JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 00, Number 00, 2024

Pp. 000–000
DOI: 10.1089/cmb.2023.0382

Pairwise Distances and the Problem of Multiple Optima

RAN LIBESKIND-HADAS

ABSTRACT

Discrete optimization problems arise in many biological contexts and, in many cases, we seek to make inferences from the optimal solutions. However, the number of optimal solutions is frequently very large and making inferences from any single solution may result in conclusions that are not supported by other optimal solutions. We describe a general approach for efficiently (polynomial time) and exactly (without sampling) computing statistics on the space of optimal solutions. These statistics provide insights into the space of optimal solutions that can be used to support the use of a single optimum (e.g., when the optimal solutions are similar) or justify the need for selecting multiple optima (e.g., when the solution space is large and diverse) from which to make inferences. We demonstrate this approach on two well-known problems and identify the properties of these problems that make them amenable to this method.

Keywords: dynamic programming, multiple optima.

1. INTRODUCTION

Many problems in the life sciences are formulated as discrete optimization problems and the solutions to those problems are often the bases of biological inferences. For example, the evolution of phenotypic traits may be inferred from solutions to the small parsimony problem that takes as input a phylogenetic tree whose leaves (extant taxa) have associated labels (e.g., traits) and seeks a labeling of the internal nodes that minimizes the total number of differences between the labels on parent and child nodes (Fitch, 1971).

As other examples, the evolutionary histories of molecular sequences are inferred from solutions to the global sequence alignment problem (Needleman and Wunsch, 1970), the secondary structures of RNA are inferred from solutions to the RNA folding problem Eddy (2004), and the co-evolutionary histories of hosts and parasites are inferred from solutions to the phylogenetic reconciliation problem (Bansal et al., 2012).

These—and many other—bioinformatic optimization problems can be solved efficiently with dynamic programming (DP) algorithms. However, the number of optimal solutions to these optimization problems can be exponentially large in the size of the problem instance, presenting challenges in interpreting solutions. Any one solution may not be representative of the entire solution space and making inferences from a single solution may lead to incomplete, or even erroneous, conclusions.

To illustrate this challenge, we generated 100 random binary phylogenetic trees, each with 100 leaves labeled randomly from a set of 20 distinct characters. The number of optimal solutions to the small parsimony problem

Kravis Department of Integrated Sciences, Claremont McKenna College, Claremont, California, USA.

ranged from 7×10^9 to over 3×10^{25}. Without information about the diversity of this solution space, making inferences about ancestral states from any one optimal solution may significantly misrepresent other equally plausible scenarios and result in incorrect conclusions.

In a second experiment involving 100 randomly generated pairs of strings of length 1000 over an alphabet of size 4, the number of optimal solutions to the edit distance problem (a simple version of a family of pairwise sequence alignment problems) ranged from 5×10^{47} to 5×10^{65}. The same pattern has been reported with real data. For example, for RNA folding using the Nussinov Algorithm (Nussinov et al., 1978), Kiirala et al. (2019) found that on 117 23S rRNA sequences with average length 2726, the average number of optimal solutions was $\sim 10^{130}$. For the phylogenetic tree reconciliation problem in the Duplication-Transfer-Loss (DTL) model, Bansal et al., 2013 found that in a Tree of Life data set with 100 species and 4849 gene trees, >15% of solutions had between 10^4 and 10^{39} optimal solutions.

A number of approaches have been suggested for dealing with the large optimal solution spaces arising in such bioinformatic optimization problems (Bansal et al., 2013; Heitsch and Poznanovik, 2014; Huber et al., 2018; Kiirala et al., 2019; Liu et al., 2022; Miklós and Darling, 2009; Mikls et al., 2014; Rogers and Heitsch, 2016; Salmela and Tomescu, 2018; Santichaivekin et al., 2019; Vingron and Argos, 1990).

One approach is to find parts of solutions that are shared by all optimal solutions for the given problem instance. Such partial solutions are called *safe*. For example, algorithms for finding all safe partial solutions have been developed for the protein sequence alignment problem (Vingron and Argos, 1990), the gap filling problem (Salmela and Tomescu, 2018), the contig assembly problem (Tomescu and Medvedev, 2017), and the RNA folding problem (Kiirala et al., 2019).

Another approach for addressing the large space of optimal solutions is to find a single "best" representative solution. For the phylogenetic tree reconciliation problem, Nguyen et al. (2013) provided a technique for efficiently computing a median solution, an optimal solution that minimizes the maximum distance to all other optimal solutions with respect to a given distance metric between solutions. One challenge with this approach is that the median is not unique and, in fact, there can be an exponential number of medians. Thus, any single median may still not adequately represent the entire solution space. Yet another useful approach is to partition the large solution space into a much smaller number of equivalence classes (Wang et al., 2023).

Recently, several efforts have been made to efficiently compute statistics and distributions that characterize the potentially large space of optimal solutions. In particular, Huber et al. (2018) proposed computing the *diameter* of the space of optimal solutions for the phylogentic reconciliation problem. The diameter is defined to be the maximum distance between all pairs of optimal solutions with respect to a distance metric. For example, if the solution space is large but the diameter is small, then a single optimal solution may adequately represent the solution space. The diameter allows us to report, for example, that all optimal solutions differ by no more than a specified amount from the single selected optimal solution.

More generally, Huber et al. proposed computing the distribution of distances between all pairs of optimal solutions. The distribution of pairwise distances provides insights into the solution space and allows us to compute summary statistics in addition to the diameter (which is the maximum pairwise distance). These summary statistics may help support reporting a single solution (e.g., if the mean pairwise distance and standard deviation are relatively small) or indicate a need for multiple representative optimal solutions. In particular, if the distribution of pairwise distances is multimodal, we may conclude that there are clusters of optimal solutions and seek to find representative solutions from each cluster (Mawhorter and Libeskind-Hadas, 2019).

Since the optimal solution space can be exponentially large, Huber et al. proposed approximating the set of pairwise distances using a random sample of the solution space (Huber et al., 2018). Subsequently, Haack et al. (2018) and Santichaivekin et al. (2019) showed that, for the tree reconciliation problem, both the diameter and the distribution of pairwise distances can be computed *exactly* (i.e., without sampling) in polynomial time.

In this article, we demonstrate a broadly applicable approach to computing diameters and pairwise distances exactly (without sampling) and efficiently (polynomial time). This generalizes the results of Haack et al. (2018) and Santichaivekin et al. (2019) to other problems and demonstrates the methods without the technically complicated details that are particular to the tree reconciliation problem.

In summary, the contributions of this article are as follows:

1. A systematic approach for computing diameters, demonstrated using two well-known bioinformatic problems.
2. An extension of that method to compute pairwise distance vectors.
3. Criteria for which these methods can be applied.

The remainder of this article is organized as follows: In Section 2, we define key concepts, including the distance metric under consideration, diameter, and pairwise distances. In Section 3, we demonstrate the framework for computing the diameter and pairwise distances using the edit distance problem as an example. In Section 4, we demonstrate the framework for more complex problems, using the small parsimony problem as an example.

2. DIAMETER AND PAIRWISE DISTANCES BY EXAMPLE

In this section, we define diameter and pairwise distance and demonstrate these concepts with two examples, the edit distance problem and the small parsimony problem in phylogenetic trees. In the next two sections, we show how to efficiently compute these diameters and pairwise distances for those two problems and describe the conditions that allow this approach to be applied to other problems.

2.1. Diameter

The diameter of a finite space is defined as the maximum distance between all pairs of points in that space with respect to a given distance metric. In this article, we use the *symmetric difference distance* metric because of its versatility (Agius et al., 2010; Liu et al., 2022; Nguyen et al., 2013; Santichaivekin et al., 2019). The symmetric difference distance between two sets is the number of elements that are found in one of the two sets but not in both sets. We note, however, that what constitutes the set of elements in a solution is problem-dependent. The results described here may also extend to other distance metrics.

As our first example, we consider the edit distance problem (Levenshtein, 1966) that seeks to find the minimum number of insertion, deletion, and substitution events that are required to transform one string S_1 to a second string S_2. We adopt the convention of indexing strings beginning with index 1 and use the notation $S[i]$ to indicate the symbol in string S at index i.

Consider $S_1 =$ "AT" and $S_2 =$ "TA". The edit distance between these two strings is 2. One optimal solution uses a substitution to replace the "T" at $S_1[2]$ with an "A" and a second substitution to replace the "A" at $S_1[1]$ with a "T." This solution has a corresponding alignment shown in Figure 1a. Another optimal solution inserts an "A" at the end of S_1 and deletes the "A" at $S_1[1]$, also requiring two operations. This solution gives rise to the alignment in Figure 1b. Finally, a third optimal solution deletes the "T" at $S_1[2]$ and inserts a "T" at the front of S_1, again requiring two operations and giving rise to the alignment shown in Figure 1c. Two symbols in the same column in the alignment are said to be *matched*. Note that in these alignments, a substitution is indicated by differing matched symbols. An insertion in S_1 is indicated by a gap character in S_1 and a deletion in S_1 is indicated by a gap character in S_2.

The distance between two solutions is given by the symmetric difference distance between their alignments. The first two solutions, represented by the alignments in Figure 1a and b, differ at all indices and thus have distance 5. More precisely, in the solution in (Fig. 1a), the matching (due to substitution) of $S_1[1]$ and $S_2[1]$ is not found in solution (Fig. 1b), and thus contributes 1 to the distance. The matching (due to substitution) of $S_1[2]$ to $S_2[2]$ is also not found in (Fig. 1b) and also contributes 1 to the distance. Similarly, in (Fig. 1b), the matching of $S_1[1]$ to a gap character (due to deletion), the matching of $S_1[2]$ to $S_2[1]$, and the matching of a gap character at the end of S_1 to $S_2[2]$ are not found in (Fig. 1a), contributing a total of 3 to the distance. Similarly, the solutions in (Fig. 1a) and (Fig. 1c) differ by 5. However, the solutions in (Fig. 1b) and (Fig. 1c) differ by six and thus the diameter of this optimal solution space is 6.

In the instance of the small parsimony problem shown in Figure 2a with four leaves labeled "A," "A," "T," and "G," the objective is to find a labeling of the internal nodes that minimizes the number of differences between parent and child nodes. There are five optimal solutions, each with cost 2, shown in (Fig. 2b–f).

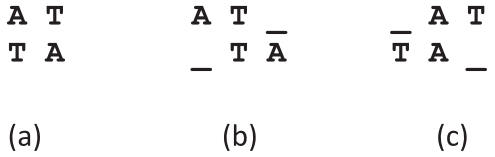

FIG. 1. The three optimal alignments for strings "AT" and "TA."

The distance between two solutions is the number of internal nodes with different labels. For example, in Figure 2, solutions (b) and (c) differ only at internal node 2 and the distance is, therefore, 1. Similarly, solutions (e) and (f) have distance 2 (since they differ at nodes 0 and 2). In this example of the small parsimony problem the diameter is 2 since the maximum distance between any two optimal solutions is 2.

As noted earlier, the size of the optimal solution space may be exponential in the size of the problem instance. Naively computing the diameter of the optimal solution space would require exponential time. However, as we show in this article, for many optimization problems, the diameter can be computed exactly in polynomial time by exploiting properties of the DP tables that are used to compute optimal solutions.

2.2. *Pairwise distances*

Although the diameter provides an insight into the space of optimal solutions, any single statistic is not fully informative. For example, the diameter of an optimal solution space may be large due to two outliers with large distance, whereas all of other optimal solutions may have small distances between them.

A more informative measure of the optimal solution space is the *pairwise distance vector* defined as a vector v such that v_i denotes the number of pairs of optimal solutions whose distance is exactly i. Note that v_0 is the total number of optimal solutions since for a pair of solutions (x, y) to have distance 0, it must be that $x = y$. In addition, if d is the maximum index such that $v_d > 0$, then d is the diameter of the space. Thus, the pairwise distance vector provides the size of the optimal solution space, its diameter, and a digest of the differences between all pairs of optimal solutions.

For example, for the edit distance problem with strings "AT" and "TA" (Fig. 1), the pairwise distance vector is $(3, 0, 0, 0, 0, 2, 1)$ indicating that there are three optimal solutions and thus $\binom{3}{2} = 3$ pairs of solutions, two pairs have distance 5 and one pair has distance 6, which is the diameter of this space. For the small parsimony problem example (Fig. 2) the pairwise distance vector is $(5,5,5)$. There are five solutions and thus $\binom{5}{2} = 10$ pairs of solutions. Five of those pairs have distance 1 and five of those pairs have distance 2, which is the diameter of this space.

2.3. *Overview of the general principle*

The edit distance problem and the small parsimony problem are two examples of problems that have the *optimal substructure property*, the property that a solution to the problem can be found from solutions to smaller

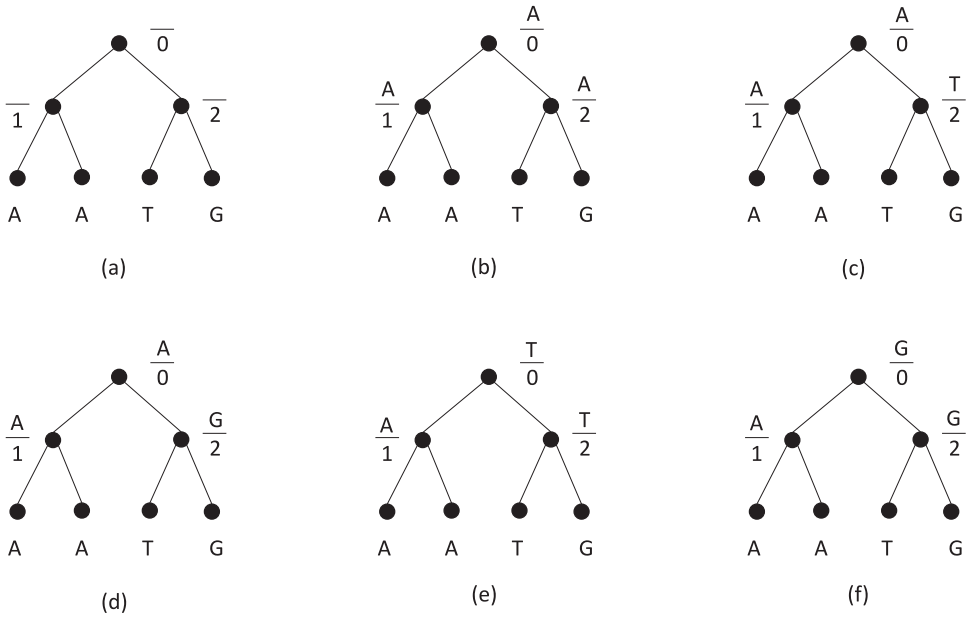

FIG. 2. **(a)** An instance of the small parsimony problem and **(b–f)** five optimal solutions.

subproblems of the same type. When the number of subproblems that are required is polynomially bounded, this optimal substructure property can be exploited to solve the problem with an efficient DP algorithm.

In the next two sections we use the edit distance and the small parsimony problems to first demonstrate how diameters can be computed by efficient algorithms and then extend this to computing the pairwise distance vectors. This approach is based on first constructing the DP table for the original problem. The DP table is annotated to record which choices give the minimum cost, giving rise to a graph where the vertices are the entries in the DP table and the edges correspond to the annotations. Finally, we apply a second DP algorithm on this graph to compute the diameter and, subsequently, the pairwise distance vector.

3. THE EDIT DISTANCE PROBLEM

Recall that the edit distance between two strings S_1 and S_2 is defined as the minimum number of insertions, deletions, and substitutions required to transform string S_1 to S_2. Each such sequence of operations has a corresponding alignment of the two strings. Other pairwise alignment problems have similar algorithmic solutions and the results presented here for edit distance extend naturally to those problems.

3.1. Computing the edit distance

For completeness, the recursive algorithm for computing the edit distance is given in pseudocode in Figure 3 (Levenshtein, 1966). In this implementation, the two strings are denoted S_1 and S_2. If S_1 has length m and S_2 has length n, then the two strings are indexed from 1 to m and 1 to n, respectively. The Edit Distance function, ED, takes the two strings S_1 and S_2 and indices $i \geq 0$ and $j \geq 0$ as input and returns the edit distance from the substring of S_1 indexed from 1 to i and the substring of S_2 indexed from 1 to j. Note that when i (or j) is zero, this substring is the empty string.

Since the worst-case running time of this recursive algorithm is exponential, it is implemented using DP by maintaining an $(m+1) \times (n+1)$ DP table in which cell (i,j) represents the value that would be computed by ED (S_1, S_2, i, j). The table is filled row-by-row beginning with row 0 $(i=0)$ by increasing value of j. Each cell is filled by using the recursion in the algorithm in Figure 3, but each recursive call now becomes a constant-time lookup in the corresponding cells in the table. Cell (m,n), therefore, contains the edit distance between the two strings and the algorithm has asymptotic time complexity $O(mn)$.

While filling in the DP table, each cell can be annotated to record which options give the minimum cost. When the table is filled, the annotations can be traced back from cell (m,n) to reconstruct optimal solutions. For example, the DP table for the strings S_1 = "AT" and S_2 = "TA" is shown in Figure 4.

The cells in the DP table correspond to vertices and the annotations correspond to directed edges. Thus, we use the terminology of cells and vertices interchangeably and basic graph theory definitions for convenience. The DP table is, therefore, a directed acyclic graph.

In addition, note that each path from cell (m,n) to cell $(0,0)$ corresponds to a distinct optimal solution to the edit distance problem and thus a distinct optimal alignment. For example, for the strings "AT" and "TA," the path $(2,2),(1,1),(0,0)$ in the table in Figure 4 represents two substitutions, which corresponds to the alignment in Figure 1a. The path $(2,2),(2,1),(1,0),(0,0)$ corresponds to inserting an "A" at the end of "AT,"

```
ED(S1, S2, i, j):
  if i == 0: return j
  elif j == 0: return i
  else:
    if S1[i] == S2[j]:  # Last symbols match
      return ED(S1, S2, i-1, j-1)
    else:
      option1 = 1 + ED(S1, S2, i-1, j-1) # substitution
      option2 = 1 + ED(S1, S2, i, j-1)   # insertion
      option2 = 1 + ED(S1, S2, i-1, j)   # deletion
      return min(option1, option2, option3)
```

FIG. 3. Recursive algorithm for computing the edit distance between two strings.

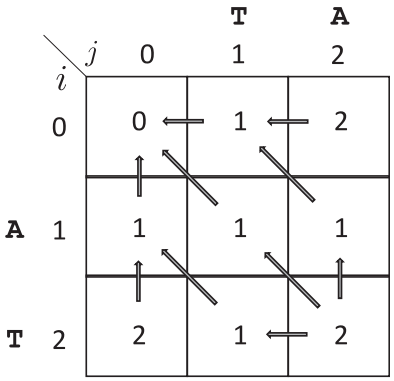

FIG. 4. Edit distance dynamic programming table for the strings $S_1 =$ "AT" and $S_2 =$ "TA" with annotations indicated by arrows.

matching the "T" symbols (no operation), and deleting the "A" from the front, corresponding to the alignment in Figure 1b. Finally, the path $(2, 2), (1, 2), (0, 1), (0, 0)$ corresponds to deleting the "T" from the end of "AT," matching the "A" symbols (no operation), and adding a "T" at the front of S_1, corresponding to the alignment in Figure 4c.

Thus, we refer to a path from (m, n) to $(0, 0)$ as an *optimal solution path*. Observe that the diameter of the space of optimal solutions is the maximum number of edges in which two optimal solution paths differ. In this example, the paths $(2, 2), (2, 1), (1, 0), (0, 0)$ and $(2, 2), (1, 2), (0, 1), (0, 0)$ differ in six edges, which is the largest distance among the three pairs of paths, giving a diameter of 6.

3.2. Diameter

As noted earlier, computing the diameter is equivalent to the problem of computing the maximum number of differences in edges between all pairs of optimal solution paths, that is, between all pairs of paths from cell (m, n) to cell $(0, 0)$ in the annotated DP table. This problem can be solved by a second DP algorithm operating on the annotated DP table.

We begin by describing the recursive solution for computing the diameter and then the DP implementation follows immediately. We henceforth assume that the annotated DP table for the pair of input strings has been computed. Let **edges**(i, j) denote the set of edges leaving cell (vertex) (i, j) in the DP table, corresponding to the set of operations that are part of optimal solutions to the subproblem represented by (i, j). Let **neighbors**(i, j) denote the set of cells (vertices) at the endpoints of each of the edges leaving (i, j), that is, the set of cells with an edge entering from cell (vertex) (i, j). For example, in the DP table in Figure 4, the cell at $(2,2)$ has three outgoing edges to cells $(2, 1)$, $(1, 1)$, and $(1, 2)$. Thus, **neighbors**$(2, 2) = \{(2, 1), (1, 1), (1, 2)\}$. Let **maxdifference** (i, j, i', j') denote the maximum number of differences in edges between two paths in the annotated DP table where one path begins at cell (i, j) and ends at $(0,0)$ and the other begins at cell (i', j') and ends at $(0, 0)$. Then, the diameter of the optimal solution space is **maxdifference** (m, n, m, n).

Consider two cells in the table, one at (i, j) and the other at (i', j'). Then, we say that (i, j) *subsumes* (i', j') if $i \geq i', j \geq j'$ and either $i > i'$ or $j > j'$. The significance of (i, j) subsuming (i', j') is that the first edge on any path from (i, j) to $(0\,0)$ cannot be part of a path from (i', j') to $(0\,0)$. Thus, in this case

$$\textbf{maxdifference}(i, j, i', j') = \max_{(s,t)\in\textbf{neighbors}(i,j)} 1 + \textbf{maxdifference}(s, t, i', j'). \tag{1}$$

Similarly, if (i', j') subsumes (i, j) then

$$\textbf{maxdifference}(i, j, i', j') = \max_{(s,t)\in\textbf{neighbors}(i',j')} 1 + \textbf{maxdifference}(i, j, s, t). \tag{2}$$

If neither (i, j) subsumes (i', j') nor (i', j') subsumes (i, j) and $(i, j) \neq (i', j')$ then the first edge on path from (i, j) to $(0,0)$ cannot appear on a path from (i', j') to $(0,0)$ and vice versa; thus **maxdifference**(i, j, i', j') can be computed using either of Equation (1) or (2). Finally, if $(i, j) = (i', j')$ then two most different paths

from (i,j) to $(0,0)$ may begin with either the same edge or two different edges. Let $\delta(x,y)=0$ if $x=y$ and 1 otherwise. Then

$$\textbf{maxdifference}(i,j,i,j) = \max_{(s,t),(u,v)\in\textbf{neighbors}(i,j)} 2\delta((s,t),(u,v)) + \textbf{maxdifference}(s,t,u,v). \tag{3}$$

Note that the term $2\delta((s,t),(u,v))$ contributes 0 to the distance if $(s,t)=(u,v)$ and thus the two paths begin with the same edge. This term contributes 2 to the distance if $(s,t)\neq(u,v)$ and the two paths begin with two different edges. The base case for this recursion is

$$\textbf{maxdifference}(0,0,0,0)=0. \tag{4}$$

This set of rules defines a recursive algorithm and that algorithm is then implemented as a dynamic program. Since there are four arguments in the recursive function, the DP table is four-dimensional. Moreover, for this problem, each cell has only a constant number of neighbors, and thus the asymptotic running time of the dynamic program is, therefore, $O(m^2n^2)$.

3.3. Pairwise distance vector

Recall that the pairwise distance vector is a vector v such that v_i denotes the number of pairs of optimal solutions whose distance is exactly i. The pairwise distance vector can be computed using the same recursive structure as that for the diameter, but now using vector operations rather than integer arithmetic.

Recall that pairwise distance vectors are indexed from 0 to the diameter of the space, d. Therefore, $d+1$-dimensional vectors suffice. However, rather than first computing the diameter to establish the dimension, we can use an upper-bound on the diameter as the dimension of the vectors and then compute the pairwise distance vector directly.

For this problem, observe that the diameter is upper-bounded by $2(m+n)$ since the longest path from (m,n) to $(0,0)$ has length $m+n$ and, in theory, two paths could differ on every edge. Let **dim** denote the dimension of the vectors used in the algorithm. In this problem, we use $\textbf{dim}=2(m+n)+1$.

Let **unit** denote the vector $(1,0,\ldots,0)$, let $+$ denote the standard vector addition operator, and let $\textbf{shift}_k(v)$ denote the vector in which each entry of v is shifted k places to the right. That is, $\textbf{shift}_k(v)$ is a vector w of dimension **dim** such that for each $0 \leq i < \textbf{dim}$:

$$w_i = \begin{cases} v_{i-k} & : i \geq k \\ 0 & : i < k \end{cases} \tag{5}$$

Let $v=\textbf{pdv}(i,j,i',j')$ denote the vector of dimension **dim** such that v_k denotes the number of pairs of paths in the DP table, one from (i,j) to $(0,0)$ and the other from (i',j') to $(0,0)$ that differ by exactly k edges. Then, the pairwise distance vector is $\textbf{pdv}(m,n,m,n)$.

By analogy to the computation of diameter, if (i,j) subsumes (i',j'), then first edge on any path from (i,j) to $(0,0)$ cannot be part of a path from (i',j') to $(0,0)$. Thus, in this case, $\textbf{pdv}(i,j,i',j')$ is the sum of the vectors for each of the neighbors of (i,j) shifted one position to the right because each distance increases by one due to the first edge from (i,j) to a neighbor, which is present in the solution beginning at (i,j) but not in the solution beginning at (i',j'):

$$\textbf{pdv}(i,j,i',j') = \sum_{(s,t)\in\textbf{neighbors}(i,j)} \textbf{shift}_1(\textbf{pdv}(s,t,i',j')). \tag{6}$$

Similarly, if (i',j') subsumes (i,j) then

$$\textbf{pdv}(i,j,i',j') = \sum_{(s,t)\in\textbf{neighbors}(i',j')} \textbf{shift}_1(\textbf{pdv}(i,j,s,t)). \tag{7}$$

If neither (i,j) subsumes (i',j') nor (i',j') subsumes (i,j) and $(i,j)\neq(i',j')$ then the first edge on path from (i,j) to $(0,0)$ cannot appear on a path from (i',j') to $(0,0)$ and vice versa; thus $\textbf{pdv}(i,j,i',j')$ can be computed using either of equations (6) or (7). Finally, if $(i,j)=(i',j')$ then two most different paths from (i,j) to $(0,0)$ may begin with either the same edge or two different edges. Again letting $\delta((s,t),(u,v))=1$ if $(s,t)\neq(u,v)$ and 0 otherwise. Then,

$$\mathbf{pdv}(i,j,i,j) = \sum_{(s,t),(u,v)\in \mathbf{neighbors}(i,j)} \mathbf{shift}_{2\delta((s,t),(u,v))}(\mathbf{pdv}(s,t,u,v)). \tag{8}$$

Since there is exactly one pair of paths (the empty path and itself) from $(0,0)$ to $(0,0)$ at distance 0 from one another, the base case for this recursion is

$$\mathbf{maxdifference}(0,0,0,0) = \mathbf{unit}. \tag{9}$$

Implementing this recursive algorithm using DP results in a four-dimensional table with $O(m^2n^2)$ cells and computing each entry in that table requires a constant number of vector additions and shift operations, each taking time $O(m+n)$. Therefore, the asymptotic running time of the algorithm is $O(m^2n^2(m+n))$.

As an example, Figure 5 shows a visualization of the pairwise distance vectors for two different instances of the edit distance problem. Each instance comprises a pair of randomly generated strings of length 100 where each symbol is generated uniformly at random from a set of four characters. The horizontal axis represents indices of the pairwise distance vector (symmetric difference distance) and the vertical axis represents the value of the vector at that index (number of pairs of solutions with that distance). The distribution in (Fig. 5a) has a single peak at distance ~ 60, whereas the distribution in (Fig. 5b) is multimodal, suggesting that there are likely to be two or more clusters of solutions.

Although the edit distance problem is relatively simple, this example is illustrative of a general principle that is broadly applicable. First, the diameter of the optimal solution space is induced by the paths in the DP table that differ in the largest number of edges, where edges are the annotations that are recorded while solving the DP. Second, finding a pair of paths that differ in the largest number of edges can be solved by a recursive algorithm whose arguments are pairs of vertices (cells) in the annotated DP table.

Third, that recursive algorithm can then be implemented using DP to obtain a polynomial-time algorithm to compute the diameter. Finally, the polynomial-time algorithm for computing the diameter can then be directly extended to compute pairwise distance vectors by replacing integer operations with corresponding vector operations.

In the next section, we illustrate this approach again for a problem that is somewhat more complicated but, nonetheless, can be solved using the same approach.

4. THE SMALL PARSIMONY PROBLEM

The small parsimony problem (also known as the *ancestral state reconstruction problem*) takes a binary tree as input with leaves labeled from a set of characters. These characters are typically phenotypic or molecular characters. The objective is to find an assignment of characters to internal nodes that minimizes the total number of differences between the labels of parent and child nodes. The small parsimony problem is used as a

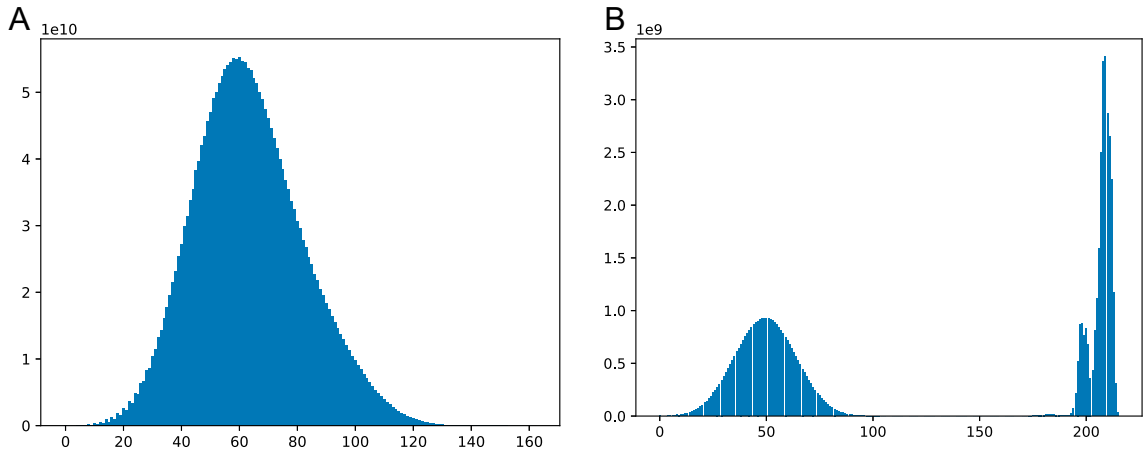

FIG. 5. Pairwise distance vectors for two random pairs of strings, each of length 100 over an alphabet of size 4. The horizontal axis is the pairwise distance and the vertical axis is the number of optimal pairs of solutions at this distance. **(A)** More than 1.9×10^7 optimal solutions, diameter 162, vertical scale $\times10^{10}$. **(B)** More than 1.4×10^6 optimal solutions, diameter 215, vertical scale $\times10^9$.

subproblem of the large parsimony problem, in which a phylogenetic tree is sought that minimizes the small parsimony score, but is also used in evolutionary studies where the phylogenetic tree is already established and the objective is to infer the characters of ancestral species. In the latter case in particular, the fact that there can be a very large number of equally optimal solutions to the small parsimony problem presents challenges because each solution is a different putative evolutionary history.

4.1. Computing the small parsimony score

The small parsimony problem can be solved using well-known recursive algorithms such as the Fitch Algorithm (Fitch, 1971). For completeness, we describe the basic recursive algorithm here. Let T denote a binary phylogenetic tree and let $L(T)$ denote its leaves. Let C denote a finite set of characters (e.g., molecular or phenotypic characters) and let $\ell : L(T) \to C$ denote a labeling of the leaves with characters. Let $\mathbf{left}(v)$ and $\mathbf{right}(v)$ denote the left and right children, respectively, of an internal node v and let r denote the root of the tree.

For each node, v and each character $c \in C$, let $\mathbf{cost}(v, c)$ denote the minimum cost of a subtree rooted at v if it is labeled with c. Then the minimum cost for the tree is $\min_{c \in C} \mathbf{cost}(r, c)$. The recursive algorithm for computing $\mathbf{cost}(v, c)$ first computes $\mathbf{cost}(\mathbf{right}(v), d)$ and $\mathbf{cost}(\mathbf{left}(v), d)$ for each $d \in C$. Then, again letting $\delta(x, y) = 1$ be if $x \neq y$ and 0, we now compute

$$\begin{aligned}\mathbf{cost}(v, c) = &\min_{d \in C}(\mathbf{cost}(\mathbf{left}(v), d) + \delta(c, d)) \\ &+ \min_{d \in C}(\mathbf{cost}(\mathbf{right}(v), d) + \delta(c, d)).\end{aligned} \tag{10}$$

The base case is for $v \in L(T)$

$$\mathbf{cost}(v, c) = \begin{cases} 0 & \ell(v) = c \\ \infty & \ell(v) \neq c \end{cases}. \tag{11}$$

Although this algorithm can be implemented bottom up as a dynamic program (e.g., the Fitch Algorithm), the recursive function is polynomial-time since this recursion makes no duplicated recursive calls. In either case, the complexity is $O(nk)$ where n is the number of nodes in the tree and k is the size of the character set. In either case, we can record (either in the top-down recursion or in the bottom-up dynamic program) annotations for each node and character to indicate the subproblems that give rise to optimal solutions. That is, for each (v, c) pair comprising a node v and character c, we can record all of the labels on $\mathbf{left}(v)$ and $\mathbf{r}ight(v)$ that give rise to the optimal score for (v, c). Let $\mathbf{leftopt}(v, c)$ denote the set of all labels d such that $\mathbf{left}(v)$ has label d in some optimal solution in which v is labeled c. Similarly, let $\mathbf{rightopt}(v, c)$ denote the set of all labels d such that $\mathbf{right}(v)$ has label d in some optimal solution in which v is labeled c.

4.2. Diameter

Consider the graph in which (v, c) pairs are vertices and the annotations computed in the recursion or DP table are directed edges. Specifically, for each vertex (v, c), there is a directed edge to each vertex $(\mathbf{left}(v), d)$ such that $d \in \mathbf{leftopt}(v, c)$ and, analogously, there is a directed edge from (v, c) to each vertex $(\mathbf{right}(v), d)$ such that $d \in \mathbf{rightopt}(v, c)$.

Let $\mathbf{OPT} = \min_{c \in C} \mathbf{cost}(r, c)$ and let $start = \{c \mid \mathbf{cost}(r, c) = \mathbf{OPT}\}$. In other words, *start* represents all characters that are assigned to the root of the phylogenetic tree in some optimal solution. An optimal solution to the small parsimony problem corresponds to a tree rooted at (r, c), $c \in start$, containing some node $(v, \cdot)$ for each v in the phylogenetic tree.

Equivalently, a solution corresponds to a set of pairs $(v, \cdot)$ for each v in the phylogenetic tree. Therefore, the symmetric set difference between two solutions is the number of $(v, \cdot)$ pairs in which they differ. Thus, if one solution assigns label c_1 to vertex v and the other assigns c_2 to vertex v, vertex v contributes 0 to the distance if $c_1 = c_2$ and contributes 2 otherwise. It is more natural, however, to consider this as contributing a distance of 1 since vertex v is simply labeled differently in the two solutions. Thus, for this problem, we define the distance between two solutions to be half of the symmetric set difference between the solutions. The diameter of the optimal solution space is, therefore, the maximum number of differences between any two optimal solutions.

Note that those trees may or may not begin with the same node (r, c). That is, the diameter may involve two trees in which the root node has the same or different characters.

Let $\mathbf{maxdist}(v, c_1, c_2)$ denote the maximum distance between two subtrees rooted at v, where one associates v with label c_1 and the other with c_2 (noting, as aforementioned, that c_1 may be equal to c_2). Then, the diameter is

$$\max_{c_1, c_2 \in \mathbf{start}} \mathbf{maxdist}(r, c_1, c_2). \tag{12}$$

In general, $\mathbf{maxdist}(v, c_1, c_2)$ can be computed as follows:

$$\mathbf{maxdist}(v, c_1, c_2) = \delta(c_1, c_2) \tag{13}$$

$$+ \max_{\substack{d_1 \in \mathbf{leftopt}(v, c_1) \\ d_2 \in \mathbf{leftopt}(v, c_2)}} \mathbf{maxdist}(\mathbf{left}(v), d_1, d_2) \tag{14}$$

$$+ \max_{\substack{d_1 \in \mathbf{rightopt}(v, c_1) \\ d_2 \in \mathbf{rightopt}(v, c_2)}} \mathbf{maxdist}(\mathbf{right}(v), d_1, d_2). \tag{15}$$

The base case is when v is a leaf node, in which case $\mathbf{maxdist}(v, c, c) = 0$ for $\ell(v) = c$. (No characters other than $\ell(v) = c$ are relevant).

Note that this function can make multiple repeated recursive calls because two different labels on a node v may have the same optimal solutions to left or right subproblems. Thus, this recursive algorithm is implemented with DP to compute the values bottom up or with memoization to avoid repeated recursive calls. We assume that C is a fixed set (e.g., nucleotides and phylogenetic characters) and thus treat $|C|$ as a constant. The asymptotic running time is $O(n)$ because there are $n|C|^2$ DP table entries entries of the form $\mathbf{maxdist}(v, c_1, c_2)$ and each such entry can explore at most $|C|^2$ combinations of labels on each of the left and right children of v.

We make several observations about this problem. First, as in the case of the diameter for edit distance, the diameter is found through a recursive algorithm (and ultimately, a DP implementation) that seeks the "traversals" of maximum distance in the graph induced by the DP table. Whereas a traversal in the edit distance problem was a path from one corner of the table to the opposite corner, in the small parsimony problem a traversal is a tree. However, the recursive calls on the left and right subtrees are independent of one another, and thus can be solved using two separate and independent recursive calls. In other words, not only does the small phylogeny problem have the optimal substructure property (Cormen et al., 2009) required for its efficient solution, but the diameter problem *also* has the optimal substructure property, allowing it too to be solved efficiently. Finally, whereas in the edit distance problem, the distance between two optimal solutions is the number of edges in which two traversals in the DP table differ, in the small parsimony problem it is more natural to define the distance as the number of vertices $(v, \cdot)$ in which two traversals in the DP table differ.

4.3. Pairwise distance vector

The pairwise distance vector for the small parsimony problem can be solved again by extending the algorithm for diameter and replacing arithmetic operations with vector operations. Let n denote the number of nodes in the phylogenetic tree. Then the number of internal nodes is $\frac{n-1}{2}$ and two labelings of the tree can differ in at most that many places. Thus, we use vectors with $\mathbf{dim} = \frac{n-1}{2} + 1$.

In this problem, the addition of solutions for the left and right subproblems is replaced by the convolution of vectors. Let $\otimes$ denote the convolution of two vectors: $x \otimes y$ is the vector v such that

$$v_i = \sum_{0 \le j \le i} x_j y_{i-j}, 0 \le i < \mathbf{dim}. \tag{16}$$

By analogy to the computation of the diameter, let $\mathbf{pdv}(v, c_1, c_2)$ denote the vector of dimension $\mathbf{dim}$ whose kth element is the number of pairs of subtrees in the DP table, one rooted at (v, c_1) and the other at (v, c_2), that differ by k.

Then, the pairwise distance vector is

$$\sum_{(c_1, c_2) \in \mathbf{start} \times \mathbf{start}} \mathbf{pdv}(r, c_1, c_2), \tag{17}$$

where $\times$ represents the Cartesian product and summation represents standard vector summation. The summation is used here since the pairwise distance vector counts the total number of pairs of solution at each possible distance.

For non-root internal nodes v, the pairwise distance vector for one subtree rooted at (v, c_1) and one rooted at (v, c_2) can be computed by first finding the pairwise distance vectors for each pair of subtrees for the left child of v with labels found in some optimal solution and, analogously, for each pair of subtrees for the right child of v with labels found in some optimal solution. Those vectors are then combined using convolution since the total number of solutions that differ in a total of k edges is the sum of all of the ways that k differences arise between the left and right subtrees: 0 on the left and k on the right, 1 on the left and $k-1$ on the right, and so forth. Finally, if $c_1 \neq c_2$, then there is an additional difference in the two subtrees and thus all of the distance counts must be shifted one position to the right in the pairwise distance vector. Thus, the general recursive step is

$$\mathbf{pdv}(v, c_1, c_2) = \mathbf{shift}\delta(c_1, c_2)\left(\sum_{\substack{d_1 \in \mathbf{leftopt}(v, c_1) \\ d_2 \in \mathbf{leftopt}(v, c_2)}} \mathbf{pdv}(\mathbf{left}(v), d_1, d_2) \otimes \sum_{\substack{d_1 \in \mathbf{rightopt}(v, c_1) \\ d_2 \in \mathbf{rightopt}(v, c_2)}} \mathbf{pdv}(\mathbf{right}(v), d_1, d_2)\right). \tag{18}$$

The base case for this recursion arises when v is a leaf node. The only label for a leaf node is the one defined by the given association $L(T)$ of leaf nodes to labels, and thus

$$\mathbf{pdv}(v, c_1, c_2) = \begin{cases} \mathbf{unit} : \ell(v) = c_1 = c_2 \\ \mathbf{zero} : \text{otherwise} \end{cases}. \tag{19}$$

Note that **unit** and **zero** correspond to the unit and zero vectors, respectively, with dimension **dim**.

The asymptotic time complexity of the DP implementation of this algorithm is derived as follows. There are $O(n|C|^2)$ cells in the table, one for each $\mathbf{pdv}(v, c_1, c_2)$ in Equation (18). Each cell involves one shift operation, a constant number (at most $|C|^2$) addition operations, and one convolution operation. The shift and addition operations take $O(n)$ time and the convolution operation takes $O(n\log n)$ time using a discrete Fourier transform (Cormen et al., 2009). Therefore, the asymptotic running time is $O(n^2 \log n)$.

As an example, Figure 6 shows a visualization of the pairwise distance vectors for two different randomly generated phylogenetic trees with 101 leaves (100 internal nodes) labeled randomly from a set of 20 characters. Each of the two trees is randomly generated and the labels of their leaves are selected at random.

The horizontal axis represents indices of the pairwise distance vector (symmetric difference distance) and the vertical axis represents the value of the vector at that index (number of pairs of optimal solutions with that distance). The peak of part (b) of Figure 6 is at 60, indicating that the modal difference between pairs of solutions is 60% (60 differences among 100 internal nodes). In addition, the bimodal distribution suggests that there are clusters of solutions. These facts suggest that a single optimal solution does not adequately represent the diversity of the solution space for this problem instance.

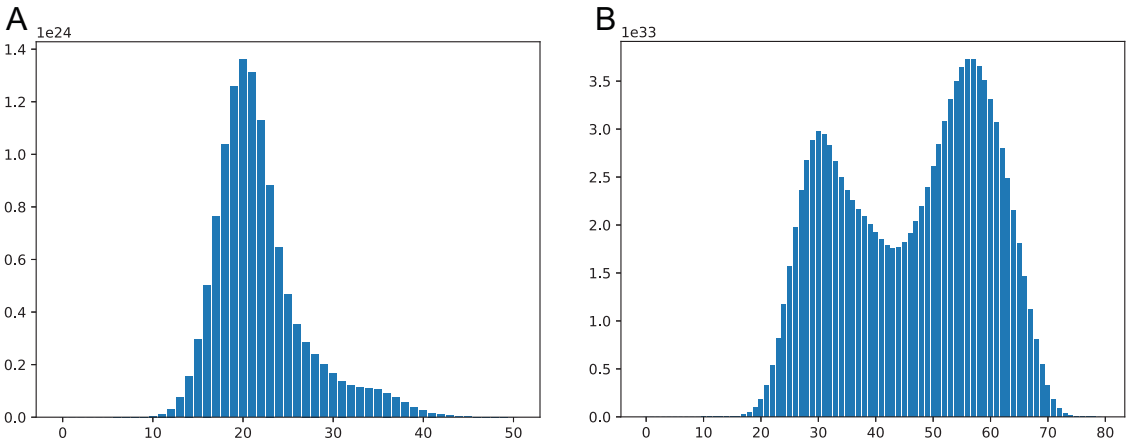

FIG. 6. Pairwise distance vectors for two random binary trees with 101 leaves (100 internal nodes) and 20 characters. The horizontal axis is the pairwise distance and the vertical axis is the number of optimal pairs of solutions at this distance. **(A)** More than 4×10^{12} optimal solutions, diameter 50, vertical scale is $\times 10^{24}$. **(B)** More than 4×10^{17} optimal solutions, diameter 79, vertical scale is $\times 10^{33}$.

5. CONCLUSION

Many applications in bioinformatics involve making inferences from the solutions to discrete optimization problems. These problems often give rise to very large spaces of equally optimal solutions. When the solution space is diverse, making inferences from any single solution may lead to conclusions that are not supported by other equally optimal solutions.

In this article, we have described a general method for efficiently computing statistics on the spaces of optimal solutions. These statistics may be useful in justifying the use of a single optimal solution (e.g., when the diameter or other pairwise distance statistics suggest that most solutions are similar to one another) or establishing the need for multiple representative solutions (e.g., when the solution space is diverse). Our method for computing the diameter, and by extension by the pairwise distance distribution, is based on the fact that not only does the underlying optimization problem (e.g., edit distance or small parsimony) have the optimal substructure property, but the diameter problem (and thus the pairwise distance problem) *also* has the optimal substructure property. This results in efficient DP algorithms for the diameter and pairwise distance problems. Those algorithms rely on first constructing the annotated DP table for the underlying problem and then constructing a second DP algorithm that operates on that table.

Although we have provided two examples of this method, one open problem is that of characterizing which problems are amenable to this approach. It is currently not known whether this approach can be applied to all problems with efficient DP algorithms. For example, the RNA folding problem (Nussinov et al., 1978; Zuker, 1989) has efficient DP algorithms based on optimal substructure of the recursive formulations, but computing the diameter for the space of optimal foldings remains an open problem (Liu et al., 2022). In particular, the RNA folding problem involves considering multiple pairs of subproblems and selecting the pair that gives the best folding score. Two different optimal solutions may, therefore, comprise solutions to different pairs of subproblems. Consequently, when seeking to compute the diameter for this problem, the subproblems can evidently overlap in complicated ways that did not arise in the simpler problems that we have considered here. The problem is potentially further complicated when using more realistic energy functions as in the Zuker Algorithm (Zuker, 1989). Thus, further study is required to determine whether more powerful methods can be used to compute diameters and pairwise distance vectors efficiently for such problems or whether computation of those statistics is computationally intractable in some cases.

ACKNOWLEDGMENTS

The author wishes to thank Dr. Shibu Yooseph, Danzhe Chen, and Ross Mawhorter for valuable conversations about this research as well as the anonymous reviewers and editorial staff whose suggestions improved the article.

CODE

The code used in this article is freely available at https://github.com/RanLH/PairwiseDistances

AUTHOR DISCLOSURE STATEMENT

No competing financial interests exist.

FUNDING INFORMATION

This study was funded by the U.S. National Science Foundation under grant number IIS-1419739.

REFERENCES

Agius P, Bennett K, Zuker, M. Comparing RNA secondary structures using a relaxed base-pair score. RNA 2010; 16(5):865–878.

Bansal MS, Alm EJ, Kellis M. Reconciliation revisited: handling multiple optima when reconciling with duplication, transfer, and loss. J Comput Biol 2013;20(10):738–754; doi: 10.1089/cmb.2013.0073

Bansal M, Alm E, Kellis, M. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. Bioinformatics 2012;28(12):i283–i291.

Cormen T, Leiserson C, Rivest R, et al. Introduction to Algorithms, 3rd ed. The MIT Press, Cambridge, Massachusetts; 2009.

Eddy S. How do RNA folding algorithms work? Nat Biotechnol 2004;22(11):1457–1458.

Fitch W. Toward defining the course of evolution: Minimum change for a specific tree topology. Syst Biol 1971;20(4): 406–416.

Haack J, Zupke E, Ramirez A, et al. Computing the diameter of the space of maximum parsimony reconciliations in the duplication-transfer-loss model. IEEE/ACM Trans Comp Biol 2018;16(1):14–22.

Heitsch C, Poznanovik S. Combinatorial Insights into RNA Secondary Structure. Springer:Berlin; 2014.

Huber K, Moulton V, Sagot, M.-F, et al. Exploring and visualizing spaces of tree reconciliations. Syst Biol 2018;68(4): 607–618; doi: 10.1093/sysbio/syy075

Kiirala N, Salmela L, Tomescu A. Safe and Complete Algorithms for Dynamic Programming Problems, with an Application to RNA Folding. In: 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany 2019;8:1–8:16.

Levenshtein V. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Phys Doklady 1966;10: 707–710.

Liu J, Duan I, Santichaivekin S, et al. Distance Profiles of Optimal RNA Foldings. In: International Symposium on Bioinformatics Research and Applications. Springer, Berlin 2022; pp. 315–329.

Mawhorter R, Libeskind-Hadas R. Hierarchical clustering of maximum parsimony reconciliations. BMC Bioinformatics 2019;20:1–12.

Miklós I, Darling A. Efficient sampling of parsimonious inversion histories with application to genome rearrangement in *Yersinia*. Genome Biol Evol 2009;1:153–164.

Mikls I, Kiss S, Tannier E. Counting and sampling SCJ small parsimony solutions. Theor Comput Sci 2014;552:83–98; doi: 10.1016/j.tcs.2014.07.027

Needleman S, Wunsch C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 1970;48(3):443–453.

Nguyen, T, Ranwez V, Berry V, et al. Support measures to estimate the reliability of evolutionary events predicted by reconciliation methods. PLoS One 2013;8(10):e73667.

Nussinov R, Pieczenik G, Griggs J, et al. Algorithms for loop matchings. SIAM J Appl Math 1978;35(1):68–82.

Rogers E, Heitsch C. New insights from cluster analysis methods for RNA secondary structure prediction. Wiley Interdiscip Rev RNA 2016;7(3):278–294; doi: 10.1002/wrna.1334

Salmela L, Tomescu AI. Safely filling gaps with partial solutions common to all solutions. IEEE/ACM Trans Comp Biol 2018;16(2):617–626.

Santichaivekin S, Mawhorter R, Libeskind-Hadas R. An efficient exact algorithm for computing all pairwise distances between reconciliations in the duplication-transfer-loss model. BMC Bioinformatics 2019;20(Suppl 20):636.

Tomescu A, Medvedev P. Safe and complete contig assembly through omnitigs. J Comp Biol 2017;24(6):590–602.

Vingron M. Argos P. Determination of reliable regions in protein sequence alignments. Protein Eng Des Select 1990; 3(7):565–569; doi: 10.1093/protein/3.7.565

Wang Y, Mary A, Sagot M.-F, et al. A general framework for enumerating equivalence classes of solutions. Algorithmica 2023;85(10):3003–3023.

Zuker M. On finding all suboptimal foldings of an RNA molecule. Science 1989;244(4900):48–52.

Address correspondence to:
Dr. Ran Libeskind-Hadas
Kravis Department of Integrated Sciences
Claremont McKenna College
Claremont
CA 91711
USA

E-mail: rhadas@cmc.edu