internal node), define *binary* variable $M_{i,j}$, which indicates whether the unique path between nodes i and j in T is cut at some branches or not. That is, $M_{i,j} = 1$ iff the path between i and j is *not* cut. It is easy to see that the type one constraint for an incompatible triple i,j and k can be then expressed as $M_{i,j} + M_{i,k} \leq 1$. Similarly, the type two constraint for an incompatible twin path (i,j) and (k,l) can be expressed as $M_{i,j} + M_{k,l} \leq 1$.

Now we describe how to create constraints for $M_{i,j}$ for two nodes i and j. First, if $i = j$, we have $M_{i,i} = 1$. Let p be the node next to i on the path between i and j, and let the branch e be the one with p and i as two nodes. Then, the following three constraints are created:

$$\begin{aligned} M_{i,j} - M_{p,j} &\leq 0 \\ M_{i,j} - M_{p,j} + C_e &\geq 0 \\ M_{i,j} + C_e &\leq 1 \end{aligned}$$

For example, consider leaves 1 and 3 in Figure 3. Let node 12 be the lowest common ancestor of leaves 1 and 2 in T. Then we have: $M_{1,3} - M_{12,3} \leq 0$, $M_{1,3} - M_{12,3} + a \geq 0$ and $M_{1,3} + a \leq 1$. For incompatible triple with leaves 1, 2 and 3, we have $M_{1,2} + M_{1,3} \leq 1$. For the incompatible pair of paths (1,2) and (3,4), we have $M_{1,2} + M_{3,4} \leq 1$. These constraints may look a little confusing, but the reasoning behind them is simple. When $M_{p,j} = 0$ (i.e. path between p and j is cut) then $M_{i,j} = 0$ no matter what happens to C_e. If $C_e = 1$ (i.e. branch e is cut), then $M_{i,j} = 0$ no matter what happens to $M_{p,j}$. If $C_e = 0$, then $M_{i,j} = M_{p,j}$.

This new ILP formulation appears to be solved faster by GLPK solver for larger trees. But for smaller tree or when CPLEX is used, the original ILP formulation is faster. In the following, our software implementation and practical results are based on this new ILP formulation. In practice, both ILP formulations are implemented and one may choose one to get better running time.

The ILP formulation is then solved by an ILP solver. Our experience (Section 3) indicates that even with a free publicly available ILP solver, the ILP formulation can outperform many existing heuristic methods in computing the $_r$SPR distance. The solution of the ILP formulation indicates a set of branches to cut

in T, which also implies the trees in a MAF. Once the ILP is solved, the $_r$SPR distance is equal to the value of the solution. This is implied by Lemma 1.1 since the number of trees in the MAF is one larger than the number of removed edges. See Semple (2007) for ideas if one wants to determine an optimal set of SPR operations that transforms T to T' based on the found MAF.

Moreover, to improve the efficiency of ILP solving, we first perform simple preprocessing for the two trees by contracting identical subtrees in both T and T'. It is easy to see that such contraction does not change the $_r$SPR distance between two trees.

3 EXPERIMENTAL RESULTS

We have implemented an ILP-based software tool called SPRDist to compute the $_r$SPR distance between two rooted binary trees. The tool is written in C++ and uses the GNU GLPK ILP solver. Our experience shows CPLEX (a commercial ILP solver) is often faster and more robust than GLPK. The main reason for using GLPK as the ILP solver is for better availability of our software tool for users without a CPLEX license.

To test the effectiveness of SPRDist, we compute the $_r$SPR distance for a number of tree pairs. The trees are from both simulated data and biological data. The experiment was performed on a 3192 MHz Intel Xeon workstation. To see how well the open source GLPK solver works, we also list the running time when commercial CPLEX solver is used. As shown below, it appears that although CPLEX solver is faster, the difference between CPLEX and GLPK for our particular problem is not very large for many datasets. On the other hand, for datasets where GLPK is too slow, it may help to run CPLEX. For example, as shown in Table 2, for the ndhF and ITS trees, CPLEX takes about 4 h to find the optimal solution while GLPK takes more than 1 day. It appears that a more severe problem for GLPK is that GLPK may abort when solving very large ILP formulations. We did experience this problem for one dataset we tried. This is shown by one test run in Section 3.1. Initially we thought this is a lack of memory issue. But now we believe it is more likely due to some implementation issues with GLPK on some large problem instances. We run GLPK-based simulations on both Windows and Linux platforms. It appears that the Linux version of SPRDist is more stable. In general, CPLEX is more robust than GLPK when solving large ILP formulations.

For comparison, we list the performance from related previous methods, including both heuristic methods and an exact method. Note that the running time is only for reference: the running time of the other methods being compared with is quoted directly from the respective literature.

3.1 Simulated data

The simulated data are from Beiko and Hamilton (2006) when they developed a software tool called EEEP. Program EEEP is designed to find the parsimonious explanation of tree transformation with SPR operations. For datasets we tested, each tree contains 100 leaves. Beiko and Hamilton first generated a random rooted binary tree and then applied a set of 10 random SPR operations to obtain another rooted binary tree. The input contains the topology of both the original and the resulting tree. Note that the $_r$SPR distance is at most 10 for the pairs of trees tested. Beiko and Hamilton (2006) then tested several different methods (including some of their own) for finding plausible tree rearrangements using $_r$SPR. Note that none of these methods is guaranteed to find the most parsimonious SPR operations. Since our purpose here is to test the efficiency of SPRDist, we only pick the 10 datasets with the largest size (100 leaves) and the most random SPR operations performed (10). For these chosen datasets, only program Lattrans (Hallett and Lagergren, 2001) gives solutions to some of the datasets. Several other methods tested in Beiko and Hamilton (2006), including EEEP, could not finish within 5 h. Note that one of the reasons that program EEEP is slower than some programs mentioned here is that EEEP is designed to handle more complex situations, such as multifurcating trees and timing constraints.

Table 1. Performance of program SPRDist on 10 simulated data from Beiko and Hamilton (2006)

	SPRDist			Lattrans	
Data	$_r$SPR dist.	**GLPK**	CPLEX (s)	Best $_r$SPR	Time (s)
1	10	1.4 h	313	**13**	520
2	10	–	17	10	67
3	9	11 s	314	9	36
4	9	992 s	4	9	36
5	10	29 s	18	10	68
6	9	7 s	5	–	–
7	10	43 s	15	–	–
8	10	68 s	22	**13**	648
9	10	618 s	135	10	70
10	10	37 s	10	10	67

For comparison, we also list the results from program Lattrans (Hallett and Lagergren, 2001). Note that 10 random SPR operations are performed when the datasets are generated. When no results are found (either because the program crashes or stop without finding solutions), we use symbol '-'. For all the 10 data, program EEEP does not give results. Time is measured in seconds (s). GLPK is highlighted in bold as this is likely the version that is to be used in practice. Results of Lattrans for datasets 1 and 8 are highlighted in bold as Lattrans significantly over-estimated the SPR distance for these two data.

From Table 1, it can be found that our program SPRDist outperforms in terms of both accuracy and efficiency. Here, we use 10 (the original number of $_r$SPR operations performed in the *true* history) as a rough guide on accuracy in inferring tree rearrangements. Recall that this means $_r$SPR distance is at most 10, but could be smaller than 10 if the true history is not the most parsimonious. Program Lattrans significantly over estimates the number of $_r$SPR operations for datasets 1 and 8. Also, both SPRDist and Lattrans can under estimate the true number of $_r$SPR operations (e.g. for datasets 3 and 4). These results show that the $_r$SPR distance matches the true number of rearrangements 6 out of ten datasets. Thus, for relatively large trees and relative small number of tree rearrangements (in this case trees with 100 leaves and 10 true rearrangements), the $_r$SPR distance appears to give quite good estimates on the number of true rearrangement operations.

Moreover, program SPRDist is surprisingly efficient for most of the 10 datasets. Except for dataset 2, SPRDist finds the exact $_r$SPR distance, and with running time <1 min for 5 out of 10 datasets. However, the running time of SPRDist has large variance: for dataset 1, it takes >1 h while for dataset 5, it only takes 29 s. Note that this running time is achieved with GLPK, a relative poor ILP solver. Our experience indicates that when commercial ILP solver such as

Table 2. Performance of program SPRDist on 15 pairs of trees for the Poaceae data

Pair	Data			SPRDist			Bordewich *et al.*	
	1	2	# taxa	$_r$SPR	**Time (GLPK)**	Time (CPLEX)	hybridize	Time
1	ndhF	phyB	40	12	75 s	26 s	14	11 h
2	ndhF	rbcL	36	10	20 s	7 s	13	11.8 h
3	ndhF	rpoC2	34	11	53 s	11 s	12	26.3 h
4	ndhF	waxy	19	7	8 s	2 s	9	320 s
5	ndhF	ITS	46	**19**	**34.7 h**	**4.1 h**	≥15	2 d
6	phyB	rbcL	21	4	10 s	1 s	4	1 s
7	phyB	rpoC2	21	6	3 s	3 s	7	180 s
8	phyB	waxy	14	3	3 s	1 s	3	1 s
9	phyB	ITS	30	8	21 s	7 s	8	19 s
10	rbcL	rpoC2	26	11	66 s	19 s	13	29.5 h
11	rbcL	waxy	12	6	3 s	1 s	7	230 s
12	rbcL	ITS	29	**13**	211 s	86 s	≥9	2 d
13	rpoC2	waxy	10	1	1 s	1 s	1	1 s
14	rpoC2	ITS	31	**14**	626 s	277 s	≥10	2 d
15	waxy	ITS	15	7	14 s	3 s	8	620 s

For comparison, we also list the results from Bordewich *et al.* (2007). Note that the minimum hybridization number is not exactly the same as the $_r$SPR distance. But these two values are related. $_r$SPR: the $_r$SPR distance; hybridize: the minimum hybridization number. Time is measured in seconds (s), hours (h) and days (d). Results from three datasets (5,12,14) are highlighted in bold because these are the larger examples and program HybridNumber fails to compute the exact hybridization number while our new method finds the exact $_r$SPR distance.

CPLEX is used, the range of applicability of our methodologies can be enhanced. For example, GLPK solver aborts for dataset 2 in Table 1. The CPLEX solver, on the other hand, solves this dataset in just 17 s.

Overall, program SPRDist outperforms program Lattrans in terms of both accuracy and efficiency. The main advantage of our method is that it always gives optimal solutions, which is unlike the heuristic approaches such as program Lattrans.

3.2 Biological data

To demonstrate that our method works for true biological data, we also test our method on the following biological data. Here, we apply our method to tree pairs for a Poaceae dataset. The dataset is originally from the Grass Phylogeny Working Group (Grass PWG, 2001). The dataset contains sequences for six loci: internal transcribed spacer of ribosomal DNA (ITS); NADH dehydrogenase, subunit F (ndhF); phytochrome B (phyB); ribulose 1,5-biphosphate carboxylase/oxygenase, large subunit (rbcL); RNA polymerase II, subunit β'' (rpoC2); and granule bound starch synthase I (waxy). The Poaceae dataset was previously analyzed by Schmidt (2003), who generated the inferred rooted binary trees for these loci. Bordewich *et al.* (2007) computed the minimum hybridization number for each of the 15 pair of trees. Note that when a pair of trees is processed, only shared taxa is kept. To test how well our method performs on these biological trees, we compute the exact $_r$SPR distance for the same 15 pairs of trees. In this section, we focus on exact methods. The only other available software tool we know of that computes exact values similar to (but not the same as) the $_r$SPR distance is program HybridNumber (Bordewich *et al.*, 2007). In the following, we test the scalability and efficiency of our method, and compare with program HybridNumber. See Table 2 for the results.

The experimental results in Table 2 clearly indicate that our program SPRDist is more efficient than program HybridNumber by Bordewich *et al.* (2007). In fact, except for one pair of tree (ndhF and ITS), our program only takes seconds or minutes, while program HybridNumber sometimes takes hours or even fails to give optimal results after 2 days (tree pairs 5, 12 and 14). So, our program SPRDist is more scalable than program HybridNumber. Computing the $_r$SPR distance between two large and topologically far apart trees is still challenging, but feasible. As an example, for ndhF and ITS trees, the GLPK solver runs for >1 day and CPLEX solver runs for 4 h to find the optimal solution for this dataset. One should note that program HybridNumber computes the minimum hybridization number which, although related, is not exactly the same as the $_r$SPR distance. Thus, we are not suggesting here our program outperforms program HybridNumber. Again, the point is to compare how scalable exact methods can be on computing some similar quantity.

An additional feature of program SPRDist is that it checks whether the found MAF obeys the additional timing constraints imposed by the hybridization problem. That is, if the found MAF satisfies the timing constraints, then program SPRDist also solves the minimum hybridization problem. Of course, this does not always happen. There are data as shown in Baroni *et al.* (2005) where the $_r$SPR distance is different from the minimum hybridization number. We also note that the $_r$SPR distance gives a lower bound on the minimum hybridization number. In Table 2, for two sets of trees rbcL/ITS and rpoC2/ITS, the $_r$SPR distances computed by program SPRDist give significantly higher estimates on the minimum hybridization number than those given by program HybridNumber. Thus, our method may also be of interests for hybridization problems.

4 DISCUSSION

We have presented a simple and effective method to compute the exact $_r$SPR distance between two rooted binary trees. As far as we know, this is the first practical method that can compute the exact $_r$SPR distances for relatively large trees with large $_r$SPR distance.

The ILP formulation can be fairly efficiently solved by the open source GLPK solver for many datasets. Only for a small number of datasets we tried CPLEX is needed to find the optimal solution within a reasonable amount of time. This has practical importance because most users will not have CPLEX licenses. Our new method may be useful for several computational problems in biology. Note that if one or both input trees are un-rooted, one can first try all possible rooting for the un-rooted trees, and then apply our method on the rooted trees.

Comparing with previous work (Bonet *et al*., 2006; Bordewich *et al*., 2008; Hein *et al*., 1996; Rodrigues *et al*., 2001) on the $_r$SPR distance problem, our method is much simpler. We take advantages of the power of ILP. We believe that ILP could also help for other phylogenetics problems.

One of the major remaining issues is to further improve the performance of our method. A main issue is to reduce the size of ILP formulation used by the ILP solver and speedup the solving process. In future work, we will exploit more structural properties in the $_r$SPR distance problem, as shown by Bordewich *et al*. (2008). These structural properties, when combined with the ILP formulation, may greatly reduce the size of the ILP formulation and thus make the solution more efficient.

ACKNOWLEDGEMENTS

I thank Yun S. Song, Frederick Matsen and Dan Gusfield for useful discussions and Simone Linz for sharing the grass trees. I also thank Frederick Matsen, Simone Linz and three anonymous referees for reading and helping to improve the manuscript.

Funding: National Science Foundation (IIS-0803440) and University of Connecticut; National Science Foundation (partial CCF-0515278, IIS-0513910).

Conflict of Interest: none declared.

REFERENCES

Baroni,M. *et al*. (2005) Bounding the number of hybridisation events for a consistent evolutionary history. *J. Math. Biol.*, **51**, 171–182.

Beiko,R.G. and Hamilton,N. (2006) Phylogenetic identification of lateral genetic transfer events. *BMC Evol. Biol.*, **6**, 159–169.

Bonet,M.L. *et al*. (2006) Approximating subtree distances between phylogenies. *J. Comp. Bio.*, **13**, 1419–1434.

Bordewich,M. and Semple,C. (2004) On the computational complexity of the rooted subtree prune and regraft distance. *Ann. Combinatorics*, **8**, 409–423.

Bordewich,M. *et al*. (2007) A reduction algorithm for computing the hybridization number of two trees. *Evol. Bioinform.*, **3**, 86–98.

Bordewich,M. *et al*. (2008) A 3-approximation algorithm for the subtree distance between phylogenies. *J. Discrete Algorithm*, **6**, 458–471.

Felsenstein,J. (2004) *Inferring Phylogenies*, Sinauer, Sunderland, MA.

Grass Phylogeny Working Group (2001) Phylogeny and subfamilial classification of the grasses (poaceae). *Ann. Mo. Bot. Gard.*, **88**, 373–457.

Gusfield,D. (2003) Haplotype inference by pure parsimony. In Baeza-Yates,R. *et al*. (eds), *14th Annual Symposium on Combinatorial Pattern Matching (CPM'03), LNCS, vol. 2676*. Springer, pp. 144–155.

Hallett,M. and Lagergren,J. (2001) Efficient algorithms for lateral gene transfer problems. In *Proceedings of fifth annual conference on Research in Computational Molecular Biology* (RECOMB 2001), pp. 149–156.

Hein,J. (1990) Reconstructing evolution of sequences subject to recombination using parsimony. *Math. Biosci.*, **98**, 185–200.

Hein,J. (1993) A heuristic method to reconstruct the history of sequences subject to recombination. *J. Mol. Evol.*, **36**, 396–405.

Hein,J. *et al*. (1996) On the complexity of comparing evolutionary trees. *Discrete Appl. Math.*, **71**, 153–169.

MacLeod,D. *et al*. (2005) Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement. *BMC Evol. Biol.*, **5**, 27.

Rodrigues,E.M. *et al*. (2001) Some approximation results for the maximum agreement forest problem. In *LNCS 2129*. Springer, London, pp. 159–169.

Schmidt,H.A. (2003) Phylogenetic trees from large datasets. PhD thesis, Heinrich-Heine-Universitat, Dusseldorf.

Semple,C. (2007) Hybridization networks. In Gascuel,O. and Steel,M. (eds.), Reconstructing Evolution: New Mathematical and Computational Advances. Oxford University Press, New York, pp. 277–314.

Semple,C. and Steel,M. (2003) *Phylogenetics*. Oxford University Press, New York.

Song,Y.S. and Hein,J. (2005) Constructing minimal ancestral recombination graphs. *J. Comp. Bio.*, **12**, 159–178.