Minimizing the Average Distance to a Closest Leaf in a Phylogenetic Tree

FREDERICK A. MATSEN IV*, AARON GALLAGHER, AND CONNOR O. MCCOY

Program in Computational Biology, Fred Hutchinson Cancer Research Center, Seattle, WA 91802, USA
*Correspondence to be sent to: Program in Computational Biology, Fred Hutchinson Cancer Research Center, Seattle, WA 91802, USA;
E-mail: matsen@fhcrc.org.

Abstract.—When performing an analysis on a collection of molecular sequences, it can be convenient to reduce the number of sequences under consideration while maintaining some characteristic of a larger collection of sequences. For example, one may wish to select a subset of high-quality sequences that represent the diversity of a larger collection of sequences. One may also wish to specialize a large database of characterized "reference sequences" to a smaller subset that is as close as possible on average to a collection of "query sequences" of interest. Such a representative subset can be useful whenever one wishes to find a set of reference sequences that is appropriate to use for comparative analysis of environmentally derived sequences, such as for selecting "reference tree" sequences for phylogenetic placement of metagenomic reads. In this article, we formalize these problems in terms of the minimization of the Average Distance to the Closest Leaf (ADCL) and investigate algorithms to perform the relevant minimization. We show that the greedy algorithm is not effective, show that a variant of the Partitioning Around Medoids (PAM) heuristic gets stuck in local minima, and develop an exact dynamic programming approach. Using this exact program we note that the performance of PAM appears to be good for simulated trees, and is faster than the exact algorithm for small trees. On the other hand, the exact program gives solutions for all numbers of leaves less than or equal to the given desired number of leaves, whereas PAM only gives a solution for the prespecified number of leaves. Via application to real data, we show that the ADCL criterion chooses chimeric sequences less often than random subsets, whereas the maximization of phylogenetic diversity chooses them more often than random. These algorithms have been implemented in publicly available software. [Mass transport; phylogenetic diversity; sequence selection.]

This article introduces a method for selecting a subset of sequences of a given size from a pool of candidate sequences in order to solve one of two problems. The first problem is to find a subset of a given collection of sequences that are representative of the diversity of that collection in some general sense. The second is to find a set of "reference" sequences that are as close as possible on average to a collection of "query" sequences.

Algorithms for the first problem, selecting a diverse subset of sequences from a pool based on a phylogenetic criterion, have a long history. The most well used such criterion is maximization of *phylogenetic diversity* (PD), the total branch length spanned by a subset of the leaves (Faith, 1992). The most commonly cited applications of these methods is to either select species to preserve (Faith, 1992) or to expend resources to perform sequencing (Pardi and Goldman 2005; Wu et al. 2009). It is also commonly used for selecting sequences that are to be used as a representative subset that span the diversity of a set of sequences.

PD is a useful objective function that can be maximized very efficiently (Minh et al. 2006), but it has some limitations when used for the selection of representative sequences. Because maximizing the phylogenetic diversity function explicitly tries to choose sequences that are distant from one another, it tends to select sequences on long pendant branches (Bordewich et al. 2008); these sequences can be of low quality or otherwise different than the rest of the sequences. Furthermore, PD has no notion of weighting sequences by abundance, and as such it can select artifactual sequences or other rare sequences that may not form part of the desired set of sequences. This motivates the development of algorithms that strike a balance between centrality and diversity, such that one finds central sequences within a broad diversity of clusters.

The second problem addressed by the present work is motivated by modern genetics and genomics studies, where it is very common to learn about organisms by sequencing their genetic material. When doing so, it is often necessary to find a collection of sequences of known origin with which to do comparative analysis. Once these relevant "reference" sequences are in hand, a hypothesis or hypotheses on the unknown "query" sequences can be tested.

Although it is easy to pick out reference sequences that are close to an individual query sequence using sequence similarity searches such as BLAST, we are not aware of the methods that attempt to find a collection of reference sequences that are close on average to a *collection* of query sequences. Such a method would have many applications in studies that use phylogenetics. In phylogenomics, sequences of known function are used to infer the function of sequences of unknown function (Eisen et al. 1995; Eisen 1998; Engelhardt et al. 2005). In the study of HIV infections, hypotheses about the history of infection events can be phrased in terms of clade structure in phylogenetic trees built from both query and reference sequences (Piantadosi et al. 2007). In metagenomics, it is now common to "place" a read of unknown origin into a previously constructed phylogeny (Matsen et al. 2010; Berger et al. 2011). Each of these settings requires a set of reference sequences that are close on average to the collection of query sequences.

One approach to picking reference sequences would be to pick every potentially relevant sequence, such as all HIV reference sequences of the relevant subtype, but this strategy is not always practical. Although many

strategies in phylogenetics have been developed to speed inference, most analyses still require quadratic or greater execution time. On the other hand, the number of sequences available to do comparative analysis is growing at an exponential pace. This motivates strategies to pick useful subsets of sequences.

It may seem ironic that in order to find a useful subset of sequences for phylogenetics, we propose a fairly complex algorithm to use on a tree that has already been built; there are several reasons that have led us to develop this methodology. First, tree-building methods vary widely in their running time, from subquadratic time methods (Price et al. 2010) to very computationally expensive Bayesian methods that model many aspects of the mutation process. Similarly, analytic methods taking a tree as input may scale poorly with large numbers of taxa. When a dataset is too large for an expensive method, our algorithm can be used in conjunction with a fast/approximate phylogenetic method to pick a subset of sequences to use in the more complex method. Second, we note that there is a remarkable quantity of sequences for certain loci, such as over 2 million 16s sequences from Release 10 of the RDP database (Cole et al. 2009). Because this number will continue to increase, and many of these sequences are redundant, we feel the need to have a principled method useful for curators to pick sequences that can form a representative subset of these large databases. Others can then use the results of this curation process without having to run the algorithm themselves.

The objective is simple: select the collection of sequences that are on average as close as possible in terms of phylogenetic relatedness to the set of input sequences. We now more formally state the two problems described above (Fig. 1).

Problem 1. For a given phylogenetic tree T and desired number of leaves k, find a k-element subset X of the leaves L that minimizes the Average Distance from each leaf in L to its Closest Leaf (ADCL) in X.

We emphasize that the distance is calculated between each leaf and its closest representative in X.

Problem 2. Given T and k as before, but let $R \subset L$ be a set of "reference sequences." Find the k-element subset X of R that minimizes the Average Distance of the leaves in $L \setminus R$ (the "query sequences") to their Closest Leaf in X.

Recalling that the branch length between two sequences is typically the expected number of substitutions per site for those sequences, we are usually calculating the average expected number of substitutions relating each sequence to its closest selected leaf.

These criteria, along with generalizations, can be expressed in a single framework in terms of "mass transport" (Villani, 2003) on trees as follows (Fig. 1). In this framework, the "work" needed to move a mass m a distance d is defined to be m times d. For Problem 1 above, assume we are interested in selecting sequences according to the first criterion on a tree with n sequences. Distribute mass $1/n$ at each of the leaves, and then find

the set X of k leaves such that the work required to move the mass to one of the leaves in X is minimized. This is equivalent to minimizing the ADCL criterion, because the optimal solution will have all of the mass for a single leaf being transported to its closest included leaf, incurring a cost of that distance divided by n; the sum of these individual quantities of work will be equal to the ADCL.

In a similar way, the second criterion can be phrased as evenly dividing a unit of mass among the tips of $L \setminus R$, and finding a set X of leaves in R minimizing work as before. In this second criterion, call the tree induced by the reference sequences R the "reference tree." Because mass can only be transported to reference tree leaves and not query leaves, all of the mass of the query sequences must first be transported somewhere on the reference tree. This amount of work is a fixed cost, and thus we can just think of the mass for a subtree composed of only query sequences as appearing at the attachment location of that query-only subtree to the reference tree. This change of perspective will change the magnitude of, but not the differences between, the ADCL values, thus giving an equivalent solution to Problem 2.

A further motivation for considering mass at internal nodes of a tree comes from *phylogenetic placement*, that is the mapping of query sequences into a tree built from the reference sequences. This collection of placements can then be thought of as a collection of mass on the tree, and the optimization can proceed as above. The transition of placements to mass distribution can include "spreading" out mass according to uncertainty in placement (Matsen et al. 2010; Evans and Matsen 2012). Because of the speed of placement algorithms, this can be a useful way of proceeding when the set of query sequences is large. We have previously used mass transport to measure the differences between collections of placements (Evans and Matsen, 2012). In this context, a collection of query sequences can be mapped onto the tree and used to pick an optimal subset of reference sequences.

Arbitrary distributions of mass on the tree are also possible. These distributions may arise from transforms of placement distributions. Alternatively, they may arise by assigning an arbitrary value to various locations on the tree; these values may convey the importance of regions of a tree for some analysis.

Because all of these can be formulated in terms of mass on a phylogenetic tree, rather than considering Problems 1 and 2 separately, we solve the following generalization of both of them.

Problem 3. Given a mass distribution μ on a phylogenetic tree T with n leaves and some $0 < k < n$, find the k-element subset X of the leaves of T such that the work required to move the mass $\mu(x)$ at point $x \in T$ to x's closest leaf in X is minimized across all k-element subsets of the leaves of T.

We will still call this problem "minimizing ADCL" because it attempts to minimize the average distance to the closest leaf, where now the average is weighted by the mass distribution. It should also be pointed out
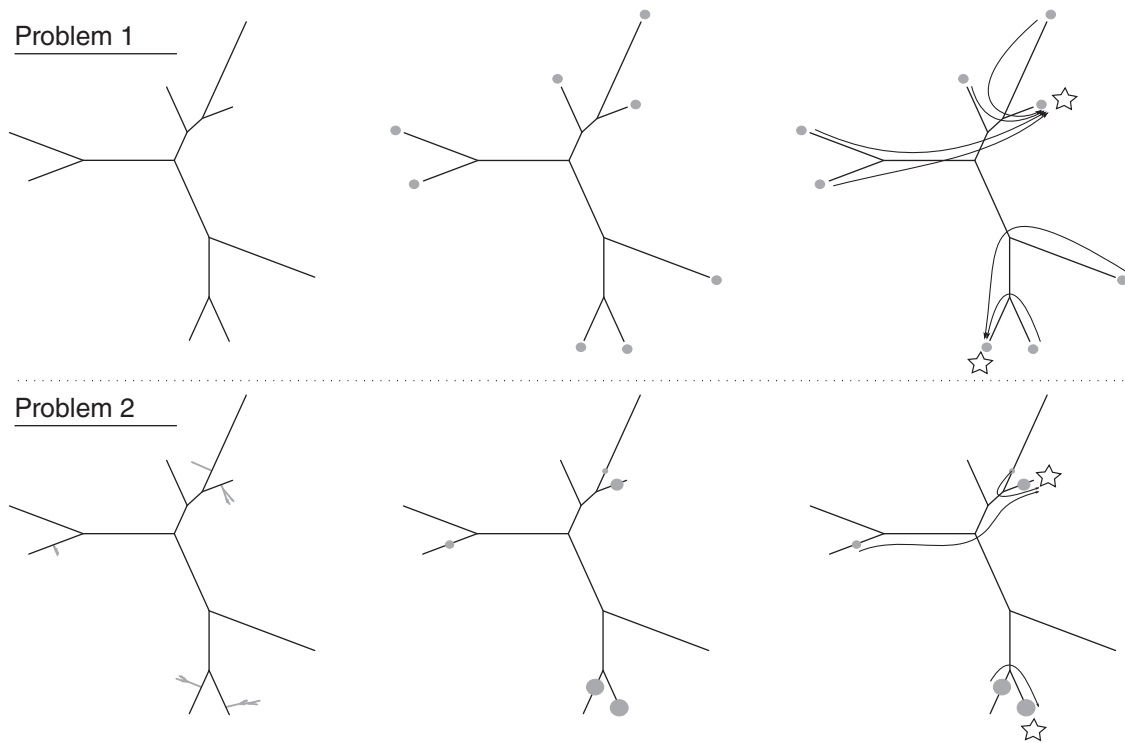
FIGURE 1.   A diagram showing two example ADCL minimization problems. The k selected leaves are marked with hollow stars; in this case $k = 2$. Problem 1 is to minimize the average distance from each leaf to its closest selected leaf. Problem 2 is to minimize the average distance from the query sequences (gray branches) to their closest reference sequence (reference sequence subtree in black). Both of these problems can be thought of as instances of Problem 3, which is to minimize the work required to move mass (gray circles) to a subset of k leaves. In Problem 1, a unit of mass is uniformly divided among the leaves of the tree. In Problem 2, mass is distributed in proportion to the number of query leaves that attach at that point.

that the distances used in the ADCL framework are the distances in the original tree. When leaves are pruned out of the tree, a tree built *de novo* on this reduced set will have different branch lengths than the original tree with branches pruned out; we do not attempt to correct for that effect here. A more formal statement of Problem 3 is made in the Appendix.

Problem 1 is equivalent to the DC1 criteria independently described in chapter 5 of Barbara Holland's Ph.D. thesis (Holland, 2001). She writes out the criterion (among others), discusses why it might be biologically relevant, describes the computational complexity of the brute-force algorithm, and does some experiments comparing the brute-force to the greedy algorithm.

We also note that the work described here shares some similarities with the Maximizing Minimum Distance (MMD) criterion of Bordewich et al. (2008). In that criterion, the idea is to select the subset X of leaves such that the minimum distance between any two leaves in X is maximized across subsets of size k. The MMD criterion has more similarities with PD maximization than it does with Problem 1. Moreover, the MMD analog of Problem 2 (minimizing the maximum distance of a reference sequence to a query sequence) would be highly susceptible to off-target query sequences, such as sequences that are similar to but not actually homologous to the reference set. Because of this

difference in objective functions, we have not attempted a comparison with MMD here.

The analogous problem in the general nonphylogenetic setting is the classical k-medoids problem where k "centers" are found minimizing the average distance from each point to its closest center. The Partitioning Around Medoids (PAM) algorithm (described below) is a general heuristic for such problems, although our exact algorithm, which is based on additivity of distances in a tree, will not work. It appears that the complexity of exact k-medoids is not known and is only bounded above by the obvious brute-force bound. The simpler setting of k-means in the plane has been shown to be NP-hard by Mahajan et al. (2009).

We emphasize that for the purposes of this article, we assume that k has been chosen ahead of time. Although we consider choosing an appropriate k to be an interesting direction for future work, as described in the discussion, the choice will depend substantially on the goals of the user.

METHODS

In this section, we will investigate methods to minimize ADCL for a given mass distribution as in Problem 3. We will first show that the greedy algorithm

fails to provide an optimal solution, then describe a variant of the PAM algorithm that finds a local minimum of ADCL, and then describe our dynamic program that is guaranteed to find a global minimum.

For all of these algorithms, the structure of the tree is not reestimated and branch lengths not reoptimized after removal of leaves. Indeed, the tree is not changed at all, rather the removal is from the set of selected leaves.

Optimization via Greedy Leaf Pruning

The PD minimization problem is known to be solved exactly by a greedy leaf pruning algorithm (Steel, 2005), and by analogy a reasonable first attempt might be to try to apply the same approach.

Algorithm 4. (**Greedy leaf pruning**) Given a tree T, a mass distribution μ, and a desired number of leaves k, start with X being all of the leaves of the tree.

1. If $|X| = k$ then stop.
2. Find the ℓ minimizing the ADCL of $X \setminus \{\ell\}$
3. Remove ℓ from X and return to (1).

A similar algorithm, which instead greedily *adds* sequences to the chosen set, was independently described by Barbara Holland in her thesis (Holland, 2001).

Proposition 5. Algorithm 4 does not find an optimal solution to Problem 3.

Proof. Consider the problem of finding the ADCL-minimizing leaf set for $k = 1$ on a three taxon tree. Fix a three-taxon tree with a single internal node, and label the leaves n_0, n_1, and n_2. Assign mass m to leaves n_0 and n_1, and assign mass $m - \epsilon$ to leaf n_2. Let the edges going to n_0 and n_1 have length x, and the edge going to n_2 have length y. Choose these values satisfying $0 < \epsilon < m$ and $0 < y < x$, and such that $\epsilon \cdot (x + y) < m \cdot (x - y)$.

A greedy algorithm will delete leaf n_2 as a first step, because deleting either n_0 or n_1 increases ADCL by $m \cdot (x + y)$, whereas deleting n_2 increases it by $(m - \epsilon) \cdot (x + y)$. However, deleting n_0 and n_1 at once leads to an ADCL of $2m \cdot (x + y)$, whereas the other options give an ADCL of $m \cdot 2x + (m - \epsilon) \cdot (x + y)$. By our choice of m, ϵ, x, and y,

$$2m \cdot (x + y) < m \cdot 2x + (m - \epsilon) \cdot (x + y).$$

Therefore removing n_0 and n_1 is optimal, whereas the greedy algorithm removes n_2 in its first step. ∎

This approach has shown poor enough performance in practice compared with the one in the next section (results not shown) that we have not pursued efficient optimization.

Optimization via PAM

A different way to minimize ADCL is to adapt heuristic algorithms for the so-called k-medoids problem. The objective of k-medoid clustering is the same as k-means clustering, except that the cluster centers X must be chosen to be elements of the set L being clustered. Those chosen centers $X \subset L$ are called "medoids." That is, the objective is to find a subset X of elements minimizing the average distance from each element y of L to the closest element x_y of X. Problem 1 can be expressed as a standard k-medoid problem, where the points are leaves of the tree and the distances between them are distances between those leaves of the tree. One common approach for the k-medoid problem is called the PAM algorithm (Theodoridis and Koutroumbas, 2006). This algorithm starts with a random selection of k medoids, then executes a hill-climbing heuristic to improve the relevant objective function.

Problem 2 can also be formulated as a variant of PAM that we now describe. The same algorithm can be used for Problem 3 in the (common) case of a discrete mass distribution. Let J be an arbitrary set of items. Assume we are given a distance matrix M that measures the distance from a set of "leaves" L to the items in J, as well as some $0 < k \leq |L|$. The goal is to find a k-element set of leaves $X \subset L$ minimizing the objective function, which is the average distance of each item in J to its closest leaf in X.

Algorithm 6. (**PAM variant**) Initialize $X \subset L$ as a random selection of k leaves. Repeat the following process until an iteration over every value in X does not strictly decrease the objective function.

1. For a single i in X; remove it from X and try adding every other $j \in L \setminus X$ to X in its place.
2. Keep the best such exchange if it decreases the objective function.
3. Continue with the next i in X.

This differs from the traditional formulation of PAM in two ways. First, the set J is not necessarily identical to L. Second, whereas PAM examines every combination from $X \times (L \setminus X)$, choosing the exchange that most decreases the objective function, this variant only examines the potential exchanges for a single medoid at a time, making the exchange of these that most decreases the objective function before continuing with the next medoid.

The complexity of this PAM variant is $O(k(|X| - k)|J|)$ for every iteration.

PAM will always find a local minimum in terms of these pairwise exchanges; however, it will not always find a global minimum as shown in Figure 2 and Table 1. In that example, there are four masses $\{a, b, c, d\}$ on a four taxon tree. Assume we are trying to find the pair of leaves minimizing ADCL, and PAM selects $\{n_1, n_2\}$ as a starting set. The optimal solution is to take $\{n_0, n_3\}$. Because it only swaps a single pair of sequences, and the ADCL increases for every such swap from $\{n_1, n_2\}$, it will not be able to escape this local minimum (Table 1).

Exact Algorithm

We also present the following exact dynamic program. We note that the exact algorithm gives solutions for all
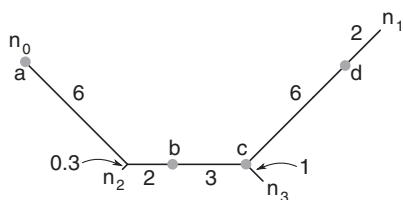
FIGURE 2.   An example where the PAM algorithm gets stuck in a local minimum. Assume masses of equal magnitude $a, \ldots, d$ on a tree with leaves $n_0, \ldots, n_3$, and two leaves are desired from the algorithm (i.e., $k=2$). Branch lengths are as marked on the tree. The optimal solution is to take $\{n_0, n_3\}$. However, if PAM starts with $\{n_1, n_2\}$, it will not be able to make it to the optimal solution by changing one leaf at a time because the ADCL values for every other pair is greater than that for $\{n_0, n_3\}$ and $\{n_1, n_2\}$ (Table 1).

TABLE 1.   ADCL values for all subsets of 2 leaves for the example in Figure 2

subset	ADCL
$\{n_0, n_1\}$	4.500
$\{n_0, n_2\}$	4.725
$\{n_0, n_3\}$	3.000
$\{n_1, n_2\}$	3.975
$\{n_1, n_3\}$	4.750
$\{n_2, n_3\}$	4.150

numbers of leaves less than or equal to the given desired number of leaves k, which can be useful when the best k is to be inferred from the data. This section will give a high-level overview of the exact algorithm; for a complete description see the Appendix.

Our exact algorithm is a dynamic program that proceeds from a chosen "root" out to the leaves then back to the root of the tree. Assume the dynamic program has descended into a subtree S of T. The optimal solution will allow some number of leaves to be used within S, and will have some amount, direction, and distance of mass transport through the root of S. However, by the nature of a dynamic program, this number of leaves and mass transportation characteristics are necessarily not known when the algorithm is visiting the subtree S. For this reason, the algorithm builds a collection of "partial solutions" for every subtree and number of selected leaves less than or equal to k; these partial solutions are indexed by the amount and direction of mass transport going through the root of S and only specify where the mass within that subtree will go (Fig. 3). When progressing proximally (toward the root) through an internal node that is the root of a subtree S, the candidate list of partial solutions for S is built from every combination of partial solutions for the subtrees of S. Because this combination is done for every allocation of leaves to subtrees of total number of leaves less than or equal to k, the final output of this algorithm is solutions for every number of allowed leaves less than or equal to k.

These solution sets can become very large, and it is necessary to cut them down in order to have a practical algorithm. Ideally, this dynamic program would only maintain partial solutions that could be optimal for
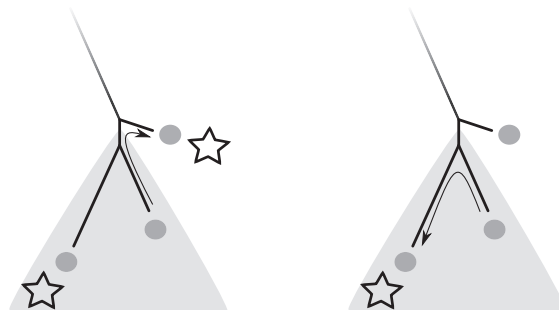

FIGURE 3.   The movement of mass within a subtree depends on the selection of leaves outside the subtree, motivating a dynamic program that keeps solutions that could be optimal for a variety of circumstances outside of the tree. Here, stars represent selected leaves and filled circles represent masses.

some number of leaves and some amount and direction of mass transport through the root of S. In fact it is possible to only keep *exactly* that set of partial solutions using methods from geometry. The partial solutions can be partitioned by the number of leaves used and the direction of mass transport through the root of S. These solutions will either be *root mass distal* (RMD) solutions, where the solution may accept mass to flow into the subtree from the outside, or *root mass proximal* (RMP) solutions, where the solution sends mass through the root of the subtree to the outside. Note that the distinction between these two types of partial solutions concerns the flow of the mass through the root of the subtree only. For example, RMP solutions with a nonempty leaf set have some mass flowing toward those leaves and a (possibly empty) amount of mass flowing proximally out of the root. Whether a partial solution is an RMP or an RMD solution specifies its *mass direction class*.

Thus for the dynamic program, we will solve the optimal mass transport for all possible contexts of the rest of the tree: when the partial solution has *proximal mass* going proximally through the root of S to some leaf of unknown distance in the rest of the tree, or some unknown amount of mass descending to a leaf in S. For an RMP solution, the amount of work required in a given partial solution to move the mass in S to some selected leaf is equal to the mass transport within S plus the amount of work required to move the proximal mass of that partial solution some distance x away from the root of S. The amount of work is linear in x, with y-intercept the amount of work within S, and with slope equal to the amount of mass that moves outside of S. For a given partial solution, we can plot the contribution of the mass in S to the ADCL as a function of x. In a similar way, we can plot the amount of work required for an RMD solution, except this time the appropriate parameter x to use is the amount of mass that comes distally (away from the root) through the root of S. The work is again linear in x, and the y-intercept is again the amount of work within S, but with slope equal to the distance to the closest leaf that is selected in S. These lines will be
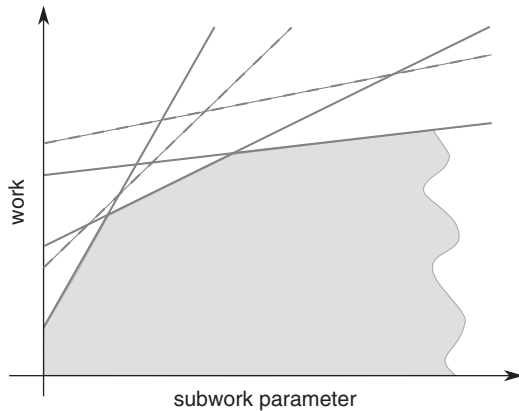
FIGURE 4.   A visual depiction of the method of removing partial solutions that could not be optimal for any setting in the rest of the tree. Each line represents the total work for a partial solution that has subwork parameter x. Because the dashed lines are not minimal for any value of x, they can be discarded.

called *subwork* lines, and the parameter x will be called the *subwork parameter*.

The only partial solutions that could form part of an optimal solution are those that are optimal for some value of the subwork parameter (Fig. 4). This optimization can be done using well-known algorithms in geometry. Imagine that instead of considering the minimum of a collection of subwork lines, we are using these lines to describe a subset of the plane using inequalities. Some of these inequalities are spurious and can be thrown away without changing the subset of the plane; the rest are called *facets* (Ziegler, 1995). Our implementation uses Komei Fukuda's cddlib implementation (Fukuda, 2012) of the double description method (Fukuda and Prodon, 1996) to find facets. The complexity of our exact algorithm is difficult to assess, given that in the words of Fukuda and Prodon (1996), "we can hardly state any interesting theorems on [the double description algorithm's] time and space complexities." We note that our code uses the floating point, rather than exact arithmetic, version of cddlib because branch lengths are typically obtained by numerical optimization to a certain precision. For this reason, our implementation is susceptible to rounding errors, and ADCLs are only compared to within a certain precision in the implementation.

RESULTS

We have developed and implemented algorithms to minimize the ADCL among subsets of leaves of the reference tree. These algorithms are implemented as the min_adcl_tree (Problem 1) and min_adcl (Problems 1, 3) subcommands of the rppr binary that is part of the pplacer (http://matsen.fhcrc.org/pplacer/) suite of programs. The code for all of the pplacer suite is freely available (http://github.com/matsen/pplacer).

Our PAM implementation follows Algorithm 6. We found this variant, which makes the best exchange

at each medoid rather than the best exchange over all medoids, to converge two orders of magnitude more rapidly than a traditional PAM implementation (Supplementary Figs. S3 and S4).

We used simulation to understand the frequency with which PAM local minima are not global minima as in Figure 2, as well as the relative speed of PAM and the exact algorithm. For tree simulation, random trees were generated according to the Yule process (Yule, 1925); branch lengths were drawn from the γ distribution with shape parameter 2 and scale parameter 1. We evaluated three datasets: two "tree" sets and one "mass" set. For the "tree" test sets, trees were randomly generated as described, resulting in 5 trees of 1000 leaves each, and a collection of trees with 10 to 2500 leaves in increments of 10 leaves. Problem 1 was then solved using each algorithm for each "tree" test set; for the 1000 leaf trees k was set to each number from 1 to 991 congruent to 1 mod 10, and for the large collection of trees k was set to half the number of leaves. For the "mass" test set, one tree was built for each number of leaves from 5 to 55 (inclusive); for each of these trees, m masses were assigned to uniformly selected edges of the tree at uniform positions on the edges, where m ranged from 5 to 95 masses in 10-mass increments. All of these simulated test sets have been deposited in Dryad (http://datadryad.org/handle/10255/dryad.41611). Problem 2 was then solved using each algorithm with k equal to (the ceiling of) half the number of leaves.

The PAM heuristic typically works well. Although we have shown above that the PAM algorithm does get stuck in local minima (Fig. 2), it did so rarely on the "mass" dataset (Fig. 5); similar results were obtained for the "tree" dataset (results not shown). As might be expected, PAM displays the greatest speed advantage in when k is rather large on the "tree" dataset (Fig. 6). PAM is slowest for k equal to $n/2$ because that value of k has the largest number of possible k-subsets; once $k > n/2$ it gets faster because there are fewer choices as far as what to select. PAM is faster for small trees than the exact algorithm on the "tree" dataset (Fig. 7), and uses less memory (Supplementary Fig. S2). We note in passing that the ADCL improvement from PAM is not monotonically nonincreasing, which is to say that it is possible to have a small improvement followed by a large improvement.

The currently available tools for automatic selection of reference sequences include the use of an algorithm that maximizes PD or using a random set of sequences (Redd et al. 2011). Others have pointed out that long pendant branch lengths are preferentially chosen by PD maximization, even when additional "real" diversity is available (Bordewich et al. 2008). Long pendant branch lengths can be indicative of problematic sequences, such as chimeras or sequencing error.

We designed an experiment to measure the extent to which the ADCL algorithm would pick problematic sequences compared with PD maximization and random selection. We downloaded sequences with taxonomic annotations from GenBank belonging to the family
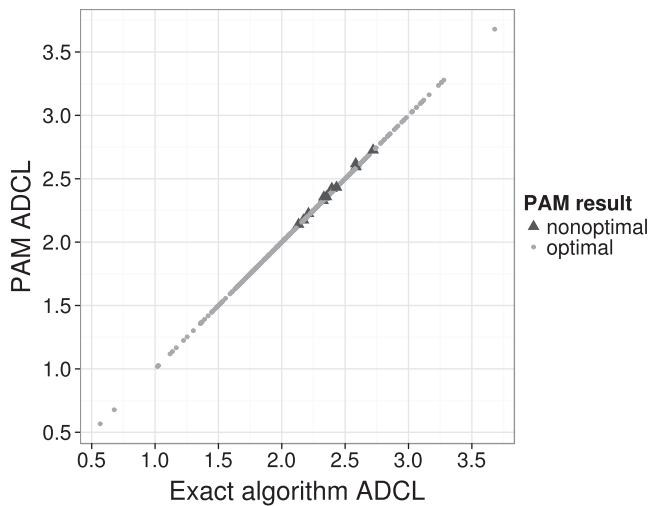
FIGURE 5.   Comparison between the ADCL results obtained by the exact algorithm and the PAM heuristic on the "mass" test set. The points are black triangles when the difference in ADCL between PAM and the exact algorithm is greater than 10^{-5}.
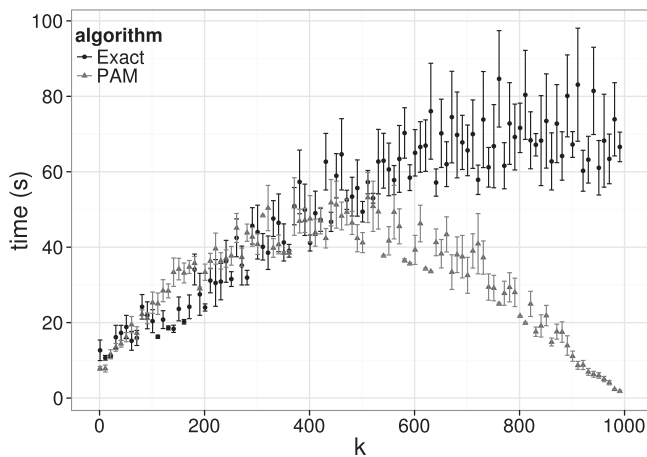

FIGURE 6.   Comparison of the time required to run the exact algorithm versus PAM with respect to the number of leaves selected to keep. Five trees were generated of 1000 leaves each, and for each number k from 1 to 991 congruent to 1 mod 10, both algorithms were run to keep k of the leaves. The mean and standard errors are shown here.
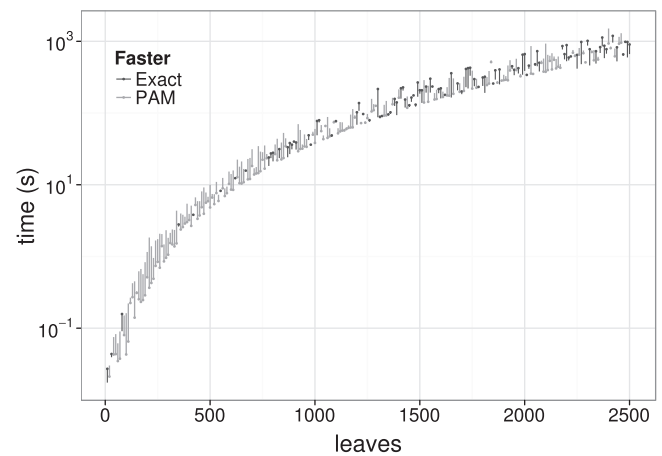

FIGURE 7.   Comparison of the time required to run the exact algorithm versus PAM with respect to the number of leaves in the original tree. Trees were generated with 10 to 2500 leaves in increments of 10 leaves; each tree was pruned to half of the original number of leaves. The result of running each algorithm on a given tree is shown here as a single line with x-position equal to the number of leaves of that tree, whereas the two y-positions of the line show the times taken by the two algorithms; when the exact algorithm was faster the line is black and when PAM was faster the line is gray. A point on each line shows the time for the PAM algorithm.

Enterobacteriaceae, and identified chimeric sequences using UCHIME (Edgar et al. 2011). We built a tree of consisting of these sequences and other sequences from the same species from the RDP database release 10, update 28 (Cole et al. 2009). Five sequences were chosen for each species (when such a set existed). Remaining RDP sequences from these species were then placed on this tree using pplacer (Matsen et al. 2010) and the full algorithm was used to pick some fraction x of the reference sequences. In this case, using the full algorithm to minimize ADCL was less likely to choose chimeric sequences than choosing at random, whereas PD maximization was more likely to choose chimeric sequences (Fig. 8). The ADCL for the full algorithm was substantially lower than for either PD maximization or random subset selection, as would be expected given that ADCL is explicitly minimized in our algorithms.

We are not proposing this algorithm as a new way to find sequences of poor quality, rather, it is a way of picking sequences that are representative of the local diversity in the tree. The chimera work above was to make the point that artifactual sequences clearly not representative of actual diversity do not get chosen, while they do using the PD criterion. We also note that because bootstrap resampling can change branch length and tree topology, the minimum ADCL set is not guaranteed to be stable under bootstrapping. However, in the cases we have evaluated, the minimum ADCL value itself is relatively stable to bootstrap resampling when k is not too small (Supplementary Fig. S1, doi:10.5061/dryad.md023/6).

DISCUSSION

In this article, we described a simple new criterion, minimizing the ADCL, for finding a subset of sequences that either represent the diversity of the sequences in a sample or are close on average to a set of query sequences. In doing so, abundance information is taken into account in an attempt to strike a balance between optimality and centrality in the tree. In particular, this criterion is the only way of which we are aware to pick sequences that are phylogenetically close on average to a set of query sequences. We have also investigated means of minimizing the ADCL, including a heuristic that performs well in practice and an exact dynamic program. ADCL minimization appears to avoid picking chimeric sequences.
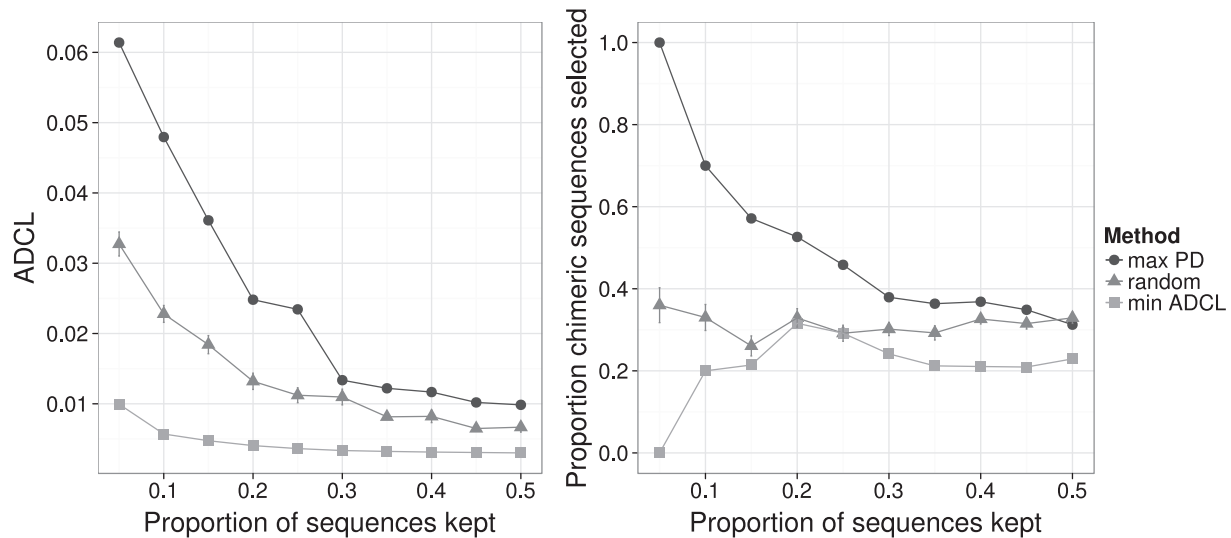
FIGURE 8. ADCL values and proportion of chimeric sequences kept for random selection, PD maximization, and ADCL minimization run on a set of Enterobacteriaceae 16s sequences along with chimeras from the same family identified with UCHIME (Edgar et al. 2011).

The current implementations are useful for moderate-size trees; improved algorithms will be needed for large-scale use (Fig. 7). We have found present algorithms to be quite useful in a pipeline that clusters query sequences by pairwise distance first, then retrieves a collection of potential reference sequences per clustered group of query sequences, then uses the ADCL criterion for selecting potential reference sequences among those (manuscript under preparation). This has the advantage of keeping the number of input sequences within a manageable range, as well as ensuring that the number of reference sequences is comprehensive across the tree.

The computational complexity class of the ADCL optimization problem is not yet clear.

Because of the special geometric structure of the problem, there is almost certainly room for improvement in the algorithms used to optimize ADCL. Only a subset of the possible exchanges need to be tried in each step of the PAM algorithm, and more intelligent means could be used for deciding which mass needs to be reassigned, similar to the methods of Zhang and Couloigner (2005). A better understanding of situations such as those illustrated by Figure 4 could lead to an understanding of when PAM becomes stuck in local optima. The geometric structure of the optimality intervals could be better leveraged for a more efficient exact algorithm. The PAM algorithm may also reach a near-optimal solution quickly, then use substantial time making minimal improvements to converge to the minimum ADCL (Supplementary Fig. S3). If an approximate solution is acceptable, alternate stopping criteria could be used.

In future work, we also plan on investigating the question of what k is appropriate to use for a given phylogenetic tree given certain desirable characteristics of the cut-down set. We note that using the exact algorithm makes it easy to find a k that corresponds to an upper bound for ADCL, but the choice of an appropriate upper bound depends on the application and priorities of the user. For example, in taxonomic assignment, some may require subspecies-level precision, whereas others require assignments only at the genus or higher level; the needs for ADCL would be different in these different cases.

SUPPLEMENTARY MATERIAL

FUNDING

ACKNOWLEDGMENTS

APPENDIX: A MORE COMPLETE DESCRIPTION OF ADCL MINIMIZATION

The distance $d(\cdot,\cdot)$ between points on a tree is defined to be the length of the shortest path between those points. A *rooted subtree* is a subtree that can be obtained from a rooted tree T by removing an edge of T and taking the component that does not contain the original root of T. The *proximal* direction in a rooted tree means toward the root, whereas the *distal* direction means away from the root. We emphasize that the phylogenetic trees here are considered as collections of points with distances between them, that is metric spaces, such that by a subset of a phylogenetic tree we mean a subset of those points.

Introduction to ADCL

Definition 1 A *mass map* on a tree T is a Borel measure on T. A *mass distribution* on a tree T is a Borel probability measure on T.

Definition 2 Given a subset $X \subset L(T)$ of leaves and a mass distribution μ, define the ADCL to be the expected distance of a random point distributed according to μ to its closest leaf in X. That is,

$$\mathrm{ADCL}_\mu(X) = \mathbb{E}\left[\min_{\ell \in X} d(P,\ell)\right],$$

where $P \sim \mu$.

Problem 7. Minimize ADCL for a given number of allowed leaves. That is, given $0 < k \leq |L(T)|$ and probability measure μ, find the $X \subset L(T)$ with $|X| = k$ minimizing $\mathrm{ADCL}_\mu(X)$.

The expected distance of a randomly sampled point $P \sim \mu$ to a fixed point ℓ is equal to the amount of work required to move mass distributed according μ to the point ℓ, when work is defined as mass times distance.

Voronoi Regions

In this section, we connect the above description with the geometric concept of a Voronoi diagram.

Definition 3 Given a subset $X \subset L(T)$ of leaves and $\ell \in X$, the *Voronoi region* $V(\ell, X)$ for leaf ℓ is the set of points of T such that the distance to ℓ is less than or equal to the distance to any other leaf in X. The *Voronoi diagram* for a leaf set X in the tree is the collection of Voronoi regions for the leaves in X.

Note that the Voronoi regions by this definition are closed sets that are not disjoint; they intersect each other in discrete points where the distances to leaves are equal.

Definition 4 Given a subset Z of T, a mass distribution μ and a leaf ℓ, let $\delta_\mu(Z, \ell)$ be the work needed to move the mass of μ in Z to the leaf ℓ.

The following simple lemma allows us to express the ADCL in terms of the Voronoi regions.

Lemma 8. Let μ be a mass distribution. Then

$$\mathrm{ADCL}_\mu(X) = \sum_{\ell \in X} \delta_\mu(V(\ell, X), \ell). \qquad (1)$$

Proof. For a given $p \in T$,

$$\min_{\ell \in X} d(p,\ell) = \sum_{\ell \in X} 1_{V(\ell,X)}(p) d(p,\ell),$$

where $1_{V(\ell,X)}$ is the indicator function for the set $V(\ell, X)$. Let $P \sim \mu$. Then

$$\mathbb{E}\left[\min_{\ell \in X} d(P,\ell)\right] = \int_{p \in T} \min_{\ell \in X} d(p,\ell) d\mu$$

$$= \sum_{\ell \in X} \int_{p \in V(\ell, X)} d(p,\ell) d\mu$$

$$= \sum_{\ell \in X} \delta_\mu(V(\ell, X), \ell)$$

where the last step is by the optimization definition of the KR distance (Evans and Matsen, 2012). ∎

Dynamic Program

Background.—This section presents a full solution to Problem 7 via a dynamic program. This dynamic program will descend through the tree selecting each rooted subtree S in a depth-first manner, and solving the optimization for every amount and direction of mass transport through the root of S. Because the algorithm constructs every solution that is not suboptimal, the algorithm is exact.

Bubbles.—A first observation for the exact algorithm is that the tree can be divided into connected sets, that we call *bubbles*, with the following property: irrespective of the set of leaves X of the tree T that are selected, any pair of points in the same bubble will share the same closest leaf in X. Because of this characteristic, all that is needed to decide the fate of every particle of mass is to decide optimal mass transport on a per-bubble basis. Indeed, if it is optimal to move mass at point p to a leaf ℓ, then the same must be true for every other point q in in the bubble. This observation turns the search for an optimal subset into an optimal assignment of bubbles to leaves of the tree, such that the total number of leaves that are assigned a bubble has cardinality at most k. As described below, the partition of the tree into bubbles is in fact the refinement of all possible Voronoi diagrams for all subsets of the leaves (along with the partition by edges, which is put in for convenience).

Recall that partitions of a set are partially ordered by inclusion, such that $\mathcal{A} \leq \mathcal{A}'$ for two partitions $\mathcal{A}$ and $\mathcal{A}'$ iff every $V \in \mathcal{A}$ is contained in a $V' \in \mathcal{A}'$. Partitions are a complete lattice with this partial order, thus there exists a greatest lower bound for any collection of partitions; define $\mathcal{A} \wedge \mathcal{B}$ be the greatest lower bound for

any partitions $\mathcal{A}$ and $\mathcal{B}$. In practice, this means finding the "coarsest" partition such that pairwise intersections of sets in the partitions are represented: for example,

$$\{A, X \setminus A\} \wedge \{B, X \setminus B\} = \{A \cap B, A \setminus B, B \setminus A, X \setminus (A \cup B)\}.$$

We will be interested in partitions of phylogenetic trees, and the boundaries of partitions can be thought of as "cuts" on edges or internal nodes. Thus if $\mathcal{A}$ and $\mathcal{B}$ are two partitions of a tree T, $\mathcal{A} \wedge \mathcal{B}$ is the partition with every "cut."

Let $\mathcal{V}(X)$ be the Voronoi diagram of T for some subset X of the leaves of T. Let $\mathcal{E}$ be the partition of T such that the edges of T are the sets of the partition.

Definition 5 The *bubble partition* of a tree T is the coarsest partition refining all of the Voronoi decompositions of T and the edge partition:

$$\mathcal{B}(T) := \mathcal{E} \wedge \bigwedge_{X \subset L(T)} \mathcal{V}(X).$$

This partition forms the basis of our approach. By Lemma 8, an exhaustive approach to Problem 7 would involve trying all Voronoi diagrams for a given tree and transporting the mass in each of the regions to the closest leaf. However, $\mathcal{B}(T)$ is the refinement of all Voronoi partitions. Because every point in a bubble has the same closest leaf, every optimal solution can be completely described by deciding to what leaf the mass in B gets sent for each $B \in \mathcal{B}(T)$. The number of bubbles is quadratic in the number of leaves irrespective of the given mass distribution.

In particular, by describing the optimization algorithm in terms of bubbles, it will work in the case of a continuous mass distribution. What is needed is a way to calculate the amount of work needed to move a continuous mass distribution to one side of a bubble. This can be done using a simple integral as described in (Evans and Matsen, 2012), however, the above-described rppr implementation is in terms of a discrete distribution of mass.

Recursion introduction.—In this section, we will describe a recursion that will solve Problem 7 as described above. Fix the number of allowed leaves k. The recursion is depth-first starting at a root, which can be arbitrarily assigned if the tree is not already rooted. *Partial solutions* start at the leaves, are modified and reproduce as they travel through bubbles, then get combined at internal nodes. We remind the reader that a solution to Problem 7 is completely specified by the destination of the mass for each bubble.

These partial solutions will be denoted by labeled tuples, either $\text{RMD}(X, \chi, \omega)$ for a root mass distal solution or $\text{RMP}(X, \chi, \pi, \omega)$ for a root mass proximal solution as follows. The X component of the partial solution is the *leaf set*: the leaves that have been selected for the partial solution to the ADCL problem. The χ component is the *closest leaf distance*: the distance to the closest leaf in the partial solution. The π component is the

proximal mass: the mass that is moved to the root of S by this partial solution (RMD solutions always have zero proximal mass). The ω component is the *work subtotal*: the amount of work needed for the partial solution to move the mass in S to either the root of S or a leaf in X. Whether a partial solution is an RMP or an RMD solution defines its *mass direction class*.

The depth-first recursion will maintain a list of partial solutions that gets updated upon traversing bubbles and internal nodes. We remind the reader that bubbles never span more than a single edge.

Base case at a leaf.—There are two base cases at a leaf: that of not including the leaf and that of including the leaf. The partial solution that corresponds to including the leaf ℓ is $\text{RMD}(\{\ell\}, 0, 0)$ and that of not including the leaf is $\text{RMP}(\emptyset, \infty, 0, 0)$. From there, we move proximally through the bubbles along the edges and through the internal nodes as follows. Note that these partial solutions at each leaf are then passed through the bubbles directly proximal to their leaf.

Moving through a bubble.—Assume the algorithm is traversing a bubble along an edge, such that the edge length in the bubble is λ and the amount of mass in the bubble is μ. Define α to be the amount of work required to move the mass in the bubble to the distal side of the bubble, and β to be the corresponding amount of work for the proximal side. We now describe the steps required to update this collection of partial solutions going from the distal to the proximal side of the bubble.

The first step is to update the existing partial solutions. In this updating step, the mass-direction class will not be changed (in the second step we will construct RMP solutions from RMD solutions). RMD solutions get updated as follows:

$$\text{RMD}(X, \chi, \omega)$$

maps to

$$\text{RMD}(X, \chi + \lambda, \omega + \alpha + \mu \cdot \chi)$$

as RMD solutions must move the mass of the current bubble to a leaf, and moving it to the closest selected leaf is optimal.

On the other hand, RMP solutions will have the mass of the current bubble moving away from the leaves of S. Thus

$$\text{RMP}(X, \chi, \pi, \omega)$$

maps to

$$\text{RMP}(X, \chi + \lambda, \pi + \mu, \omega + \beta + \pi \cdot \lambda).$$

The second step is to consider solutions such that the mass transport on the distal and proximal sides of the bubble are not the same. In that case, the optimal directions of mass movement on distal and proximal edges for a given bubble must be pointing away from each other; the alternative could not be optimal. Thus here we consider adding an RMP solution based on a previous RMD solution. This solution has all of the mass

going to the leaves in S except that of the current bubble, which moves proximally. This step can be ignored if $\mu = 0$. Given a RMD(X, χ, ω) with $X \neq \emptyset$, add the resulting

$$\text{RMP}(X, \chi + \lambda, \mu, \omega + \beta)$$

to the list of possible solutions if $\beta < \alpha + \mu \cdot \chi$.

The following simple lemma reduces the number of bubbles that must be considered.

Lemma 9. Given two neighboring bubbles on a single edge, such that the proximal bubble has zero mass. The recursive step in this section after progressing through these two bubbles is identical to one where the two bubbles are merged. ∎

Moving proximally through an internal node.—When encountering an internal node, the algorithm first combines all tuples of partial solutions for each subtree as follows. Assume we are given one φ_i from each of the subtrees, where X_i, χ_i, π_i, and ω_i are as above for the i-th tree (define $\pi_i = 0$ for RMD solutions).

At least one, and possibly two, partial solutions can be constructed from the partial solutions in the subtrees. There is always one solution where the proximal mass, if it exists, continues moving away from the leaves of S; we will call this the "continuing" solution. Sometimes it is also possible for the proximal mass to go to a leaf in one of the subtrees, giving another solution we will call the "absorbing" solution.

The continuing solution in the case that all solutions are RMD solutions is

$$\text{RMD}\left(\bigcup_i X_i, \min_i \chi_i, \sum_i \omega_i\right).$$

If any of the solutions are RMP solutions, then the resulting solution is

$$\text{RMP}\left(\bigcup_i X_i, \min_i \chi_i, \sum_i \pi_i, \sum_i \omega_i\right)$$

where $\pi_i = 0$ for RMD solutions.

Now, if at least one leaf is selected in one of the subtrees, then it could be optimal to move the proximal mass to the closest leaf of the existing RMD solutions to make an absorbing solution; in that case we also have the new RMD solution

$$\text{RMD}\left(\bigcup_i X_i, \min_i \chi_i, \sum_i \omega_i + \left(\min_i \chi_i\right) \cdot \sum_i \pi_i\right).$$

Given an internal node with k subtrees, combine all partial solutions of the subtrees in this way. The complete collection of solutions given a set of ψ_i for each subtree is the union of this process applied to every element of the Cartesian product of the ψ_i. Throw away any partial solutions such that the cardinality of the X is greater than k.

Termination at the root.—Select the RMP solution RMP(X, χ, ω) with $|X| = k$ with the smallest ω. Note that in fact all solutions for number of chosen leaves less than k are generated by this algorithm.

Avoiding computation of suboptimal solutions.—By naïvely combining all of the solutions described above we may get solutions that cannot be optimal for any structure of the rest of the tree. These solutions greatly reduce the speed of the algorithm when carried along as described above. In this section, we describe a way to avoid these solutions using geometry (Fig. 4).

The culling strategy employed here is to eliminate partial solutions that would not be optimal for any amount and direction of mass transport through the root of S. This is achieved by first binning the partial solutions by the number of leaves k they employ then further binning them by mass-direction class.

For an RMP solution, the amount of work required in a given partial solution to move the mass in S to some selected leaf is equal to the mass transport within S plus the amount of work required to move the proximal mass of that partial solution to some leaf proximal to S. Imagine that this proximal leaf is distance x away from the root of S. For a given partial solution, we can plot the amount of work to move the mass in S to some selected leaf with respect to x. This will be $\omega + x \cdot \pi$.

Similar logic applies for RMD solutions but where the appropriate parameter x to use is the amount of mass that comes proximally through ρ. The total amount of work in that case then, is $\omega + x \cdot \chi$.

These considerations motivate the following definition.

Definition 6 The *subwork* f_φ for a partial solution φ is the function

$$f_\varphi(x) = \begin{cases} \omega + x \cdot \pi & \text{if } \varphi = \text{RMP}(X, \chi, \pi, \omega) \\ \omega + x \cdot \chi & \text{if } \varphi = \text{RMD}(X, \chi, \omega) \end{cases}$$

Let the x in $f_\varphi(x)$ be called the *subwork parameter*.

Definition 7 Assume ψ is a set of partial solutions with the same number of leaves and the same root mass direction. The *optimality interval* $I_\psi(\varphi)$ is the interval for which φ is optimal compared with the other solutions in ψ, namely

$$I_\psi(\varphi) = \left\{x \in [0, \infty) : f_\varphi(x) \leq f_{\varphi'}(x) \forall \varphi' \in \psi\right\}.$$

It can be easily seen that the set defined in this way is actually an interval. We can ignore partial solutions that have an empty optimality interval.

These optimality intervals can be found by using the double description algorithm as described in the algorithm introduction. Specifically, a line with equation $y = mx + b$ will get translated into the half-plane constraint $y \leq mx + b$. The set of points in the plane that satisfy this collection of inequalities is called a *convex polytope*. A convex polytope can be equivalently described as the intersection of a number of half-spaces (a so-called *H-description*) or as the convex hull of a

number of points (a so-called *V-description*). In this setting, the set of x values between vertices are the optimality intervals, and the lines that contact pairs of neighboring vertices correspond to partial solutions that are optimal for some value of x (Fig. 4).

The "double description" algorithm (Motzkin et al. 1983; Fukuda and Prodon 1996) is an efficient way to go from an H-description to a V-description which also collects information on what linear constraints contact what facets. One way to use this perspective on optimal solutions is to simply throw away partial solutions that have empty optimality intervals after combining. Another is to use optimality intervals to guide the combination of solutions in at internal nodes as described in the next section.

We perform a prefiltering step before running the double description algorithm to discard lines that could never be facets. Denote lines as being pairs of (m,b), where m is a slope and b is a y-intercept. Clearly if $m_1 < m_2$ and $b_1 < b_2$ then the line (m_1,b_1) will lie below (m_2,b_2) for all positive x. Therefore (m_2,b_2) will never be a facet and can be ignored. We quickly eliminate some of these clearly suboptimal partial solutions by sorting the (m,b) pairs in terms of increasing m. If in this ordering, the line (m_1,b_1) precedes (m_2,b_2) with $b_2 \geq b_1$ then we discard (m_2,b_2).

We can also include some global information as inequalities. For RMP solutions, the subwork parameter is bounded between the closest and farthest leaves in the proximal part of the tree. For RMD solutions, the subwork parameter is bounded between zero and the total amount of mass in the proximal part of the tree. These additional constraints further cut down optimality intervals and reduce the number of solutions.

Optimality intervals and solution combination.—Here we explore ways of using optimality intervals to reduce the number of partial solutions that must be combined between subtrees. We will describe the partial solutions combination in terms of combining pairs of subtrees. If a given internal node has more than two subtrees, say $T_1,\ldots,T_k$, then we can combine over partial solutions for T_1 and T_2, then combine those results with T_3, and so on.

Assume we are given two partial solutions φ_1 and φ_2, and these have two optimality intervals $I_1 = (l_1,u_1)$ and $I_2 = (l_2,u_2)$, respectively. The criteria used to check if a given combination could be optimal depend on the mass-direction class, and we will describe the criteria on a case-by-case basis. Let φ be the solution that is formed from the combination of φ_1 and φ_2, and denote its optimality interval with I.

If φ_1 and φ_2 are both RMP solutions, the subwork parameter for each will be the distance to the closest leaf in the proximal part of the tree. Since the combination will also be an RMP solution, to be optimal for a subwork parameter x each partial solution will have to be optimal for that x. Thus $I = I_1 \cap I_2$, and φ is viable if $I \neq \emptyset$.

If φ_1 and φ_2 are both RMD solutions, the subwork parameter is the amount of mass that comes proximally through the root of S. When mass comes proximally through the root, it will go to the subtree that has the closest leaf. For this reason, I will be the optimality interval of the partial solution with the smallest χ. If the subtrees have identical χ's, then I is the smallest interval containing I_1 and I_2.

Recall that when one partial solution is an RMP solution and the other is an RMD solution, then we can get either an RMP or an RMD solution. Assume first φ_1 is an RMP solution, φ_2 is an RMD solution, and φ is an RMP solution. Since φ is an RMP solution, no mass will be sent into T_2 from outside of it. Thus, φ should only be used if φ_2 has the smallest ω across all partial solutions that have the same number of leaves as φ_2. Also, the solution could be optimal only if χ_2 is greater than the upper bound of I_1, because otherwise it would be optimal to send the proximal mass of T_1 proximally into T_2. If φ_1 is an RMP solution, φ_2 is an RMD solution, and φ is an RMD solution, the subwork parameter for φ is the amount of mass coming from above. Because the proximal mass of φ_1 will be sent into T_2, the optimality interval I is $I_2 \cap [\pi_1,\infty)$. The corresponding partial solution is valid if I is nonempty and $\chi_2 \in I_1$.

References

Berger S.A., Krompass D., Stamatakis A. 2011. Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood. Syst. Biol. 60:291–302.

Bordewich M., Rodrigo A.G., Semple C. 2008. Selecting taxa to save or sequence: desirable criteria and a greedy solution. Syst. Biol. 57:825–834.

Cole J.R., Wang Q., Cardenas E., Fish J., Chai B., Farris R.J., Kulam-Syed-Mohideen A.S., McGarrell D.M., Marsh T., Garrity G.M. et al. 2009. The Ribosomal Database Project: improved alignments and new tools for rRNA analysis. Nucleic Acids Res. 37(suppl 1):D141–D145.

Edgar R.C., Haas B.J., Clemente J.C., Quince C., Knight R. 2011. Uchime improves sensitivity and speed of chimera detection. Bioinformatics, 27:2194–2200.

Eisen J.A., Sweder K.S., Hanawalt P.C. 1995. Evolution of the SNF2 family of proteins: subfamilies with distinct sequences and functions. Nucleic Acids Res. 23:2715–2723.

Eisen J.A. 1998. Phylogenomics: improving functional predictions for uncharacterized genes by evolutionary analysis. Genome Res. 8:163–167.

Engelhardt B.E., Jordan M.I., Muratore K.E., Brenner S.E. 2005. Protein molecular function prediction by Bayesian phylogenomics. PLOS Comp. Biol. 1:e45.

Evans S.N., Matsen F.A. 2012. The phylogenetic Kantorovich-Rubinstein metric for environmental sequence samples. J. Royal Stat. Soc. (B). 74:569–592.

Faith D.P. 1992. Conservation evaluation and phylogenetic diversity. Biol. Conserv. 61:1–10.

Fukuda K., Prodon A. 1996. Double description method revisited. In Deza, M., Euler, R., Manoussakis, I., editors. Combinatorics and Computer Science, Vol. 1120. 91–111.

Fukuda, K. 2012. cddlib library. Available from: http://www.cs.mcgill. ca/~fukuda/soft/cdd_home/cdd.html, version 094f.

Holland B. 2001. Evolutionary analyses of large datasets: trees and beyond. [PhD thesis], Massey University. Available from http://hdl.handle.net/10179/2078.

Mahajan M., Nimbhorkar P. Varadarajan, K. 2009. The planar k-means problem is NP-hard. In WALCOM: Algorithms and Computation. Chicago: Springer. 274–285.

Matsen, F.A., Kodner, R.B., Armbrust, E.V. 2010. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. BMC bioinfo. 11:538.

Minh B.Q., Klaere S., Von Haeseler A. 2006. Phylogenetic diversity within seconds. Syst. Biol. 55:769–773.

Motzkin T.S. Ralffa H., Thompson G.L., Thrall R.M. 1983. The double description method. In: Cantor, D., Gordon, B., Rothschild, B., Boston, B., editors. p. 81.

Pardi F. and Goldman N. 2005. Species choice for comparative genomics: Being greedy works. PLOS Genet. 1:e71.

Piantadosi A., Chohan B., Chohan V., McClelland R.S., Overbaugh, J. 2007. Chronic HIV-1 infection frequently fails to protect against superinfection. PLOS Pathogens, 3:e177.

Price M.N., Dehal P.S., Arkin, A.P. 2010. Fasttree 2–approximately maximum-likelihood trees for large alignments. PLOS ONE, 5:e9490.

Redd A.D., Collinson-Streng A., Martens C., Ricklefs S., Mullis C.E.. Manucci J., Tobian A.A.R., Selig E.J., Laeyendecker O., Sewankambo, N. et al. 2011. Identification of HIV superinfection in seroconcordant couples in Rakai, Uganda, by use of next-generation deep sequencing. J. Clin. Microbiol. 49:2859–2867.

Steel M. 2005. Phylogenetic diversity and the greedy algorithm. Syst. Biol. 54:527–529.

Theodoridis S., Koutroumbas K. 2006. Pattern recognition. 3rd ed. San Diego: Academic Press.

Villani. C. 2003. Topics in optimal transportation, Vol. 58. Providence: American Mathematical Society.

Wu D., Hugenholtz P., Mavromatis K., Pukall R., Dalin E., Ivanova N.N., Kunin V., Goodwin L., Wu M., Tindall B.J. et al. 2009. A phylogeny-driven genomic encyclopaedia of Bacteria and Archaea. Nature. 462:1056–1060.

Yule G.U. 1925. A mathematical theory of evolution, based on the conclusions of Dr. JC Willis, FRS. Phil. Tran. Royal Soc. London (B). 213:21–87.

Zhang Q., Couloigner I. 2005. A new and efficient k-medoid algorithm for spatial clustering. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. editors. Comp. Sci. and Its Appl.–ICCSA 2005. Berlin Heidelberg: Springer. p. 207–224.

Ziegler G.M. 1995. Lectures on polytopes, vol. 152 of Graduate Texts in Mathematics. New York: Springer.