Computing Steiner Minimum Trees in Hamming Metric

Ernst Althaus* Rouven Naujoks†

Abstract

Computing Steiner minimum trees in Hamming metric is a well studied problem that has applications in several fields of science such as computational linguistics and computational biology. Among all methods for finding such trees, algorithms using variations of a branch and bound method developed by Penny and Hendy have been the fastest for more than 20 years. In this paper we describe a new pruning approach that is superior to previous methods and its implementation.

1 Introduction

We are interested in computing a Steiner minimum tree in Hamming metric. Given a set $T \subseteq U$ of required points (terminals) in an universe U and a cost function $c : U \times U \mapsto \mathbb{R}$, a Steiner tree is a tree connecting $T \cup S$ for a subset $S \subseteq U$. The elements of S are called the Steiner points. A Steiner minimum tree $\mathrm{SMT}(T)$, is a Steiner tree of minimal cost.

The Steiner tree problem is one of the most studied NP-hard optimization problems. In this paper we elaborate on the variant where U is the set of strings of size d over a finite alphabet Σ and where c is the Hamming distance between two strings, i.e. the number of characters in which two strings differ. As the general version this variant is known to be NP-hard too [6].

One main application of the Steiner tree problem in Hamming metric is the computation of evolutionary trees in bioinformatics (in this context also called phylogenetic trees) and in computational linguistics. We elaborate on the use in bioinformatics.

The problem in bioinformatics reads as follows: Given a set of species, we want to determine their ancestral relationship. In order to build the tree, we compare specific features of the species under the natural assumption that species with similar features are closely related. Building a tree implicitly makes the assumption that all species have evolved from a common ancestor and no recombination has occurred (which is generally not true). Classic phylogenetics dealt mainly with physical features whereas modern phylogeny is based mainly on DNA and protein sequences. The literature on phylogenetics is very rich. For a starting point, we refer to [10].

The problem of computing a phylogenetic tree is to find one or all trees that optimize a certain objective function. There are two major versions of the problem. The maximum parsimony tree is the tree which minimizes the number of changes of the genetic code along the edges of the tree. This problem is equivalent to the Steiner minimum tree in Hamming metric problem as described above. The second prominent version is the maximum likelihood tree where the assumption is made that the characters are pairwise independent and that the branching is a Markov process. The optimal tree is the most likely one under this process. Even though both methods are currently used by biologists there is no clear answer to the question which of the methods provides a more "realistic" reconstruction of a phylogeny. We refer to [16] for a more detailed discussion of this prolem.

Almost all algorithms for the Steiner tree in Hamming metric problem enumerate tree structures spanning the terminals and a set of Steiner points. For each of these tree structures (also called topologies) they determine an SMT and choose among them the ones with minimal cost. Usually an $O(nd|\Sigma|)$ time algorithm found by Fitch [5] is used to determine such an SMT for a given topology.

Clearly, we profit from a set of topologies that is as small as possible. Therefore, we enumerate only topologies of the special form that each terminal has degree exactly one and each Steiner point has degree exactly three. The existence of such an SMT is folklore. Even under this assumption on the degrees, the number of topologies grows super-exponential with the number of terminals. More precisely, for an instance with n terminals the number of topologies is $\prod_{i=2}^{n-1}(2i-3)$. For $n = 20$ this value is about $2.2 \cdot 10^{20}$ which shows that even for very small instances doing a brute force search is not applicable.

In general, we restrict our attention to SMTs whose topologies satisfy this degree assumption unless we state otherwise.

There are many implementations that tackle the

*LORIA, Université Henri Poincaré, Vandœuvre-lès-Nancy, France,althaus@loria.fr

†Max-Planck-Institut für Informatik, Saarbrücken, Germany, naujoks@mpi-sb.mpg.de

maximum parsimony problem (see e.g. [19] for a big collection). All (reasonable) exact versions we are aware of (e.g. [4, 17, 9]) are based on variants of the following branch-&-bound approach found by Hendy and Penny [7]. Start with the unique topology $\mathcal{T}$ of three (cleverly chosen) terminals. We recursively add a (cleverly chosen) terminal in all possible ways (in a clever order) to the current topology. If the cost of the current topology $\mathcal{T}$ plus a lower bound for adding the remaining terminals to it is larger than the best known Steiner tree, we can stop the recursion. For details we refer to [12].

We now sketch our approach and elaborate on the differences. The algorithm is described in more detail in Section 4. We call a topology spanning all terminals a *full topology* otherwise a *partial topology*. We consider the topologies as rooted binary trees by inserting an artificial root into an arbitrary edge.

First we construct all rooted topologies of all subsets of the terminals up to cardinality $\lfloor n/2 \rfloor$, which we cannot provably exclude from being part of an optimal topology.

In a second phase we construct the full topologies by combining all possible triples of partial topologies whose terminals are a partition of T.

We consider terminals as partial topologies, where the terminal itself is the root. All other partial topologies are created by connecting a new root to two roots of existing partial topologies. The partial topologies spanning k terminals are constructed by combining all partial topologies spanning i and $k - i$ terminals ($1 \leq i < \lfloor k/2 \rfloor$) whose terminal sets are disjoint.

Note that both approaches enumerate substructures of all topologies and try to prune many of those substructures. The main difference of our approach is that the structure of a partial topology is conserved. For example, if we want to add a terminal in the old approach, we can do this at any position. In our approach, we have to add it at the root. This extends the possibilities of the pruning tests (compare e.g. Section 5). Note that due to the super-exponential growth of the number of topologies, the number of substructures enumerated is smaller than the number of full topologies. Nevertheless, we can show that if nothing can be pruned, we enumerate less substructures than in the old approach.

2 Related Work

There are many papers on computational methods for solving the maximum parsimony problem. Almost all papers (we are aware of) that describe exact methods to solve the maximum parsimony problem describe variants of the branch-&-bound algorithm outlined in the introduction (e.g. [4, 17, 9]).

Holland et al. [8] try to prove the optimality of parsimony trees by computing lower bounds and hoping for a match with the upper bound. They use one- and two-column discrepancy bounds (see Section 6) and try to combine these to lower bounds by a very simple relaxation of the integer linear program defined in Section 6.

A similar approach to ours is used to solve the Steiner minimum tree problem in the Euclidean plane [18]. We outline this approach and elaborate on the differences. For a Steiner tree ST, we call the maximum subtrees whose internal vertices are Steiner points the full components of ST. We can subdivide the computation of an Steiner tree into two phases. We first compute a superset X of the full components of ST and then find the minimum cost subset of X that yields a tree. The second part is typically solved by an integer linear programming approach. The superset of the full components is computed by enumerating topologies in the same way as in our algorithm. One main difference in the case of the Euclidean plane is that the size of the full components is very small (typically less than 10, even on instances with 10.000 terminals). Although there are instances where the algorithm will create an exponential number of candidate full components, experiments indicate an about linear number of full components on random instances. In Hamming metric, the SMT consists typically only of a small number (if not even 1) of full components. Since the efficiency of their approach heavily depends on partitioning the instance into many small full components one cannot apply this technique to our problem setting.

Therefore, we decided to enumerate topologies. Nevertheless the pruning ideas of creating full components can be adopted and generalized to prune partial topologies.

The most successful algorithm to solve the Steiner minimum tree problem in graphs (i.e. the universe is the set of all vertices of the graph, the terminals are a subset of the vertices and the distances are given by costs of the edges) is also based (among other techniques) on successfully pruning parts of the graph. All known techniques are described in the recent PhD thesis of Polzin [11]. We can adopt some of these pruning techniques for our work.

We now restate Fitch's Algorithm: Given a topology $\mathcal{T}$, we want to compute strings for the Steiner points of $\mathcal{T}$ such that the cost of the corresponding tree is minimal. This can be done as follows:

First notice that we can consider the characters independently of each other one by one. Thus, when describing the algorithm, we can assume that the strings have length one. Assume we want to connect three

terminals to a common Steiner point s. The letter of s has to be chosen as the majority - if there is one - of the letters of the three terminals. If all three letters are different, any of the three letters can be chosen.

We consider the topology as a rooted tree, as described in the introduction. The main insight is that we can chose the letter of a Steiner point s such that the cost of its subtree is optimal as the connection of s to its parent costs at most one less for any other choice for the letter of s.

In general, we have a two-phase approach. In the first phase, we traverse the tree bottom up computing the set of letters for the Steiner points such that the cost of the subtree below the Steiner point is minimized. We call this set of letters the candidate set. Having computed the candidate set of the root of the topology, we arbitrarily fix the letter of the root to one of the candidates and traverse the tree from top to bottom fixing the letters of the Steiner points appropriately.

Let u be a node of the tree and $range(v)$ and $range(w)$ the candidate sets of the two children v and w, respectively. If $range(v) \cap range(w) \neq \emptyset$, we can select any letter of this intersection to construct a tree whose cost is the sum of the costs of the two subtrees. Hence the candidate set of u is $range(v) \cap range(w)$ in this case. If $range(v) \cap range(w) = \emptyset$, the children of u will definitely have different letters. Hence the minimal cost subtree will be attained by one element in $range(v) \cup range(w)$ and with a cost one more than the sum of the cost of the subtrees.

The second phase is simple. If we know the letter of the parent of a node u, we select the same letter for u if it is in the candidate set. Otherwise, we select an arbitrary letter from the candidate set.

For a detailed description and a complete correctness proof, we refer to [5].

3 Notation and Definitions

Given a set $T \subseteq \Sigma^d$ of n required points (*terminals*) and a cost function $c : \Sigma^d \times \Sigma^d \mapsto \mathbb{R}$, a *Steiner tree* is a tree connecting $T \cup S$ for a subset $S \subseteq \Sigma^d$. The elements of S are called the Steiner points. The cost of a Steiner tree is the sum of the costs of its edges. A *Steiner minimum tree* SMT(T), is a Steiner tree of minimal cost. For a string s we denote by s_i the i-th character of s. For $s, t \in \Sigma^d : c(s,t) := \|s,t\|$, where $\|s,t\|$ denotes the *Hamming distance* between s and t, i.e. $\|s,t\| = |\{i \in \{1,\ldots,d\} \mid s_i \neq t_i\}|$.

We call $p \in \Sigma^d$ a *point* and $r \in P(\Sigma)^d$ a *range*, where $P(\Sigma)$ denotes the power set of Σ. For a point p and a range r we write $p \in r$ if $p_i \in r_i \; \forall \; i \in \{1,\ldots,d\}$. Given a Steiner tree ST of a subset $T' \subseteq T$, we call the underlying graph structure the *topology* $\mathcal{T}$ of ST. We call $\mathcal{T}$ a *full topology* if $T' = T$ and we call $\mathcal{T}$ a *partial topology* if $T' \subsetneq T$. $\mathcal{FT}$ denotes the set of all full topologies and $\mathcal{PT}$ the set of all partial topologies. The size of a topology $\mathcal{T}$ is defined as the number of terminals spanned by $\mathcal{T}$. We say, a topology $\overline{\mathcal{T}}$ is a *subtopology* of the topology $\mathcal{T}$ if $\overline{\mathcal{T}}$ is a subtree of $\mathcal{T}$. Given two partial topologies $\mathcal{P}_i$ and $\mathcal{P}_j$, we write $\mathcal{P}_i \cdot \mathcal{P}_j$ for the concatenation of $\mathcal{P}_i$ and $\mathcal{P}_j$, i.e. we connect the roots of P_i and P_j to a new root. We write $r(\mathcal{P})$ to denote the root of a topology $\mathcal{P}$. The range of the new root is computed from the ranges of the roots of P_i and P_j using Fitch's algorithm. We define a mapping from a node in a topology $\mathcal{P}$ to a range: For a node $u \in \mathcal{P}$, let $range(u)$ be the range assigned in the first phase of Fitch's algorithm. Depending on the context, a terminal is a leaf in a topology, the related point, or the related range with components of cardinality one. For a topology $\mathcal{T}$, let $span(\mathcal{T})$ be the set of terminals spanned by $\mathcal{T}$. We represent the partial topologies as rooted trees and write $r(\mathcal{T})$ for the root node of $\mathcal{T}$. Further, let $c(\mathcal{T})$ be the cost of the topology $\mathcal{T}$, i.e. the cost computed by Fitch's algorithm.

Let $\mathcal{PT}_i = \{\mathcal{P} \in \mathcal{PT} \mid |span(\mathcal{P})| = i\}$ be the set of partial topologies of cardinality i. Given a set of terminals T, the *Bottleneck Steiner distance* $bsd(u,v)$ between two nodes $u, v \in T$ is defined as the cost of the most expensive edge on the path between u and v in a minimum spanning tree. Note that this cost is independent of the choice of the minimum spanning tree. Finally, SMT(T) and MST(T) denote an arbitrary but fixed Steiner minimum tree and minimum spanning tree, respectively.

4 Pruning Algorithm

The pruning algorithm we use is conceptually quite simple: We start with the set $X := T$ of all terminals, i.e. topologies of size 1. We now construct consecutively bigger topologies using the smaller topologies contained in X and add them to X. Before adding a partial topology $\mathcal{P}$, it is checked by several pruning tests if $\mathcal{P}$ can be part of an optimal solution. If not, $\mathcal{P}$ will be discarded. After this construction phase the set of full topologies $\mathcal{FT}'$ is created by concatenating the topologies in X. Among them, the optimum is determined by brute force. For each topology $\mathcal{T}$ there is a Steiner point s such that each partial topology created by removing s and its adjacent edges from $\mathcal{T}$ spans at most $\lfloor n/2 \rfloor$[1] terminals. Hence, it is sufficient

[1] The intuition is as follows: Let s be any Steiner point and assume that the cardinality of one component that is created by removing s is more than $\lfloor n/2 \rfloor$. Consider the Steiner point s' adjacent to s in the component that is too large. Notice that for the Steiner point s', the size of the largest component created

to construct only the topologies up to a size of $\lfloor n/2 \rfloor$.

Algorithm 1 Pruning Algorithm

```
X := T
for i = 2 ... ⌊n/2⌋ do
  for all P_i, P_j ∈ X do
    if R(P_i) ∩ R(P_j) = ∅ and
       |R(P_i)| + |R(P_j)| = i then
      if ¬ prunable( P_i · P_j ) then
        X = X ∪ {P_i · P_j}
      end if
    end if
  end for
end for
construct full topologies FT' ⊆ FT
search for optimum in FT' by brute force
```

For constructing the full topologies we sort the non pruned partial topologies lexicographically: First we sort them by the spanned terminals and then by increasing length. This allows a fast construction of $\mathcal{FT}'$, and guarantees that each full topology in $\mathcal{FT}'$ is counted exactly once. A full topology is assembled by three topologies. Now assume that you have already build the topology $p_1 \cdot p_2$ using $p_1, p_2 \in X$. Then we have to concatenate $p_1 \cdot p_2$ with all $p_3 \in X$ such that $span(p_1 \cdot p_2 \cdot p_3) = T$. Therefore, we perform a binary search on the sorted list for the first topology t such that $span(t) = T \setminus span(p_1 \cdot p_2)$. Then we iterate over this list creating the full topology for each element until either a topology does not span the correct terminals anymore or $|p_1 \cdot p_2| + |p_3|$ is greater than the length of the minimal full topology found so far.

5 Pruning Tests

The efficiency of our algorithm mainly depends on the pruning tests. Given a partial topology $\mathcal{P} \in \mathcal{PT}$, we derive efficiently computable properties of $\mathcal{P}$ that exclude $\mathcal{P}$ from being part of an SMT. In this section, let $\mathcal{F}$ be a full topology containing $\mathcal{P}$ as a subtopology.

The following pruning tests are currently used in our implementation. Some of the ideas are adopted from known pruning tests of other variants of the Steiner tree problem. Note that due to lack of space we drop most of the details and present just the ideas and argue about correctness when it is important.

by removing s' is smaller than for s as the large component is split and the sum of the cardinalities of two components that are merged is at most $\lfloor n/2 \rfloor$. Hence, we can repeat this argument until all three components have size of at most $\lfloor n/2 \rfloor$.

We use the following notation: $\|x, y\|_{min}$ denotes the minimal distance between two ranges x and y, i.e. $\|x, y\|_{min} := \min\{ \|p_x, p_y\| \mid p_x \in x,\ p_y \in y \}$. Similarly, we define $\|x, y\|_{max} := \max\{ \|p_x, p_y\| \mid p_x \in x,\ p_y \in y \}$. Given an edge $e = (u, v)$ in $\mathcal{P}$ we write $\|e\|_{min} := \min\{ \|p_u, p_v\| \mid p_u \in range(u),\ p_v \in range(v) \}$.

At some points one of the simplifications we make is the assumption that edge lengths in a given topology are fixed. This is clearly not true in general unless the endpoints are terminals. A consideration of these situations usually does not change the idea of the tests itself but makes it more difficult to explain.

edge-length: Consider an edge $(u, v) \in \mathcal{P}$ and assume $\mathcal{P}$ is part an the optimal topology $\mathcal{T}$. If we remove (u, v) from $\mathcal{T}$, $\mathcal{T}$ will be split into two components. The length of an edge $(u, v) \in \mathcal{P}$ may not exceed the minimal distance between two points in different components. Otherwise we can construct a shorter Steiner tree by reconnecting the tree with a shortest of those edges.

MST-lb: We can prune $\mathcal{P}$ if the cost of the minimum spanning tree with respect to the bottleneck Steiner distances of $span(\mathcal{P})$ plus a lower bound on the length of an edge connecting $r(\mathcal{P})$ with a terminal $z \in span(\mathcal{P})$ is smaller than $c(\mathcal{P})$. This is in principle an extension of the above test to more than one edge. A proof (of a slight variant) can be found in [11]. We use two bounds on the lengths of the edge connecting $r(\mathcal{P})$ to a terminal $z \in span(\mathcal{P})$: 1.) the one of the pruning test above and 2.) $\min_{z \in span(\mathcal{P})} \max_{u \in range(r(\mathcal{P}))} \|z, u\|$.

lower bounds: Given an upper bound U for the SMT, we can prune $\mathcal{P}$, if the cost of $\mathcal{P}$ plus a lower bound for connecting the remaining terminals $T \setminus span(\mathcal{P})$ to the root $r(\mathcal{P})$ is larger than U. In the next section, we derive lower bounds for the length of an SMT.

empty ball property: Given an edge $e = (u, v)$ in $\mathcal{P}$, let x be a Steiner point of $\mathcal{P}$ or an arbitrary terminal (not necessarily in $\mathcal{P}$). Let $\mathcal{F}$ be a full topology such that $\mathcal{P}$ is a subtopology of $\mathcal{F}$. By removing e, the full topology $\mathcal{F}$ is decomposed into two components $\mathcal{F}_u$ and $\mathcal{F}_v$ such that $u \in \mathcal{F}_u$ and $v \in \mathcal{F}_v$. If $x \in \mathcal{P}$ it is clear in which component x lies. Otherwise x must have been connected to the component that contained the root of $\mathcal{P}$. Let us assume w.l.o.g. that $x \in \mathcal{F}_u$. We can prune $\mathcal{P}$ if $\|v, x\| < \|u, v\|$ since the resulting topology

by reconnecting $\mathcal{F}_u$ and $\mathcal{F}_v$ and by introducing the edge (x, v) has cost less than $\mathcal{F}$.

bottleneck Steiner distances: Consider an edge $e = (u, v)$ in an SMT ST. It is a well known fact, that $\|e\| \leq \min\{\ bsd(x, y) \mid x, y$ are terminals s.t. e is in the path from x to y in $ST\ \}$. In our situation this translates to: given an edge $e = (u, v)$ in $\mathcal{P}$, $\mathcal{P}$ can be pruned if $\|e\|_{min} > bsd(x, y)$. But we can even strengthen this pruning test: Consider a Steiner point $s \in \mathcal{P}$ and its adjacent nodes u, v and w. From the bottleneck Steiner distances we get bounds on the lengths of the edges (s, u), (s, v) and (s, w). Hence, we check whether we can select a position for s such that these three bounds are met. Another point is that we do not have to consider only terminals in $\mathcal{P}$. Using the same arguments as above we know to which "side" of s the non spanned terminals will be connected. Thus, we can consider all terminals instead of considering only the ones in $span(\mathcal{P})$.

triangle connectivity: Given a Steiner point $s \in \mathcal{P}$, removing s decomposes $\mathcal{P}$ into three components $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$. If there is a triple (x, y, z) with $x \in \mathcal{P}_1$, $y \in \mathcal{P}_2$, $z \in \mathcal{P}_3$ such that reconnecting the components via these points over a Steiner point leads to a topology that in turn leads to cheaper full topologies, $\mathcal{P}$ can be pruned.

incompatibility: Given a topology $\mathcal{P}'$ such that $span(\mathcal{P}) = span(\mathcal{P}')$, $\mathcal{P}$ can be pruned if one of the following conditions holds:

1. $|\mathcal{P}| > |\mathcal{P}'| + \|r(\mathcal{P}), r(\mathcal{P}')\|_{max}$
2. $|\mathcal{P}| + \|r(\mathcal{P}), T \setminus span(\mathcal{P})\|_* > |\mathcal{P}'| + min\{\ bsd(x, y) \mid x \in span(\mathcal{P}), y \in (T \setminus span(\mathcal{P}))\ \}$

where $\|r(\mathcal{P}), T \setminus span(\mathcal{P})\|_*$ denotes the minimal distance between the root of $\mathcal{P}$ and the root of a topology spanning the terminals that are not spanned by $\mathcal{P}$. It can be easily seen that $\|r(\mathcal{P}), T \setminus span(\mathcal{P})\|_* \geq \mid \{c \in \{1, \ldots, d\} \mid \forall t \in T : r(\mathcal{P})_c \cap t_c = \emptyset\} \mid$. In other words, P can be pruned because it can be substituted by P' such that the resulting topology is cheaper.

Until now we described conditions that can show the non-optimality of a given partial topology. In the following, we give methods that improve other pruning tests or that shrink the search space in advance, i.e. they decrease $|\mathcal{PT}|$ before we start with the pruning phase of the algorithm.

implicit lower bounds: Even if pruning tests like the "empty ball property" fail, they give upper bounds on the edge lengths. Consider a Steiner point $s \in \mathcal{P}$ and its three adjacent points p_1, p_2, p_3. Given an upper bound C on $|(p_1, s)| + |(p_2, s)| + |(p_3, s)|$ that can easily be computed and two upper bounds $|(p_1, s)| \leq b_1$ and $|(p_2, s)| \leq b_2$, we can deduce the lower bound $|(p_3, s)| \geq C - b_1 - b_2$. Since many pruning tests involve lower bounds on edge lengths, we can rerun these pruning tests with the deduced lower bounds.

pruning ranges: Since in general we have to cope with bounds on edge lengths, it is reasonable to make these bounds as tight as possible. One way is to prune ranges in a topology $\mathcal{P}$: Consider a column c of the root R of $\mathcal{P}$. If $R_c \cap t_c = \emptyset\ \forall\ t \in T \setminus span(\mathcal{P})$, we can clearly set $R_c := \{x\}$ for an arbitrary $x \in R_c$. Otherwise if there is an $y \in R_c$ such that $y \notin t_c\ \forall\ t \in T \setminus span(\mathcal{P})$, we can set $R_c := R_c \setminus \{y\}$.

pruning non-informative columns: In [10] the authors describe a preprocessing technique for pruning columns: a column is called "parsimony informative" if there are at least two letters that appear at least twice. If a column c is not parsimony informative, an optimal topology for the instance excluding c is also optimal for the complete instance. Instead of doing this only once, we recurse on the result as long as we can prune columns: after the non informative columns have been pruned, we check if some sequences are identical. In this case we delete these duplicates. Since by removing the identical sequences the occurrences of letters in a column can change, it can happen that an informative column becomes non informative.

In our implementation we also use specialized versions of the pruning tests which are faster to compute before we call the general pruning tests. This allows us to save some calls of more expensive pruning tests. As a simple example: before we compute the bottleneck-Steiner-distance test, we determine the biggest existing bottleneck Steiner distance D and check for all edges in $\mathcal{P}$ if they can be pruned using D. This is clearly a very cheap test since it is linear in the complexity of $\mathcal{P}$ but it usually prunes a lot of the topologies for which we would have to perform the more costly bottleneck-Steiner-distance test.

6 Lower Bounds

In this section we derive new lower bounds for SMTs. Note that we want to compute lower bounds for sets of ranges. Again for sake of simplicity we describe

our approach for points with fixed coordinates. The approach can easily be generalized for ranges.

The following simple lower bound is well known (see e.g. [8]). For each column, we compute the cost of the SMT only considering this column. Clearly the cost of the SMT is the number of different letters in the column minus one. This is called the *discrepancy* of this column. The sum of the discrepancies of the columns is a lower bound, called the *single column discrepany* lower bound.

Using the so called *partitioning theorem* (see [8]) this bound can be generalized as follows: Compute lower bounds lb_k for the SMT over subsets $C_k \subseteq \{1, \ldots, d\}$ of the columns. Then combine these lower bounds to a lower bound for the SMT over all columns with a relaxation of the following integer linear program (*ILP*) (see [8]). Assume the cost of the SMT in column i is x_i. Then we know that the cost of the SMT is $\sum_{i=1}^{d} x_i$. We know for the lower bound lb_k of the subset C_k that $\sum_{i \in C_k} x_i \geq lb_k$. Minimizing $\sum_{i=1}^{d} x_i$ over these constraints using the integer condition on x_i gives a lower bound on the cost of an SMT. In Section 6.3 we show how we solve the linear programming (*LP*) relaxation of this integer linear program combinatorially, when $|C_k| \leq 2$ for all computed lower bounds.

In the sequel, we consider lower bounds for several columns. The simplest lower bound is the number of pairwise distinct terminals minus one (which works for any number of columns and sometimes improves the single column discrepancy bound).

We now derive better bounds for two and three columns.

6.1 Two Column SMTs

LEMMA 6.1. *Let T be a set of strings of length two. $\mathrm{MST}(T)$ is an SMT, i.e., the cost of a Steiner Minimum Tree is equal to the cost of a Minimum Spanning Tree.*

Proof. Let $ST = \mathrm{MST}(T \cup S)$ be an SMT with the minimal number of Steiner points (no assumption on the length of the edges or on the degree of Steiner points). If such an SMT is not unique, choose one with minimal $\sum_{s \in S} \deg(s)$.

We want to show $S = \emptyset$. Assume otherwise and let $s \in S$. Clearly the degree of s is at least two (otherwise we could remove it and the incident edge). If the degree of s is equal to two, we can remove s from S and connect the two points adjacent to obtain a tree of at most the same cost as ST with fewer Steiner points. Similarly, we can show that we are able to remove s if one of the edges incident to s has length 0.

Now assume that $\deg(s) \geq 3$. If the length of an edge (u, s) incident to s is 2, i.e. maximal, we can remove this edge from ST and obtain two connected components, both containing at least one terminal - otherwise the other part would be a smaller Steiner tree. We connect u to a terminal in the other component and get a Steiner tree of the same length as ST where the total degree of the Steiner points is smaller.

Thus we can assume that all edges incident to s have length 1. We partition the set of vertices adjacent to s into two sets S_1, S_2, where the letter in the ith column of the vertices in S_i is the letter of s in the ith column. Removing all edges incident to s and connecting the vertices in S_1 and S_2 (with edges of length 1) gives two connected components. The total cost is the cost of ST minus 2. Connecting two terminals in the two components gives a Steiner tree of the same cost as ST whose number of Steiner points is smaller. Hence, $S = \emptyset$.

Note that we are only interested in the cost of such an MST, not in its structure. Exploiting this and the fact that we have only distances in $\{0, 1, 2\}$ we can determine this cost very efficiently using a variant of Kruskal's algorithm (see [2]).

We want to compute clusters with pairwise distance two such that the minimum spanning tree within the clusters can be build of edges of length one. When c is the number of such clusters, the cost of an MST is $2(c-1)$ plus the sum of sizes of the clusters minus 1, i.e. $T' + c - 2$ where T' is number of distinct terminals in T. Let $\Sigma_i := \bigcup_{t \in T} t_i$ for $i \in \{1, 2\}$ and let $\Sigma_{\min} := \arg\min_{S \in \{\Sigma_1, \Sigma_2\}} \{|S|\}$. We assume that two initialized arrays of size $|\Sigma|$ have been created in the preprocessing phase. Determining $\Sigma_{\min}$ takes $O(|T|)$ set operations. Let w.l.o.g. $\Sigma_1 = \Sigma_{\min}$. First, we create the $|\Sigma_1|$ clusters of the terminals that share the first letter. For each such cluster, we only store the set of all letters that appear in the second column. As long as there are clusters with pairwise distance we merge them. This can be done in $|\Sigma_{\min}|^2$ set operations. All together the algorithm needs $O(|T| + |\Sigma_{\min}|^2)$ set operations over subsets of Σ. Since $|\Sigma|$ is usually a small constant (4 for DNA and 20 for protein sequences) not bigger than $|T|$ we can use word parallelism for set operations over subsets of Σ to achieve constant time set operations.

6.2 Three Column SMTs

LEMMA 6.2. *Let T be a set of terminals of length 3. Let $C_1, \ldots C_k$ be the connected components of the graph $G := (T, \{(u, v) \in T \times T \mid \|uv\| \leq 2\})$ and c_i be an arbitrary representative of C_i. Then $\bigcup_{i=1}^{k} SMT(C_k) \bigcup MST(\bigcup_{i=1}^{k} \{c_i\})$ is an SMT for T, i.e., if we have components in T whose distance is three, we can compute the SMTs of the components and*

connect these SMTs to give an SMT over all terminals T by arbitrary connections.

Proof. Given an SMT, its full components are the maximal subtrees whose internal vertices are Steiner points. Consider an SMT with a maximal number of full components. It suffices to show that there is no full component that contains terminals from different connected components. Assume otherwise and let t be a full component containing points from different components. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two partial topologies of t that are connected to each other and such that within $span(\mathcal{P}_i)$ there are no terminals from different connected components but the terminals from $\mathcal{P}_1$ and $\mathcal{P}_2$ are from different components. Let r be the root they are connected to. Note that the ranges of the roots of $\mathcal{P}_1$ and $\mathcal{P}_2$ are disjoint. Assume z is the position that is finally chosen for r. If there are several possible choices, we choose z such that the sum of distances to the roots of $\mathcal{P}_1$ and $\mathcal{P}_2$ is maximal. If $\|z, r(\mathcal{P}_i)\| = 3$ for one partial topology, we can remove this edge from t and connect a terminal of $\mathcal{P}_1$ to a terminal of $\mathcal{P}_2$ and get an SMT with more full components. If $\|z, r(pt_1)\| = 2$, we know that there is a terminal t in $T \setminus (span(\mathcal{P}_1) \cup span(\mathcal{P}_2))$ that shares at least one component with some element of a range of $r(\mathcal{P}_1)$. Further, there is a terminal t' in $span(\mathcal{P}_1)$ that has exactly this letter in this column. Remove the edge $(z, r(\mathcal{P}_1))$ from the SMT and add tt'. This yields an SMT with more full components. Finally, $\|z, r(\mathcal{P}_1)\| \leq 1$ and $\|z, r(\mathcal{P}_2)\| \leq 1$ is not possible as the minimal distance between $r(\mathcal{P}_1)$ and $r(\mathcal{P}_2)$ is 3.

LEMMA 6.3. *Let T be a set of pairwise distinct terminals of length 3 and $C_1, \dots C_k$ be the connected components of the graph $G := (T, \{(u, v) \in T \times T \mid \|uv\| = 1\})$. $SMT(T) \geq |T| - 1 + (k-1)/2$.*

Proof. For an SMT ST, let K_i be the number of connected components induced by C_i, i.e., the number of components of the graph with vertex set C_i, where $u, v \in C_i$ are connected by an edge, if there is an edge in ST between these two vertices.

Let ST be an SMT whose edges have length exactly one and such that $\sum_{i=1}^{k} K_i$ is minimal.

We first prove that $K_i = 1$ for each $1 \leq i \leq k$. Assume otherwise and let $u, v \in C_i$ not be connected in the subgraph induced by C_i. Remove an edge st from ST on the path from u to v, such that either s or t is a Steiner point or s and t are from different components (they must exist as u and v are not connected within C_i) and add the edge (u, v) (which has cost 1). The new tree has the same cost as ST, the number of components induced by C_i decreases by one and the number of components induced by other C_j remains the same. This is a contradiction to the choice of ST.

Let F' be the full components of ST and let F be the non-trivial full components, i.e., the full components having at least one Steiner point. Notice that each full component $f \in F'$ is an SMT of the terminals it spans.

For a full component $f \in F$, let $span(f)$ be the set of terminals spanned by f. Note that the distance between two terminals of $span(f)$ is at least two. We first show that if $SMT(f) \geq 3(|span(f)| - 1)/2$, for all full components $f \in F$ of ST, then $c(ST) \geq |T| - 1 + (k-1)/2$.

The number of trivial full components is $|T| - k$ as the trivial full components connect all terminals in the same component. Further, $\sum_{f \in F} |span(f)| = k + |F| - 1$ as a full component of t terminals connects $t - 1$ components. Hence, $c(T) = \sum_{f \in F'} c(f) = |T| - k + \sum_{f \in F} c(f) \geq |T| - k + \sum_{f \in F} (3(t(F) - 1)/2) \geq |T| - k + 3(k-1)/2 + 3(|F| - 1)/2 \geq |T| - 1 + (k-1)/2$.

Now consider a full component f and assume its cost is less than $3(|span(f)| - 1)/2$. We consider the smallest counterexample, i.e., let T' be a set of strings of length 3 with pairwise distance at least two whose SMT is a full component such that its cost is smaller than $3(|T'| - 1)/2$. Further, assume $|T'|$ is minimal. Note that an SMT for any strict subset of $R \subsetneq T'$ has cost at least $3(|R| - 1)/2$, no matter whether the SMT is a full component or not (as any full component of the SMT of R has less than $|T'|$ terminals).

We assume again that all edges of T' have length exactly one. For $|T'| = 1$ there is nothing to show. Otherwise T' contains a Steiner point s (as all edges have length one and the distance between terminals is at least two). Let $N_i, 1 \leq i \leq 3$, be the set of points adjacent to s whose ith letter differs from the ith letter of s (recall that every point adjacent to s differs from s in exactly one column). Reconnect the sets N_i (by an MST over N_i) with $|N_i| - 1$ edges of length 1 and let T_i be the terminals reachable from N_i. Note that T_i are Steiner trees spanning fewer terminals than T' (they are not necessarily full components). If two of the N_i are empty, one T_i contains all terminals and we have constructed a Steiner tree shorter than T'. If one N_i is empty, say N_3, we have $c(T') = c(T_1) + c(T_2) + 2 \geq 3(|T_1| - 1)/2 + 3(|T_2| - 1)/2 + 2 \geq 3(|T| - 1)/2$. If no N_i is empty, we have $c(T') = c(T_1) + c(T_2) + c(T_3) + 3 \geq 3(|T_1| - 1)/2 + 3(|T_2| - 1)/2 + 3(|T_3| - 1)/2 + 3 \geq 3(|T| - 1)/2$.

We can conclude $SMT(T) \geq |T| - 1 + \lceil (k-1)/2 \rceil$ as the cost of an SMT is integral. Further, it can be shown that the analysis is tight since there are terminal sets T whose SMTs have exactly this cost (for all k).

6.3 Combinatorial Solution of the Linear Programming Relaxation

As stated earlier solving the ILP of section 6.3 yields a lower bound for the cost of an SMT. Unfortunately vertex cover is a special case of this ILP when setting $|C| = 2$ and $l_C \in \{0, 1\}$. On the other hand, the LP-relaxation of this ILP also yields a lower bound and can be computed efficiently.

To see this we replace x_i by $y_i := x_i - l_{\{i\}}$, i.e. we subtract the single column discrepancy from every variable. Hence, we can restrict our attention to subsets of cardinality two. Now we can solve the LP combinatorially as its dual

$$\begin{array}{ll} \max & \sum_{(i,j)\in\{1,\ldots,d\}\times\{1,\ldots d\}} l'_{ij} z_{i,j} \\ s.t. & \sum_{j=1}^{d} z_{i,j} \leq 1 \quad \text{for all } 1 \leq i \leq d \end{array}$$

where $l'_{ij} = l_{\{i,j\}} - l_{\{i\}} - l_{\{j\}}$, corresponds to the well known 2-matching polytope, with the only difference that all right hands sides are divided by two. It can be solved as follows (see [15]).

Consider the complete bipartite graph G with vertex sets $\{u_1, \ldots, u_d\}$ and $\{v_1, \ldots v_d\}$. We define the weight w of an edge (u_i, v_j) as $w(u_i, v_j) := l_{\{i,j\}}$. We show that the objective value of the LP is exactly half the weight of the maximum weight matching in G: For any matching M, we can construct a solution of the LP with a cost of $w(M)/2$ by setting $z_{i,j} = |\{u_i v_j, u_j v_i\} \cap M|/2$. On the other hand, let z be an LP solution and let $x_{u_i v_j} = x_{v_i u_j} = z_{ij}$. It is easy to see that x is contained in the matching polytope of the bipartite graph. Thus there is a matching of weight at least $w(x)$ which is twice the value of z.

Note that the lower bound $\frac{1}{(d-1)} \sum_{1\leq i<j\leq d} l_{ij}$ of [8] is worse than this LP bound as $x_{ij} = 1/(d-1)$ is a feasible solution of the dual LP with this objective value.

In the preprocessing phase, we compute a global lower bound for all terminals as described above. During the construction of the partial topologies, it is necessary to compute many lower bounds. As the running time becomes too expensive, we use several methods to speed up the computation like storing computed bounds in a hash table, using matching heuristics like the one in [3] and computing SMTs over pairs of columns only if

1. the pair of columns gave an improvement over the *single column discrepancy* bound over all terminals
2. both columns are still parsimony-informative.

7 Experiments

We compare our C++ implementation of the pruning algorithm with two famous phylogeny programs from the public domain: MEGA in version 3 (see [9]) and PHYLIP in version 3.63 (see [4]). Both programs include very fast variations and extensions of Penny's branch and bound method. While PHYLIP is probably the most cited phylogeny package, MEGA can be considered as one of the fastest free available programs for computing the maximum parsimony problem.

As a global upper bound we use a heuristic as implemented in the PHYLIP package that can be computed within a few seconds.

Since the choice of the test data is very crucial for the results, we decided not to create random instances by our own. Instead we use real life test data and generated data sets from a public source. All tests have been performed on a 1.6 Ghz Pentium M machine equipped with 2 GByte of RAM.

real life data: As real life data we use genetic sequences of RNA viruses discussed in [1]. Since these data sets vary much in size and complexity, they reflect a wide spectrum of different difficulty levels in computing a maximum parsimony tree. Since our implementation does not support unknown or partially known nucleotides at the moment, we only use data sets that do not contain such nucleotides. Table 2 shows a comparison of the running times of these problems using our pruning approach, MEGA and PHYLIP to solve them. The given running times are the CPU-times in seconds consumed by the solvers and include also the time needed to read and parse the data files. The lower part of the table lists the problems that could not be solved by any of the programs within 6 hours.

generated data: The results of the real life data show clearly the superiority of the pruning approach, especially when the problems become more complex. To examine this behavior we created several instances with the tool Seq-Gen [13] that simulates the evolution of amino acid sequences along a phylogeny. The instances that we created have been build on a tree with 24 leaves that was chosen randomly from a huge set of trees created by Vincent Ranwez [14] . Table 1 compares the running times of MEGA and our algorithm. The numbers in the data set names describe branch length scaling factors that can be seen as a measure for the evolutionary distance between the species. With $|T_{pp}|$ we denote the number of terminals after the initial pruning step described in section 5. The columns *spts* and *spts*(%) give the absolute number of the survived partial topologies and the percentage of all possible partial topologies of sizes at most $\lfloor |T_{pp}|/2 \rfloor$. Furthermore, we list the optimal

Data Set	$\|T\|$	$\|T_{pp}\|$	spts	lb_{mst}	opt	spts(%)	Running Times (s) pruning	MEGA
Seq-Gen 50	24	23	49	885	885	$5.3 \cdot 10^{-12}$	0.26	0.51
Seq-Gen 100	24	23	113	1580	1585	$1.2 \cdot 10^{-11}$	1.20	4.21
Seq-Gen 150	24	24	1227	2143	2188	$3.3 \cdot 10^{-12}$	10.72	717.40
Seq-Gen 200	24	24	6488	2473	2560	$1.7 \cdot 10^{-11}$	98.49	10813.80
Seq-Gen 250	24	24	28181	2750	2894	$7.2 \cdot 10^{-11}$	1043.48	–
Seq-Gen 300	24	24	139122	2921	3124	$3.6 \cdot 10^{-10}$	16904.43	–

Table 1: **running time comparison - generated data:** "Seq-Gen 250" and "Seq-Gen 300" MEGA were interrupted after five hours of computation.

solutions *opt* together with 2-column lower bounds lb_{mst} using exact matchings.

8 Conclusion

We have presented a new approach for solving the Steiner tree problem in Hamming distance. It outperforms all previous methods. Still we see various ways of improving the algorithm. Clearly, the algorithm can be improved by finding further properties that exclude partial topologies from being part of an SMT that can be tested efficiently or by finding new ways of preprocessing the data.

One major key for the efficiency of our approach is the tight lower bounds we presented. These lower bounds cannot be used in the old approach as their applicability heavily depends on the fact that the terminals not spanned by the current topology are all connected to the root of the topology.

Preliminary experiments show that the lower bound can be improved by actually computing the lower bounds also for triples of columns. Unfortunately, the computing time to improve the lower bound in both ways dominates the benefit of these tighter bounds. Hence, we work on fast methods that improve the lower bounds.

References

[1] E. Chare, E. Gould, and E. Holmes. Phylogenetic analysis reveals a low rate of homologous recombination in negative-sense rna viruses. *J Gen Virol*, 84, 2003.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[3] D. E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, Feb. 2003.

[4] J. Felsenstein. PHYLIP (phylogeny inference package) version 3.6, 2004. Distributed by the author.

[5] W. M. Fitch. Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20, 1971.

[6] L. Foulds and R. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3, 1982.

[7] M. D. Hendy and D. Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, 59, 1982.

[8] B. R. Holland, K. T. Huber, D. Penny, and V. Moulton. The minmax squeeze: Guaranteeing a minimal tree for population data. *Molecular Biology and Evolution*, 22(2), 2005.

[9] S. Kumar, K. Tamura, and M. Nei. Mega 3: Integrated software for molecular evolutionary genetics analysis and sequence alignment. *Briefings in Bioinformatics*, 5, 2004.

[10] N. Nei and S. Kumar. *Molecular Evolution and Phylogenetics*. Oxford University Press, 2000.

[11] T. Polzin. *Algorithms for the Steiner Problem in Networks*. PhD thesis, Universität des Saarlandes, 2003.

[12] P. Purdom, P. Bradford, K. Tamura, and S. Kumar. Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees. *Bioinformatics*, 16, 2000.

[13] A. Rambaut and N. Grassly. Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.*, 13, 1997.

[14] V. Ranwez and O. Gascuel. Quartet based phylogenetic inference: improvement and limits. *Molecular Biology and Evolution*, 18, 2001. http://www.lirmm.fr/ ranwez/.

[15] A. Schrijver. *Combinatorial optimization : polyhedra and efficiency*. Springer, 2003.

[16] M. Steel. Should phylogenetic models be trying to 'fit an elephant'? *Trends in Genetics*, 21(6), 2005.

[17] D. L. Swofford. PAUP*. Phylogenetic Analysis Using Parsimony (*and other methods). Version 4., 2003.

[18] D. M. Warme, P. Winter, and M. Zachariasen. Exact algorithms for plane steiner tree problems: A computational study. In *Advances in Steiner Trees*. Kluwer Academic Publishers, 2000.

[19] Phylogeny programs. `http://evolution.genetics.washington.edu/phylip/software.html`.

Family	Species	Gene	$\|T\|$	Running Times (s) pruning	MEGA	PHYLIP
Bornaviridae	*Borna disease virus*	G	17	0.04	0.80	37.75
		M	19	0.01	2.03	142.68
Bunyaviridae	*Crimean-Congo hemorrhagic fever virus*	G	11	2.15	0.70	0.28
	Dobrava virus	N	23	10.99	4.03	357.51
	Hantaan virus	G1	15	0.98	0.76	1.92
		N	20	2.98	0.78	294.09
	La Crosse Virus	G	9	0.09	0.65	0.09
	Oropouche virus	NP	28	1122.12	11256.00	$-^3$
	Puumala virus	G1	14	6.45	0.87	10.92
		G2	27	31.92	242.38	1066.42
	Rift Valley Fever Virus	G2	19	0.13	0.79	299.51
	Sin Nombre virus	N	10	<0.01	0.62	0.06
	Tomato spotted wilt virus	N	16	0.01	0.67	24.07
Orthomyxoviridae	*Influenza B virus*	M	26	0.54	13.56	$-^3$
		PA	19	0.12	0.64	109.18
	Influenza C virus	CM2	20	0.01	0.69	56.62
		NP	28	0.19	13.67	$-^1$
Paramyxoviridae	*Avian pneumovirus*	F	14	0.01	4.29	72.31
		M	21	0.02	16442.00	2038.71
		N	14	<0.01	50.83	34.28
		P	15	0.01	1.21	34.31
	Bovine respiratory syncytial virus	G	24	0.06	0.81	640.78
	Canine distemper virus	H	31	7856.98	$-^1$	$-^1$
	Human parainfluenza virus 3	HN	13	0.04	0.72	10.33
	Human respiratory syncytial virus A	N	24	0.46	2.63	$-^3$
	Human respiratory syncytial virus	F	22	1.0	1.32	1082.48
	Measles virus	F	22	1.09	28.94	$-^3$
		HA	20	0.04	0.77	206.36
		L	27	6.28	13.06	$-^3$
	Mumps virus	L	11	0.02	0.7	0.11
	Newcastle disease virus	N	27	1298.52	$-^1$	$-^1$
		P	29	136.8	$-^1$	$-^1$
	Sendai virus	M	27	<0.01	$-^2$	$-^3$
Rhabdoviridae	*Infectious hematopoietic virus*	G	19	0.07	0.86	145.38
	Vesicular stomatitis Indiana virus	G	26	3.68	$-^2$	$-^3$
	Vesicular stomatitis virus	G	25	2.07	1.00	4877.56
	Viral hemorrhagic septicaemia virus	G	15	0.05	0.33	91.16
		N	10	0.01	0.29	0.13
Unclassified	*Rice stripe Virus*	JN	11	< 0.01	0.66	0.40
Arenaviridae	*Junin virus*	NP	45	–	–	–
	Lassa virus	NP	59	–	–	–
Bunyaviridae	*Puumala virus*	NP	48	–	–	–
Paramyxoviridae	*Measles virus*	N	135	–	–	–

Table 2: **running time comparison - real life data:** Entries with a "–" sign mark an interrupted call of the corresponding program. We interrupted a program when it consumed more than 2 GByte of RAM or when its running time exceeded a certain threshold that depends on the complexity of the instance: $-^1$: terminated after 6 hours of computation and a progress of less than 3% (stated by the program); $-^2$: process allocated more than 2 GByte of RAM after 1.5 hours of computation; $-^3$: process was terminated after 3 hours of computation.