Clustering

Rudi Cilibrasi and Paul M.B. Vitányi

Abstract

The problem is to construct an optimal weight tree from the $3\binom{n}{4}$ weighted quartet topologies on n objects, where optimality means that the summed weight of the embedded quartet topologies is optimal (so it can be the case that the optimal tree embeds all quartets as nonoptimal topologies). We present a Monte Carlo heuristic, based on randomized hill climbing, for approximating the optimal weight tree, given the quartet topology weights. The method repeatedly transforms a bifurcating tree, with all objects involved as leaves, achieving a monotonic approximation to the exact single globally optimal tree. The method has been extensively used for general hierarchical clustering of nontree-like (non-phylogeny) data in various domains and across domains with heterogenous data, and is implemented and available, as part of the CompLearn package. We compare performance and running time with those of UPGMA, BioNJ, and NJ, as implemented in the SplitsTree package on genomic data for which the latter are optimized.

Keywords— Data and knowledge visualization, Pattern matching–Clustering–Algorithms/Similarity measures, Pattern matching–Applications,

Index Terms— hirarchical clustering, global optimization, Monte Carlo method, quartet method, randomized hill-climbing,

I. Introduction

We present a quartet method for phylogenetic construction in biology, and more generally for hierarchical clustering of nontree-like data in non-biological areas. It is a Monte Carlo method, as opposed to deterministic methods like local search. Our method is based on a novel

Rudi Cilibrasi is with the Center for Mathematics and Computer Science (CWI). Address: CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: Rudi.Cilibrasi@cwi.nl. Part of his work was supported by the Netherlands BSIK/BRICKS project, and by NWO project 612.55.002. Paul Vitányi is with the Center for Mathematics and Computer Science (CWI), and the University of Amsterdam. Address: CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: Paul.Vitanyi@cwi.nl. He was supported in part by the EU project RESQ, IST–2001–37559, the NoE QUIPROCONE IST–1999–29064, the ESF QiT Programmme, and the EU NoE PASCAL, the Netherlands BSIK/BRICKS project.

does not address the problem of how to obtain the quartet topology weights from sequence data [22], [28], [30], but takes as input the weights of all quartet topologies and executes the step of how to reconstruct the phylogeny from there. The algorithm produces a sequence of candidate trees with the objects as leaves. Each such candidate tree is scored as to how well the tree represents the information in the weighted quartet topologies on a scale of 0 to 1. If a new candidate scores better than the previous best candidate, the former becomes the new best candidate. The globally optimal tree has the highest score, so the algorithm monotonically approximates the global optimum. The algorithm terminates on a given termination condition. The performance is compared with that of other modern phylogeny methods, on artificial and natural data sets. We compare performance and running time with those of UPGMA, BioNJ, and NJ, as iplemented in the SplitsTree package. The method was developed as part of the CompLearn software [8], and used in, among many others, [10], [11], [12]. We focussed on a quartet method for tree reconstruction believing it to be more sensitive and objective than other methods. Since the available quartet tree methods were too slow when they were exact or global, and too inaccurate or uncertain when they were statistical incremental, we developed a new approach.

A. Relation with Previous Work:

The Minimum Quartet Tree Cost (MQTC) problem below for which we give a new computational heuristic is related to the Quartet Puzzling problem, [41]. There, the quartet topologies are provided with a probability value, and for each quartet the topology with the highest probability is selected (randomly, if there are more than one) as the maximum-likelihood optimal topology. The goal is to find a bifurcating tree that embeds these optimal quartet topologies. In the biological setting it is assumed that the observed genomic data are the result of an evolution in time, and hence can be represented as the leaves of an evolutionary tree. Once we obtain a proper probabilistic evolutionary model to quantify the evolutionary relations between the data we can search for the true tree. In a quartet method one determines the most likely quartet topology under the given assumptions, and then searches for a tree that represents as many of such topologies as is possible. If the theory and data were perfect then there was a tree that represented precisely all most likely quartet topologies. Unfortunately, in real life the theory is not perfect, there is reticulation and hence the data are network-like, the data are

many most likely quartet topologies as possible, do error correction on the quartet topologies, and so on. Hence in phylogeny, finding the best tree according to an optimization criterion may not be the same thing as inferring the tree underlying the data set (which we tend to believe, but usually cannot prove, to exist). For n objects, there are $(2n-5)!! = (2n-5)\times(2n-3)\times\cdots\times 3$ unrooted bifurcating trees. For n large, exhaustive search for the optimal tree turns out to be NP–hard, see Section III-A, and hence infeasible in general. There are two main avenues that have been taken:

(i) Incrementally grow the tree in random order by stepwise addition of objects in the locally optimal way, repeat this for different object orders, and add agreement values on the branches, like DNAML [18], or quartet puzzling [41]. These methods are fast, but a possible problem is as follows. Suppose we have just 32 items. With quartet puzzling we incrementally construct an quartet tree from a randomly ordered list of elements, where each next element is optimally connected to the current tree comprising the previous elements. We repeat this process for, say, 100 permutations. Subsequently, we look for percentage agreement of subtrees common to all such trees. But the number of permutations is about 2^{160}, so why would the incrementally locally optimal trees derived from 100 of them be a representative sample from which we can conclude anything about the globally optimal tree?

(ii) Approximate the global optimum monotonically or compute it, using a geometric algorithm or dynamic programming [3], linear programming [44], or semi-definite programming [39]. These latter methods, other methods, as well as methods related to the MQT problem, cannot handle more than 15–30 objects [44], [32], [34], [4], [39] directly, even while using farms of desktops. To handle more objects one needs to construct a supertree from the constituent quartet trees for subsets of the original data sets, [36], as in [32], [34].

B. This Work

In 2003 in [10], [11] we considered a new approach, like [44], and possibly predating it. Our goal was to use a quartet method to obtain high-quality hierarchical clustering of data from arbitrary (possibly heterogeneous) domains, not necessarily only biological phylogeny data. We thus do not assume that there exists a true evolutionary tree, and our aim is not to just embed as many optimal quartet topologies as is possible. Instead, for n objects we consider all $3\binom{n}{4}$ possible quartet topologies, each with a given weight, and our goal is to find the tree

simple randomized hill-climbing heuristic that monotonically approximates this optimum, and a figure of merit that quantifies the quality of the best current candidate tree on a linear scale. We give an explicit proof of NP–hardness of this problem (which was claimed but not proven in previous literature) Moreover, if a PTAS for the problem exists, then P=NP. Given the NP–hardness of phylogeny reconstruction in general relative to most commonly-used criteria as well as the non-trivial algorithmic and run-time complexity of all previously-proposed quartet-based heuristics, such a simple heuristic is potentially of great use. We also give evidence that the natural data sets we consider have qualities of smoothness so that the monotonic heuristic obtains the global optimum in a feasible number of steps.

C. Materials and Methods

The data samples we used, here or in referred-to previous work, were obtained from standard data bases accessible on the world-wide web, generated by ourselves, or obtained from research groups in the field of investigation. The clustering heuristic generates a tree together with a goodness score. The latter is called standardized benefit score or $S(T)$ value in the sequel. Contrary to other phylogeny methods, we do not have agreement values on the branches: we generate the best tree possible, globally balancing all requirements. Generating trees from the same weighted quartet topologies many times resulted in the same tree in case of high $S(T)$ value, or a similar tree in case of moderately high $S(T)$ value, for every weighting we used, even though the heuristic is randomized. That is, there is only one way to be right, but increasingly many ways to be increasingly wrong which can all be realized by different runs of the randomized algorithm. The quality of the results depends on how well the hierarchical tree represents the information in the set of weighted quartet topologies. That quality is measured by the $S(T)$ value, and is given with each experiment. In certain natural data sets, such as H5N1 genomic sequences, consistently high $S(T)$ values are returned even for large sets of objects of 100 or more nodes, [9]. In other discordant natural data sets however, as treated in [10], [11], the $S(T)$ value deteriorates more and more with increasing number of elements being put in the same tree. The reason is that with increasing size of a discordant natural data set the projection of the information in the cost function into a ternary tree gets necessarily increasingly distorted because the underlying structure in the data is incommensurate with any tree shape whatsoever. In this way, larger structures may induce additional “stress” in the mapping that is visible as

genomic comparisons. Experience shows that in both cases the hierarchical clustering methods seem to work best for small sets of data, up to 25 items, and to deteriorate for some (but not all) larger sets, say 40 items or more. This deterioration is directly observable in the $S(T)$ score and degrades solutions in two common forms: tree instability when different or very different solutions are returned on successive runs or tree "overlinearization" when some data sets produce caterpillar-like structures only or predominantly. In case a large set of objects, say 100 objects, clusters with high $S(T)$ value this is evidence that the data are of themselves tree-like, and the quartet-topology weights, or underlying distances, truely represent to similarity relationships between the data.

II. The Quartet Method

Given a set N of n objects, we consider every set of four elements from our set of n elements; there are $\binom{n}{4}$ such sets. From each set $\{u, v, w, x\}$ we construct a tree of arity 3, which implies that the tree consists of two subtrees of two leaves each. Let us call such a tree a *quartet topology*. The set of $3\binom{n}{4}$ quartet topologies induced by N is denoted by Q. We denote a partition $\{u, v\}, \{w, x\}$ of $\{u, v, w, x\}$ by $uv|wx$. There are three possibilities to partition $\{u, v, w, x\}$ into two subsets of two elements each: (i) $uv|wx$, (ii) $uw|vx$, and (iii) $ux|vw$. In terms of the tree topologies: a vertical bar divides the two pairs of leaf nodes into two disjoint subtrees (Figure 1).

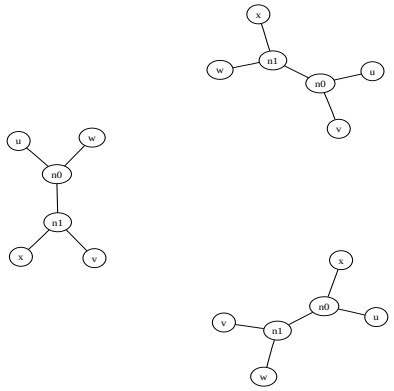

Fig. 1. The three possible quartet topologies for the set of leaf labels *u,v,w,x*

For the moment we consider the class $\mathcal{T}$ of undirected trees of arity 3 with $n \geq 4$ leaves, labeled with the elements of N. Such trees have n leaves and $n - 2$ internal nodes.

$u, v, w, x \in N$, we say T is *consistent* with $uv|wx$, or the quartet topology $uv|wx$ is *embedded* in T, if and only if the path from u to v does not cross the path from w to x.

It is easy to see that precisely one of the three possible quartet topologies for any set of 4 labels is consistent for a given tree from the above class, and therefore a tree from $\mathcal{T}$ contains precisely $\binom{n}{4}$ different quartet topologies. We may think of a large tree having many smaller

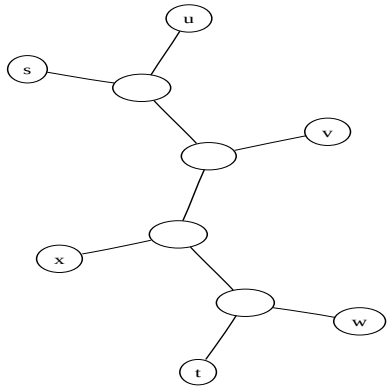

Fig. 2. An example tree consistent with quartet topology $uv|wx$

quartet topologies embedded within its structure. Commonly the goal in the quartet method is to find (or approximate as closely as possible) the tree that embeds the maximal number of consistent (possibly weighted) quartet topologies from a given set $P \subseteq Q$ of quartet topologies [21] (Figure 2). A *weight function* $W : P \rightarrow \mathcal{R}$, with $\mathcal{R}$ the set of real numbers determines the weights. The unweighted case is when $W(uv|wx) = 1$ for all $uv|wx \in P$.

Definition 2.2: The (weighted) *Maximum Quartet Consistency (MQC) optimization* problem is defined as follows:

GIVEN: N, P, and W.

QUESTION: Find $T_0 = \max_T \sum\{W(uv|wx) : uv|wx \in P$ and $uv|wx$ is consistent with $T\}$.

III. Minimum Quartet Tree Cost

The rationale for the MQC optimization problem is the assumption that there is exists a tree T_0 as desired in the class $\mathcal{T}$ under consideration, and our only problem is to find it. This assumption reflects the genesis of the method in the phylogeny community. Under the

now existing species, there is a phylogeny P (tree in $\mathcal{T}$) that represents that evolution. The set of quartet topologies consistent with this tree, has one quartet topology per quartet which is the true one. The quartet topologies in P are the ones which we assume to be among the true quartet topologies, and weights are used to express our relative certainty about this assumption concerning the individual quartet topologies in P.

However, the data may be corrupted so that this assumption is no longer true. In the general case of hierarchical clustering we do not even have a priori knowledge that certain quartet topologies are objectively true and must be embedded. Rather, we are in the position that we can somehow assign a relative importance to the different quartet topologies. Our task is then to balance the importance of embedding different quartet topologies against one another, leading to a tree that represents the concerns as well as possible. We start from a cost-assignment to the quartet topologies: Given a set N of n objects, let Q be the set of quartet topologies, and let $C : Q \rightarrow \mathcal{R}$ be a *cost function* assigning a real valued cost $C(uv|wx)$ to each quartet $uv|wx \in Q$.

Definition 3.1: The *cost* C_T of a tree T with a set N of leaves (external nodes of degree 1) is defined by $C_T = \sum_{\{u,v,w,x\} \subseteq N} \{C(uv|wx) : T \text{ is consistent with } uv|wx\}$—the sum of the costs of all its consistent quartet topologies.

Definition 3.2: Given N and C, the *Minimum Quartet Tree Cost (MQTC)* is $\min_T \{C_T : T$ is a tree with the set N labeling its leaves$\}$.

We normalize the problem of finding the MQTC as follows: Consider the list of all possible quartet topologies for all four-tuples of labels under consideration. For each group of three possible quartet topologies for a given set of four labels u, v, w, x, calculate a best (minimal) cost $m(u, v, w, x) = \min\{C(uv|wx), C(uw|vx), C(ux|vw)\}$, and a worst (maximal) cost $M(u, v, w, x) = \max\{C(uv|wx), C(uw|vx), C(ux|vw)\}$. Summing all best quartet topologies yields the best (minimal) cost $m = \sum_{\{u,v,w,x\} \subseteq N} m(u, v, w, x)$. Conversely, summing all worst quartet topologies yields the worst (maximal) cost $M = \sum_{\{u,v,w,x\} \subseteq N} M(u, v, w, x)$. For some cost functions, these minimal and maximal values can not be attained by actual trees; however, the score C_T of every tree T will lie between these two values. In order to be able to compare the scores of quartet trees for different numbers of objects in a uniform way, we now rescale the score linearly such that the worst score maps to 0, and the best score maps to 1:

Definition 3.3: The *normalized tree benefit score* $S(T)$ is defined by $S(T) = (M -$

Our goal is to find a full tree with a maximum value of $S(T)$, which is to say, the lowest total cost. Now we can rephrase the MQTC problem in such a way that solutions of instances of different sizes can be uniformly compared in terms of relative quality:

Definition 3.4: Definition of the *MQTC optimization problem*:

GIVEN: N and C.

QUESTION: Find a tree T_0 with $S(T_0) = \max\{S(T) : T$ is a tree with the set N labeling its leaves$\}$.

Definition 3.5: Definition of the *MQTC decision problem*:

GIVEN: N and C and a rational number $0 \leq k \leq 1$.

QUESTION: Is there a binary tree T with the set N labeling its leaves and $S(T) \geq k$.

A. Computational Hardness

The hardness of Quartet Puzzling is informally mentioned in the literature [44], [32], [34], but we provide explicit proofs. To express the notion of computational difficulty one uses the notion of "nondeterministic polynomial time (NP)". If a problem concerning n objects is NP–hard this means that the best known algorithm for this (and a wide class of significant problems) requires computation time at least exponential in n. That is, it is infeasible in practice. Let N be a set of n objects, let T be a tree of which the n leaves are labeled by the objects, and let Q be the set of quartet topologies and Q_T be the set of quartet topologies embedded in T.

Definition 3.6: The *MQC decision problem* is the following:

GIVEN: A set of quartet topologies $P \subseteq Q$, and an integer k.

DECIDE: Is there a binary tree T such that $P \bigcap Q_T > k$.

In [40] it is shown that the MQC decision problem is NP–hard. Sometimes this problem is called the *incomplete* MQC decision problem. The less general *complete MQC decision problem* requires P to contain precisely one quartet topology per quartet out of N, and is proven to be NP–hard as well in [4].

Theorem 3.7: (i) The MQTC decision problem is NP–hard.

(ii) The MQTC optimization problem is NP–hard.

Proof: (i) By reduction from the MQC decision problem. For every MQC decision problem one can define a corresponding MQTC decision problem that has the same solution: give the quartet topologies in P cost 0 and the ones in $Q - P$ cost 1. Consider the MQTC decision

alternative equivalent formulation is: is there a tree T with the set N labeling its leaves such that

$$S(T) > \frac{M - \binom{n}{4} + k}{M - m}?$$

Note that every tree T with the set N labeling its leaves has precisely one out of the three quartet topologies of every of the $\binom{n}{4}$ quartets embedded in it. Therefore, the cost $C_T = \binom{n}{4} - |P \bigcap Q_T|$. If the answer to the above question is affirmative, then the number of quartet topologies in P that are embedded in the tree exceeds k; if it is not then there is no tree such that the number of quartet topologies in P embedded in it exceeds k. This way the MQC decision problem can be reduced to the MQTC decision problem, which shows also the latter to be NP–hard.

(ii) An algorithm for the MQTC optimization problem yields an algorithm for the MQTC decision problem with the same running time up to a polynomial additive term: If the answer to the MQTC optimization problem is a tree T_0, then we determine $S(T_0)$ in $O(n^4)$ time. Let k be the bound of the MQTC decision problem. If $S(T_0) \geq k$ then the answer to the decision problem is "yes," otherwise "no." ■

The proof shows that negative complexity results for MQT carry over to MQTC. A polynomial time approximation scheme (PTAS) is is a polynomial time approximation algorithm for an optimization problem with a performance guaranty. It takes an instance of an optimization problem and a parameter $\epsilon > 0$, and produces a solution of an optimization problem that is optimal up to an ϵ fraction. For example, for the MQC optimization problem as defined above, a PTAS would produce a tree embedding at least $(1-\epsilon)|P|$ quartets from P. The running time of a PTAS is required to be polynomial in the size of the problem concerned for every fixed ϵ, but can be different for different ϵ. In [4] a PTAS for a restricted version of the MQC optimization problem, namely the "complete" MQC optimization problem defined above, is exhibited. This is a theoretical approximation that would run in something like n^{19}. For general (what we have called "incomplete") MQC optimization it is shown that even such a theoretical algorithm does not exist, unless P=NP.

Theorem 3.8: If a PTAS for the MQTC optimization problem exists, then P=NP.

Proof: The reduction in the proof of Theorem 3.7 yields a restricted version of the MQTC optimization problem that is equivalent to the MQT optimization problem. There is an isomorphism between every partial solution, including the optimal solutions involved: For every tree T with N labeling the leaves, the MQTC cost $C_T = \binom{n}{4} - |P \bigcap Q_T|$ where $P \bigcap Q_T$ is the

since the reduction gives a linear time computable isomorphic version of the MQTC problem instance for each MQT problem instance. Since [4] has shown that a PTAS for the MQT optimization problem does not exist unless P=NP, it also holds for this restricted version of the MQTC optimization problem that a PTAS does not exist unless P=NP, The full MQTC optimization problem is at least as hard to approximate by a PTAS, from which the theorem follows. ■

Is it possible that the best $S(T)$ value is always one, that is, there always exists a tree that embeds all quartets at minimum cost quartet topologies? Consider the case $n = |N| = 4$. Since there is only one quartet, we can set T_0 equal to the minimum cost quartet topology, and have $S(T_0) = 1$. A priori we cannot exclude the possibility that for every N and C there always is a tree T_0 with $S(T_0) = 1$. In that case, the MQTC optimization problem reduces to finding that T_0. However, the situation turns out to be more complex. Note first that the set of quartet topologies uniquely determines a tree in $\mathcal{T}$, [6].

Lemma 3.9: Let T, T' be different labeled trees in $\mathcal{T}$ and let $Q_T, Q_{T'}$ be the sets of embedded quartet topologies, respectively. Then, $Q_T \neq Q_{T'}$.

A *complete set* of quartet topologies on N is a set containing precisely one quartet topology per quartet. There are $3^{\binom{n}{4}}$ such combinations, but only $2^{\binom{n}{2}}$ labeled undirected graphs on n nodes (and therefore $|\mathcal{T}| \leq 2^{\binom{n}{2}}$). Hence, not every complete set of quartet topologies corresponds to a tree in $\mathcal{T}$. This already suggests that we can weight the quartet topologies in such a way that the full combination of all quartet topologies at minimal costs does not correspond to a tree in $\mathcal{T}$, and hence $S(T_0) < 1$ for $T_0 \in \mathcal{T}$ realizing the MQTC optimum. For an explicit example of this, we use that a complete set corresponding to a tree in $\mathcal{T}$ must satisfy certain transitivity properties, [13], [14]:

Lemma 3.10: Let T be a tree in the considered class with leaves N, Q the set of quartet topologies and $Q_0 \subseteq Q$. Then Q_0 uniquely determines T if

(i) Q_0 contains precisely one quartet topology for every quartet, and

(ii) For all $\{a, b, c, d, e\} \subseteq N$, if $ab|bc, ab|de \in Q$ then $ab|ce \in Q$, as well as if $ab|cd, bc|de \in Q$ then $ab|de \in Q$.

Theorem 3.11: There are N (with $n = |N| = 5$) and a cost function C such that, for every $T \in \mathcal{T}$, $S(T)$ does not exceed $4/5$.

$C(ux|vw) = 0$, $C(xy|uv) = C(wy|uv) = C(uy|wx) = C(vy|wx) = 0$, and $C(ab|cd) = 1$ for all remaining quartet topologies $ab|cd \in Q$. We see that $M = 5 - \epsilon$, $m = 0$. The tree $T_0 = (y, ((u, v), (w, x)))$ has cost $C_{T_0} = 1 - \epsilon$, since it embeds quartet topologies $uw|xv, xy|uv, wy|uv, uy|wx, vy|wx$. We show that T_0 achieves the MQTC optimum.

Case 1: If a tree $T \neq T_0$ embeds $uv|wx$, then it must by Item (i) of Lemma 3.10 also embed a quartet topology containing y that has cost 1.

Case 2: If a tree $T \neq T_0$ embeds $uw|xv$ and $xy|uv$, then it must by Item (ii) of the Lemma 3.10 also embed $uw|xy$, and hence have cost $C_T \geq 1$. Similarly, all other remaining cases of embedding a combination of a quartet topology not containing y of 0 cost with a quartet topology containing y of 0 cost in T, imply an embedded quartet topology of cost 1 in T. ■

Altogether, the MQTC optimization problem is infeasible in practice, and natural data can have an optimal $S(T) < 1$. In fact, it follows from the above analysis that to determine the optimal $S(T)$ in general is NP–hard. If the deterministic approximation of this optimum to within a given precision, can be done in polynomial time, then that implies the generally disbelieved conjecture P=NP. Therefore, any practical approach to obtain or approximate the MQTC optimum requires some type of heuristics, for example Monte Carlo methods.

IV. Monte Carlo Heuristic

Our algorithm is a Monte Carlo heuristic, essentially randomized hill-climbing, using parallelized Genetic Programming, where undirected trees evolve in a random walk driven by a prescribed fitness function. We are given a set N of n objects and a cost function C.

Definition 4.1: We define a *simple mutation* on a labeled undirected ternary tree as one of three possible transformations:

1) A *leaf swap*, which consists of randomly choosing two leaf nodes and swapping them.
2) A *subtree swap*, which consists of randomly choosing two internal nodes and swapping the subtrees rooted at those nodes.
3) A *subtree transfer*, whereby a randomly chosen subtree (possibly a leaf) is detached and reattached in another place, maintaining arity invariants.

Each of these simple mutations keeps the number of leaf nodes and internal nodes in the tree invariant; only the structure and placements change.

nodes (with 1 connecting edge) labeled with the names of the data items, and $n-2$ non-leaf or *internal* nodes labeled with the lowercase letter "k" followed by a unique integer identifier. Each internal node has exactly three connecting edges.

Step 2: For this tree T, we calculate the summed total cost of all embedded quartet topologies, and compute $S(T)$.

Step 3: The *currently best known tree* variable T_0 is set to T: $T_0 \leftarrow T$.

Step 4: Pick a number k with probability $p(k) = c/(k(\log k)^2)$ where $1/c = \sum_{k=1}^{\infty} 1/(k(\log k)^2)$.

Step 5: Compose a k-mutation by, for each of the constituent sequence of k simple mutations, choosing one of the three types listed above with equal probability. For each of these simple mutations, we uniformly at random select leaves or internal nodes, as appropriate.

Step 6: In order to search for a better tree, we simply apply the k-mutation constructed in *Step 5* to T_0 to obtain T, and then calculate $S(T)$. If $S(T) > S(T_0)$, then replace the current candidate in T_0 by T (as the new best tree): $T_0 \leftarrow T$.

Step 7: If $S(T_0) = 1$ or a *termination condition* to be discussed below holds, then output the tree in T_0 as the best tree and halt. Otherwise, go to *Step 4*.

Fig. 3. The Algorithm

Definition 4.2: A *k-mutation* is a sequence of k simple mutations. Thus, a simple mutation is a 1-mutation.

A. Algorithm

The algorithm is given in Figure 3. We comment on the different steps:

Comment on Step 2: A tree is consistent with precisely $\frac{1}{3}$ of all quartet topologies, one for every quartet. A random tree is likely to be consistent with about $\frac{1}{3}$ of the best quartet topologies—but because of dependencies this figure is not precise.

Comment on Step 3: This T_0 is used as the basis for further searching.

Comment on Step 4: This number k is the number of simple mutations that we will perform in the next k-mutation. The probability distribution $p(k)$ is easily generated by running a random

is, if $x = x_1 \dots x_k \in \{0,1\}^k$ ($|x| = k \geq 1$), then $\bar{x} = 1^{k-1}0x$, $x' = \overline{|x|}x$, and $x'' = \overline{|x'|}x'$. Thus, the length $|x''| = k + \log k + 2\log\log k$. The probability of generating x'' corresponding to a given x of length k by fair coin flips is $2^{-|x''|} = 2^{-k-\log k-2\log\log k} = 2^{-k}/(k(\log k)^2)$. The probability of generating x'' corresponding to *some* x of length k is 2^k times as large, that is, $1/(k(\log k)^2)$. In practice, we used a "shifted" fat tail distribution $1/((k+2)(\log k+2)^2)$

Comment on Step 5: Notice that trees which are close to the original tree (in terms of number of simple mutation steps in between) are examined often, while trees that are far away from the original tree will eventually be examined, but not very frequently.

Remark 4.3: We have chosen $p(k)$ to be a "fat-tail" distribution, with the fattest tail possible, to concentrate maximal probability also on the larger values of k. That way, the likelihood of getting trapped in local minima is minimized. In contrast, if one would choose an exponential scheme, like $q(k) = ce^{-k}$, then the larger values of k would arise so scarcely that practically speaking the distinction between being absolutely trapped in a local optimum, and the very low escape probability, would be insignificant. Considering positive-valued probability mass functions $q : \mathcal{N} \to (0,1]$, with $\mathcal{N}$ the natural numbers, as we do here, we note that such a function (i) $\lim_{k\to\infty} q(k) = 0$, and (ii) $\sum_{k=1}^{\infty} q(k) = 1$. Thus, every function of the natural numbers that has strictly positive values and converges can be normalized to such a probability mass function. For smooth analytic functions that can be expressed a series of fractional powers and logarithms, the borderline between converging and diverging is as follows: $\sum 1/k, \sum 1/(k\log k)$, $\sum 1/(k\log k\log\log k)$ and so on diverge, while $\sum 1/k^2, \sum 1/(k(\log k)^2)$,$\sum 1/(k\log k(\log\log k)^2)$ and so on converge. Therefore, the maximal fat tail of a "smooth" function $f(x)$ with $\sum f(x) < \infty$ arises for functions at the edge of the convergence family. The distribution $p(k) = c/(k(\log k)^2)$ is as close to the edge as is reasonable, and because the used coding $x \to x''$ is a prefix code we have $\sum 1/(k(\log k)^2) \leq 1$ by the Kraft Inequality (see for example [31]) and therefore $c \geq 1$. Let us see what this means for our algorithm using the chosen distribution $p(k)$. For $N = 64$, say, we can change any tree in $\mathcal{T}$ to any other tree in $\mathcal{T}$ with a 64-mutation. The probability of such a complex mutation occurring is quite large with such a fat tail: $1/(64 \cdot 6^2) = 1/2304$, that is, more than 40 times in 100,000 generations. If we can get out of a local minimum with already a 32-mutation, then this occurs with probability at least $1/800$, so 125 times, and with a 16-mutation with probability at least $1/196$, so 510 times. $\diamondsuit$

The main problem with hill-climbing algorithms is that they can get stuck in a local optimum. However, by randomly selecting a sequence of simple mutations, longer sequences with decreasing probability, we essentially run a Metropolis Monte Carlo algorithm [33], reminiscent of simulated annealing [24] at random temperatures. Since there is a nonzero probability for every tree in $\mathcal{T}$ being transformed into every other tree in $\mathcal{T}$, there is zero probability that we get trapped forever in a local optimum that is not a global optimum. That is, trivially:

Lemma 4.4: (i) The algorithm approximates the MQTC optimal solution monotonically in each run.

(ii) The algorithm without termination condition solves the MQTC optimization problem eventually with probability 1 (but we do not in general know when the optimum has been reached in a particular run).

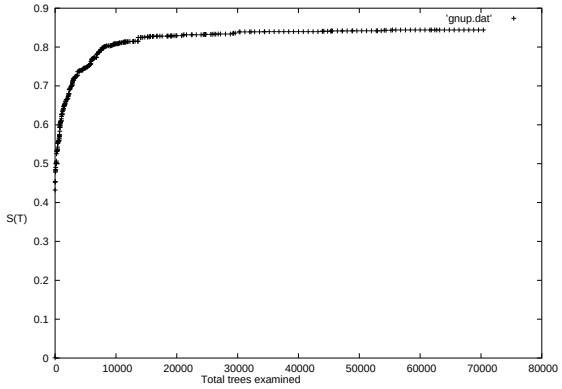

Fig. 4. Progress of a 60-item data set experiment over time

The main question therefore is the convergence speed of the algorithm on natural data, and a termination criterion to terminate the algorithm when we have an acceptable approximation. From the Theorem 3.8 we know that there is no polynomial approximation scheme for MQTC optimization, and whether our scheme is expected polynomial time seems to require proving that the involved Metropolis chain is rapidly mixing [42], a notoriously hard and generally unsolved problem. In practice, in our experiments there is unanimous evidence that for the natural data and the cost function we have used, convergence is always fast. We have to determine the cost of $\binom{n}{4}$ quartets to determine each $S(T)$ value. Hence, trivially,

Lemma 4.5: The algorithm in Figure 3 runs in time $\Omega(n^4)$.

Remark 4.6: If one constructs the quartet-topology costs from more basic quantities, such

$d(\cdot,\cdot)$, then one can use the additional structure thus supplied to speed up the algorithm. In the above case the algorithm could be sped up to $O(n^2)$, and we were able to analyze a 260-node tree in about 3 hours cpu time reaching $S(T) \approx 0.98$. $\diamondsuit$

In experiments we found that for the same data set different runs consistently showed the same behavior, for example Figure 4 for a 60-object computation. There the $S(T)$ value leveled off at about 70,000 examined trees, and the termination condition was "no improvement in 5,000 trees." Different random runs of the algorithm nearly always gave the same behavior, returning a tree with the same $S(T)$ value, albeit a different tree in most cases since here $S(T) \approx 0.865$, a relatively low value. That is, there are many ways to find a tree of optimal $S(T)$ value, and apparently the algorithm never got trapped in a lower local optimum. For problems with high $S(T)$ value, as we see later, the algorithm consistently returned the same tree. This situation is perhaps similar to the behavior of the Simplex method in linear programming, that can be shown to run in exponential time on a badly chosen problem instance, but in practice on natural problems consistently runs in linear time.

Note that if a tree is ever found such that $S(T) = 1$, then we can stop because we can be certain that this tree is optimal, as no tree could have a lower cost. In fact, this perfect tree result is achieved in our artificial tree reconstruction experiment (Section IV-E) reliably for 18-node trees in a few minutes, and 32-node trees in a few hours. For real-world data, $S(T)$ reaches a maximum somewhat less than 1, presumably reflecting distortion of the information in the cost function data by the best possible tree representation, as noted above, or indicating getting stuck in a local optimum or a search space too large to find the global optimum. On many typical problems of up to 40 objects this tree-search gives a tree with $S(T) \geq 0.9$ within half an hour. For large numbers of objects, tree scoring itself can be slow: as this takes order n^4 computation steps. Current single computers can score a tree of this size in about a minute. Additionally, the space of trees is large, so the algorithm may slow down substantially. For larger experiments, we used the C program called partree (part of the CompLearn package [8]) with MPI (Message Passing Interface, a common standard used on massively parallel computers) on a cluster of workstations in parallel to find trees more rapidly. We can consider the graph mapping the achieved $S(T)$ score as a function of the number of trees examined. Progress occurs typically in a sigmoidal fashion towards a maximal value ≤ 1, Figure 4.

The *termination condition* is of two types and which type is used determines the number of objects we can handle.

Simple termination condition: We simply run the algorithm until it seems no better trees are being found in a reasonable amount of time. Here we typically terminate if no improvement in $S(T)$ value is achieved within 100,000 examined trees. This criterion is simple enough to enable us to hierarchically cluster data sets up to 80 objects in a few hours. This is way above the 15–30 objects in the previous exact (non-incremental) methods (see Introduction).

Agreement termination condition: In this more sophisticated method we select a number $2 \leq r \leq 6$ of runs, and we run r invocations of the algorithm in parallel. Each time an $S(T)$ value in run $i = 1, \ldots, r$ is increased in this process it is compared with the $S(T)$ values in all the other runs. If they are all equal, then the candidate trees of the runs are compared. This can be done by simply comparing the ordered lists of embedded quartet topologies, in some standard order, since the set of embedded quartet topologies uniquely determines the quartet tree by [6]. If the r candidate trees are identical, then terminate with this quartet tree as output, otherwise continue the algorithm.

This termination condition takes (for the same number of steps per run) about r times as long as the simple termination condition. But the termination condition is much more rigorous, provided we choose r appropriate to the number n of objects being clustered. Since all the runs are randomized independently at startup, it seems very unlikely that with natural data all of them get stuck in the same local optimum with the same quartet tree instance, provided the number n of objects being clustered is not too small. For $n = 5$ and the number of invocations $r = 2$, there is a reasonable probability that the two different runs by chance hit the same tree in the same step. This phenomenon leads us to require more than two successive runs with exact agreement before we may reach a final answer for small n. In the case of $4 \leq n \leq 5$, we require 6 dovetailed runs to agree precisely before termination. For $6 \leq n \leq 9$, $r = 5$. For $10 \leq n \leq 15$, $r = 4$. For $16 \leq n \leq 17$, $r = 3$. For all other $n \geq 18$, $r = 2$. This yields a reasonable tradeoff between speed and accuracy. These specifications of r-values relative to n are partially common sense, partially empirically derived.

It is clear that there is only one tree with $S(T) = 1$ (if that is possible for the data), and random trees (the majority of all possible quartet trees) have $S(T) \approx 1/3$ (above). This gives evidence that the number of quartet trees with large $S(T)$ values is much smaller than the

depends on the data set involved, and hence cannot be expressed by a general formula without further assumptions on the data. However, we can safely state that small data sets, of say ≤ 15 objects, that in our experience often lead to $S(T)$ values close to 1 have very few quartet trees realizing the optimal $S(T)$ value. On the other hand, those large sets of 60 or more objects that contain some inconsistency and thus lead to a low final $S(T)$ value also tend to exhibit more variation as one might expect. This suggests that in the agreement termination method each run will get stuck in a different quartet tree of a similar $S(T)$ value, so termination with the same tree is not possible. Experiments show that with the rigorous agreement termination we can handle sets of up to 40 objects, and with the simple termination up to at least 80 objects on a single computer or 100-200 objects using a cluster of computers in parallel. Basically the algorithm evaluates all quartet topologies in each generated tree, which leads to an $O(n^4)$ algorithm (per generation). The costs of the quartet topologies are often obtained (like in most our examples) by addung the distance of the siblings, that is, for the quartet topology $ab|cd$ we assign a cost $d(a,b) + d(c,d)$. Here $d(a,b)$ is the distance between a and b. But now we can use several improvements of the algorithm, since in this case the costs of different quartet topologies depends on one another, leading to an $O(n^2)$ per generation algorithm. This way, one can attack problems of up to 200 objects. Recently, [15] has used various other heuristics different from the one presented here to obtain a method that is both faster and yields better results than the initial heuristic in this paper.

D. Tree Building Statistics

We used the CompLearn package, [8], to analyze a "10-mammals" example with *zlib* compression yielding a 10×10 distance matrix, similar to the examples in Section VI-B. The algorithm starts with four randomly initialized trees. It tries to improve each one randomly and finishes when they match. Thus, every run produces an output tree, a maximum score associated with this tree, and has examined some total number of trees, T, before it finished. Figure 5 shows a graph displaying a histogram of T over one thousand runs of the distance matrix. The x-axis represents a number of trees examined in a single run of the program, measured in thousands of trees and binned in 1000-wide histogram bars. The maximum number is about 12000 trees examined. The graph suggests a Poisson distribution. About $2/3$rd of the trials take less than 4000 trees. In the thousand trials above, 994 ended with the optimal $S(T) = 0.999514$.

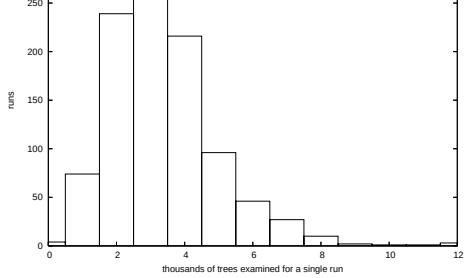

Fig. 5. Histogram of run-time number of trees examined before termination.

The remaining six runs returned 5 cases of the second-highest score, $S(T) = 0.995198$ and one case of $S(T) = 0.992222$. It is important to realize that outcome stability is dependent on input matrix particulars.

Another interesting distribution is the mutation stepsize. Recall that the mutation length is drawn from a shifted fat-tail distribution. But if we restrict our attention to just the mutations that improve the $S(T)$ value, then we may examine these statistics to look for evidence of a modification to this distribution due to, for example, the presence of very many isolated areas that have only long-distance ways to escape. Figure 6 shows the histogram of successful

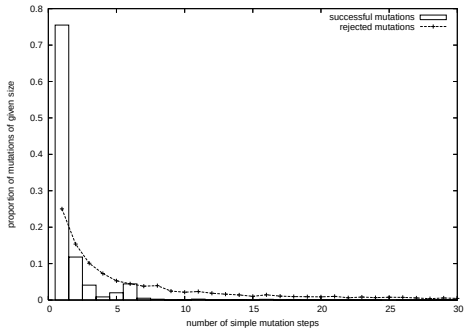

Fig. 6. Histogram comparing distributions of k-mutations per run.

mutation lengths (that is, number of simple mutations composing a single kept complex mutation) and rejected lengths (both normalized) which shows that this is not the case. Here the x-axis is the number of mutation steps and the y-axis is the normalized proportion of times that step size occurred. This gives good empirical evidence that in this case, at least, we have a relatively easy search space, without large gaps.

With natural data sets, say genomic data, one may have the preconception (or prejudice) that primates should be clustered together, rodents should be clustered together, and so should ferungulates. However, the genome of a marsupial may resemble the genome of a rodent more than that of a monotreme, or vice versa—the very question one wants to resolve. Thus, natural data sets may have ambiguous, conflicting, or counterintuitive outcomes. In other words, the experiments on natural data sets have the drawback of not having an objective clear "correct" answer that can function as a benchmark for assessing our experimental outcomes, but only intuitive or traditional preconceptions. We discuss experiments that show that our program indeed does what it is supposed to do—at least in artificial situations where we know in advance what the correct answer is.

V. The CompLearn Toolkit

Recall that the quartet method of phylogeny assembly consists of three parts: (i) extracting a distance matrix from the data, (ii) extracting the quartet topology costs from the distance matrix, and (iii) constructing a quartet tree from the quartet topologies and associated costs. The CompLearn Toolkit [8] uses the heuristic introduced in this paper for item (iii). We now discuss what CompLearn uses for items (ii) and (i). In our experiments it is most realistic to derive the quartet topology costs from a distance matrix. For simplicity, we choose to define the *cost of a quartet topology* as the sum of the distances between each pair of neighbors; that is,

$$C(uv|wx) = d(u, v) + d(w, x),$$

where d is chosen as the compression distance NCD below.

A. Compression-based Distance

To be able to make unbiased comparisons between phylogeny reconstruction algorithms that take distance matrices as input, we use a new compression-based distance, called NCD . This metric distance was co-developed by us in [28], [29], [30], as a normalized version of the "information metric" of [31], [1]. The mathematics used is based on Kolmogorov complexity theory [31], which is approximated using real-world compression software. Roughly speaking, two objects are deemed close if we can significantly "compress" one given the information in the other, the idea being that if two pieces are more similar, then we can more succinctly

compressor Z (for example "gzip", "bzip2", or "PPMZ"). The *normalized compression distance (NCD)* is defined as

$$NCD(x,y) = \frac{Z(xy) - \min\{Z(x), Z(y)\}}{\max\{Z(x), Z(y)\}},$$

which is actually a family of distances parameterized with the compressor Z. The better Z is, the better the results are, [11]. Precursors of the NCD using phylogeny reconstruction methods from standard Biological packages have been applied to, among others, alignment-free whole genome phylogeny, [28], [29], [30], chain letter phylogeny [2], constructing language trees [30], plagiarism detection [7]. The NCD method is also used for general clustering and classification of natural data in arbitrary domains, for clustering of heterogeneous data, and for anomaly detection across domains [11]. It is in fact a parameter-free, feature-free, data-mining tool. It has been experimentally tested on all time sequence data used in all the major data-mining conferences in the last decade [23]. Comparing the compression method with all major methods used in those conferences they established clear superiority of the compression method for clustering heterogeneous data, and for anomaly detection, and competitiveness in clustering domain data. The NCD method turns out to be robust under change of the underlying compressor-types: statistical (PPMZ), Lempel-Ziv based dictionary (gzip), block based (bzip2), or special purpose (Gencompress). While there may be more appropriate special-purpose distance measures for biological phylogeny, incorporating decades of research, the NCD is a robust objective platform to test the unbiased performance of the competing phylogeny reconstruction algorithms.

B. Previous Experiments

Oblivious to the problem area concerned, simply using the distances according to the NCD above and the derived quartet topology costs, the quartet heuristic described in this paper fully automatically clusters the objects concerned. The method has been released in the public domain as open-source software: The CompLearn Toolkit [8] is a suite of simple utilities that one can use to apply compression techniques to the process of discovering and learning patterns in completely different domains, and hierarchically cluster them using the new quartet method described in this paper. In fact, this method is so general that it requires no background knowledge about any particular subject area. There are no domain-specific parameters to set, and only a handful of general settings.

evolution, by reconstructing the phylogeny from the mitochondrial genomes of 24 species. These were downloaded from the GenBank Database on the world-wide web. In another experiment, we used the mitochondrial genomes of molds and yeasts. We clustered the SARS virus after its sequenced genome was made publicly available, in relation to potential similar virii. The NCD distance matrix was computed using the compressor bzip2. The relations with $S(T) = 0.988$ were very similar to the definitive tree based on medical-macrobio-genomics analysis, appearing later in the New England Journal of Medicine, [25]. In [9], 100 different H5N1 sample genomes were downloaded from the NCBI/NIH database online, to analyze the geographical spreading of the Bird Flu H5N1 Virus in a large example.

In general hierarchical clustering, we constructed language trees, cluster both Russian authors in Russian, Russian authors in English translation, English authors, handwritten digits given as two-dimensional ocr data, and astronomical data. We also tested gross classification of files based on heterogeneous data of markedly different file types: genomes, novel excerpts, music files in MIDI format, Linux x86 ELF executables, and compiled Java class files. The program correctly classifies each of the different types of files together with like near like. No features of any specific domain of application are used. We believe that there is no other method known that can cluster data that is so heterogeneous this reliably. This is borne out by the massive experiments with the method in [23]. In [10] we used MIDI data to cluster classical music, distinguish between genres like pop, rock, and classical, and do music classification. In [43], the CompLearn package was used to analyze network traffic and to cluster computer worms and virusses. CompLearn was used to analyze medical clinical data in clustering fetal heart rate tracings [16]. Other applications by different authors are in software metrics and obfuscation, web page authorship, topic and domain identification, protein sequence/structure classification, phylogenetic reconstruction, hurricane risk assessment, ortholog detection, and other topics. Using code-word lengths obtained from the page-hit counts returned by Google from the web, we obtain a semantic distance between *names* for objects (rather than the objects themselves) using the NCD formula and viewing Google as a compressor. Using the CompLearn package, this has been useful for data mining, text comprehension, classification, and translation, [12]. In [9], relations between Nobel Laureates in literature, as well as among Euro-Parlamentarians were determined.

We compare the performance of our method as implemented in the CompLearn package against that of a leading application to compute phylogenetic trees, a program called SplitsTree [20]. We chose SplitsTree version 4.6 for comparison and selected three tree reconstruction methods to benchmark: NJ, BioNJ, and UPGMA. To make comparison possible, we require a tree reconstruction implementation that takes a distance matrix as input. This requirement ruled out some other possiblities, and motivated our choice. To score the quality of the produced trees we used the $S(T)$ values, where the quartet topology costs were derived from the distance matrix concerned, Section V. The UPGMA method consistently performed worse than the other two methods; in several trials it failed to produce an answer at all (throwing an unhandled Java Exception), which may be due to an implementation problem. Therefore, attention was focussed on the other two methods. NJ [37] and BioNJ [19] are neighbor-joining type methods. In all tested cases they produced the same trees, therefore we will treat them as the same in this discussion.

A. Testing on Artificial Data

We first test whether the quartet-based tree construction heuristic and the SplitsTree methods are trustworthy. We generated 100 random samples of an unrooted binary tree T with 32 leaves as follows: We started with a tree made in linear fashion with each node connected to one leaf node, a prior kernel node, and a successive kernel node. The ends have two leaf nodes instead. This starting tree was then mutated 1000 times using randomly generated instantiations of the complex mutation operation defined earlier. Next, we derived a a metric from the scrambled tree by defining the distance between two nodes as follows: Given the length of the path from a to b, in an integer number of edges, as $L(a, b)$, let

$$d(a, b) = \frac{L(a, b) + 1}{32},$$

except when $a = b$, in which case $d(a, b) = 0$. It is easy to verify that this simple formula always gives a number between 0 and 1, is monotonic with path length, and is the resulting matrix is symmetric. Given only the 32×32 matrix of these normalized distances, our quartet method exactly reconstructed the original tree one hundred times out of one hundred random trials. SplitsTree NJ and BioNJ also reconstructed each of these correctly, however UPGMA was unable to cope with this test. It appears there is a mismatch of assumptions in

Amia calva	Bowfin fish	Melanogrammus aeglefinus	Haddock
Anguilla japonica	Japanese eel	Metaseiulus occidentalis	Western predatory mite
Anopheles funestus	Mosquito	Neolamprologus brichard	Lyretail cichlid fish
Arctoscopus japonicus	Sailfin sandfish	Nephila clavata	Orb web spider
Asterias amurensis	Northern Pacific seastar	Oreochromis mossambicus	Mozambique tilapia fish
Astronotus ocellatus	Tiger oscar	Oscarella carmela	Sponge
Cervus nippon taiouanus	Formosan sika deer	Phacochoerus africanus	Warthog
Cobitis sinensis	Siberian spiny loach fish	Plasmodium knowlesi	Primate malaria parasite
Diphyllobothrium latum	Broad tapeworm	Plasmodium vivax	Tersian malaria parasite
Drosophila melanogaster	Fruit fly	Polypterus ornatipinnis	Ornate bichir fish
Engraulis japonicus	Japanese anchovy	Psephurus gladius	Chinese paddlefish
Gavia stellata	Red throated diver	Pterodroma brevirostris	Kerguelen petrel
Gymnogobius petschiliensis	Floating goby fish	Savalia savaglia	Encrusting anemone
Gymnothorax kidako	Moray eel	Schistosoma haematobium	Vesical blood fluke
Hexamermis agrotis	Roundworm Nematode	Schistosoma spindale	Cattle fluke
Hexatrygon bickelli	Sixgill stingray	Synodus variegatus	Variegated lizardfish
Homo sapiens	Human	Theragra finnmarchica	Norwegian pollock fish
Hynobius arisanensis	Arisian salamander	Tigriopus californicus	Tidepool copepod
Hynobius formosanus	Formosa salamander	Tropheus duboisi	White spotted cichlid fish
Lepeophtheirus salmonis	Sea lice		

Fig. 7. Listing of scientific and corresponding common names of 41 (out of 45) species used. The remaining four are dogs, with common breed names Chinese Crested, Irish Setter, Old English Sheepdog, Saint Bernard. There are no scientific names distinguishing them, as far as we know.

this experimental ensemble and the UPGMA preconditions, or there may be an error in the SplitsTree implementation. The running time of CompLearn was about 3 hours per example, SplitsTree was much faster with a few seconds per example.

B. Testing on Natural Data

In the biological setting the data are (parts of) genomes of currently existing species, and the purpose is to reconstruct the evolutionary tree that led to those species. Thus, the species are labels of the leaves, and the tree is traditionally binary branching with each branching representing a split in lineages. The internal nodes and the root of the tree correspond with

The root of the tree is commonly determined by adding an object that is known to be less related to all other objects than the original objects are with respect to each other. Where the unrelated object joins the tree is where we put the root. In these settings, the direction from the root to the leaves represents an evolution in time, and the assumption is that there is a true tree we have to discover. However, we can also use the method for hierarchical clustering, resulting an unrooted ternary tree. The interpretation is that objects in a given subtree are pairwise closer (more similar) to each other than any of those objects is with respect to any object in a disjoint subtree. To evaluate the quality of tree reconstruction for natural genomic data, we downloaded 45 mitochondrial gene sequences, Figure 7, and randomly selected 100 subsets of 32 species each. We used CompLearn with PPMD to compute NCD matrices for each of the 100 trials and fed these matrices (as Nexus files) to both CompLearn and SplitsTree. CompLearn took substantially longer than SplitsTree; for these trials it took about 10 hours per tree but usually produced trees with a higher S(T) score than SplitsTree taking about 10 seconds. In all but one case, CompLearn performed better than the best method from SplitsTree and the results are shown in the histogram Figure 8. CompLearn had an average $S(T)$ of 0.99487068. SplitsTree

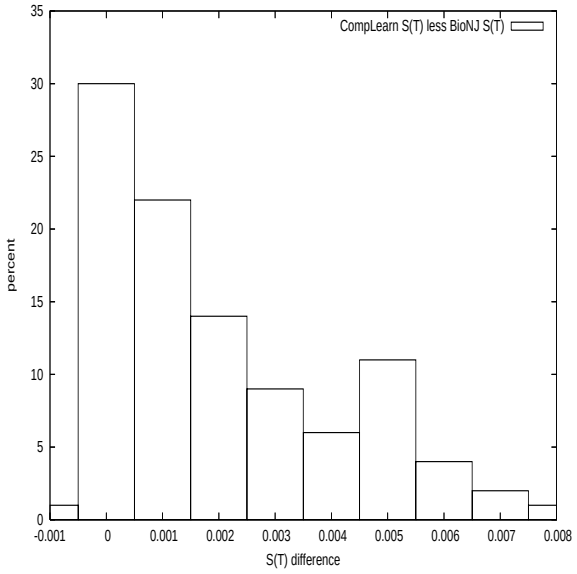

Fig. 8. Histogram showing CompLearn S(T) advantage over SplitsTree S(T)

achieved the best $S(T)$ with both NJ and BioNJ at 0.99243944. At this high level the absolute magnitude of the difference is small, yet it can still imply significant changes in the structure of the tree. Figure 9 and Figure 10 depict one example showing both BioNJ and CompLearn

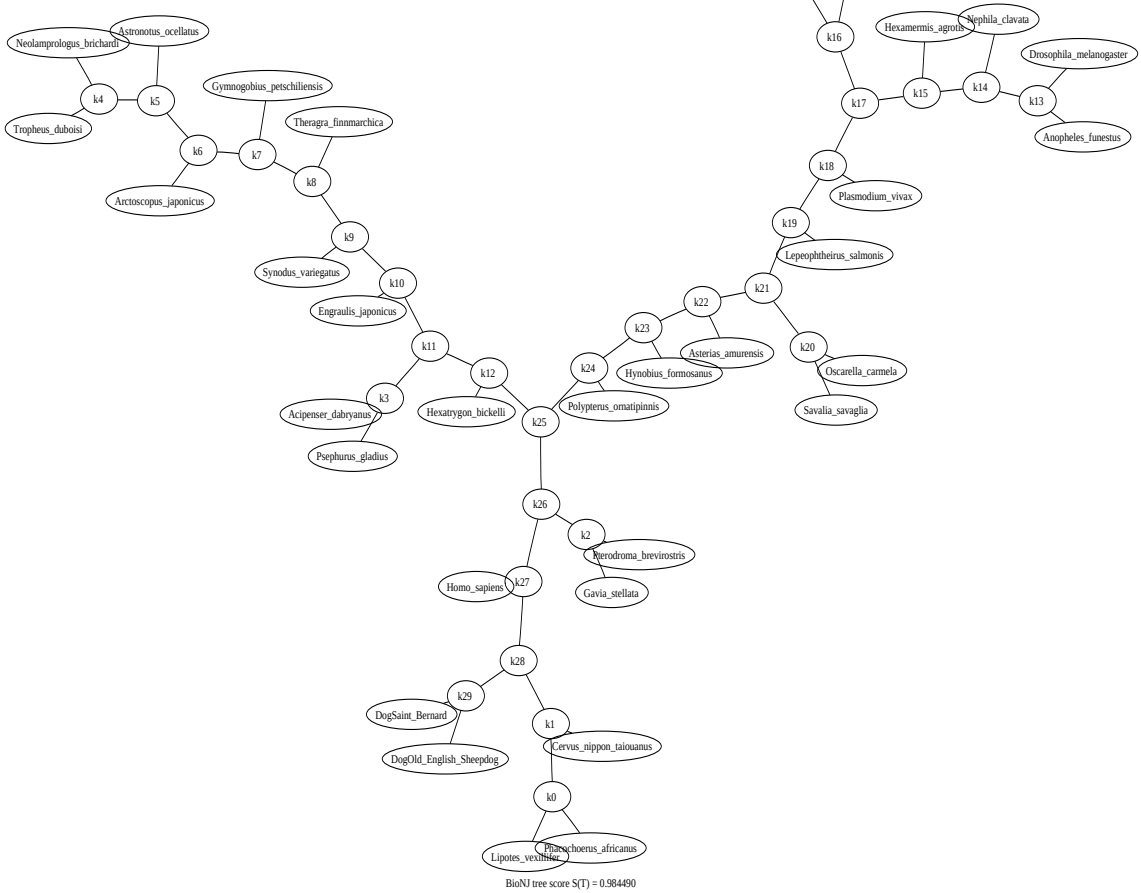

Fig. 9. BioNJ tree from SplitsTree

trees applied to the same input matrix from one of the natural data test cases described above. In this case there are important differences in placement of at least two species; *Hexatrygon bickelli* and *Synodus variegatus*. Although we can not know for sure the true maximum value that can be attained for $S(T)$ given an arbitrary distance matrix, we can still define a useful quantity

$$R(T) = 1.0 - S(T)$$

and term $R(T)$ the *room for improvement* for tree T, especially in cases like the present one when we know that the optimal $S(T)$ is close to 1. We may then define $R_C(T)$ to be CompLearn's $R(T)$ for a given trial, whereas $R_B(T)$ is the SplitsTree BioNJ room for improvement on a given trial. We can compute the decibel reduction $db(T)$ in room for

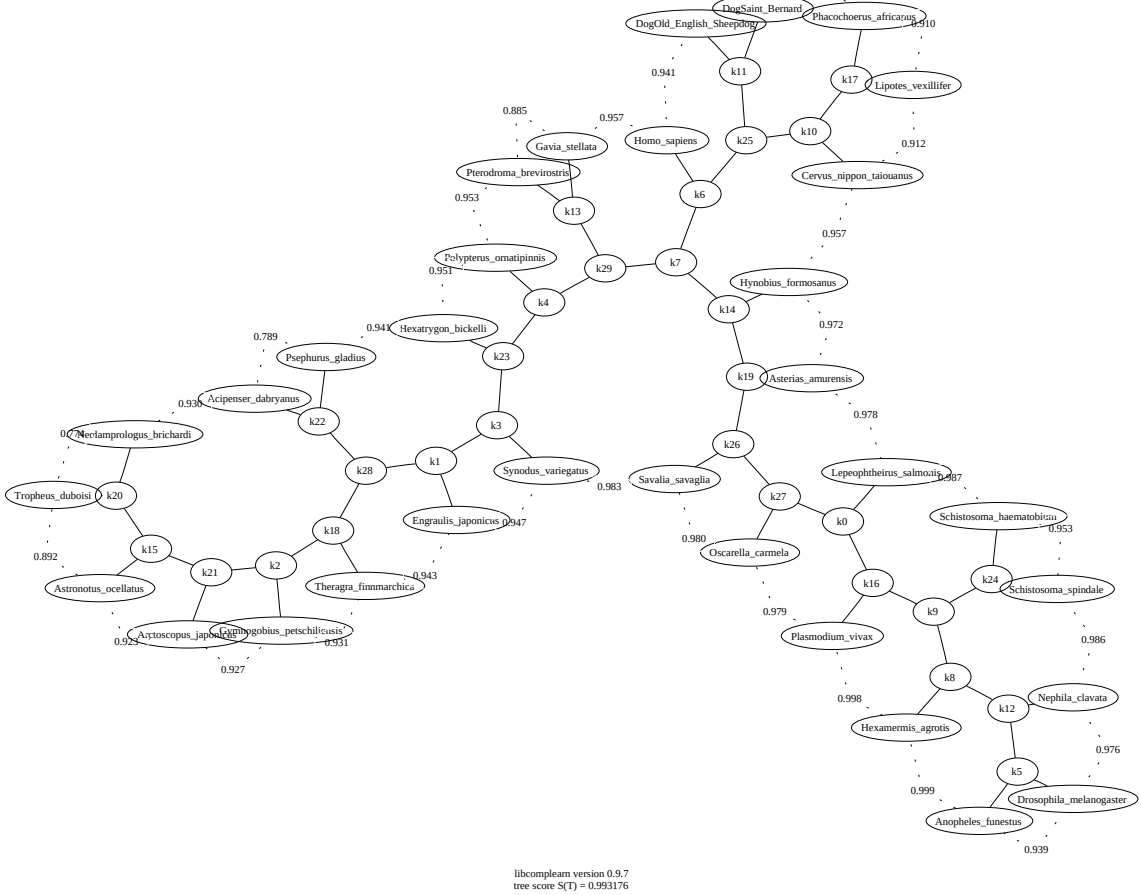

Fig. 10. CompLearn tree for comparison with previous Figure

improvement due to CompLearn's answer with the formula

$$db(T) = 10 \log_{10} \frac{R_B(T)}{R_C(T)}$$

Note that the room for improvement ratio $db(T)$ represents also a conservative estimate of the true improvement ratio in real error terms because the true maximum score of any distance matrix is less than or equal to 1. Using the $S(T_{opt})$ value of the real optimal tree T_{opt} instead of 1 would only make the ratio more extreme. We plot the decibel error reduction in Figure 11, using different binning than the earlier figure. We can see that more than 1/3 of the time CompLearn achieves at least a 2dB reduction in room for improvement as compared to SplitsTree BioNJ.

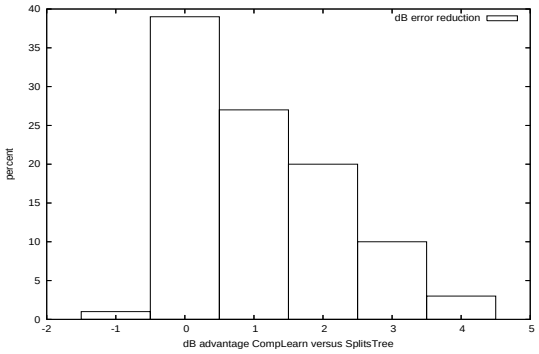

Fig. 11. Decibel error reduction from CompLearn

REFERENCES

[1] C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W. Zurek. Information Distance, *IEEE Transactions on Information Theory*, 44:4(1998), 1407–1423.

[2] C.H. Bennett, M. Li, B. Ma, Chain letters and evolutionary histories, *Scientific American*, June 2003, 76–81.

[3] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, D. Pelleg, Constructing phylogenies from quartets: Elucidation of eutherian superordinal relationships, *J. Computational Biology*, 5:3(1998), 377–390.

[4] V. Berry, T. Jiang, P. Kearney, M. Li, T. Wareham, Quartet cleaning: improved algorithms and simulations. Algorithms–Proc. 7th European Symp. (ESA99), LNCS vol. 1643, Springer Verlag, Berlin, (1999), 313-324.

[5] D. Bryant, V. Berry, P. Kearney, M. Li, T. Jiang, T. Wareham and H. Zhang. A practical algorithm for recovering the best supported edges of an evolutionary tree. *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, January 9–11, 2000, San Francisco, California, USA, 287–296, 2000.

[6] P. Buneman, The recovery of trees from measures of dissimilarity. Pp. 387–395 in: F. Hodson, D. Kenadall, P. Tautu (Eds.), Proc. of the Anlo-Romanian conference, The Royal Society of London and the Academy of the Socialist Republic of Romania, The University Press, Edinburgh, Scottland, UK.

[7] X. Chen, B. Francia, M. Li, B. McKinnon, A. Seker, Shared information and program plagiarism detection, *IEEE Trans. Inform. Th.*, 50:7(2004), 1545–1551.

[8] R. Cilibrasi, The CompLearn Toolkit, 2003, http://www.complearn.org/ .

[9] R. Cilibrasi, Statistical Inference Through Data Compression, PhD Thesis, ILLC DS-2007-01, University of Amsterdam, 2007. http://cilibrar.com/projsup/thesis.pdf

[10] R. Cilibrasi, P.M.B. Vitanyi, R. de Wolf, Algorithmic clustering of music based on string compression, *Computer Music J.*, 28:4(2004), 49-67.

[12] R.L. Cilibrasi, P.M.B. Vitanyi, The Google Similarity Distance, *IEEE Trans. Knowledge and Data Engineering*, 19:3(2007), 370-383.

[13] H. Colonius, H.H. Schulze, Trees constructed from empirical relations, *Braunschweiger Berichte as dem Institut fuer Psychologie*, 1(1977).

[14] H. Colonius, H.-H. Schulze, Tree structures for proximity data. *British Journal of Mathematical and Statistical Psychology*, 34(1981), 167-180.

[15] , S. Consoli, K. Darby-Dowman, G. Geleijnse, J. Korst and S. Pauws, Heuristic approaches for the quartet method of hierarchical clustering, Submitted to: *IEEE Trans. Knowledge Data Engin.*.

[16] C. Costa Santos, J. Bernardes, P.M.B. Vitanyi, L. Antunes, Clustering fetal heart rate tracings by compression, *Proc. 19th IEEE Symp. Computer-Based Medical Systems*, 2006, 685-690

[17] R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, 2nd Edition, Wiley Interscience, 2001.

[18] Felsenstein, J. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Molecular Evolution* 17(1981), 368–376.

[19] Gascuel O., BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. Mol. Biol. Evol., 14:685-695.

[20] D. H. Huson and D. Bryant, Application of Phylogenetic Networks in Evolutionary Studies, Mol. Biol. Evol., 23(2):254-267, 2006.

[21] T. Jiang, P. Kearney, and M. Li. A Polynomial Time Approximation Scheme for Inferring Evolutionary Trees from Quartet Topologies and its Application. *SIAM J. Computing*, 30:6(2000), 1942–1961.

[22] P.E. Kearney, Ordinal quartet method, *2nd Int'nl Conf. Comput. Molecular Biology*, 1998, 125-134.

[23] E. Keogh, S. Lonardi, and C.A. Rtanamahatana, Toward parameter-free data mining, In: *Proc. 10th ACM SIGKDD Intn'l Conf. Knowledge Discovery and Data Mining*, Seattle, Washington, USA, August 22—25, 2004, 206–215.

[24] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, *Science* 220 (1983) 671-680.

[25] T.G. Ksiazek, et.al., A Novel Coronavirus Associated with Severe Acute Respiratory Syndrome, *New England J. Medicine*, Published at www.nejm.org April 10, 2003 (10.1056/NEJMoa030781).

[26] J.R. Koza, Hierarchical genetic algorithms operating on populations of computer programs, *Proc. 11th Intn'l Joint Conf. Artificial Intell. (IJCAI'89)*, Morgan-Kaufmann, 1989, 768–774.

[27] P.S. Laplace, *A philosophical essay on probabilities*, 1819. English translation, Dover, 1951.

[28] M. Li, J.H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny, *Bioinformatics*, 17:2(2001), 149–154.

[29] M. Li and P.M.B. Vitányi. Algorithmic Complexity, pp. 376–382 in: *International Encyclopedia of the Social & Behavioral Sciences*, N.J. Smelser and P.B. Baltes, Eds., Pergamon, Oxford, 2001/2002.

[30] M. Li, X. Chen, X. Li, B. Ma, P.M.B. Vitányi. The similarity metric, *IEEE Trans. Inform. Th.*, 50:12(2004), 3250- 3264.

[31] M. Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, New York, 2nd Edition, 1997.

[32] T. Liu, J. Tang, B.M.E. Moret, Quartet methods for phylogeny reconstruction from gene orders. Dept. CS and Engin., Univ. South-Carolina, Columbia, SC, USA. Manuscript.

[33] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth. A.H. Teller and E. Teller, *J. Chem. Phys.* 21 (1953) 1087-1092.

[34] R. Piaggio-Talice, J. Gordon Burleigh, O. Eulenstein, Quartet supertrees. Chapter 4, pp. 173–191 in: O.R.P. Beninda-Edmonds (Ed.), *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*. Computational Biology, volume 3 (Dress. A., series ed.), Kluwer Academic Publishers, 2004.

phylogenies, *Nature*, 425(2003), 798–804 (25 October 2003).

[36] U. Roshan, B.M.E. Moret, T. Warnow, T.L. Williams, Performance of supertree methods on various datasets decompositions, pp 301–328 in: O.R.P. Beninda-Edmonds (Ed.), *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*. Computational Biology, volume 3 (Dress. A., series ed.), Kluwer Academic Publishers, 2004.

[37] N. Saitou, M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetic trees, *Mol. Biol. Evol.*, 4(1987), 406–425.

[38] S Snir, S Rao, Quartets Semi-Definite Programming Revisited, http://math.berkeley.edu/ ssagi/quartets-1-full.pdf

[39] S. Snir, T. Warnow and S. Rao, "Short Quartet Puzzling: A New Quartet-based Phylogeny Reconstruction Algorithm, Submitted.

[40] M. Steel, The complexity of reconstructiong trees form qualitative characters and subtrees, *Journal of Classification*, 9(1992), 91–116.

[41] K. Strimmer, A. von Haeseler, Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies, *Mol. Biol. Evol.*, 13:7(1996), 964–969.

[42] P.M.B. Vitanyi, A discipline of evolutionary programming, *Theoret. Comp. Sci.*, 241:1-2 (2000), 3–23.

[43] S. Wehner, Analyzing worms and network traffic using compression, *J. Comput. Security*, 15(2007), 303–320.

[44] J. Weyer-Menkoff, C. Devauchelle, A. Grossmann, S. Grünewald, Integer linear programming as a tool for constructing trees from quartet data, *Comput Biol Chem.* 29:3(2005), 196-203.