The minimal evolution problem in phylogenetics: polytopes, programming and interpretation.

Stefan Forcey, Gabriela Hamerlinck, Logan Keefe, William Sands

November 22, 2017

Abstract

Scientists use phylogenetic trees, or phylogenies, to visualize genetic relatedness of organisms. Phylogenetic trees can be used at many scales, from determining the population structure of a single species, to displaying all life on planet Earth. This chapter introduces the key concept of using a distance matrix (created from genetic data) in order to infer phylogenies via the principle of Balanced Minimum Evolution (BME). Then we focus on finding the phylogeny via linear programming techniques, especially branch-and-bound on relaxations of the BME polytope. Related methods for finding a BME phylogeny include edge-walking with tree moves (as in the program FastMe) and Neighbor Joining, a popular greedy algorithm for approximating the BME phylogeny. The skills used to infer and interpret a phylogenetic tree can be useful to many biological fields, including genetics and molecular biology, as well as applied research ranging from conservation of endangered species to tracking the spread of infectious diseases.

1 Introduction

A *phylogenetic tree* (or *phylogeny*) is a representation of the genetic relationships between organisms, species or genes. Mathematically, phylogenetic trees can be thought of as *partially labeled graphs* — a collection of items connected by branching edges. Phylogenies are commonly used in biology to explore the evolution, biological diversity and relatedness of species, organisms and genes. Phylogenetic trees allow us to postulate about similar adaptations and shared genes between organisms. Determining phylogenetic relationships can be a useful step in identifying the genetic basis of an observed trait between closely related species or individuals.

1.1 Phylogenetic reconstructions and interpretation

Phylogenetic reconstructions can be inferred from morphological or genetic data. For the purposes of this chapter, we focus on the genetic data used in molecular phylogenetics. The genome carries the complete genetic material of an organism.

This information is found in the deoxyribonucleic acid (DNA), which is transcribed into ribonucleic acid (RNA) where it can be processed into amino acids to form proteins. DNA is typically represented as a chain of nucleotide bases: adenine (A), guanine (G), thymine (T) or cytosine (C). In RNA, thymine is replaced by uracil (U). A nucleotide sequence is thus represented by a continuous chain of repeating letters (A, C, G and T/U). Some RNA strands encode proteins, meaning their sequence of nucleotide bases code for amino acids. There are 20 unique amino acids that, when strung together in a continuous chain, form unique proteins.

To construct a phylogenetic tree, we must first identify and align our sequences so they can be compared. These data could be DNA or RNA nucleotide sequences or an amino acid sequence. There are many programs and approaches available that will automatically align multiple sequences (see [SLV09]). For our purposes, we assume that we have a well-defined alignment of multiple sequences and explore phylogenetic reconstructions using distance based approaches, specifically the Balanced Minimal Evolution (BME) and branch and bound methods.

Distance based approaches to phylogenetic inference are one class of methods used to approximate a tree to a set of molecular data and can accommodate very large data sets. These methods use a matrix of genetic distances which estimate the genetic dissimilarity between organisms. The distance matrix is calculated by applying a model of evolution to the multisequence alignment and can be done in a variety of molecular phylogenetic programs (see [SLV09] for a thorough explanation). The BME method constructs a phylogenetic tree by minimizing a quantity called the tree length. In the case of error-free data, the tree and the corresponding tree length is uniquely determined by the pairwise distances, the dissimilarity matrix. The BME method yields an unrooted tree, without edge lengths. If wanted, edge lengths can be found by solving or approximating solutions to linear equations. Finally a root can be chosen or added based on external evidence. In this chapter, we pair the BME method with a branch and bound method. Branch and bound methodology is typically used to optimize the search for the most accurate phylogenetic reconstruction and we provide a demonstration of how branch and bound may be applied to the BME problem.

Phylogenetic trees contain a wealth of information about the evolution, biological diversity and relatedness of the species, organisms or genes represented in the tree. Each branch of the tree will end in a tip that represents the terminal or extant taxa that were used to construct the tree. The tips of phylogenetic tree branches are also called leaves. The internal branching points of the tree, called nodes, represent instances where genetic differentiation from the ancestral population has occurred. See Figure 1 for a simple example. The lengths of the branches of a phylogenetic tree can be calibrated to estimate evolutionary time. The branch length is then indicative of the amount of genetic change between organisms. For a more detailed description of the biological interpretations of evolutionary events represented in phylogenies, see [Bau08].

With recent technological and computing advances, researchers now have access to and have begun to utilize larger and larger genetic datasets in their phylo-

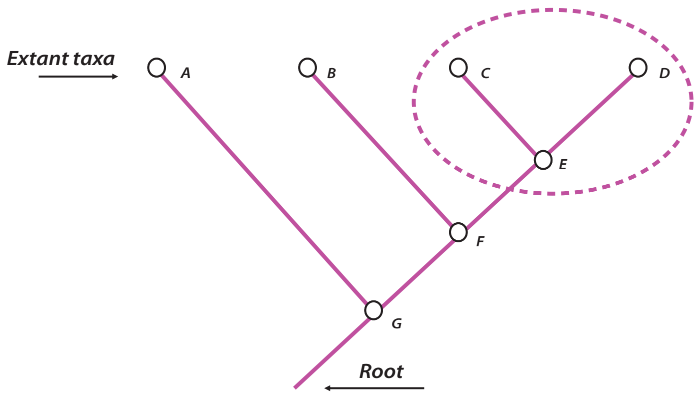


Figure 1: **Terminology and interpretation of phylogenetic trees. The root of a tree represents the ancestral population from which all the extant taxa are derived. The nodes in the body of the tree are branching points where differentiation (speciation) from the ancestral population has occurred. A monophyletic group, or clade, is marked with a dashed line. A clade with only two leaves is also known as a cherry. An example of a non-monophyletic group would include taxa B and C and their common ancestor nodes, E and F.**

genetic reconstructions–into the thousands of sequences ([MSM10] [CDvM+06]). Phylogenies produced from these large datasets have the potential to lead to more biological questions. For example, Figure 2 shows three of the current theories of angiosperm (flowering plant) evolution.

The grouping of *Chloranthus* and *Magnolia* in Figure 2 (a) is called a *clade*, specifically a clade with two leaves which is referred to as a *cherry*. A clade will include the ancestral lineage and all the descendants of that ancestor and is also called a monophyletic group. The monophyly seen with *Amborella* and the plants is disputed by other studies that have shown *Amborella* and *Nymphaeales* (water lilies) are sisters and are equally distant from the rest of the flowering plants (Figure2 (c); [BCM+00]). The inconsistencies shown between the trees in Figure 2 demonstrate a demand for new and innovative approaches to phylogenetic reconstructions as a way to approximate the true evolutionary history of organisms. The work presented in this chapter illustrates a new approach to the distance-based Balanced Minimal Evolution (BME) method of phylogenetic reconstructions.

1.1.1 Exercise

Redraw parts (a) and (c) of Figure 2 to get the alternate versions: cladogram and unrooted tree as seen in Figure 3.

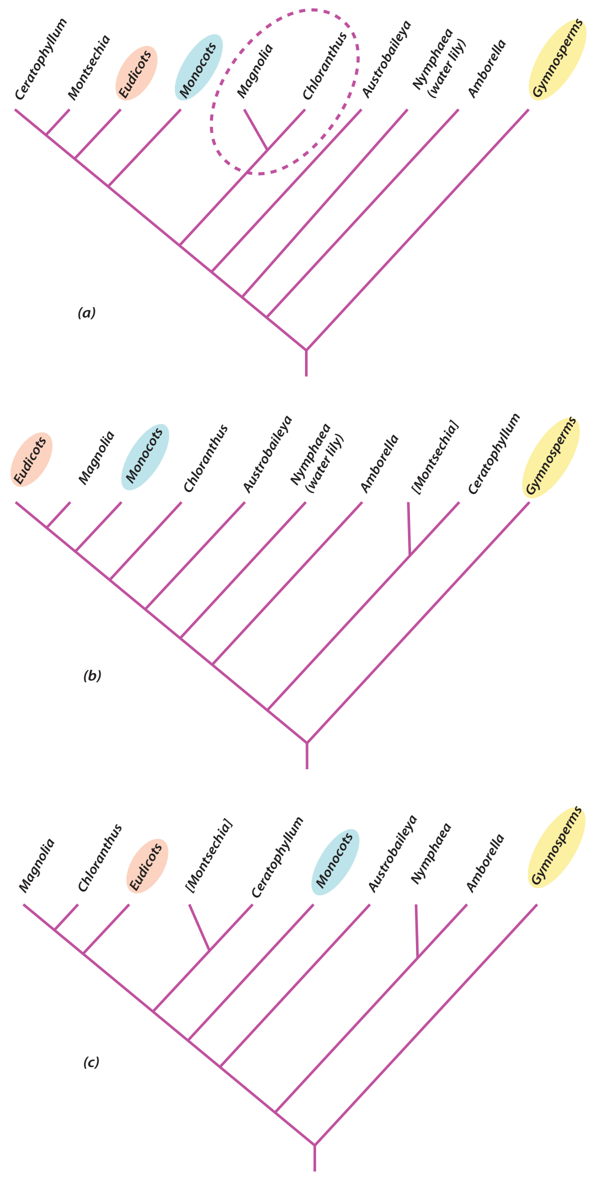


Figure 2: **Phylogenetic reconstructions for angiosperms (flowering plants), for each of which the gymnosperms contain the root ancestor. Top-to-bottom these are based on results in [GDGC+15], [Mor11], and [BCM+00]. Pictured are examples of *Amborella*, *Ceratophyllum*, and *Nymphae*. It is in [GDGC+15] that the fossil *Montsechia* appears, but we add it here to extant *Ceratophyllum* for comparison.**

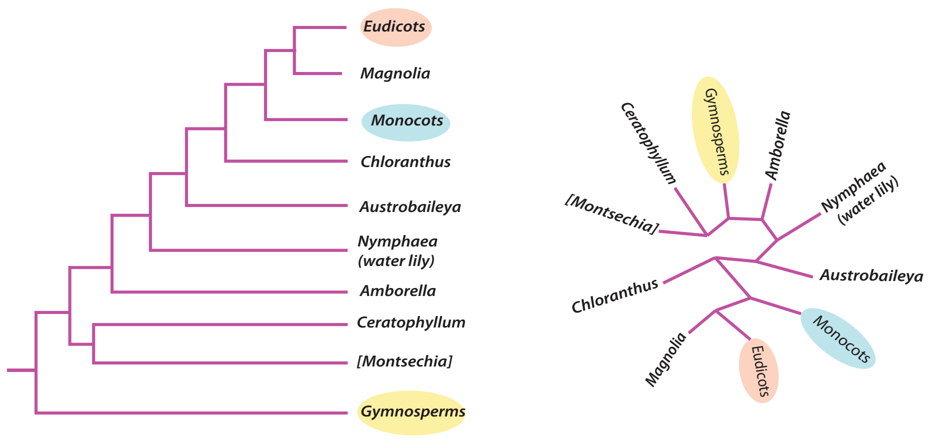


Figure 3: **Alternate tree diagrams, or cladograms for the tree in Figure 2 (b). The second diagram is the unrooted version.**

1.2 Balanced Minimal Evolution

Utilizing molecular data collected from DNA, RNA and amino acids allows us to measure *dissimilarities*, or distances, between pairs of taxa. Distance-based methods are a class of techniques that use this information to help select the best tree that is represented by a set of aligned molecular data. The Balanced Minimal Evolution (BME) method, which we consider in this chapter, finds the optimal tree by solving a minimization problem designed to pick out the unique tree whose geometry is able to yield those distances (or the tree which comes closest, in a sense to be made precise.) An advantage of the BME method is that it is known to be *statistically consistent*. This means that as we obtain more information related to the dissimilarity of species which indeed obey a tree structure, then our solution approaches the true, unique, tree representing that data. Furthermore, the correct tree can be recovered even in instances of missing or corrupted data, provided that the error is within bounds.

Various methods for obtaining solutions have been proposed over the course of several decades (See [Cat07] for a survey of current methods). In [SN87a], Saitou and Nei suggested a popular greedy algorithm, known as Neighbor-Joining (NJ), which runs in polynomial time. The primary disadvantage is that this greedy, bottom-up approach, often obtains the incorrect tree from noisy data. Developments by Gascuel and Steel [GS06], such as FastMe, work in a similar fashion, but use more sophisticated operations called *edge moves* that make the approach more robust. These algorithms are capable of handling a large number of taxa (e.g. 100).

In 2000, Pauplin [Pau00] showed how to calculate the length of a phylogenetic tree using a linear functional. Recently, it was discovered that minimizing this length over the set of phylogenetic trees is equivalent to minimizing over a geometric object known as the Balanced Minimal Evolution Polytope

[HHY11a]. Thus, the problem can be reformulated in terms of mathematical linear programming.

1.3 Definitions and Notation

To formulate our problem, we must introduce some definitions. We define $\mathcal{S} = \{1, 2, \ldots, n\}$ to be a list of distinct taxa that we wish to consider. Each element in $\mathcal{S}$ is a natural number, which corresponds to an individual taxon. Let the *dissimilarity vector* $\mathbf{d}$ with $\binom{n}{2}$ components be given, where each entry d_{ij} is positive and represents the dissimilarity between taxa i and j, for each pair $\{i, j\} \subset \mathcal{S}$. This vector is obtained from our aligned molecular data. Let $\mathcal{T}$ be the set of all binary phylogenetic trees *without* edge weights, up to isomorphism. Two trees in $\mathcal{T}$ are isomorphic, and we say they have the same tree topology, if we can label their respective internal nodes such that there is a graph isomorphism between the trees. Then, for each tree $t \in \mathcal{T}$, there is a corresponding vector $\mathbf{x}(t)$ with $\binom{n}{2}$ components $x_{ij}(t)$ for each pair $\{i, j\} \subset \mathcal{S}$. We define

$$x_{ij}(t) = 2^{n-2-l_{ij}(t)}, \tag{1}$$

where $l_{ij}(t)$ is the number of internal nodes (degree 3 vertices) in the path connecting i and j in t. The additional factor of 2^{n-2} is used to rescale Pauplin's original coordinates to be positive integers [Pau00]. We use a lexicographic ordering of the entries for vectors $\mathbf{d}$ and $\mathbf{x}$. We say a vector $\mathbf{b}$ has a lexicographic ordering if the entries b_{ij} are expressed in the form

$$\mathbf{b} = \left(b_{12}, b_{13}, \ldots, b_{1n}, b_{23}, b_{24}, \ldots, b_{2n}, \ldots, b_{(n-1)n}\right).$$

For simplicity, we chose to use a vector to express dissimilarity, rather than a *matrix*. The dissimilarity matrix D contains zeros along the main diagonal and is symmetric because $d_{ij} = d_{ji}$. Our vector $\mathbf{d}$ arises naturally from D in a simple way: It consists of rows from the upper triangular elements in D, excluding the main diagonal. If we are provided a binary tree T with non-negative weights on the edges, then we can calculate $\mathbf{d}$ by adding the weights on each of the edges connecting the path from i to j in T. Once we have calculated $\mathbf{d}$, we can determine the rescaled length of T using the path-length functional $\mathcal{L} : \mathcal{T} \longrightarrow \mathbb{R}$ defined by

$$\mathcal{L}(T) = \sum_{\substack{i,j \\ i<j}} d_{ij} 2^{n-2-l_{ij}(T)}. \tag{2}$$

Here l_{ij} does not depend on the edge lengths, just on the path-lengths. As shown by Pauplin [Pau00], $\mathcal{L}(T)$ is equal to $2^{n-2}L(T)$ where the length $L(T)$ is the sum of the edge lengths of T. Using our definition in (1), we can rewrite (2) as

$$\mathcal{L}(T) = \mathbf{d}_T \cdot \mathbf{x}(T). \tag{3}$$

Since T is its own BME tree, we are left to compute a single dot product. However, our task here is to find the BME tree represented by an arbitrary dissimilarity vector $\mathbf{d}$ using data that is potentially missing or corrupted. Therefore, we must extend this definition to handle any tree $t \in \mathcal{T}$ using a slight modification of (3). The penultimate form of our functional is

$$\mathcal{L}(t) = \mathbf{d} \cdot \mathbf{x}(t). \tag{4}$$

The ultimate form will come later, when we allow $\mathbf{x}$ to range over a region of Euclidean space. The primary difference between (3) and (4) is that the latter does not assume that the dissimilarity data comes from a known phylogeny. Therefore, it assumes the role of a more generalized objective function. However, in the case for which $\mathbf{d} = \mathbf{d}_T$ for T with positive edge lengths, the tree topology is unique and the functional $\mathcal{L}(t)$ will be minimized precisely when the minimizer t^* has the same topology as T. Thus minimizing this dot product provides a consistent way to reconstruct the tree which contributed the distance vector. Moreover, it has been shown that the method is *statistically consistent*: for any sequence of vectors $\mathbf{d}_n$ which approach $\mathbf{d}_T$, the corresponding sequence of minimizers t_n^* approach the topology of T.

Using (1), we can equivalently describe the structure of a tree t by its unique vector representation $\mathbf{x}(t)$. This allows us to minimize (4) over $\mathbf{x}(t)$, with the minimizer, now, being $\mathbf{x}(t^*)$. Observe that our rescaling will not affect the solution obtained through the minimization procedure. In general, the minimizer is only unique provided it does not contain edge weights that are identically equal to zero. In the latter case, we will have a finite collection of trees that minimize $\mathcal{L}(t)$ simultaneously.

Note that vectors are written several ways in this chapter. We use $(1, 2, 3)$ in the text, but for Matlab input and output this becomes [123]. For Polymake input and output we see $[1, 1, 2, 3]$ where an extra coordinate of '1' is placed in the first position.

1.3.1 Exercise

Find the BME tree, with $n = 4$, given the distance vector $\mathbf{d} = (6, 8, 9, 12, 7, 15)$. Procede as follows: 1) Draw all possible trees t for $n = 4$ and determine the vector $\mathbf{x}(t)$ for each tree. 2) Take dot products $\mathbf{d} \cdot \mathbf{x}(t)$ and select the tree corresponding to the minimal dot product. 3) Then reconstruct the 5 edge lengths by solving 6 linear equations simultaneously. In general there will be $\binom{n}{2}$ equations using the $2n - 3$ variable edge lengths. Note that if the original $\mathbf{d}$ is not additive, then the solution may not exist–and then we may be forced to choose approximate solutions. Luckily the tree topology is the crucial ingredient we seek. Also note that in this example, the naive approach would choose the smallest distance, 6, as the first cherry. This is known as the long-branch problem, and in the solution you will see that one branch has an outsize length.

2 Polytopes and Relaxations

2.1 What is a polytope?

A polytope is a *convex hull* of finitely many points in a Euclidean space. The definition of convex hull is as follows: A set Y is said to be *convex* if for any points $a, b \in Y$, every point on the straight line segment joining them is also in Y . The convex hull of a set of points X in Euclidean space is the smallest convex set containing X. Colloquially speaking, one way to define a polytope is a finite set of points which have been shrink wrapped. Some examples of *polytopes* include polygons, cubes, tetrahedrons, pyramids, and hypercubes, also known as tesseracts. Note that a point in Euclidean space is equivalently seen as a vector from the origin, and vice versa.

Another geometric definition of a polytope utilizes *half-spaces*, which are given by *linear inequalities.* If we take finitely many linear equalities such that the set of points which obey all of them is bounded, that set of points is a polytope. We can say that the polytope is the intersection of half-spaces described by those inequalities. Any polytope given by a convex hull can also be given in this manner.

If we cannot remove a point v without changing the convex hull itself, we will call v a vertex of the polytope. For example, each corner of a triangle is a vertex. If there exists a linear inequality such that every point in the polytope satisfies it, we call the set of points that satisfy it exactly a $face$ of the polytope. Each face of a polytope is itself a polytope. For example, all of the corners and edges of an octagon are faces of the octagon. The *dimension* of a polytope is the dimension of the smallest Euclidean space which could contain it. For example, the dimension of a pentagon is 2.

A $facet$ of a polytope is a face of the polytope with dimension one less than that of the polytope. For example, a square is a facet of a cube. A polytope can also be described combinatorially as a *partially ordered set*, or more specifically a *lattice.* Each polytope is made up of smaller polytopes– its faces– ordered by containment. The poset of faces does not record the geometric information, such as lengths and volumes, or location of points in space. Instead, it records the combinatorial type of the polytope, and we often refer to this as the polytope itself, up to equivalence.

Many polytopes occur naturally in a sequence. For easy examples, consider the simplices. This sequence begins with the degenerate cases of a point and a line segment, then the triangle and the tetrahedron. In general, as we increase the dimension by one, the n-dimensional simplex is the convex hull of $n + 1$ points in general position, such as the $n + 1$ unit vectors of a standard basis. Another example of a sequence is the n-dimensional cubes. Our polytopes of interest occur in a similar sequence, as described in the next section, but they skip dimensions.

2.2 The BME Polytope

Formulating our problem involves particular restrictions on the solutions we seek. These requirements can be developed by studying the topological properties of the trees belonging to $\mathcal{T}$. Pair-wise distances on phylogenetic trees have a structure that allows us to treat them as points in $\binom{n}{2}-$dimensional space. However, for a set $\mathcal{S}$, containing n taxa, one can show that there are exactly $(2n-5)!!$ potential trees that belong to $\mathcal{T}$ (See Table 1 for details). Minimizing our path length functional (4) over each of these points would require us to construct the vector $\mathbf{x}$ that describes the topology of every tree in $\mathcal{T}$, in addition to computing their corresponding path lengths. A useful result presented in [HHY11a], states that minimizing over the set of trees in $\mathcal{T}$ is equivalent to minimizing over the convex hull of $\mathcal{T}$. Thus, taking the convex hull of these points, denoted $Conv(\mathcal{T})$, we arrive at the BME Polytope, hereafter denoted as BME(n). For n=3, since there is only one tree with three leaves, BME(3) is a single point. For $n = 4$, the polytope BME(4) is a triangle as pictured in Figure 4.

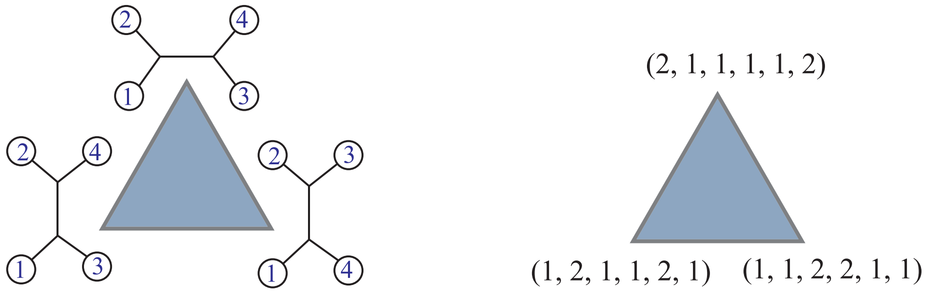


Figure 4: **The balanced minimal evolution polytope for $n = 4$. The vertices $\mathbf{x}(t)$ are shown for the respective trees.**

Polytopes have rich applications in mathematical programming problems, where the constraints take the form of inequalities. This allows us to reformulate the BME Problem as a *linear programming* problem where both the objective function and the constraint functions are linear. In this context, the BME Polytope represents the *feasible region* that is described by these inequalities. The facet inequalities are preferable in practice, because they allow us to implicitly maintain some of the geometric information of the trees without the need to explicitly determine $\mathbf{x}$ directly from a phylogeny.

2.2.1 Exercise

Find the vertices for the 5 dimensional BME polytope. Use software (such as polymake) to find the structure of the three types of 4-dimensional facets with their inequalities for the 5-dimensional BME polytope.

Taxa	**Dimension**	**Vertices**	**Facets**
3	0	1	0
4	2	3	3
5	5	15	52
6	9	105	90262
n	$\binom{n}{2} - n$	$(2n-5)!!$	?

Table 1: This table shows enumeration and formulas for some of the basic features of BME(n). The number of facets for BME(n) remains an open problem. Notice that the total number of facets from $n = 5$ to $n = 6$ increases drastically.

2.3 The Splitohedron

Consequently, obtaining a *complete* description of a polytope is often difficult. By complete, we mean that for every facet of the polytope, there exists a corresponding facet inequality that describes it. Furthermore, collections of facets for polytopes are often exponential or factorial in size, which are impractical for mathematical programming formulations. In particular, the $\mathcal{NP}$ nature of the BME problem, as noted in [CLPSG12], suggests that a complete description of the polytope is unlikely. To circumvent this, we can consider a relaxation of the BME polytope. A *relaxation* of a polytope P is a certain larger polytope R containing P. R is specifically any polytope that can be given by a subset of a list of inequalities which define P. We can develop relaxations of the BME Polytope using various combinations of the known facet inequalities. To this end, we propose several inequalities used to construct the relaxation of BME(n).

Proposition 1 (Caterpillar Facets and Cherry Faces)**.** *For every $i, j \in \mathcal{S}$, with $i \neq j$,*

$$1 \leq x_{ij} \leq 2^{n-3}. \tag{5}$$

This inequality provides both a lower bound and an upper bound on each of the decision variables used in the model. These inequalities suffice to guarantee that our polytope is bounded, since it is contained within the hypercube $\left[1, 2^{n-3}\right]$, in $\binom{n}{2}$-dimensional space. The right-hand side of the inequality follows immediately using the definition of x_{ij} and noting that every leaf must be separated using at least one internal node. These constraints are called the *Cherry Faces*. Similarly, on the left-hand side, the *Caterpillar Facets* follow because the distance between any two taxon in a tree is at most $n - 2$ internal nodes away.

Proposition 2 (Kraft Equalities)**.** *Let $i, j \in \mathcal{S}$. Then for every $i \in \mathcal{S}$,*

$$\sum_{j;j\neq i} x_{ij} = 2^{n-2}. \tag{6}$$

The Kraft Equality is a necessary condition for a path length sequence to represent a phylogeny. These equalities are commonly encountered in information theory, specifically in *Huffman trees*, which are rooted, binary trees used to represent symbols in a coding alphabet. Interestingly, Huffman trees can be described using a path length sequence [Ryt04]. Therefore, we can think of a phylogeny as a Huffman tree encoded in a binary alphabet using the taxa as symbols in the code [CLPSG12, FOR17]. We do not provide a proof of this inequality here, but one can derive this property using an inductive edge collapsing argument and an appropriate relabeling of the taxa.

Proposition 3 (Intersecting-Cherry Facets)**.** *Let $i, j, k \in \mathcal{S}$ be distinct. Then, for any collection of phylogenetic trees with either $\{i, j\}$ or $\{j, k\}$ as cherries, we have*

$$x_{ij} + x_{jk} - x_{ik} \leq 2^{n-3}. \tag{7}$$

This inequality will become strictly less than when a graph contains neither $\{i, j\}$ or $\{j, k\}$ as cherries. The proof that this inequality forms a facet of BME(n) is in Theorem 4.7 of [FKS16a]. In [CLPSG12], an equivalent form of this inequality was proposed. The authors referred to these constraints as the *triangular inequalities*.

Proposition 4 (Split Facets)**.** *Consider $\pi = \{S_1, S_2\}$, a partition of $\mathcal{S}$. Let $|S_1| := k \geq 3$ and $|S_2| := m \geq 3$. Then for $i, j \in S_1$*

$$\sum_{\substack{i,j \\ i<j}} x_{ij} \leq (k-1)2^{n-3} \tag{8}$$

Here, $|X|$ denotes the cardinality of a set X. For convenience, we will refer to types of splits using the cardinality of their partitions, e.g., we say a tree exhibits a (k, m)-split. This inequality allows us to have some control on the positioning of the taxa within a subgraph of a tree t. The split inequality is an equality for any tree that displays the split, and is a strict inequality for all others. In [FKS16b], we proved that this inequality, indeed, forms a facet of BME(n). We also showed that this inequality grows on the order of $\mathcal{O}(2^n)$, which will be relevant to our discussion on the performance of our proposed algorithm in later sections.

We define our relaxation of the BME Polytope as the intersection of half-spaces given by Propositions 1-4. This operation forms a new polytope, which we call the *Splitohedron*, denoted as $Sp(n)$. Some properties regarding faces of Sp(n) are provided in Table 2. Note that the cherry face inequalities, that is the upper bounds $x_{ij} \leq 2^{n-3}$, are actually redundant. This can be seen

Classification	**Size of Collection**	**Vertices in Faces**
Caterpillar Facets	$\binom{n}{2}$	$(n-2)!$
Cherry Faces	$\binom{n}{2}$	$(2n-7)!!$
Intersecting Cherry Facets	$\binom{n}{2}(n-2)$	$2(2n-7)!!$
Kraft Equalities	n	-
Split-Facets	$2^{n-1}-\binom{n}{2}-n-1$	$(2m-3)!!(2k-3)!!$

Table 2: We provide some statistics for the inequalities used in our relaxation, Sp(n), based on [FKS16b]. Notice that the facets of first three classes of inequalities grow polynomially in n, while the facets for (k,m)-splits grow exponentially in n. The Kraft Equalities appear in all faces of the polytope, so they trivially contain all of the vertices.

since that upper bound is also implied by the pair of intersecting-cherry inequalities based on (1) the intersecting cherries $\{i,j\}$ and $\{j,k\}$ and (2) the intersecting cherries $\{i,j\}$ and $\{i,k\}$. Adding the latter two inequalities yields the former upper bound. However, keeping the cherry inequalities can be a programming convenience. The first nontrivial polytope $Sp(4)$ is the same triangle as BME(4). After that, although the facets are fewer, there are more vertices than in BME(n). From computer calculations using *Polymake*, [GJ00], we see that $Sp(5)$ has 27 vertices and $Sp(6)$ has 2335. We now state a theorem obtained in [FKS16b] relating the vertices of Sp(n) and BME(n).

Theorem 1. *Let t be a phylogenetic tree with n taxa. If the number of cherries of t is at least $\lceil \frac{n}{4} \rceil$, then $\mathbf{x}(t)$ is a vertex in both BME(n) and Sp(n). For $n \leq 11$ the statement holds regardless of the number of cherries.*

Theorem 1 allows us to estimate when we begin losing information under the relaxation. As long as $n \leq 11$, the Splitohedron will contain all the vertices of BME(n). Otherwise, we begin losing some of the vertices of the BME Polytope.

2.3.1 Exercise

Find the vertices for the 3 and 5-dimensional splitohedra.

3 Optimizing with Linear Programming

The linearity of the half-spaces defining the Splitohedron and the underlying linear objective function suggest a linear programming based approach. Recall that, in equation (4) we defined the path-length functional that describes the length of a tree $t \in \mathcal{T}$. As previously noted, we seek a minimizer $\mathbf{x}(t^\star)$ of this functional, but we delayed defining the region containing admissible solutions. Now that we have defined our relaxation of the BME Polytope, we can use it as the feasible region for our linear programming model. Our model is as follows.

Formulation (Discrete Integer Linear Programming)**.**

$$\begin{aligned}
&\underset{\mathbf{x}}{\operatorname{argmin}} \quad \mathbf{d} \cdot \mathbf{x} \\
&\textit{subject to:} \\
&\sum_{j;j\neq i} x_{ij} = 2^{n-2}, \qquad \forall i \in \mathcal{S} && (9)\\
&x_{ij} + x_{jk} - x_{ik} \leq 2^{n-3}, \qquad i,j,k \in \mathcal{S}, i \neq j \neq k && (10)\\
&\sum_{\substack{i,j \in S_1 \\ i<j}} x_{ij} \leq (k-1)2^{n-3}, \qquad k \geq 3, m \geq 3 && (11)\\
&1 \leq x_{ij} \leq 2^{n-3}, \qquad \forall i,j \in \mathcal{S}, i \neq j && (12)\\
&x_{ij} \in \{2^k : k \in \mathbb{Z}_+ \cup \{0\}\}, \qquad i,j \in \mathcal{S}, i \neq j && (13)
\end{aligned}$$

Here we use *argmin* because we want to return the argument $\mathbf{x}$ that minimizes the rescaled path length $\mathcal{L}(\mathbf{x})$. This ultimate form of our functional considers $\mathbf{x}$ as a general vector *without* the dependence on t. This is a direct consequence of our relaxation, which introduces non-tree realizable vectors into the feasible region. Notice that (9)-(12) are the *Kraft Equalities*, *Intersecting Cherry Facets*, *Split Facets*, and the *Caterpillar and Cherry Faces*, respectively. These are the same inequalities we used to define our polytope $\mathrm{Sp}(n)$. The last constraint, (13), states that each of the variables belongs to the set of powers of 2. This allows us to avoid encountering many of the potential solutions that might not belong to the BME Polytope that are present in our relaxation. It is this constraint which makes the problem difficult.

In terms of computation, the system above, it is current state is *not* a polynomial sized formulation. The number of inequalities in constraint (11) grow according to the power set and are, therefore, $\mathcal{O}(2^n)$. While this is not an issue for smaller problems, where $n \leq 11$, larger problems are consist of many inequalities and the numerical linear algebra slows the run time considerably. Therefore, for larger problems, some simplifications would be necessary to obtain solutions within a reasonable run time.

3.0.1 Exercise

Repeat Exercise 1.3.1 using the inequalities for the polytope BME(3), and the simplex method, by hand or with your favorite software.

3.1 Discrete Integer Linear Programming: The Branch and Bound Algorithm

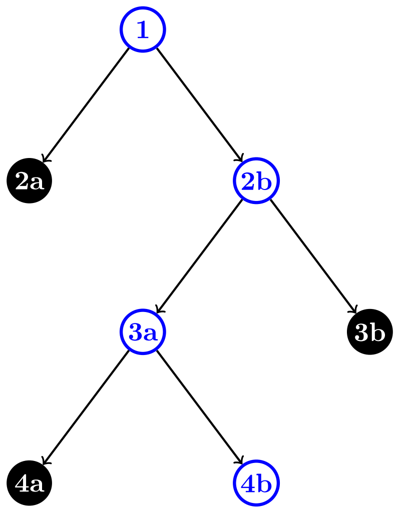


Figure 5: An example of a Branch and Bound binary search tree. Active nodes follow along the indicated blue path, while fathomed nodes are labeled in black.

The problem of finding the vertex of the BME polytope which corresponds to the tree that minimizes our product belongs to a class of problems called discrete integer linear programs. That is because we impose the requirement that values in the solution vector be powers of two. One of the most common techniques for solving this class of problems is called *branch and bound*. This process is recursive, breaking the original problem into *subproblems*, which are easier to solve. The recursive structure of this process can be visualized as traversing a rooted binary tree, where each node represents an individual linear programming problem. We provide an example of a branch and bound tree in Figure 5.

To begin, the discrete valued constraints on the decision variables are relaxed. This allows us to utilize linear programming algorithms because the decision variables, now, admit a continuum of values. The initial linear programming problem that results from this relaxation is called the *root LP*. Computing its solution allows us to determine the feasibility of the original problem. If the solution to the root LP meets our original restrictions for the decision variables, then the branch and bound routine terminates. Otherwise, we select a variable according to a *branching rule* and begin the branching process. The branching rule tells us how to divide the solution space, which results in a set of subproblems, which represent new nodes in our branch and bound tree. After separately solving each of these problems, a *selection strategy* is used to determine which nodes to explore in the branch and bound tree. If a node is not explored, we say the node was *fathomed* or *pruned*. Once a node is selected for exploration, the process is repeated.

The inequalities used in the creation of individual problems, along the path, are maintained throughout the search. Once a feasible solution satisfying the constraints on the decision variables is obtained, we can update the global bound on the objective and use it to prune subproblems, which provide a less optimal objective value. We are permitted to prune subproblems in this manner, even if the discrete constraints on the decision variables are not satisfied. We call the best, current solution the *incumbent solution*. This pruning process allows us to eliminate significant portions of the search space, which effectively reduces the algorithm's running time. Repeatedly applying this process allows us to eventually obtain the optimal solution to our original Discrete Programming problem.

3.1.1 Exercise

Perform branch and bound. Maximize $p = 6.75x + 5y$ subject to

$$-x \leq 0$$

$$-y \leq 0$$

$$y \leq 10$$

$$x \leq 8.3$$

$$79x + 18y \leq 693.5$$

Require all coordinates of answer (x, y) to be powers of 2.

3.2 Recursive Structure: Branch Selection Strategy and Fixing Values.

Designing a branch and bound algorithm is an open-ended decision process. One has a large amount of flexibility among choices for selection strategies and branching rules available for implementation. For example, if our problem required that variables belong to $\{0, 1\}$, then we could chose to branch on variables with the "most fractional" entry. For a vector $\mathbf{x}$, this is the entry i satisfying

$$\underset{i}{\operatorname{argmin}}\{|x_i - 0.5|\}. \tag{14}$$

In our problem, since we require that variables be powers of 2, we could modify (14) so that we chose the entry whose value is "farthest" from any adjoining power of 2. The expression for this is slightly more complex and takes the form

$$\underset{i}{\operatorname{argmax}}\{\min\{|x_i - 2^{\lfloor log_2 x_i \rfloor}|, |x_i - 2^{\lceil log_2 x_i \rceil}|\}\}. \tag{15}$$

The complexity of this selection strategy arises naturally from the structure of powers of 2. As the numbers in the set become larger, the distance between adjoining elements also increases. We can also think of (15) as being a bottom-up selection strategy, where the algorithm focuses on finding the cherries first. If the selection strategy (15) leads to multiple maximizers, then we can choose any of them to use as a branching variable. Our strategy involves choosing the first index. Of course, many other selection strategies exist based on machine-learning and examining statistical relationships between selection variables and the corresponding "gain" in the objective function (See for example [AKM05]). We omit these rules in an effort to maintain simplicity.

Once we have selected the variable to branch on, we must determine how to subdivide the feasible region. This effect is accomplished by *bounding functions*, which introduce new inequalities to the feasible region in an effort to remove the unwanted values. Suppose we have identified the branching variable x_j. During the branching phase, we generate two new bounds

$$x_j \leq 2^{\lfloor log_2(x_j) \rfloor} \tag{16}$$

$$x_j \geq 2^{\lceil log_2(x_j) \rceil}, \tag{17}$$

which are maintained in the constraint sets $\{A_1, \mathbf{b}_1\}$ and $\{A_2, \mathbf{b}_2\}$, respectively. Each constraint set is associated with a particular subproblem, and is passed recursively to the branch and bound algorithm, where both feasibility and validity are further examined. If the problem is infeasible, then the node is fathomed. If the problem is feasible and the solution has entries that are powers of 2, then we check the value of the objective function. If it is less than the incumbent solution (since we are minimizing), then the bound is updated. Once a valid solution has been found, its bound can be used to prune suboptimal branches in the tree.

We can also also introduce *heuristic* fixing. Heuristics, on a general level, involve incorporating problem-dependent experimental information into an algorithm to eliminate unlikely candidates in the branch and bound process and promote faster convergence to the optimizer. If variables in the solution have floating point values which are "close" to a power of 2 and remain stagnant during the process, we could reduce round-off errors by fixing the values to their closest power of 2. Therefore, given a tolerance $\epsilon > 0$ and a vector $\mathbf{x}$, we select a candidate for fixing if an entry satisfies

$$\operatorname*{argmin}_i |x_i - 2^{[log_2(x_i)]}| < \epsilon. \tag{18}$$

Suppose now that we have identified a candidate variable j that satisfies (18). Then we set

$$x_j = 2^{[log_2(x_j)]} \tag{19}$$

in the equality constraints $\{A_{eq}, \mathbf{b}_{eq}\}$. This effectively eliminates the variable x_j as an unknown. We fix variables in two places: inside the main script, after finding the root solution, and before branching. In the first instance, we fix all cherries using equality constraints and adjust the upper bounds from 2^{n-3} to 2^{n-4}. The experimental observation, here, was that linear programming solvers tended to find all cherries in the given problem after solving the root LP. Therefore, it was deemed reasonable to fix their positions in the equality constraints. In the second instance, we fix any variable according to (18) and (19). Should multiple variables be eligible for fixing, we choose one closest to its rounded value. This sort of secondary fixing is commonly referred to as a Large Neighborhood Search (LNS) since it searches for solutions to the underlying problems in a large neighborhood of the polytope, which contains the face generated by our fixed entry. As we let $\epsilon \to 0$, the algorithm relies less on fixing and performs similarly to pure branch and bound. The size of ϵ controls how liberal we wish to be with fixing. The primary issue with the LNS heuristic is that it does not effectively deal with infeasibilites. Once the algorithm encounters an infeasibility, the process terminates. This problem has been considered in strategies such as Relaxation Induced Neighborhood Search (RINS) and Guided Dives, which use sophisticated back-tracking processes to return to a feasible state. Authors in [DRLP05] develop these methods, exploring how each of the methods define, search, and diversify neighborhoods to improve incumbent solutions. In practice, such techniques have allowed for fast, successful computation of often intractable optimization problems. Heuristics, in the same regard as other apsects of branch and bound, require careful experimentation. However, it should be noted that approaches utilizing heuristics can no longer guarantee that the solution obtained is the true minimizer (or maximizer) for the original problem.

3.2.1 Exercise

Given the current solution $\mathbf{x} = (2, 4, 3.25, 3.42, 2, 3.68, 1.33, 1, 8, 4, 2, 2, 4, 8, 1.5)$, use the expression (15) to determine the branching variable and write down the two bounds generated by branching using (16) and (17).

3.3 Pseudocode for Branch and Bound Algorithm: PolySplit

Our branch and bound procedure consists of a main script to manage the entire problem, and two separate branching algorithms embedded in the main file. The purpose of the main file is to formally initialize the best current bound and evaluate the feasibility of the root LP. Before the root LP is solved, the data for the problem is collected. This data includes the given disimilarity data $\mathbf{d}$, the sets of inequality and equality constraints, choices for parameters governing run-time and termination, and a decision to apply the LNS heuristic. Once this information has been collected, it is sent to the linear programming solver to identify the solution for the root and determine feasibility. If the problem has been deemed feasible, it is sent to a recursive branching function with updated constraints based on whether or not a heuristic has been applied. We provide pseudocode for the main algorithm in Figure 6.

The algorithm decides to call a particular branching function depending on whether or not we use a heuristic. If we choose to use the heuristic, the algorithm finds cherries and fixes their positions in the constraints, which are then passed as input to the branching function. It was observed in numerous test cases that LP solvers could identify cherries immediately in the root LP and that they did not change over the course of the process. Therefore, we chose to fix their values in the solution vector.

The algorithms for the branching functions are similar with a slight modification in the heurisitc based option. We provide a detailed outline in Figure 7. The branching functions first evaluate the solution of the LP at the current node. If we are at the root node, then the problem is solved twice. This redundancy is not an issue and merely serves to simplify our code. If the solution at the current node is infeasible or the value of the objective function is worse than our incumbent solution, then the node is pruned. Otherwise, the algorithm checks the solution to see if it contains all powers of 2. If this holds, then the solution becomes the new incumbent and we can use the value of the objective function to prune suboptimal results later in the search. If we find that the solution produced does not contain all powers of 2, then we branch on the current solution. We can identify the branching variable, according to a devised rule and then create the two subproblems. Each subproblem receives one of the new inequalities (16) and (17), respectively. Finally we pass the information for the subproblems $\mathcal{P}_1$ and $\mathcal{P}_2$, recursively, to the branching function. Each subproblem communicates with the another through the incumbent solution. Once the branching produces suboptimal results, then the algorithm terminates and reports the solution to the main algorithm.

To modify the code present in Figure 7 to apply the heuristic, we need to incorporate an additional parameter, which we call "maxiter0." The algorithm is designed to work with pure branch and bound until the number of iterations reaches maxiter0. At this point, the algorithm has not found the solution, so instead, we begin applying the heuristic, according to (18) and (19). This process works in conjunction with pure branch and bound until we (1) reach the solution, (2) encounter an infeasibility, or (3) reach maxiter1 and the algorithm terminates. This adjustment is reflected in our code in Figure 8.

It is important to note that for our purposes, we used a single value for each of the tolerances in our branching algorithms presented in Figures 7 and 8. In principle, we could have defined different tolerances, say ϵ_0, ϵ_1, and ϵ_2, associated with the stopping criterion, branching variable selection, and the LNS search, respectively. Another area to explore would be variations in the iterations before the heuristic is applied. For instance, does the algorithm exhibit better performance and solution quality when the heuristic is applied earlier?

Once the algorithm terminates, it is possible to draw the phylogenetic tree simply by passing the solution to the distance function, which determines the topological distances l_{ij} of the phylogenetic tree. Using our definition for x_{ij} in (1) and inverting the exponential piece via logarithms, we obtain a formula for l_{ij}'s in terms of x_{ij}. The ability to draw the tree, of course, conditional on the tree realizability of the solution $\mathbf{x}^\star$. This was experimentally observed in the above algorithms, although a formal proof of this is yet to be complete.

Algorithm 1: The Discrete ILP Main Algorithm PolySplit

```
Require: Identification of minimizer $\mathbf{x}^\star$
Input: $\mathbf{d}$, $A$, $\mathbf{b}$, $A_{eq}$, $\mathbf{b}_{eq}$, $lb$, $ub$, $n$, maxiter0, maxiter1, $\epsilon$, heuristic
Output: $\mathbf{x}^\star$, $\mathcal{L}^\star$, status
 1: Initialization: set bound $= +\infty$ and iter $= 0$
 2: Solve the relaxation at the root node $\to \mathbf{x}_0$, $\mathcal{L}_0$, status0
 3: if status $=$ infeasible then
 4:     Return $\mathbf{x}^\star$, $\mathcal{L}^\star = \emptyset$
 5: else
 6:     if heuristic $= 0$ then
 7:         Call branch0 $\to \mathbf{x}^\star, \mathcal{L}^\star$, status
 8:     else if heuristic $= 1$ then
 9:         Find all entries s.t. $|x_{0_i} - 2^{n-3}| < \epsilon$ for $i = 1, \dots, \binom{n}{2}$
10:         Fix positions and update $A_{eq}, \mathbf{b}_{eq}$, $ub$
11:         Call branch1 $\to \mathbf{x}^\star, \mathcal{L}^\star$, status
12:     end if
13: end if
```

Figure 6: Main algorithm PolySplit for the Discrete ILP problem.

3.3.1 Exercise

The Matlab code for PolySplit is available from the Encylopedia of Combinatorial Polytope Sequences, at www.math.uakron.edu/~sf34/hedra.htm#splito. Sample input is given there as well, for instance the distance vector for a tree with nine leaves:

$$\mathbf{d} = (4, 5, 3, 2, 6, 7, 8, 8, 3, 3, 4, 4, 5, 6, 6, 4, 5, 3, 4, 5, 5, 3, 5, 6, 7, 7, 6, 7, 8, 8, 3, 4, 4, 3, 3, 2)\,.$$

Find the optimal tree using the code. The output will be the powers-of-2 vector. Use it to redraw the tree. The Matlab files are also found in [San17], available from etd.ohiolink.edu.

Algorithm 2: PolySplit Branch0 Algorithm

```
Input: $\mathbf{d}$, $A$, $\mathbf{b}$, $A_{eq}$, $\mathbf{b}_{eq}$, $lb$, $ub$, $\mathbf{x}_t, \mathcal{L}_t$, $\epsilon$, bound
Output: $\tilde{\mathbf{x}}, \tilde{\mathcal{L}}$, status, bb
1:  Solve the relaxation at the current node $\rightarrow \mathbf{x}_0, \mathcal{L}_0$, status0
2:  if status0 = infeasible or $\mathcal{L}_0 >$ bound then
3:      Return input, bb $\leftarrow$ bound
4:  else
5:      Compute $\mathcal{E} = \max_i\{\min\{|x_{0_i} - 2^{\lfloor log_2(x_{0_i})\rfloor}|, |x_{0_i} - 2^{\lceil log_2(x_{0_i})\rceil}|\}\}$
6:      if $\mathcal{E} < \epsilon$ or iter $>$ maxiter1 then
7:          if $\mathcal{L}_0 <$ bound then
8:              $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_0, \tilde{\mathcal{L}} \leftarrow \mathcal{L}_0$, bb $\leftarrow \mathcal{L}_0$
9:          else
10:             Return input, bb $\leftarrow$ bound
11:         end if
12:         Return
13:     end if
14:     Select a branching variable $x_{0_j}$
15:     Build subproblem $\mathcal{P}_1$ : Set $x_{0_j} \leq 2^{\lfloor log_2(x_{0_j})\rfloor}$ in $\{A, \mathbf{b}\} \rightarrow \{A_1, \mathbf{b}_1\}$
16:     Build subproblem $\mathcal{P}_2$ : Set $x_{0_j} \geq 2^{\lceil log_2(x_{0_j})\rceil}$ in $\{A, \mathbf{b}\} \rightarrow \{A_2, \mathbf{b}_2\}$
17:     iter $\leftarrow$ iter + 2
18:     Call branching routine for $\mathcal{P}_1 \rightarrow \mathbf{x}_1, \mathcal{L}_1$, status1, bound1
19:     if bound1 $<$ bound and status1 = feasible then
20:         $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_1, \tilde{\mathcal{L}} \leftarrow \mathcal{L}_1$, bound $\leftarrow$ bound1, bb $\leftarrow$ bound1, status $\leftarrow$ status1
21:     else
22:         Return $\mathcal{P}_1$ input data, bb $\leftarrow$ bound
23:     end if
24:     Call branching routine for $\mathcal{P}_2 \rightarrow \mathbf{x}_2, \mathcal{L}_2$, bound2, status2
25:     if bound2 $<$ bound and status2 = feasible then
26:         $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_2, \tilde{\mathcal{L}} \leftarrow \mathcal{L}_2$, bb $\leftarrow$ bound2, status $\leftarrow$ status2
27:     end if
28: end if
```

Figure 7: Branching algorithm for pure Branch and Bound.

Algorithm 3: PolySplit Branch1 Algorithm

```
Input: $\mathbf{d}$, $A$, $\mathbf{b}$, $A_{eq}$, $\mathbf{b}_{eq}$, $lb$, $ub$, $\mathbf{x}_t, \mathcal{L}_t$, $\epsilon$, bound
Output: $\tilde{\mathbf{x}}, \tilde{\mathcal{L}}$, status, bb
1:  Solve the relaxation at the current node $\rightarrow \mathbf{x}_0, \mathcal{L}_0$, status0
2:  if status0 = infeasible or $\mathcal{L}_0 >$ bound then
3:      Return input, bb $\leftarrow$ bound
4:  else
5:      Compute $\mathcal{E} = \max_i\{\min\{|x_{0_i} - 2^{\lfloor log_2(x_{0_i})\rfloor}|, |x_{0_i} - 2^{\lceil log_2(x_{0_i})\rceil}|\}\}$
6:      if $\mathcal{E} < \epsilon$ or iter > maxiter1 then
7:          if $\mathcal{L}_0 <$ bound then
8:              $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_0, \tilde{\mathcal{L}} \leftarrow \mathcal{L}_0$, bb $\leftarrow \mathcal{L}_0$
9:          else
10:             Return input, bb $\leftarrow$ bound
11:         end if
12:         Return
13:     end if
14:     if iter > maxiter0 then
15:         Find an entry $k = \operatorname{argmin}_i |x_{0_i} - 2^{\left[log_2(x_{0_i})\right]}| < \epsilon$ for $i = 1, \ldots, \binom{n}{2}$
16:         Set $x_{0_k} = 2^{\left[log_2(x_{0_k})\right]}$ and update $A_{eq}, \mathbf{b}_{eq}$
17:     end if
18:     Select a branching variable $x_{0_j}$
19:     Build subproblem $\mathcal{P}_1$ : Set $x_{0_j} \leq 2^{\lfloor log_2(x_{0_j})\rfloor}$ in $\{A, \mathbf{b}\} \rightarrow \{A_1, \mathbf{b}_1\}$
20:     Build subproblem $\mathcal{P}_2$ : Set $x_{0_j} \geq 2^{\lceil log_2(x_{0_j})\rceil}$ in $\{A, \mathbf{b}\} \rightarrow \{A_2, \mathbf{b}_2\}$
21:     iter $\leftarrow$ iter + 2
22:     Call branching routine for $\mathcal{P}_1 \rightarrow \mathbf{x}_1, \mathcal{L}_1$, status1, bound1
23:     if bound1 < bound and status1 = feasible then
24:         $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_1, \tilde{\mathcal{L}} \leftarrow \mathcal{L}_1$, bound $\leftarrow$ bound1, bb $\leftarrow$ bound1, status $\leftarrow$ status1
25:     else
26:         Return $\mathcal{P}_1$ input data, bb $\leftarrow$ bound
27:     end if
28:     Call branching routine for $\mathcal{P}_2 \rightarrow \mathbf{x}_2, \mathcal{L}_2$, bound2, status2
29:     if bound2 < bound and status2 = feasible then
30:         $\tilde{\mathbf{x}} \leftarrow \mathbf{x}_2, \tilde{\mathcal{L}} \leftarrow \mathcal{L}_2$, bb $\leftarrow$ bound2, status $\leftarrow$ status2
31:     end if
32: end if
```

Figure 8: Branching algorithm using a LNS heuristic. Once we specify a tolerance, the algorithm determines if a decision variable can be fixed before branching.

4 Neighbor Joining and Edge Walking

The most freqently used method for distance-based phylogeny reconstruction is Neighbor Joining (NJ), developed in [SN87b]. As shown in [GS06], NJ is a greedy algorithm for finding the balanced minimal evolution tree. [EHPY08] shows NJ loses accuracy quickly at the point of considering seven or eight taxa (leaves). For eight taxa the theoretical accuracy is between 69% and 72% for trees with more than two cherries, but drops to 62% for caterpillar trees. The conjecture in [EHPY08] is that the accuracy will continue to drop quickly, especially for caterpillar trees with more taxa. Our algorithm, on the other hand, experimentally shows 100% accuracy up to 11 taxa, even in the caterpillar case, with and without noise.

4.1 NNI and SPR moves, and FastMe 2.0

In an ideal situation, where the entire polytope is known via its facet inequalities, then linear programming via the simplex method has a nice geometrical interpretation. The pivot moves and row reductions in the simplex method correspond to moving from vertex to vertex of the polytope along edges–each time choosing the edge which most improves the objective function.

In the non-ideal situation, working with a relaxation of the polytope is the best we can do if we want to use the inequalities. If, alternatively, we know some subset of the edges, we can attempt a solution that uses any known edge in order to move to vertices that improve the objective.

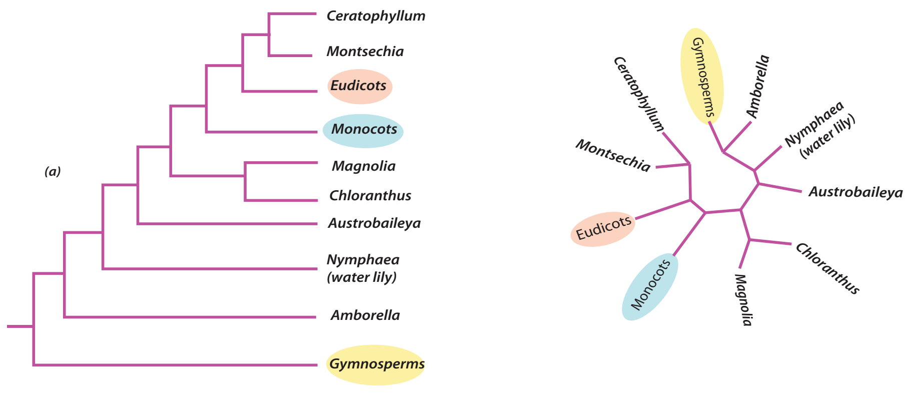
The best current published improvement on NJ is the algorithm known as FastMe and FastMe2.0, [DG02, LDG15]. This algorithm, piggy-backed on NJ, performs nearest-neighbor interchanges (NNI) and subtree-prune-and-regraftings, (SPR in 2.0) on the tree candidate. As shown in [HHY11b], both those operations correspond to edges of the BME polytope. There are many more edges in general. Thus FastMe uses edge walking on a subset of the edges of BME(n), which is essentially the bottom-up analog of our top-down use of facet inequalities. In [DG02] the FastMe algorithm is shown to improve on NJ by between 3.5% and 7% for 24 taxa, and as much as 21.3% for 96 taxa. While it is difficult to make a conclusive statement without further development of our PolySplit algorithm for greater numbers of taxa, it appears that our algorithm has the potential to improve even more, based on our 100% accuracy rate with up to 11 taxa. In order to test higher numbers, we plan to develop heuristics for selectively decreasing the number of facets from our list. The number of split facets, for instance, grows like 2^n, so we need to use a subset in order to manage the run-time for $n > 12$.

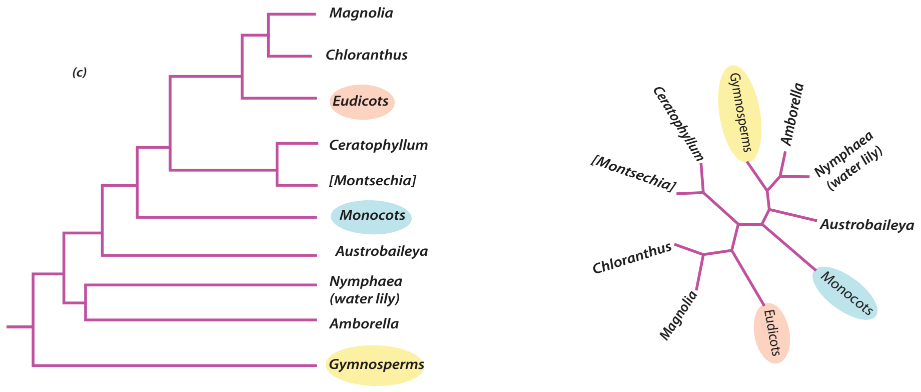
5 Summary

There are many phylogenetic questions that remain unanswered in biology and we have addressed only one method of tree reconstruction in this chapter. We have, however, demonstrated a new algorithm for finding the balanced minimal evolution tree. This approach, accompanied by the general introduction to the branch and bound method of linear programming using the discrete integer set of powers-of-two, can be used to explore these unanswered questions. We hope that the approaches in this chapter will provide a new and exciting method of phylogenetic reconstruction, capable of accounting for the very large datasets biologists are able to generate.

Additionally, there are many other methods that can be used to approximate a phylogenetic tree. For instance, in the case of a zero-noise, additive tree metric, the BME tree can be reconstructed directly by leaf insertion. Maximum likelihood (ML) and Bayesian methods are commonly used by biologists. ML methods assume a probabilistic model of mutations and choose the tree with the maximum probability. Bayesian methods require a series of model parameters which are used to identify a concensus tree. Bayesian methods tend to be preferred among biologists as they account for phylogenetic uncertainty. Probabilistic methods can be combined with BME: the ML method is used to build the dissimilarity (distance) matrix and then BME is used to construct the tree. Alternative linear programming approaches to BME, such as that in [CASP15], have also been tested on large sets of taxa. A natural extension of the work presented in this chapter would be to compare the various linear programming methods to identify areas where the approaches could be combined to further optimize reconstructions.

6 Answers to exercises, worked exercises.



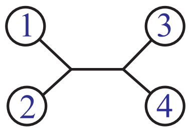
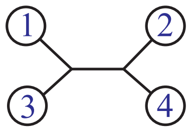
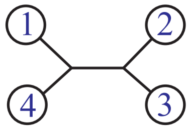


Exercise 1.1.1

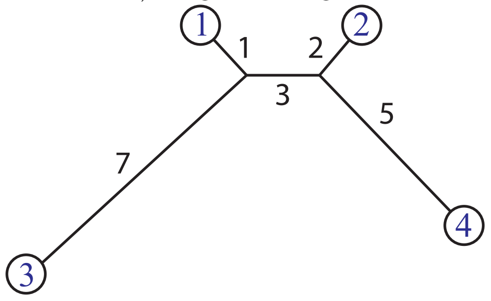
Alternate tree diagrams, or cladograms, and unrooted versions for the trees in Figure 2 (a,c).

Exercise 1.3.1

Recall that the vector of distance is $d = (6, 8, 9, 12, 7, 15)$.

t	$\boldsymbol{x}(t)$	$\boldsymbol{d}\cdot\boldsymbol{x}(t)$
	(2, 1, 1, 1, 1, 2)	78
	(1, 2, 1, 1, 2, 1)	72
	(1, 1, 2, 2, 1, 1)	78

So we choose the tree with minimal objective function value. After assigning variables to the edges and solving the 6 equations (one for each entry in the the distance vector) we get the original tree:

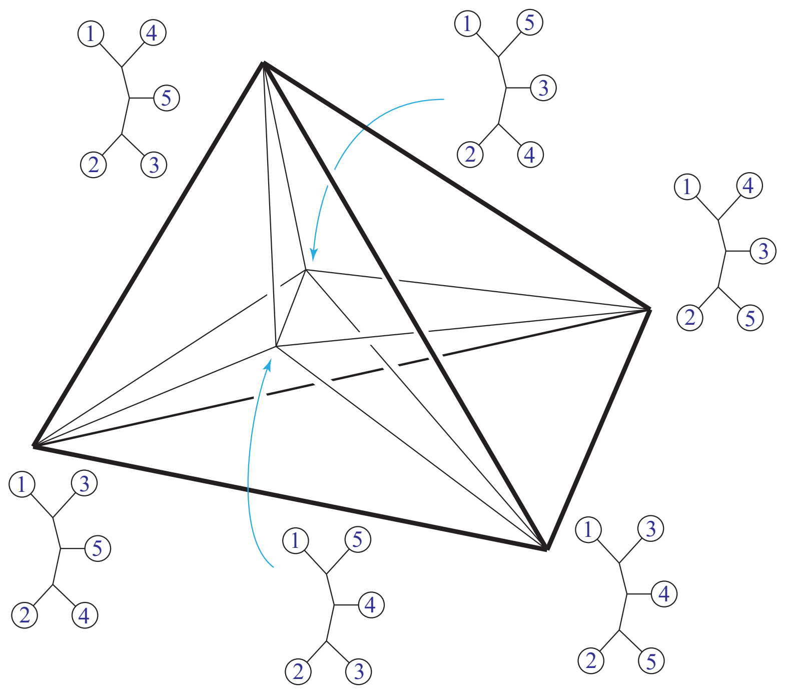


Exercise 2.2.1 Find the vertices for the 5 dimensional BME polytope. Use software (such as polymake) to find the structure of the three types of 4-dimensional facets with their inequalities for the 5-dimensional BME polytope. Here are the points, formatted for polymake with the extra initial coordinate, 1. (`shell.polymake.org`):

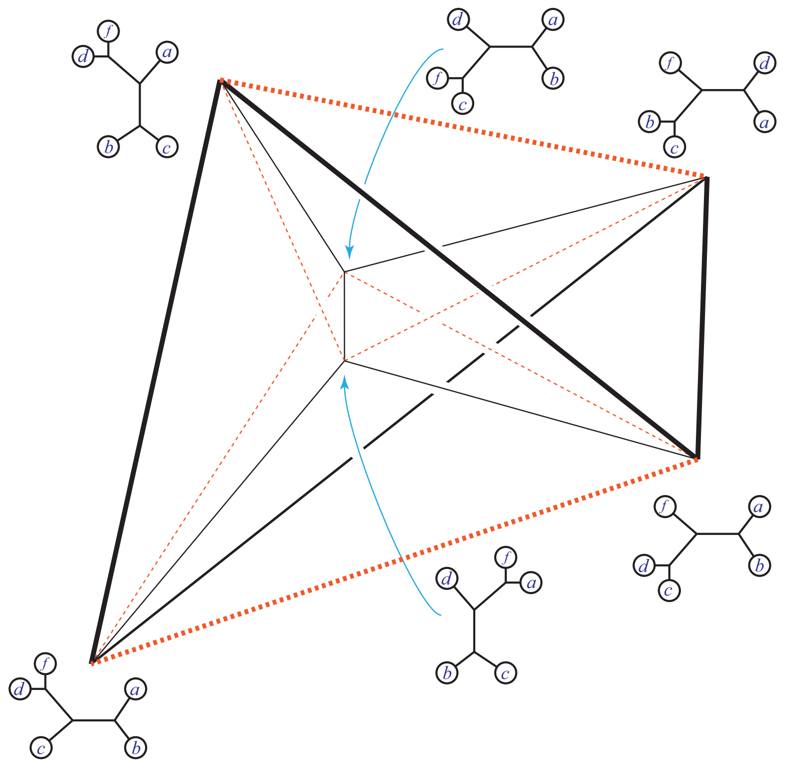
```
polytope > $points=new Matrix([[1,4,1,1,2,1,1,2,4,2,2],[1,4,2,1,1,2,1,1,2,2,4],
[1,4,1,2,1,1,2,1,2,4,2],[1,2,1,4,1,2,2,2,1,4,1],[1,2,2,2,2,1,4,1,1,4,1],
[1,1,4,1,2,1,4,2,1,2,2], [1,1,2,1,4,2,4,1,2,2,1],[1,2,1,1,4,2,2,2,4,1,1],
[1,1,1,2,4,4,2,1,2,1,2], [1,1,1,4,2,4,1,2,1,2,2],[1,2,2,2,2,4,1,1,1,1,4],
[1,2,4,1,1,2,2,2,1,1,4], [1,1,4,2,1,1,2,4,2,1,2],[1,1,2,4,1,2,1,4,2,2,1],
[1,2,2,2,2,1,1,4,4,1,1]]);
$p=new Polytope(POINTS=>$points); print $p->F_VECTOR;
...outputs... 15 105 250 210 52
```

Here are the three types of facets, with vertices labeled as in [FKS16a].

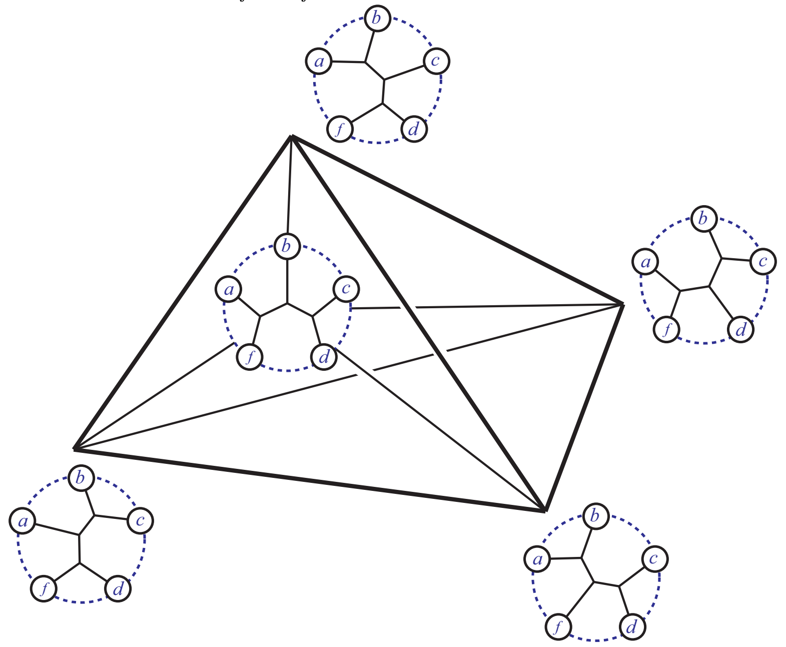
$x_{12} \geq 1$



$x_{ab} + x_{bc} - x_{ac} \leq 4$



$x_{ab} + x_{bc} + x_{cd} + x_{df} + x_{fa} \leq 13$



Exercise 2.3.1 Find the vertices for the 3 and 5-dimensional splitohedra.

The vertices of the 3-dimensional splitohedron are the same as the vertices for the 3-dimensional BME polytope. The only facets are the caterpillar and intersecting cherry facets.

The vertices of the 5-dimensional splitohedron are found by entering the caterpillar facet inequalities, intersecting-cherry facets, and cherry faces to polymake [GJ00]. The vertices output are as follows, where the highlighted vertices are from the original BME polytope–note that they are all present. Recall that polymake inserts an extra '1' as first coordinate.

```
(1, 2, 1, 4, 2, 4, 1, 2, 2, 1)
(1, 2, 4, 1, 2, 1, 4, 2, 2, 1)
(1, 4, 2, 1, 1, 2, 4, 2, 1, 2)
(1, 1, 2, 4, 4, 2, 1, 2, 1, 2)
(1, 1, 4, 2, 4, 1, 2, 1, 2, 2)
(1, 4, 1, 2, 1, 4, 2, 1, 2, 2)
(2, 1, 4, 1, 2, 2, 2, 1, 4, 1)
(8/3, 4/3, 8/3, 4/3, 4/3, 4/3, 8/3, 8/3, 8/3, 4/3)
(2, 1, 1, 4, 2, 2, 2, 4, 1, 1)
(4/3, 4/3, 8/3, 8/3, 8/3, 8/3, 4/3, 4/3, 8/3, 4/3)
(4/3, 8/3, 4/3, 8/3, 8/3, 8/3, 4/3, 4/3, 4/3, 8/3)
(4, 1, 2, 1, 1, 2, 1, 2, 4, 2)
(4, 2, 1, 1, 2, 1, 1, 2, 2, 4)
(8/3, 4/3, 4/3, 8/3, 4/3, 8/3, 4/3, 8/3, 8/3, 4/3)
(2, 2, 2, 2, 1, 1, 4, 4, 1, 1)
(2, 2, 2, 2, 1, 4, 1, 1, 4, 1)
(4/3, 8/3, 8/3, 4/3, 8/3, 4/3, 8/3, 4/3, 4/3, 8/3)
(4/3, 8/3, 8/3, 4/3, 4/3, 8/3, 8/3, 4/3, 8/3, 4/3)
(4, 1, 1, 2, 1, 1, 2, 4, 2, 2)
(8/3, 4/3, 4/3, 8/3, 8/3, 4/3, 4/3, 8/3, 4/3, 8/3)
(8/3, 4/3, 8/3, 4/3, 8/3, 4/3, 4/3, 4/3, 8/3, 8/3)
(2, 2, 2, 2, 4, 1, 1, 1, 1, 4)
(8/3, 8/3, 4/3, 4/3, 4/3, 8/3, 4/3, 4/3, 8/3, 8/3)
(8/3, 8/3, 4/3, 4/3, 4/3, 4/3, 8/3, 8/3, 4/3, 8/3)
(2, 4, 1, 1, 2, 2, 2, 1, 1, 4)
(4/3, 4/3, 8/3, 8/3, 8/3, 4/3, 8/3, 8/3, 4/3, 4/3)
(4/3, 8/3, 4/3, 8/3, 4/3, 8/3, 8/3, 8/3, 4/3, 4/3)
```

Exercise 3.1.1 Perform branch and bound. Maximize $p = 6.75x + 5y$ subject to

$$-x \leq 0$$

$$-y \leq 0$$

$$y \leq 10$$

$$x \leq 8.3$$

$$79x + 18y \leq 693.5$$

Require all coordinates of answer (x, y) to be powers of 2.

1) Run the LP solver (such as simplex method) to get answer zero: vector x_0 and objective function value p_0.

2) If x_0 has all coordinates powers of 2, then we say it is *complete*, and it is our final answer.

3) If not, then we create some new LP problems 1_A, 1_B, etc. by adding new inequalities one at a time, just enough to force an offending coordinate away from its disallowed value.

4) We solve each of these (as long as they are still *feasible*, that is, as long as the intersection of the inequalities is non-empty) to get answers x_{1A}, x_{1B}, p_{1A}, $p_{1B} \ldots$ etc.

5) For each new answer we check whether it is complete, and if not whether it *merits further branching* into more new problems; that is whether it has an objective function value better than the current best value given by a complete answer.

6) The process ends when no more branching is indicated; and the final answer is the optimal one from among the complete answers found.

Maximize p = 6.75x + 5y subject to

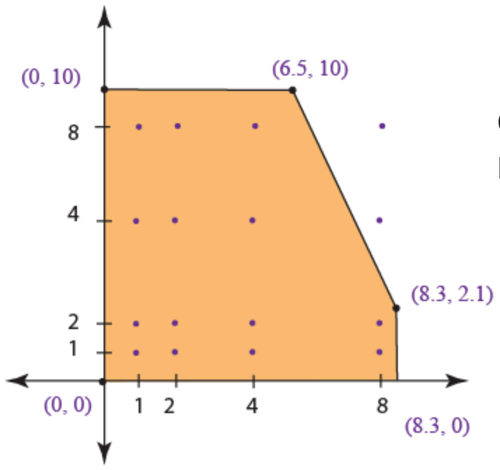
-x <= 0

-y <= 0

y <= 10

x <= 8.3

79x + 18y <= 693.5   Require all coordinates of answer (x,y) to be powers of 2.



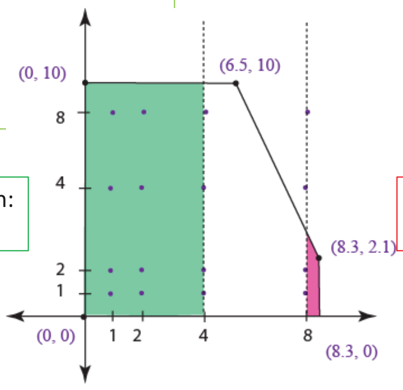
Optimal Solution:
p = 93.87; x = 6.5, y = 10

Answer zero: p = 93.87; x = 6.5, y = 10

Branches: x<=4 x>=8

Maximize p = 6.75x + 5y subject to
-x <= 0
-y <= 0
y <= 10
x <= 8.3
79x + 18y <= 693.5
x <= 4

Maximize p = 6.75x + 5y subject to
-x <= 0
-y <= 0
y <= 10
x <= 8.3
79x + 18y <= 693.5
x >= 8



1A: Optimal Solution:
p = 77; x = 4, y = 10

1B: Optimal Solution:
p = 71.08; x = 8, y = 3.417

Answer 1A: p = 77; x = 4, y = 10

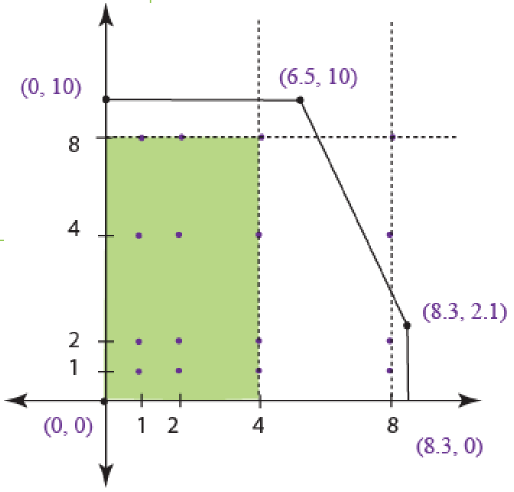
Branches: y <= 8 y >= 16

Maximize p = 6.75x + 5y subject to
-x <= 0
-y <= 0
y <= 10
x <= 8.3
79x + 18y <= 693.5
x <= 4
y <= 8

2B: Not feasible.

2A: Optimal Solution:
p = 67; x = 4, y = 8

Complete solution.

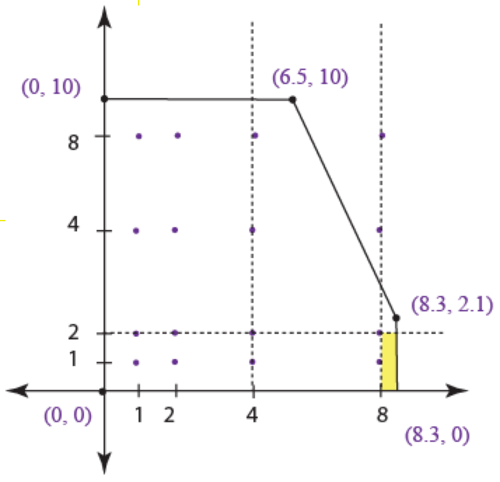


Answer 1B: p = 71.08; x = 8, y = 3.417

Branches: y <= 2 y >= 4

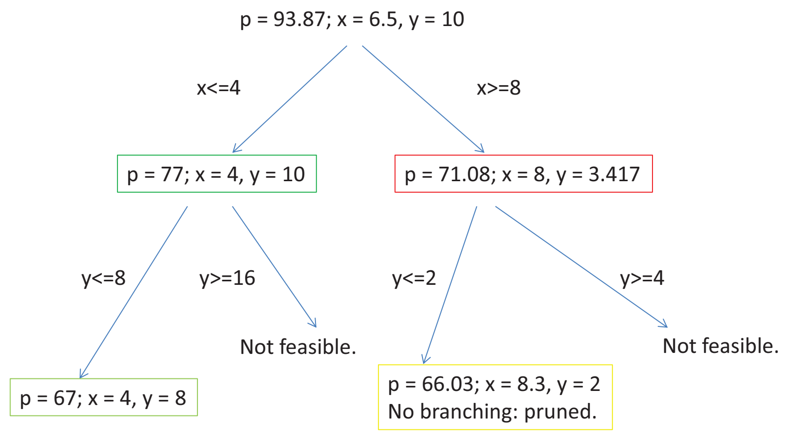
Maximize p = 6.75x + 5y subject to
-x <= 0
-y <= 0
y <= 10
x <= 8.3
79x + 18y <= 693.5
x >= 8
y <= 2

2D: Not feasible
(x >= 8 and y >= 4).



2C: Optimal Solution:
p = 66.03; x = 8.3, y = 2

66.03 < 67 (best complete solution) so: No further branching.



So final answer is p=67 at x=4, y=8.

Notice that we simply chose to branch first on x this time, whenever there was an option. That is, we explored the two subproblems created by adding inequalities involving x. Try the problem again using branching first on y, and note that the same answer will be achieved–but will require an extra level of branching. This highlights the value of a good criterion for selecting the branch variable.

Exercise 3.2.1 Given the current solution

$\mathbf{x} = (2, 4, 3.25, 3.42, 2, 3.68, 1.33, 1, 8, 4, 2, 2, 4, 8, 1.5)$, use the expression (15) to determine the branching variable and write down the two bounds generated by branching using (16) and (17).

Finding the distances from these values to the closest power of two in each case gives this vector of distances:

$$(0, 0, 0.75, 0.58, 0, 0.32, 0.33, 0, 0, 0, 0, 0, 0, 0, 0.5)$$

The maximum entry is 0.75, so the branching variable is the third variable, x_3 or $x_{1,2}$ (referring to leaves 2 and 3 of the 6 total leaves). Thus the new bounds are respectively $x_{1,2} \geq 4$ and $x_{1,2} \leq 2$.

Exercise 3.3.1 The output vector is $x(t) = [$ 16 8 32 64 4 2 1 1 32 32 16 16 8 4 4 16 8 32 16 8 8 32 8 4 2 2 4 2 1 1 32 16 16 32 32 64 $]$. Your tree will have two cherries, one for leaves 1 and 5 (drawn on the left), the other for leaves 8 and 9 (drawn on the right). The other leaves will be ordered 4,2,3,6,7 between the cherries from left to right.

References

[AKM05] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, (33):42–54, 2005.

[Bau08] David Baum. Reading a phylogenetic tree: the meaning of monophyletic groups. *Nature Education*, 1(1):190, 2008.

[BCM$^+$00] Todd J. Barkman, Gordon Chenery, Joel R. McNeal, James Lyons-Weiler, Wayne J. Ellisens, Gerry Moore, Andrea D. Wolfe, and Claude W. dePamphilis. Independent and combined analyses of sequences from all three genomic compartments converge on the root of flowering plant phylogeny. *Proceedings of the National Academy of Sciences*, 97(24):13166–13171, 2000.

[CASP15] Daniele Catanzaro, Roberto Aringhieri, Marco Di Summa, and Raffaele Pesenti. A branch-price-and-cut algorithm for the minimum evolution problem. *European Journal of Operational Research*, 244(3):753 – 765, 2015.

[Cat07] D. Catanzaro. The minimal evolution problem: Overview and classification. *Networks*, 53(2):112–125, 2007.

[CDvM$^+$06] Francesca D. Ciccarelli, Tobias Doerks, Christian von Mering, Christopher J. Creevey, Berend Snel, and Peer Bork. Toward automatic reconstruction of a highly resolved tree of life. *Science*, 311(5765):1283 – 1287, March 2006.

[CLPSG12] Daniele Catanzaro, Martine Labbé, Raffaele Pesenti, and Juan-José Salazar-González. The balanced minimal evolution problem. *INFORMS Journal on Computing*, 24(2):276–294, 2012.

[DG02] Richard Desper and Olivier Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *J. Comp. Biol.*, 9(5):687–705, 2002.

[DRLP05] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[EHPY08] K. Eickmeyer, P. Huggins, L. Pachter, and R. Yoshida. On the optimality of the neighbor-joining algorithm. *Algorithms for Molecular Biology*, 3(5), 2008.

[FKS16a] S. Forcey, L. Keefe, and W. Sands. Facets of the balanced minimal evolution polytope. *Journal of Mathematical Biology*, 73(2):447–468, 2016.

[FKS16b] S. Forcey, L. Keefe, and W. Sands. Split facets of balanced minimal evolution polytopes and the permutoassociahedron. *Bulletin of Mathematical Biology*, 79(5):975–994, 2016.

[FOR17] B. Fortz, O. Oliveira, and C. Requejo. Compact mixed integer linear programming models to the minimum weighted tree reconstruction problem. *European Journal of Operations Research*, 256:242–251, 2017.

[GDGC$^+$15] Bernard Gomez, Véronique Daviero-Gomez, Clément Coiffard, Carles Martín-Closas, and David L. Dilcher. Montsechia, an ancient aquatic angiosperm. *Proceedings of the National Academy of Sciences*, 112(35):10985–10988, 2015.

[GJ00] Ewgenij Gawrilow and Michael Joswig. polymake: a framework for analyzing convex polytopes. In Gil Kalai and Günter M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000.

[GS06] O. Gascuel and M. Steel. Neighbor-joining revealed. *Molecular Biology and Evolution*, 23:1997–2000, 2006.

[HHY11a] D. Haws, T. Hodge, and R. Yoshida. Optimality of the neighbor joining algorithm and faces of the balanced minimum evolution polytope. *Bulletin of Mathematical Biology*, 73(11):2627–2648, 2011.

[HHY11b] David C. Haws, Terrell L. Hodge, and Ruriko Yoshida. Optimality of the neighbor joining algorithm and faces of the balanced minimum evolution polytope. *Bull. Math. Biol.*, 73(11):2627–2648, 2011.

[LDG15] Vincent Lefort, Richard Desper, and Olivier Gascuel. Fastme 2.0: a comprehensive, accurate, and fast distance-based phylogeny inference program. *Molecular Biology and Evolution*, 2015.

[Mor11] Cynthia M. Morton. Newly sequenced nuclear gene (xdh) for inferring angiosperm phylogeny. *Ann. of Missouri Botanical Garden*, 98(1):63–89, 2011.

[MSM10] D.R. Maddison, K.S. Schulz, and W.P. Maddison. The tree of life web project. *Linnaeus Tercentenary: Progress in Invertebrate Taxonomy. Zootaxa 1668:1-766*, pages 19 – 40, 2010.

[Pau00] Y. Pauplin. Direct calculation of a tree length using a distance matrix. *Journal of Molecular Evolution*, 51(1):41–47, 2000.

[Ryt04] W. Rytter. Trees with minimum weighted path length. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, chapter 10, pages 1–22. Chapman and Hall/CRC, 2004.

[San17] W. Sands. Thesis: Phylogenetic inference using a discrete-integer linear programming model. *etd.ohiolink.edu*, 2017.

[SLV09] Marco Salemi, Philippe Lemey, and Anne-Mieke Vandamme. *The phylogenetic handbook: a practical approach to phylogenetic analysis and hypothesis testing*. Cambridge University Press, 2009.

[SN87a] N. Saitou and M. Nei. The neighbor joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[SN87b] N Saitou and M Nei. The neighbor joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.