Clustering the Space of Maximum Parsimony Reconciliations in the Duplication-Transfer-Loss Model*

A. Ozdemir[1], M. Sheely[1], D. Bork[2], R. Cheng[2], R. Hulett[3], J. Sung[1], J. Wang[1], and R. Libeskind-Hadas[1]

[1] Department of Computer Science, Harvey Mudd College, 301 Platt Blvd., Claremont, California, 91711 USA {aozdemir,msheely,jsung,jwang,hadas}@g.hmc.edu
[2] Department of Computer Science, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, 15213 USA {dbork,rcheng}@andrew.cmu.edu
[3] Department of Computer Science, Stanford University, 450 Serra Mall, Stanford, California, 94305 USA reyna.hulett@cs.stanford.edu

Abstract. Phylogenetic tree reconciliation is widely used in the fields of molecular evolution, cophylogenetics, parasitology, and biogeography for studying the evolutionary histories of pairs of entities. Reconciliation is often performed using maximum parsimony under the DTL (Duplication-Transfer-Loss) event model. Since the number of maximum parsimony reconciliations (MPRs) can be exponential in the sizes of the trees, an important problem is that of finding a small number of representative reconciliations. We give a polynomial time algorithm that can be used to find the cluster representatives of the space of MPRs with respect to a number of different clustering algorithms and specified number of clusters.

Keywords: Tree reconciliation, Duplication-Transfer-Loss model, clustering

1 Introduction

Phylogenetic tree reconciliation is an important technique for studying the evolutionary histories of pairs of entities such as gene families and species, parasites and their hosts, and species and their geographical habitats. The reconciliation problem takes as input two trees and the associations between their leaves and seeks to find a mapping between the trees that accounts for their topological incongruence with respect to a given set of biological events. In the widely-used DTL model the four event types are *speciation*, *duplication*, *transfer*, and *loss* [1–6, 16]. We denote the two trees as the *species tree* (S) and the *gene tree* (G), although these trees could be host and species trees or area cladograms

* This work was funded by the U.S. National Science Foundation under Grant Numbers IIS-1419739 and 1433220.

and species trees in the contexts of cophylogenetic and biogeographical studies, respectively.

Reconciliation in the DTL model is typically performed using a maximum parsimony formulation, where each event type has an assigned cost and the objective is to find a reconciliation of minimum total cost, called a *maximum parsimony reconciliation* or *MPR*. Efficient algorithms are known for finding MPRs in the DTL model [1, 15].

In general, the number of MPRs can grow exponentially with the sizes of the species and gene trees [13]. Consequently, a number of efforts have been made to summarize the vast space of MPRs. Nguyen *et al.* [10] showed that choosing a single random MPR can lead to inaccurate inferences and gave an efficient algorithm to compute a median MPR. Median MPRs were subsequently used to summarize reconciliation space in [14]. Bansal *et al.* [2] showed how a sample of MPRs can be selected uniformly at random. Ma *et al.* [9] examined the problem of finding a set of k reconciliations that collectively cover the most frequently occurring events in MPR space, for a given number k.

We study the problem of clustering the space of MPRs, both to represent the space by a small number of cluster representatives and to gain insights into the structure of the space. We first define the *reconciliation count function* that can be used to implement a number of different clustering algorithms for MPR space. We then show that the reconciliation count function can be computed exactly in polynomial time (where the degree of the polynomial depends on the number of clusters), in spite of the fact that the number of MPRs can be exponential in the size of the given trees. Our results leverage the seminal work of Scornavacca *et al.* [12] and Nguyen *et al.* [10].

We demonstrate the utility of the reconciliation count function by showing how it can be used to implement two clustering algorithms for *k-medoids* and *k-centers*. The *k-medoids* problem seeks to find a representative set of k MPRs, called *medoids*, such that the sum of the distances between each MPR and its nearest medoid is minimized. Similarly, *k-centers* seeks to identify a representative set of k MPRs, known as *centers*, such that the maximum distance between each MPR and its nearest center is minimized. These are just two examples of the many clustering algorithms that can be implemented using reconciliation counts.

This work provides new tools for exploring the structure and diversity of MPR space that may not be gleaned from a single median reconciliation or a set of randomly sampled reconciliations. Thus, the results presented here are potentially useful to practitioners who wish to better understand the space of optimal reconciliations for specific data sets as well as for researchers seeking to gain better insights into MPR space in general.

In summary, in this paper:

1. We define the *reconciliation count function* and give a polynomial-time algorithm for computing it.
2. We demonstrate the utility of the reconciliation count function by showing how it can be used to solve the k-medoids and k-centers problems for MPR space and to compute statistics on the clusterings.

3. In the Supplementary Materials, we give experimental results using the Tree of Life data set [5], comparing the k-medoids and k-centers representatives to randomly selected ones. (See `www.cs.hmc.edu/~hadas/supplement.pdf`.)
4. We provide an implementation of our algorithms. (See `www.cs.hmc.edu/~hadas/clusters.zip`).

2 Definitions

In this section we give definitions and notation used throughout this paper. In the interest of brevity, we provide the minimum background required to develop our results in the subsequent sections. For completeness, formal definitions are given in the Supplementary Materials.

2.1 Maximum Parsimony Reconciliations

An instance of the DTL maximum parsimony reconciliation problem comprises a gene tree G, a species tree S, a leaf mapping Le from the leaves of G to the leaves of S (which need not be one-to-one nor onto), and positive costs for duplication, transfer, and loss events. We assume that the trees are undated in the sense that no information is given about the relative times of speciation events in either the gene or species trees. A reconciliation is a mapping $\mathcal{M}$ of the vertices of G into the vertices of S that is consistent with the leaf mapping Le and, for each internal gene vertex g with children g' and g'', neither $\mathcal{M}(g')$ nor $\mathcal{M}(g'')$ are ancestors of $\mathcal{M}(g)$ and at least one of $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ is either equal to, or a descendant of, $\mathcal{M}(g)$.

The mapping $\mathcal{M}$ induces speciation, duplication, transfer, and loss events. While the formal definitions of these events are given in the Supplementary Materials, the following suffices for our treatment: Let g be an internal vertex in G with children g' and g''. Vertex g is a speciation vertex if $\mathcal{M}(g)$ is the most recent common ancestor of $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ and $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ are neither equal to one another nor ancestrally related. Vertex g is a duplication vertex if $\mathcal{M}(g)$ is the most recent common ancestor of $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ but $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ are either equal or ancestrally related. A duplication can be viewed as a mapping of gene vertex g onto the edge from the parent of $\mathcal{M}(g)$ to $\mathcal{M}(g)$. Vertex g is a transfer vertex if exactly one of $\mathcal{M}(g')$ or $\mathcal{M}(g'')$ is a descendant of $\mathcal{M}(g)$ and the other is not in the subtree rooted at $\mathcal{M}(g)$. A loss event arises for each internal vertex on the path in S from $\mathcal{M}(g)$ to $\mathcal{M}(g')$, for each parent-child pair (g, g').[1]

The objective of the DTL maximum parsimony reconciliation problem is to find a reconciliation that minimizes the sum of the number of duplication, transfer, and loss events weighted by their respective event costs. We henceforth refer to maximum parsimony reconciliations as *MPRs*. A number of similar

[1] This characterization slightly simplifies the way that losses are actually counted and omits details about losses arising from transfer events. While this suffices for presenting our work, full details are given in the Supplementary Materials.

dynamic programming algorithms have been given for finding MPRs in time $O(|G||S|)$ [1, 16]. Figure 1(a) shows a species tree, gene tree, and a leaf mapping and (b) and (c) show two different MPRs.

2.2 Reconciliation Graphs

Scornavacca *et al.* [12] developed a data structure called a *reconciliation graph* for compactly representing the space of all MPRs for dated trees. Ma *et al.* [9] adopted reconciliation graphs for undated trees. For the purposes of the remainder of this paper, the following characterization suffices (with full details in the Supplementary Materials):

The reconciliation graph for an instance of the DTL MPR problem (comprising trees G, S, leaf mapping Le and given DTL event costs) is a directed acyclic graph (DAG) that consists of *mapping vertices* and *event vertices* and directed edges between these two vertex types. Specifically, the graph contains a *mapping vertex* for each (g, s) pair such that $\mathcal{M}(g) = s$ for some MPR $\mathcal{M}$ and an *event vertex* for each event in which $\mathcal{M}(g) = s$, with a directed edge from mapping vertex (g, s) to each such event vertex. (Note that a mapping vertex (g, s) may have edges to multiple event vertices since, for example, g may be mapped to s as a speciation event in one MPR and as a transfer event in a different MPR.) Let g' and g'' denote the children of g. If some MPR, $\mathcal{M}$, contains an event in which $\mathcal{M}(g) = s$, $\mathcal{M}(g') = s'$, $\mathcal{M}(g'') = s''$ then that event vertex for (g, s) has a directed edge to mapping vertices (g', s') and (g'', s''). Thus, each speciation, duplication, and transfer event vertex has out-degree 2. Each loss event is represented by an event vertex with out-degree 1 corresponding to a loss induced by a particular vertex in the species tree. Finally, each leaf association $(g, Le(g))$ has a corresponding event vertex which is a sink (vertex of out-degree 0) of the reconciliation graph. The reconciliation graph can be constructed in time $O(|G||S|^2)$ [9]. The right side of Figure 1 shows the reconciliation graph for the problem instance in Figure 1(a) with DTL costs 1, 4, and 1 respectively.

This brings us to the two key results that we need in the remainder of this paper. First, there is a bijection between MPRs and subgraphs of the reconciliation graph called *reconciliation trees*. A reconciliation tree begins with a mapping vertex of the form (g, s) where g is the root of the gene tree (but s is not necessarily the root of the species tree since, as shown in Figure 1(b), a reconciliation need not involve the root of S). The mapping vertex (g, s) is followed by a directed edge to any one neighbor in the reconciliation graph, which is an event vertex corresponding to an event in which g is mapped to s. Next, both neighbors of that event vertex are included; each is a mapping vertex corresponding to the mapping of a child of g. From each such mapping vertex, we again choose any single event vertex neighbor. This process (formalized in the Supplementary Materials) is repeated until the sinks of the reconciliation graph are reached. It is not difficult to show that this process yields a tree and the bijection between these reconciliations trees and MPRs is proved in [9, 12]. Henceforth, we use the terms *reconciliation trees* and MPRs interchangeably as we do for the terms

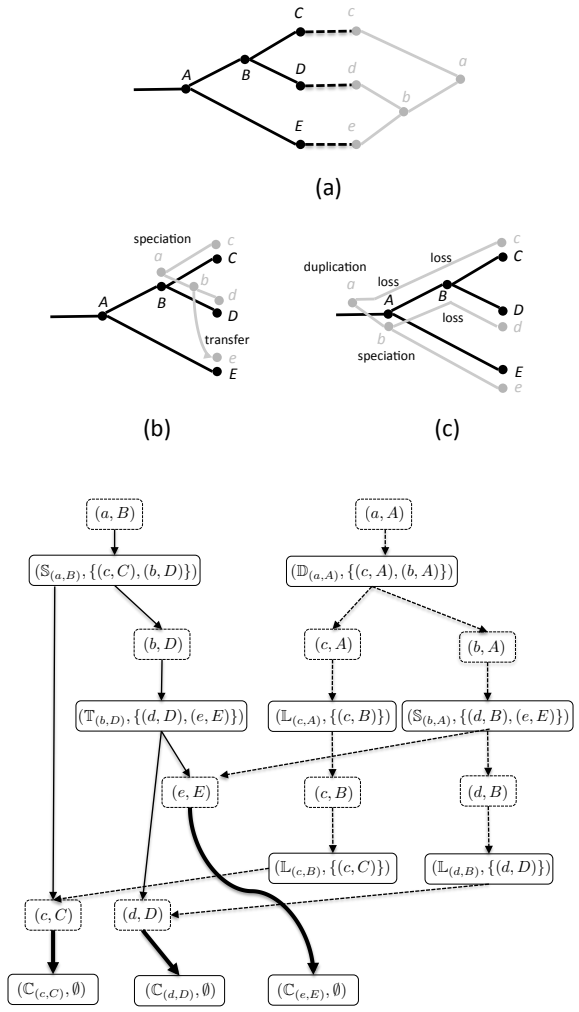

Fig. 1. Top: (a) An instance of the DTL reconciliation problem comprising species tree (black), gene tree (gray), and leaf mapping. Duplication, transfer and loss costs are 1, 4, and 1, respectively. (b) A reconciliation with one speciation and one transfer. (c) A reconciliation with one speciation, one duplication, and three losses. Both reconciliations are MPRs with total cost 4. **Bottom:** The reconciliation graph for the DTL instance in (a). Vertices with solid boundaries are event vertices and those with dashed boundaries are mapping vertices. Event vertices are designated with $\mathbb{S}$ (speciation event), $\mathbb{D}$ (duplication event), $\mathbb{T}$ (transfer event), $\mathbb{L}$ (loss event), and $\mathbb{C}$ (leaf association). The reconciliation tree indicated by solid edges corresponds to the MPR in (b) and the reconciliation tree indicated by dashed edges corresponds to the MPR in (c), with bold edges representing the shared parts of the two reconciliations.

event vertices and *events*. The right side of Figure 1 shows the two reconciliation trees corresponding to the two reconciliations in Figure 1(b) and (c).

The second major result that we need is as follows: Given a *score* function, σ, that maps vertices in the reconciliation graph to non-negative real numbers, we can find a reconciliation tree (that is, a MPR) of *maximum* total score in polynomial via a simple $O(|G||S|^2)$ time dynamic programming algorithm [9, 10]. Note that this *maximization* problem should not be confused with the problem of finding a *minimum* cost reconciliation. The vertices in the reconciliation graph *a priori* represent events and mappings in minimum cost reconciliations. The score $\sigma(v)$ can represent an arbitrary quantity that we wish to maximize over all minimum cost reconciliations.

3 Clustering Reconciliation Space

In this section we define the reconciliation count function and then show how two well-known clustering algorithms, one for medoids and one for centers, can be implemented for MPR space using this function. In Section 4 we give the algorithm for computing the reconciliation count function.

In general, our goal is to find a set of k cluster representatives for a given clustering method. We use the notation $\mathcal{T} = \{T_1, \ldots, T_k\}$ to represent such a set of k MPRs. Let R denote an MPR and let $\mathbb{E}(R)$ denote the set of events in R. For any two MPRs R_1, R_2, let the distance between R_1 and R_2, $d(R_1, R_2)$, be the size of the symmetric set difference of the event sets [10]:[2]

$$d(R_1, R_2) = |\mathbb{E}(R_1) \setminus \mathbb{E}(R_2)| + |\mathbb{E}(R_2) \setminus \mathbb{E}(R_1)|$$

For convenience, we define the distance between a reconciliation and a set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of MPRs as a k-dimensional vector that contains the distance from the reconciliation to each reconciliation in the set:

$$d(R, \mathcal{T}) = d(R, \{T_1, T_2, ..., T_k\}) = [d(R, T_1), d(R, T_2), ..., d(R, T_k)]$$

Definition 1 (Reconciliation Count Function). *Let $\mathscr{R}$ denote the set of all MPRs for a given DTL problem instance and let $\mathcal{T} = \{T_1, \ldots, T_k\}$ denote a set of k MPRs in $\mathscr{R}$. Let v denote an event vertex in the reconciliation graph. The* reconciliation count function *$Count_{\mathcal{T},v} : \mathbb{N}^k \to \mathbb{N}$ maps $\mathbf{d} \in \mathbb{N}^k$ to the number of reconciliations $R \in \mathscr{R}$ that contain event vertex v and $d(R, \mathcal{T}) = \mathbf{d}$. Let $Count_{\mathcal{T}}(\mathbf{d})$ denote the number of reconciliations $R \in \mathscr{R}$ with $d(R, \mathcal{T}) = \mathbf{d}$, regardless of whether or not they contain a particular event vertex.*

In the next two sections we demonstrate the utility of the reconciliation count function by showing how it can be used to solve the k-medoids and k-centers problems for MPR space.

[2] Other distance functions are also possible.

3.1 *k*-Medoids

Let $\mathscr{R}$ denote the set of all MPRs for a given instance of the DTL reconciliation problem. Let k be a positive integer. A set of k MPRs, $\mathcal{T} = \{T_1, T_2, ...T_k\} \subseteq \mathscr{R}$, induces a partition of $\mathscr{R}$ into clusters $C_{T_1}, C_{T_2}, ..., C_{T_k}$ such that C_{T_i} denotes the set of MPRs that are closer to T_i than to any other MPR in $\mathcal{T}$, breaking ties arbitrarily. The *k-medoids problem* seeks to find a set $\mathcal{T}$ of k MPRs that minimizes the sum of the distances between each MPR in $\mathscr{R}$ and a closest MPR in $\mathcal{T}$. More formally, the objective is to find:

$$\underset{\substack{\mathcal{T} \subset \mathscr{R} \\ |\mathcal{T}|=k}}{\arg\min} \sum_{T_i \in \mathcal{T}} \sum_{R \in C_{T_i}} d(R, T_i)$$

The elements of $\mathcal{T}$ are called *medoids*. Since the k-medoids problem is NP-hard in general [7], heuristics are used to find good, but not necessarily optimal, solutions. Park *et al.* [11] offer a simple and effective local search algorithm, with $\mathcal{T}$ initially set to k arbitrarily chosen points (MPRs in our case). In each iteration, the approximate medoid of each cluster is replaced with the MPR in that cluster that minimizes the sum of distances within that cluster. In practice, the algorithm iterates until some termination condition is reached (e.g., a maximum number of iterations). The algorithm is given below.

Algorithm 1 k-medoids heuristic [11]

1: **procedure** K-MEDOIDS($\mathscr{R}, k$)
2: $\quad\mathcal{T} \leftarrow k$ arbitrary MPRs in $\mathscr{R}$
3: $\quad$**while** some termination condition is not satisfied **do**
4: $\quad\quad$**for** $T_i \in \mathcal{T}$ **do**
5: $\quad\quad\quad T_i \leftarrow \arg\min_{M \in \mathscr{R}} \sum_{R \in C_{T_i}} d(R, M)$
6: $\quad$**return** $\mathcal{T}$

Since the number of MPRs can be exponentially large, we cannot compute line 5 in polynomial time by evaluating $\sum_{R \in C_{T_i}} d(R, M)$ for each M.

To address this problem, we define the set of functions $\{g_i : \mathbb{N}^k \to \{0, 1\}\}$, each of which takes a vector of the distances from an MPR $R \in \mathscr{R}$ to each in $T_j \in \mathcal{T}$ as an input and determines whether R is in C_{T_i}. That is,

$$g_i(\mathbf{d} = [d_1, d_2, \ldots, d_k]) = \begin{cases} 1 & \text{if } i \text{ is the least index such that } d_i \leq d_j \text{ for all } 1 \leq j \leq k \\ 0 & \text{otherwise} \end{cases}$$

Now, we can rewrite line 5 in the algorithm as

$$\underset{M \in \mathscr{R}}{\arg\min} \sum_{R \in \mathscr{R}} d(R, M) \cdot g_i(d(R, \mathcal{T})) \tag{1}$$

By definition, $d(R, M) = |\mathbb{E}(R) \setminus \mathbb{E}(M)| + |\mathbb{E}(M) \setminus \mathbb{E}(R)|$. Moreover:

$$\begin{aligned}|\mathbb{E}(R) \setminus \mathbb{E}(M)| + |\mathbb{E}(M) \setminus \mathbb{E}(R)| &= |\mathbb{E}(R) \cup \mathbb{E}(M)| - |\mathbb{E}(R) \cap \mathbb{E}(M)| \\ &= |\mathbb{E}(R)| + |\mathbb{E}(M)| - 2|\mathbb{E}(R) \cap \mathbb{E}(M)|\end{aligned}$$

Thus, we can rewrite (1) as:

$$\arg\min_{M \in \mathscr{R}} \sum_{R \in \mathscr{R}} \big(|\mathbb{E}(R)| - 2|\mathbb{E}(M) \cap \mathbb{E}(R)| + |\mathbb{E}(M)|\big) \cdot g_i\left(d(R, \mathcal{T})\right) \tag{2}$$

Since $|\mathbb{E}(R)|$ does not depend on M, the minimization problem in (2) is equivalent to the following maximization problem:

$$\arg\max_{M \in \mathscr{R}} \sum_{R \in \mathscr{R}} \big(2|\mathbb{E}(M) \cap \mathbb{E}(R)| - |\mathbb{E}(M)|\big) \cdot g_i\left(d(R, \mathcal{T})\right) \tag{3}$$

Next, we rewrite this as a summation over the events in M:

$$\arg\max_{M \in \mathscr{R}} \sum_{e \in \mathbb{E}(M)} \sum_{R \in \mathscr{R}} \big(2\ |\{e\} \cap \mathbb{E}(R)| - 1\big) \cdot g_i\left(d(R, \mathcal{T})\right) \tag{4}$$

We then split the sum to yield:

$$\arg\max_{M \in \mathscr{R}} \sum_{e \in \mathbb{E}(M)} \left(\sum_{R \in \mathscr{R}} 2\ |\{e\} \cap \mathbb{E}(R)| \cdot g_i\left(d(R, \mathcal{T})\right) - \sum_{R \in \mathscr{R}} g_i\left(d(R, \mathcal{T})\right) \right) \tag{5}$$

Define $S(e)$ to be the set of all reconciliations containing event e. We rewrite the first inner summation as a sum over $S(e)$, since $|\{e\} \cap \mathbb{E}(R)|$ is 1 for all $R \in S(e)$ and 0 for all $R \notin S(e)$:

$$\arg\max_{M \in \mathscr{R}} \sum_{e \in \mathbb{E}(M)} \left(\sum_{R \in S(e)} 2 \cdot g_i\left(d(R, \mathcal{T})\right) - \sum_{R \in \mathscr{R}} g_i\left(d(R, \mathcal{T})\right) \right) \tag{6}$$

Define $f(\mathbf{d}, X)$ to be the set of reconciliations in X such that $d(R, \mathcal{T}) = \mathbf{d}$. Notice that we can partition $S(e)$ as $\{\ f(\mathbf{d}, S(e)) \mid \mathbf{d} \in \mathbb{N}^k\}$ and $\mathscr{R}$ as $\{\ f(\mathbf{d}, \mathscr{R}) \mid \mathbf{d} \in \mathbb{N}^k\}$. Then we can rewrite our sum over these partitions as:

$$\arg\max_{M \in \mathscr{R}} \sum_{e \in \mathbb{E}(M)} \sum_{\mathbf{d} \in \mathbb{N}^k} \left(\sum_{R \in f(\mathbf{d}, S(e))} 2 \cdot g_i\left(d(R, \mathcal{T})\right) - \sum_{R \in f(\mathbf{d}, \mathscr{R})} g_i\left(d(R, \mathcal{T})\right) \right) \tag{7}$$

The inner terms can now be simplified, yielding:

$$\arg\max_{M \in \mathscr{R}} \sum_{e \in \mathbb{E}(M)} \sum_{\mathbf{d} \in \mathbb{N}^k} \left(\sum_{R \in f(\mathbf{d}, S(e))} 2 \cdot g_i(\mathbf{d}) - \sum_{R \in f(\mathbf{d}, \mathscr{R})} g_i(\mathbf{d}) \right) \tag{8}$$

Now we define $\sigma : V(\mathcal{G}) \to \mathbb{R}$ as a function from the vertices of the reconciliation graph to the reals as follows:

$$\sigma(v) = \begin{cases} \sum_{\mathbf{d} \in \mathbb{N}^k} \left(2 \cdot |f(\mathbf{d}, S(v))| - |f(\mathbf{d}, \mathscr{R})|\right) \cdot g_i(\mathbf{d}) & \text{if } v \text{ is an event vertex} \\ 0 & \text{if } v \text{ is a mapping vertex} \end{cases}$$

Notice that $|f(\mathbf{d}, S(v))| = \text{Count}_{\mathcal{T},v}(\mathbf{d})$ and $|f(\mathbf{d}, \mathscr{R})| = \text{Count}_{\mathcal{T}}(\mathbf{d})$ so we have:

$$\sigma(v) = \begin{cases} \sum_{\mathbf{d} \in \mathbb{N}^k} \left(2 \cdot \text{Count}_{\mathcal{T},v}(\mathbf{d}) - \text{Count}_{\mathcal{T}}(\mathbf{d})\right) \cdot g_i(\mathbf{d}) & \text{if } v \text{ is an event vertex} \\ 0 & \text{if } v \text{ is a mapping vertex} \end{cases}$$

This reduces to finding:

$$\arg\max_{M \in \mathscr{R}} \sum_{v \in V(M)} \sigma(v) \tag{9}$$

As noted in the previous section, the problem of finding a reconciliation that maximizes the score of its constituent event vertices can be solved in $O(|G||S|^2)$ time by dynamic programming [9,10]. Thus, we have shown that the widely-used k-medoids heuristic of Park *et al.* [11] can be applied to MPR space by computing the reconciliation count function $\text{Count}_{\mathcal{T},v}$.

3.2 *k*-Centers

The objective of the k-centers problem is to find a set $\mathcal{T}$ of k MPRs that minimizes the *covering radius*, the maximum distance between any MPR and the nearest element in $\mathcal{T}$. More formally, letting MC denote the function that returns the minimum component of a vector, we seek to find:

$$\arg\min_{\substack{\mathcal{T} \subset \mathscr{R} \\ |\mathcal{T}|=k}} \max_{R \in \mathscr{R}} MC(d(R, \mathcal{T}))$$

While this problem is NP-complete [7], Gonzalez [8] proved that the following algorithm finds solutions that are guaranteed to be within a factor of two of optimal:

Algorithm 2 k-centers 2-approximation [8]

```
1: procedure K-CENTERS(ℛ, k)
2:     𝒯 ← {arbitrary R in ℛ}
3:     for k − 1 iterations do
4:         T ← arg max_{R∈ℛ} MC(d(R, 𝒯))
5:         𝒯 ← 𝒯 ∪ {T}
6:     return 𝒯
```

Line 4 cannot be efficiently computed by simple iteration because there can be exponentially many MPRs. Again, the reconciliation count function will be applied to perform this step in polynomial time.

Given some number of current centers, line 4 of Algorithm 2 seeks a reconciliation that maximizes the minimum distance from the rest. Given the reconciliation counts to the current centers, we can find this distance by the following algorithm.

Algorithm 3 Farthest from centers

```
1: procedure FFC(𝒯)
2:     d* ← 0
3:     for d ∈ ℕ^k do
4:         if Count_𝒯(d) > 0 then
5:             if MC(d) > MC(d*) then
6:                 d* ← d
7:     return d*
```

Although the loop in Algorithm 3 iterates over all k-dimensional vectors over the natural numbers, we will see later that only a polynomial number of vectors need be considered.

Algorithm 3 gives us a distance vector, but not the desired reconciliation at this distance. However, we can augment the algorithm for computing reconciliation counts (given in the next section) to record one such reconciliation. That is, we can extend the reconciliation counts algorithm with annotations such that $\text{Count}_{\mathcal{T},v}(\mathbf{d}) = (n, R)$ if and only if there are n distinct reconciliations at distance $\mathbf{d}$ from $\mathcal{T}$ which contain event v, and R is one such reconciliation. Therefore, a polynomial time method for computing reconciliation counts allows us to find a 2-approximation to k-centers for MPRs.

4 Computing the Reconciliation Count Function

In this section, we give an algorithm for computing the reconciliation count function, $\text{Count}_{\mathcal{T},v} : \mathbb{N}^k \to \mathbb{N}$. The algorithm uses three basic operations: sum, convolution, and shift. Let f and g be functions from $\mathbb{Z}^k$ to $\mathbb{N}$. The *sum*, $h = f + g$, is defined by

$$h(\mathbf{n}) = f(\mathbf{n}) + g(\mathbf{n})$$

The *convolution*, $h = f * g$ is defined by

$$h(\mathbf{n}) = \sum_{\mathbf{m} \in \mathbb{Z}^k} f(\mathbf{m}) g(\mathbf{n} - \mathbf{m})$$

The *shift*, $h = f \gg \mathbf{m}$ for some $\mathbf{m} \in \mathbb{Z}^k$ is defined by

$$h(\mathbf{n}) = f(\mathbf{n} - \mathbf{m})$$

In addition, for a given $\mathbf{d} \in \mathbb{Z}^k$ the function $\delta_{\mathbf{d}} : \mathbb{Z}^k \to \mathbb{N}$ is defined by

$$\delta_{\mathbf{d}}(\mathbf{d}) = 1 \text{ and } \delta_{\mathbf{d}}(\mathbf{n}) = 0 \text{ for } \mathbf{n} \neq \mathbf{d}$$

Finally, for a vertex v in the reconciliation graph $\mathcal{G}$ and a MPR T, we define the function $D(v,T)$ as follows:

$$D(v,T) = \begin{cases} -1 & \text{if } v \in \mathbb{E}(T) \\ 1 & \text{if } v \notin \mathbb{E}(T) \text{ and } v \in \mathbb{E}(\mathcal{G}) \\ 0 & \text{if } v \notin \mathbb{E}(\mathcal{G}) \end{cases}$$

For $\mathcal{T} = \{T_1, \ldots, T_k\}$, we define $D(v,\mathcal{T}) = [D(v,T_1), \cdots, D(v,T_k)]$.

Note that while the convolution is defined using an infinite summation, in our usage it will only take a polynomial number of non-zero values and will be shown to be computable in polynomial time.

In Algorithm 4 we give three procedures: The main Count function for computing $\text{Count}_{\mathcal{T},v}$ which uses procedures SubCount and SuperCount. For clarity, these procedures are described recursively, but the implementation (described in the Supplementary Materials) applies dynamic programming to compute the values bottom-up in polynomial time. The algorithm uses the sum, convolution, shift and $\delta_{\mathbf{d}}$ functions described above.

Algorithm 4 Reconciliation Count

```
1: procedure Count_T(v)
2:     return (SubCount_T(v) * SuperCount_T(v)) ≫ [|E(T_1)|, ..., |E(T_k)|]
3:
4: procedure SubCount_T(v)
5:     if v is a leaf of G then
6:         return δ_[-1,-1,...,-1]
7:     if v is a mapping vertex then
8:         for each child c_i of v do
9:             f_i ← SubCount_T(c_i)
10:        return Σ_i f_i
11:    if v is an event vertex then
12:        if v has one child c then
13:            return SubCount_T(c) ≫ D(v, T)
14:        else v has two children c_1, c_2
15:            return (SubCount_T(c_1) * SubCount_T(c_2)) ≫ D(v, T)
16:
17: procedure SuperCount_T(v)
18:    if v is a root of G then
19:        return δ_[0,0,...,0]
20:    if v is a mapping vertex then
21:        for each parent p_i of v do
22:            f_i ← SuperCount_T(p) ≫ D(p, T)
23:            if v has a sibling s under p_i then
24:                f_i ← f_i * SubCount_T(s)
25:        return Σ_i f_i
26:    if v is an event vertex then
27:        p ← the parent of v
28:        return SuperCount_T(p)
```

Theorem 2. *Given a reconciliation graph $\mathcal{G}$ for a DTL problem instance and a set $\mathcal{T}$ of k MPRs, the procedure* $\text{Count}_{\mathcal{T}}(v)$ *in Algorithm 4 correctly computes the reconciliation count function.*

Let n denote the larger of the size of the gene tree and the size of the species tree in a DTL instance and let k be a fixed constant representing the size of the representative set $\mathcal{T}$.

Theorem 3. *The worst-case time complexity of Algorithm 4 is* $O(n^{k+3} \log n)$.

Theorem 4. *The worst-case time complexity for performing* I *iterations of the* k*-medoids algorithm is* $O(In^{k+3} \log n)$.

Theorem 5. *The worst-case time complexity of the* k*-centers algorithm is* $O(n^{k+3} \log n)$.

Proofs of these theorems as well as experimental results are available in Supplementary Materials at `www.cs.hmc.edu/~hadas/supplement.pdf`.

5 Conclusion

In this paper we have studied the problem of clustering the exponentially large space of MPRs in order to find a small number of representative reconciliations and to better understand the structure of MPR space. We have defined a *reconciliation count function*, shown that it can be computed in polynomial time, and demonstrated how this function can be used to implement some well-known clustering algorithms. These results can be extended to other clustering algorithms and to reconciliations whose cost is within some bound of MPR cost.

Finally, there are a number of promising directions for future research. First, while the reconciliation count algorithm runs in polynomial time for any constant k, the running time of $O(n^{k+3} \log n)$ becomes impractical for large values of k. While our initial experimental results suggest that small values of k are likely to be of greatest interest, it may be possible to compute the reconciliation count function more efficiently. Second, the reconciliation count function described here appears to have broad utility in implementing other clustering methods and algorithms and in computing a variety of statistics on clusterings of MPR space. While our experimental results demonstrate the viability of clustering, systematic empirical studies are needed to better understand what clusterings can reveal about the structure of MPR space.

Acknowledgements

This work was funded by the U.S. National Science Foundation under Grant Numbers IIS-1419739 and 1433220. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors wish to thank Yi-Chieh Wu and Mukul Bansal for valuable advice and feedback.

References

1. Bansal, M.S., Alm, E.J., Kellis, M.: Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. Bioinformatics 28(12), i283–i291 (2012)
2. Bansal, M.S., Alm, E.J., Kellis, M.: Reconciliation revisited: handling multiple optima when reconciling with duplication, transfer, and loss. In: Research in Computational Molecular Biology. pp. 1–13. Springer (2013)
3. Charleston, M.A., Perkins, S.L.: Traversing the tangle: algorithms and applications for cophylogenetic studies. Journal of Biomedical Informatics 39(1), 62–71 (2006)
4. Conow, C., Fielder, D., Ovadia, Y., Libeskind-Hadas, R.: Jane: A new tool for cophylogeny reconstruction problem. Algorithms for Molecular Biology 5(16) (2010)
5. David, L.A., Alm, E.J.: Rapid evolutionary innovation during an archaean genetic expansion. Nature 469, 93–96 (2011)
6. Doyon, J.P., Scornavacca, C., Gorbunov, K.Y., Szöllősi, G.J., Ranwez, V., Berry, V.: An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications and transfers. Comparative Genomics 6398, 93–108 (2011)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
8. González, T.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38, 293 – 306 (1985)
9. Ma, W., Smirnov, D., Forman, J., Schweickart, A., Slocum, C., Srinivasan, S., Libeskind-Hadas, R.: DTL-RnB: Algorithms and tools for summarizing the space of DTL reconciliations. IEEE/ACM Trans. on Comp. Bio and Bioinfo. (2016)
10. Nguyen, T.H., Ranwez, V., Berry, V., Scornavacca, C.: Support measures to estimate the reliability of evolutionary events predicted by reconciliation methods. PloS one 8(10), e73667 (2013)
11. Park, H.S., Jun, C.H.: A simple and fast algorithm for k-medoids clustering. Expert Systems with Applications 36(2), 3336–3341 (2009)
12. Scornavacca, C., Paprotny, W., Berry, V., Ranwez, V.: Representing a set of reconciliations in a compact way. Journal of Bioinformatics and Computational Biology 11(02), 1250025 (2013), pMID: 23600816
13. Than, C., Ruths, D., Innan, H., Nakhleh, L.: Confounding factors in hgt detection: statistical error, coalescent effects, and multiple solutions. Journal of Computational Biology 14(4), 517–535 (2007)
14. To, T.H., Jacox, E., Ranwez, V., Scornavacca, C.: A fast method for calculating reliable event supports in tree reconciliations via pareto optimality. BMC Bioinformatics 16, 384 (2015)
15. Tofigh, A.: Using Trees to Capture Reticulate Evolution: Lateral Gene Transfers and Cancer Progression. Ph.D. thesis, KTH Royal Institute of Technology (2009)
16. Tofigh, A., Hallett, M.T., Lagergren, J.: Simultaneous identification of duplications and lateral gene transfers. IEEE/ACM Trans. on Comp. Bio and Bioinfo. 8(2), 517–535 (2011)