Alcom-FT Technical Report Series
ALCOMFT-TR-03-73

Computing Refined Buneman Trees in Cubic Time

Gerth Stølting Brodal[*,†,‡] Rolf Fagerberg[*] Anna Östlin[§]
Christian N. S. Pedersen[*,¶] S. Srinivasa Rao [‖]

Abstract

Reconstructing the evolutionary tree for a set of n species based on pairwise distances between the species is a fundamental problem in bioinformatics. Neighbor joining is a popular distance based tree reconstruction method. It always proposes fully resolved binary trees despite missing evidence in the underlying distance data. Distance based methods based on the theory of Buneman trees and refined Buneman trees avoid this problem by only proposing evolutionary trees whose edges satisfy a number of constraints. These trees might not be fully resolved but there is strong combinatorial evidence for each proposed edge. The currently best algorithm for computing the refined Buneman tree from a given distance measure has a running time of $O(n^5)$ and a space consumption of $O(n^4)$. In this paper, we present an algorithm with running time $O(n^3)$ and space consumption $O(n^2)$. The improved complexity of our algorithm makes the method of refined Buneman trees computational competitive to methods based on neighbor joining.

Keywords: Evolutionary trees, reconstruction, refined Buneman trees, Buneman trees, Q^* method.

1 Introduction

The evolutionary relationship for a set of species is commonly described by an evolutionary tree, also called a phylogeny, where leaves correspond to species and internal nodes correspond to points in time where the evolution has diverged in different directions. Reconstructing an unknown evolutionary tree for a set of species from obtainable information about the species is a fundamental problem in bioinformatics. A multitude of models and methods for reconstructing evolutionary trees have been proposed in the literature, see e.g. [13] for an overview. A large class of methods use the evolutionary distance between each *pair* of

[*]BRICS (Basic Research in Computer Science, www.brics.dk, funded by the Danish National Research Foundation), Department of Computer Science, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark. E-mail: {gerth,rolf,cstorm}@brics.dk. Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

[†]Supported by the Carlsberg Foundation (contract number ANS-0257/20).

[‡]Contact author

[§]IT University of Copenhagen, Glentevej 67, DK-2400 Copenhagen NV. E-mail: annao@it-c.dk. Work done while at BRICS

[¶]Bioinformatics Research Center (BiRC), www.birc.dk, funded by Aarhus University Research Foundation, Department of Computer Science, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark.

[‖]School of Computer Science, University of Waterloo, 200, University Avenue West, Waterloo, Ontario W2L 3G1. E-mail: ssrao@monod.uwaterloo.ca. Work done while visiting BRICS.

species as their primary source of information for reconstructing the unknown evolutionary relationships. Distance data can e.g. be obtained from sequence data from the species by estimating the evolutionary distance between homologous sequences in a model of sequence evolution.

A widely used distance based method is the neighbor joining method by Saitou and Nei [14], which can be implemented with running time $O(n^3)$ and space consumption $O(n^2)$, where n is the number of species. A common critique of neighbor joining based methods is that they always reconstruct fully resolved evolutionary trees, i.e. unrooted trees where all internal nodes have degree three. A fully resolved tree can be misleading because many of its internal edges can be artifacts of the reconstructing method insisting on a fully resolved tree even though the underlying distance data contains little phylogenetic evidence hereof. To avoid this problem, a number of distance based methods have been studied which only propose evolutionary trees whose edges are well supported by constraints expressed in terms of *quartets*. A quartet is the topological subtree induced by four species. Every edge in an evolutionary tree induces a set of quartets consisting of the quartets with two species in each of the two subtrees induced by removing the edge.

The Q^* method [1, 4], which relates to the tree construction method introduced by Buneman in [7], imposes constraints on the proposed edges by requiring that all induced quartets must have positive weight for some given weight function. The running time of the general Q^* method is $O(n^4)$, where n is the number of species, and it has been experimentally shown to introduce very few incorrect edges [4]. If quartets are weighted according to their *Buneman score* the resulting evolutionary tree which satisfies that all induced quartets have positive Buneman score is called a *Buneman tree*. Berry and Bryant [3] show how to compute the Buneman tree for a set of n species in time $O(n^3)$ and space consumption $O(n^2)$. The Buneman tree is a conservative but reliable estimate of the evolutionary tree. However, as discussed in [3, 5] and illustrated in [3, Figure 1], the severe constraints of positive Buneman score for all induced quartets often result in a proposed evolutionary tree with few resolved edges. This shortcoming was addressed by Moulton and Steel [12] who proposed the *refined Buneman tree* which loosens the constraints by allowing a limited number of the induced quartets to have negative score. This tree is a refinement of the Buneman tree in the sense that it contains at least the edges in the Buneman tree. The refined Buneman tree thus takes up a useful middle ground between the neighbor joining method and the classic Buneman tree.

Bryant and Moulton [6] presented the first polynomial time algorithm to compute the refined Buneman tree. The running time is $O(n^6)$. Berry and Bryant [3, 5] gave an improved algorithm with running time $O(n^5)$ and space consumption $O(n^4)$. In this paper we present a method for constructing the refined Buneman tree for a set of n species in time $O(n^3)$ and space $O(n^2)$.

Our algorithm is based on an incremental approach also used in the algorithms presented in [3, 5, 6]. The central difference is that we do not construct a sequence of refined Buneman trees, but instead construct a sequence of over-approximations to refined Buneman trees from which we can extract the desired refined Buneman tree at the end.

The improved running time and the simplicity of our algorithm makes the method of refined Buneman trees computational competitive to methods based on neighbor joining and on plain Buneman trees. It will also make it possible to perform comprehensive experiments on biological data to examine the virtues of refined Buneman trees against trees produced by these other methods. An implementation of our algorithm is currently being made, and it is planned to be part of release 4.0 of the well-known SplitsTree package [10].

The rest of this paper is organized as follows. In Section 2 we introduce notation and earlier results related to Buneman and refined Buneman trees. In Section 3 we describe how to maintain a set of compatible splits. In Sections 4 and 5 we present our improved algorithm for computing refined Buneman trees.

2 Preliminaries

In the following we let the set of species be denoted $X = \{x_1, \ldots, x_n\}$, and for an integer $k \in \{1, \ldots, n\}$ we let $X_k = \{x_1, \ldots, x_k\}$.

Evolutionary tree An evolutionary tree (or X-tree) for a set of species X is an unrooted tree $T = (V, E)$ together with an injective labeling of the leaves by members of X.

Dissimilarity measure A dissimilarity (or distance) measure δ on a set of species X is a symmetric function $\delta : X^2 \to \mathbb{R}_+$ where $\delta(x, x) = 0$ for all $x \in X$.

Quartets To every set of four species $a, b, c, d \in X$, there are four ways to associate a leaf-labeled tree, as shown in Figure 1. The three possible binary tree resolutions, *quartets*, are denoted by $ab|cd$, $ac|bd$ and $ad|bc$, indicating how the central edge of the binary tree bipartitions the four species. We say that an edge e in an X-tree induces a quartet $ab|cd$ if e bipartitions the four species in the same way as the central edge of the quartet.

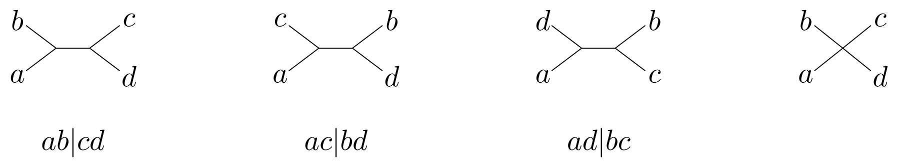

Figure 1: The possible topologies of four species.

Splits The partition of a finite set into two non-empty parts U and V is denoted a *split* $U|V$. In this paper we represent a split $U|V$ as a bit-vector A such that $x_i \in U$ if and only if $A[i] = 0$. If $|U| = 1$ or $|V| = 1$ the split is called trivial. Removing an edge e from an X-tree T partitions the leaf set of the tree into two parts. This is called the split of T associated with the edge e. The complete set of splits associated with each of its edges is denoted $splits(T)$. The lemma below is proved in [9], but also follows from the construction in Section 3.

Lemma 1 (Gusfield [9]) *Any unrooted X-tree with n leaves can be constructed from its set of non-trivial splits in time $O(kn)$, where k is the number of non-trivial splits.*

The set of quartets associated with a split $U|V$ is defined by $q(U|V) = \{uu'|vv' : u, u' \in U \wedge v, v' \in V\}$. Here u and u' (similarly v and v') need not be distinct.

Compatibility A set of splits S is *compatible* if $S \subseteq splits(T)$ for some tree T.

Lemma 2 (Buneman [7]) *Two splits $A|B$ and $C|D$ are compatible if and only if one of $A \cap C$, $A \cap D$, $B \cap C$ or $B \cap D$ is empty. A set of splits is compatible if and only if it is pairwise compatible.*

Buneman trees Buneman [7] shows how to construct a weighted unrooted tree from a dissimilarity measure δ on X by considering quartets. The *Buneman score* of a quartet $q = ab|cd$, where $a, b, c, d \in X$ is defined as:

$$\beta_q = \frac{1}{2}(\min\{ac + bd, ad + bc\} - (ab + cd)) \ , \tag{1}$$

where ab denotes $\delta(a, b)$ for $a, b \in X$. Two distinct quartets q_1 and q_2 for the same four species satisfy

$$\beta_{q_1} + \beta_{q_2} \leq 0 \ . \tag{2}$$

The *Buneman index* of a split $\sigma = U|V$ of X is

$$\mu_\sigma(\delta) = \min_{u,u' \in U, v,v' \in V} \beta_{uu'|vv'} \ .$$

Buneman showed that the set of splits $B(\delta) = \{\sigma : \mu_\sigma(\delta) > 0\}$ is compatible. The *Buneman tree* corresponding to a given dissimilarity measure δ is defined to be the weighted unrooted tree whose edges represent the splits $\sigma \in B(\delta)$ and are weighted according to $\mu_\sigma(\delta)$.

Anchored Buneman tree One relaxation of the condition that $\mu_{U|V} > 0$ is to only look at quartets containing a certain fixed species $x \in X$. For each split $U|V$ with $x \in U$ define

$$\mu^x_{U|V}(\delta) = \min_{u \in U, v,v' \in V} \beta_{xu|vv'} \ ,$$

and let $B_x(\delta) = \{U|V : \mu^x_{U|V} > 0\}$. Clearly $B(\delta) \subseteq B_x(\delta)$. Bryant and Moulton show that the set of splits $B_x(\delta)$ is compatible [6, Lemma 1]. The weighted unrooted tree representing $B_x(\delta)$ with the edge representing a split $\sigma \in B_x(\delta)$ given the weight $\mu^x_\sigma(\delta)$, is called the *Buneman tree anchored at x*.

Lemma 3 (Bryant and Moulton [6, Proposition 2]) $B(\delta) = \cap_{x \in X} B_x(\delta)$.

Lemma 4 (Berry and Bryant [3, Section 3.2]) *$B_x(\delta)$ can be computed in time (and space) $O(n^2)$.*

Refined Buneman tree Given a split σ for a set size n, let $m = |q(\sigma)|$ and let $q_1, \ldots, q_m$ be an ordering of the elements of $q(\sigma)$ in non-decreasing order of their Buneman scores. The refined Buneman index of the split σ is defined as

$$\bar{\mu}_\sigma(\delta) = \frac{1}{n-3} \sum_{i=1}^{n-3} \beta_{q_i} \ . \tag{3}$$

Moulton and Steel show that the set of splits $\{\sigma : \bar{\mu}_\sigma > 0\}$ is compatible [12, Corollary 5.1]. They define the *refined Buneman tree* as the weighted unrooted tree representing the set $RB(\delta) = \{\sigma : \bar{\mu}_\sigma > 0\}$, with the edge representing the split $\sigma \in RB(\delta)$ given the weight $\bar{\mu}_\sigma(\delta)$.

Lemma 5 *Given two incompatible splits σ_1 and σ_2, there exists an $i \in \{1, 2\}$ such that $\bar{\mu}_{\sigma_i} \leq 0$, and this can be computed in $O(n)$ time.*

Proof. Let $\sigma_1 = U_1|V_1$ and $\sigma_2 = U_2|V_2$. Since σ_1 and σ_2 are incompatible, the sets $A = U_1 \cap U_2$, $B = U_1 \cap V_2$, $C = V_1 \cap U_2$ and $D = V_1 \cap V_2$ are all non-empty. From the bitvector representations of σ_1 and σ_2 these four sets can be computed in time $O(n)$. Since $|A| \cdot |B| \cdot |C| \cdot |D| \geq n - 3$ and $\beta_{ab|cd} + \beta_{ac|bd} \leq 0$ for every $a \in A, b \in B, c \in C$ and $d \in D$ by (2), we can find at least $n-3$ pairs of quartets (q_i^1, q_i^2), where q_i^1 and q_i^2 contain the same four species and $1 \leq i \leq n-3$, such that $q_i^1 \in q(\sigma_1), q_i^2 \in q(\sigma_2)$ and $\beta_{q_i^1} + \beta_{q_i^2} \leq 0$. Thus, we have

$$\sum_{i=1}^{n-3} \beta_{q_i^1} + \beta_{q_i^2} \leq 0 \ ,$$

which implies

$$\sum_{i=1}^{n-3} \beta_{q_i^1} \leq 0 \quad \text{or} \quad \sum_{i=1}^{n-3} \beta_{q_i^2} \leq 0 \ .$$

It follows that $\bar{\mu}_{\sigma_1} \leq 0$ or $\bar{\mu}_{\sigma_2} \leq 0$. By calculating the two sums $\sum_{i=1}^{n-3} \beta_{q_i^1}$ and $\sum_{i=1}^{n-3} \beta_{q_i^2}$ in time $O(n)$ we get two upper bounds for $\bar{\mu}_{\sigma_1}$ and $\bar{\mu}_{\sigma_2}$ and can discard at least one of the two splits. □

The following lemma is due to Bryant and Moulton and forms the basis of the incremental algorithms presented in [3, 5, 6] as well as the algorithm we present in this paper.

Lemma 6 (Bryant and Moulton [6, Proposition 3])
Suppose $|X| > 4$, and fix $x \in X$. If $\sigma = U|V$ is a split in $RB(\delta)$ with $x \in U$, and $|U| > 2$, then either $U|V \in B_x(\delta)$ or $U - \{x\}|V \in RB(\delta|_{X-\{x\}})$ or both.

3 Maintaining a set of compatible splits

The running time of our algorithm for computing refined Buneman trees is dominated by the maintenance of a set of compatible splits represented by an X-tree T. In this section, we consider how to support the operations below on X-trees. Recall that we represent a split $U|V$ by a bit-vector A such that $x_i \in U$ if and only if $A[i] = 0$.

- Incompatible(T, σ) Return a split σ' in T that is incompatible with σ. If all splits in T are pairwise compatible with σ then return nil.
- Insert(T, σ) Insert a new split σ into T. It is assumed that σ is pairwise compatible with all existing splits in T.
- Delete(T, σ) Remove the split σ from T.

Theorem 1 *The operations* Incompatible, Insert, *and* Delete *can be supported in time $O(n)$, where $n = |X|$.*

Proof. For the operation $\mathsf{Incompatible}(T, \sigma)$, where $\sigma = U|V$, we root T at an arbitrary leaf, and by a depth first traversal of T for each node v of T compute the number of leaves below v which are in respectively U and V in time $O(n)$. If the parent edge of a node v represents the split $\sigma' = U'|V'$, where U' are the elements below v, then the two counts represent respectively $|U' \cap U|$ and $|U' \cap V|$. From the equalities $|V' \cap U| = |U| - |U' \cap U|$ and $|V' \cap V| = |V| - |U' \cap V|$, we can now in constant time decide if σ' is incompatible with σ, since σ and σ' by definition are incompatible if and only if $|U' \cap U|$, $|U' \cap V|$, $|V' \cap U|$, and $|V' \cap V|$ are all non-zero. We return the first incompatible split found during the traversal of T. If no edge represents a split incompatible with σ, we return nil.

To perform $\mathsf{Delete}(T, \sigma)$ we in linear time find the unique edge (v, u) representing the split σ, by performing a depth first traversal to locate the node v, where the subtree rooted at v contains all elements from U and no element from V or vice versa. Finally we remove the edge (v, u), where u is the parent of v, by contracting v and u into a single node inheriting the incident edges of both nodes.

Finally consider $\mathsf{Insert}(T, \sigma)$, where $\sigma = U|V$. We claim that, since σ is assumed pairwise compatible with all splits in T, there exist a node v, such that removing v and its incident edges leaves us with a set of subtrees where each subtree contains only elements from either U or V. We prove the existence of v below. To locate v we similar to the $\mathsf{Incompatible}$ operation root T at an arbitrary leaf and bottom-up calculate for each node the number of leaves below in respectively U and V. We stop when we find the node v described above. We replace v by two nodes v_U and v_V connected by the edge $e = (v_U, v_V)$. Each subtree incident to v containing only elements from respectively U or V is made incident to respectively v_U or v_V. This ensures that e represents the split σ, and that all other edges remain representing the same set of splits.

What remains is to show that such a node v exists. If $|U| = 1$ or $|V| = 1$ the statement is trivially true. Otherwise, assume $|U| > 1$ and $|V| > 1$, and that the root r is a leaf in U. Let a be a leaf in V. We now argue that the lowest node v on the path from a to the root r containing at least one element from U in its subtree is the node required. Let u be the predecessor of v on the path from a to the root r. By definition u only contains elements from V in its subtree. Let u' be a sibling of u that contains at least one leaf b from U in its subtree. Assume now for the sake of contradiction that removing v and its incident edges leaves us with a set of subtrees including a subtree containing elements $c \in U$ and $d \in V$. Consider the case that c and d are not contained in the subtree of u', but in a subtree that was connected to v with an edge representing a split $U'|V'$, where $c \in V'$ and $d \in V'$. Then the splits $U|V$ and $U'|V'$ are incompatible, since $a \in V \cap U'$, $b \in U \cap U'$, $c \in U \cap V'$, and $d \in V \cap V'$. Otherwise if c and d are contained in the subtree of u', then let $U'|V'$ be the split represented by the edge (u', v). The splits $U|V$ and $U'|V'$ are then incompatible by $a \in V \cap U'$, $r \in U \cap U'$, $c \in U \cap V'$, and $d \in V \cap V'$. □

4 Computing refined Buneman trees

We compute the refined Buneman tree for X by computing a sequence of sets of splits $C_4, \ldots, C_n$, such that each C_k is a set of compatible splits that is an over-approximation of the refined Buneman splits for X_k, i.e. $C_k \supseteq RB(\delta_k)$, where $\delta_k = \delta|_{X_k}$. Each iteration makes essential use of the characterization given by Lemma 6, that enables us to compute C_{k+1} from C_k together with the anchored Buneman tree for X_{k+1} with anchor x_{k+1}. To avoid a blow up in the number of splits, we use the observation that given two incompatible splits, we by Lemma 5 can discard one of the splits as not being a refined Buneman split. By computing the refined Buneman scores for the final set of splits C_n we can exclude all splits with a non-positive refined Buneman score, and obtain the refined Buneman tree $RB(\delta)$ for X. In the following we assume that all sets of compatible splits over X_k are represented by their X_k-tree, i.e. the space usage for storing a compatible set of splits is $O(k)$.

Theorem 2 *Given a dissimilarity measure δ for n species, the refined Buneman tree $RB(\delta)$ can be computed in time $O(n^3)$ and space $O(n^2)$.*

Proof. Pseudo code for the algorithm is contained in Figure 2. The operations Insert, Delete and Incompatible are the operations on a set of compatible splits as described in Section 3. The operation DiscardRight? takes two incompatible splits and returns true/false if the second/first split has been verified not to be a refined Buneman split, c.f. Lemma 5.

1. $C_4 := B_{x_4}(\delta_4)$
2. for $k = 5$ to n
3. $C_k := B_{x_k}(\delta_k)$
4.   for $U|V \in C_{k-1}$
5.   for $\sigma \in \{U \cup \{x_k\}|V \,,\, U|V \cup \{x_k\}\}$
6. $\sigma' :=$ Incompatible(C_k, σ)
7.   while $\sigma' \neq$ nil and DiscardRight?(σ, σ')
8.   Delete(C_k, σ')
9. $\sigma' :=$ Incompatible(C_k, σ)
10.   if $\sigma' =$ nil
11.   Insert(C_k, σ)
12. Compute refined Buneman index for C_n and discard splits with a non-positive score

Figure 2: The overall algorithm for computing the refined Buneman tree

In lines 1-11 we compute a sequence of sets of compatible splits $C_4, \ldots, C_n$, such that $C_k \supseteq RB(\delta_k)$. In line 1 we let C_4 be an over-approximation of $B_{x_4}(\delta_4)$, which satisfies $C_4 \supseteq RB(\delta_4)$ since each refined Buneman split for a set of size four must also be contained in any anchored Buneman split. In lines 2-11 we (based on Lemma 6) inductively compute C_k from C_{k-1} by letting C_k be the set of splits

$$B_{x_k}(\delta_k) \;\cup \bigcup_{U|V \in C_{k-1}} \{U \cup \{x_k\}|V \,,\, U|V \cup \{x_k\}\} \;,$$

except for some incompatible splits that we explicitly verify not being in $RB(\delta_k)$ (lines 6-11).

Since C_k and $B_{x_k}(\delta_k)$ are sets of compatible splits, both contain at most $2k-3$ splits. It follows that in an iteration of lines $3-11$ at most $(2k-3)+2(2(k-1)-3) \leq 6k$ splits can be inserted and deleted from C_k. The number of calls to DiscardRight? and Incompatible is bounded by the number of insertions and deletions of splits. Since by Lemma 4 computing $B_{x_k}(\delta_k)$ takes time $O(k^2)$ and each operation on a set of compatible splits takes time $O(k)$, it follows that the total time spent in an iteration of lines 3-11 is $O(k^2)$, i.e. for lines 1-11 the total time used is $O(n^3)$. Since we for each iteration of the for loop in line 2 only require access to C_{k-1} and C_k, which are represented by X-trees, it follows that the space usage for lines 1-11 is $O(n)$ (not counting the space usage for the dissimilarity measure), if we discard C_{k-2} at the beginning of iteration k.

In Section 5 we describe how to compute the refined Buneman indexes for C_n in time $O(n^3)$ and space $O(n^2)$, i.e. it follows that the total time and space usage is respectively $O(n^3)$ and $O(n^2)$. $\square$

The algorithms in [3, 5, 6] are based on a similar approach as the algorithm described above, but use the stronger requirement that $C_k = RB(\delta_k)$. A central feature of our relaxed computation is that the number of computations of refined Buneman scores for a set of compatible splits is reduced from $n-3$ to a single computation as the final step of the algorithm.

The algorithm described above in line 3 initializes C_k to be the anchored Buneman tree $B_{x_k}(\delta_k)$ by applying Lemma 4. As a simplification, we note that since the algorithm is based on over-approximation, it remains valid if we in line 3 just require C_k to be a compatible set of splits containing at least all splits of $B_{x_k}(\delta_k)$. Berry and Bryant [3, Theorem 2] prove that the single linkage clustering tree for x_k has this property. Single linkage trees can be found in time $O(n^2)$ using spanning tree based methods [2, 8, 11].

5 Refined Buneman indexes

Given an X-tree where the edges E represent a set of compatible splits, we in this section describe how to compute the refined Buneman indexes for the set of splits in time $O(n^3)$ and space $O(n^2)$. The previously best algorithm for this subtask uses time $O(n^4)$ [5, Lemma 3.2] assuming that the scores of the quartets are given in sorted order.

For each edge e, our algorithm finds the $n-3$ quartets of smallest Buneman score induced by e. The refined Buneman indexes for all edges can then be computed according to (3) in time $O(n^2)$.

To identify for each split the quartets with smallest Buneman score, we assume an arbitrary ordering of the species and adopt the following terminology. Let $ab|cd$ be a quartet, where a is the smallest named species among the four species a, b, c and d in the assumed ordering of all species. Motivated by the definition of Buneman scores (1), we consider each quartet $ab|cd$ as two *diagonal* quartets which we denote $ab||cd$ and $ab||dc$. The score of a diagonal quartet $ab||cd$ is defined as $\eta_{ab||cd} = (\delta(b,c) - \delta(a,b) + \delta(a,d) - \delta(d,c))/2$. From the definitions we have $\beta_{ab|cd} = \min\{\eta_{ab||cd}, \eta_{ab||dc}\}$.

Instead of searching for quartets with increasing score we search for diagonal quartets with increasing score. This has the disadvantage that each quartet can be found up to two times (only one time if $c = d$). We say that $ab||cd$ is the minimum diagonal of $ab|cd$, if $\eta_{ab||cd} < \eta_{ab||dc}$ or $\eta_{ab||cd} = \eta_{ab||dc}$ and c is the smallest named species among c and d. Otherwise $ab||dc$ is the minimum diagonal. Note that the Buneman score of $ab|cd$ equals the score of the minimum diagonal. When identifying $ab||cd$ we can by inspecting the quartet check if $ab||cd$ is the minimum diagonal of $ab|cd$; if so we identify $ab|cd$. Otherwise, $ab|cd$ has already been identified by $ab||dc$ since the diagonal quartets are visited in order of increasing score.

The main property of diagonal quartets which we exploit is that for fixed a and c, we can search *independently* for b and d to find the diagonal quartet $ab||cd$ of minimum diagonal score: Find respectively b and d such that respectively $\delta(b,c)-\delta(a,b)$ and $\delta(a,d)-\delta(d,c)$ are minimal.

For an edge e defining the split $U|V$ and $a \in U$ and $c \in V$, where a is the smallest named species among a and c, let $U_{ac}^e = b_1,\dots,b_{|U_{ac}^e|} \subseteq U$ and $V_{ac}^e = d_1,\dots,d_{|V_{ac}^e|} \subseteq V$ be the sets of species named at least a, and where b_i and d_j appear in sorted order with respect to increasing $\delta(b_i,c)-\delta(a,b_i)$ and $\delta(a,d_j)-\delta(d_j,c)$ value. We can consider all $ab_i||cd_j$ as entries of a matrix M_{ac}^e, where $(M_{ac}^e)_{i,j} = \eta_{ab_i||cd_j}$. The crucial property of M_{ac}^e is that each row and column is monotonic non-decreasing. This allows us to construct M_{ac}^e in a lazy manner while exploring the diagonal quartets, starting with only computing $(M_{ac}^e)_{1,1}$ which we denote the minimal score of the pair (a,c).

For each edge e we will lazily construct a subset Q_e of the diagonal quartets induced by e. We represent each Q_e by a linked list. To identify the $n-3$ quartets with smallest Buneman score it is sufficient to identify the $2(n-3)$ pairs (a,c) with smallest minimum score. Since for a quartet there are at most two diagonal quartets, the $n-3$ quartets induced by e with smallest Buneman score will have minimum diagonal quartets with (a,c) among the $2(n-3)$ pairs found.

1. for $e \in E$
2. $Q_e := \emptyset$
3. for $(a,c) \in X^2$ and $a < c$
4.   for each edge e on the path from a to c
5.   find b_e on the same side of e as a with $\delta(b_e,c)-\delta(a,b_e)$ minimal and $b_e \geq a$
6.   for each edge e on the path from c to a
7.   find d_e on the same side of e as c with $\delta(a,d_e)-\delta(d_e,c)$ minimal and $d_e > a$
8.   for each edge e on the path from a to c
9. $Q_e := Q_e \cup \{ab_e||cd_e\}$
10.   if $|Q_e| \geq 3(n-3)$
11.   remove the $n-3$ quartets with largest score from Q_e
12. for $e \in E$
13. $S_e := \emptyset$
14.   while $|S_e| < n-3$
15. $ab_i||cd_j :=$ DeleteMin(Q_e)
16.   if $ab_i||cd_j$ is a minimum diagonal
17.   Insert($S_e, ab_i|cd_j$)
18.   Insert($Q_e, ab_{i+1}||cd_1$) provided $j=1$ and b_{i+1} exists
19.   Insert($Q_e, ab_i||cd_{j+1}$) provided d_{j+1} exists

Figure 3: Algorithm for computing the $n-3$ smallest Buneman scores induced by each edge of an X-tree

The pseudo code for the algorithm to find the $n-3$ quartets for each split is given in Figure 3. In lines 1-11 we identify between $2(n-3)$ and $3(n-3)$ pairs (a,c) with smallest minimal score.

In lines 4-5 and 6-7 we find the b_1 and d_1 species for entries $(M_{ac}^e)_{1,1}$. Note that the two loops process the edges between a and c in different directions. Since the set of possible species b (species d) increases along the path from a to c (from c to a), we can compute the species b_e (species d_e) from the minimum found so far for the predecessor edge on the path together with the new species not considered yet. For each pair (a, c) we will then spend a total time of $O(n)$ in lines 4-7.

In lines 10-11 we for an edge e remove the 1/3 of the pairs (a, c) computed with largest minimum score if $|Q_e|$ becomes $3(n-3)$, leaving the $2(n-3)$ pairs with smallest minimum score in Q_e. This ensures that for each of the n edges we at most have to store $3(n-3)$ pairs, in total bounding the space required by $O(n^2)$. Line 11 can be performed in $O(n)$ time using e.g. the selection algorithm in [15], i.e. amortized $O(1)$ time for each element deleted from Q_e. In total we spend time $O(n^3)$ in lines 1-11 and use space $O(n^2)$,

In lines 12-19 we extract for each edge e the $n-3$ quartets S_e with smallest Buneman score in sorted order. In line 15 we delete the next diagonal quartet from Q_e with smallest diagonal score. If Q_e contains several diagonal quartets with the same score we first delete those which are minimum diagonals.

In line 18 we ensure that if the j first entries of row i of M^e_{ac} have been considered, then $(M^e_{ac})_{i,j+1}$ is inserted in Q_e. Similarly in line 19 we ensure that if the i first entries in the first column of M^e_{ac} have been considered then $(M^e_{ac})_{i+1,1}$ is inserted into Q_e. To find the relevant b_{i+1} in line 18 (d_{j+1} in line 19), we make a linear scan of the subtree incident to e which contains a (respectively c). The species $b_{i+1} \geq a$ should have the smallest value $\delta(b_{i+1}, c) - \delta(a, b_{i+1}) \geq \delta(b_i, c) - \delta(a, b_i)$; in case the expressions are equal then the smallest $b_{i+1} > b_i$. Similarly, the species $d_{j+1} > a$ should have the smallest $\delta(a, d_{j+1}) - \delta(d_{j+1}, c) \geq \delta(a, d_j) - \delta(d_j, c)$; and in case the expressions are equal then the smallest $d_{j+1} > d_j$.

The for-loop in lines 12-19 is performed n times, and the while-loop in lines 14-19 is performed at most $2(n-3)$ times for each edge, since each iteration considers one diagonal quartet. Each of the $2(n-3)$ deletions from Q_e inserts at most two diagonal quartets into Q_e, i.e. $|Q_e| \leq 5(n-3)$. It follows that DeleteMin in line 15 takes time $O(n)$. Finally, lines 18 and 19 each require time $O(n)$. The total time used by the algorithm becomes $O(n^3)$ and the space usage is $O(n^2)$.

Theorem 3 *The refined Buneman indexes for all splits in a given X-tree can be computed in time $O(n^3)$ and space $O(n^2)$.*

References

[1] H.-J. Bandelt and A. W. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Advances in Applied Mathematics*, 7:309–343, 1986.

[2] J.-P. Barthélémy and A. Guénoche. *Trees and Proximity Representations.* John Wiley & Sons, 1991.

[3] V. Berry and D. Bryant. Faster reliable phylogenetic analysis. In *Proc. 3rd International Conference on Computational Molecular Biology (RECOMB)*, pages 69–69, 1999.

[4] V. Berry and O. Gascuel. Inferring evolutionary trees with strong combinatorial evidence. *Theoretical Computer Science*, 240:271–298, 2000.

[5] D. Bryant and V. Berry. A structured family of clustering and tree construction methods. *Advances in Applied Mathematics*, 27(4):705–732, 2001.

[6] D. Bryant and V. Moulton. A polynomial time algorithm for constructing the refined buneman tree. *Applied Mathematics Letters*, 12:51–56, 1999.

[7] P. Buneman. The recovery of trees from measures of dissimilarity. In F. Hodson, D. Kendall, and P. Tautu, editors, *Mathematics in Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, 1971.

[8] J. C. Gower and J. G. S. ROSS. Minimum spanning trees and single-linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

[9] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

[10] D. Huson. Splitstree: a program for analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998. (http://www-ab.informatik.uni-tuebingen.de/software/splits/welcome_en.html).

[11] B. Leclerc. Description combinatoire des altramétriqueès. *Math. Sci. Hum.*, 73:5–37, 1981.

[12] V. Moulton and M. Steel. Retractions of finite distance functions onto tree metrics. *Discrete Applied Mathematics*, 91:215–233, 1999.

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

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

[15] A. Schönhage, M. S. Paterson, and N. Pippenger. Finding the median. *Journal of Computer and System Sciences*, 13:184–199, 1976.