A Collapsing Method for the Efficient Recovery of Optimal Edges in Phylogenetic Trees

Michael Hu
University of Waterloo
mhu@math.uwaterloo.ca

Paul Kearney
Caprion Pharmaceuticals
pkearney@caprion.com

Jonathan Badger
Caprion Pharmaceuticals
jbadger@caprion.com

Abstract

As the amount of sequencing efforts and genomic data volume continue to increase at an accelerated rate, phylogenetic analysis provides an evolutionary context for understanding and interpreting this growing set of complex data. We introduce a novel quartet based method for inferring molecular based phylogeny called hypercleaning (HC*). The HC* method is based on the hypercleaning (HC) technique [2], which possesses an interesting property of recovering edges (of a phylogenetic tree) that are best supported by the witness quartet set. HC* extends HC in two regards: i)whereas HC constrains the input quartet set to be unweighted (binary valued), HC* allows any positive valued quartet scores, enabling more informative quartets to be defined. ii) HC* employs a novel collapsing technique which significantly speeds up the inference stage, making it empirically on par with quartet puzzling in terms of speed, while still guaranteeing optimal edge recovery as in HC. This paper is primarily aimed at presenting the algorithmic construction of HC*. We also report some preliminary studies on an implementation of HC* as a potentially powerful approximation scheme for maximum likelihood based inference.*

Details of mathematical proofs can be found in the tech-report at (monod.uwaterloo.ca/∼mhu/hyper/hyper.ps).

1 Introduction

Inferring phylogenetic trees on molecular sequences has wide applications in biology, from analyzing the evolutionary history of AIDS [8], to detecting regulatory elements in genetic sequences [1]. The problem of phylogenetic inference is considered hard from both a biological and computational perspective. However many biologists believe that maximum likelihood (ML) based methods are the best vehicle for conducting phylogenetic analysis [9], [14]. ML methods are, however, NP hard optimization problems requiring expensive parameter estimation and topology searching procedures, and is computationally impractical on large data sets [3]. To date many approximation methods for ML based inference have been proposed such as quartet puzzling [16], PAUP parsimony [12], and structural EM [5]. We present a novel quartet based method called hypercleaning* (HC*). HC* is based on the quartet method paradigm [6], [2], [7] and is guaranteed to return the best supported edges with respect to the quartet witness set. HC* is computationally efficient and runs with an empirical running time on the order of quartet puzzling [16]. Unlike other heuristic quartet methods such as quartet puzzling or the Short Quartet method [4], HC* is guaranteed to return those edges best supported by the witness quartet set. This effectively focuses the problem of phylogenetic inference under the quartet paradigm onto accurate construction of the witness quartet set (i.e. phylogenetic analysis on input size four). By using sophisticated ML based methods for inferring the witness quartet set, HC* can serve as an effective method for approximating ML.

2 Methods & Terminologies

Given an input set S of n sequences (e.g. homologous gene sequences on n organisms), the task is to infer the true phylogeny or tree T that captures the evolutionary relationship of these n extant leaf sequences. Note that the leaves of T are labelled by S, and the internal nodes of T represents speciation events on unobservable, ancestral objects. One class of phylogenetic inference methods are the *quartet based* methods. Most quartet based methods have two stages: 1) the witness quartet set inference stage and 2) the recombination stage. Given an input set S of size n, the first stage consists of inferring all $\binom{n}{4}$ unique quartet topologies called a witness quartet set, denoted by W. The second recombination stage, takes the inferred quartet topologies in W (i.e. hypothesis to sub-problems on input size four) and combine them into a hypothesis tree T', estimating the underlying true tree T. Figure(1) shows the high level overview of quartet based methods.

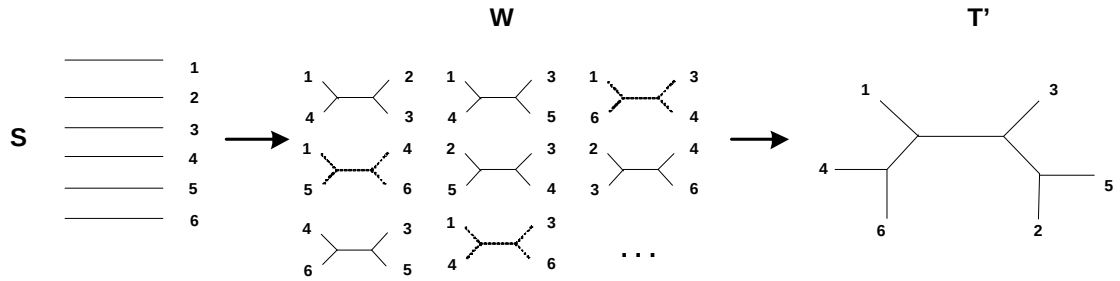

Figure 1: Overview of quartet based methods. First stage takes the input set S, and employs some existing inference method(s) to construct the witness quartet set W, shown partially. Then the quartet method pieces the information from W into a hypothesis tree T'. Note that not all quartets in set W are guaranteed to be compatible with the estimate tree T', as the ones in dashes.

Definition 1 - *A Quartet*
A quartet q in witness quartet set W, consists of the pair $\Big(\{a,b,c,d\}, \{\, w(ab|cd), w(ac|bd), w(ad|bc)\,\}\Big)$ where $a,b,c,d \in S$ and $w(ad|bc) \in \Re$ is the score (or support) that topology $ab|cd$ is contained in the underlying phylogeny T. Note that on some figures and examples to follow, we limit ourselves to *binary quartets*, where on a given quartet all the support falls onto one topology (with score of 1), and the other topologies have scores of 0. A quartet topology $ab|cd$ indicates that the path in T connecting leaves a, b is disjoint from the path connecting c, d.

The HC* algorithm has two major components: the edge inference component (HC*$_E$) and the collapsing component (HC*$_{COLLAPSE}$). For the remainder of this section, we introduce the supporting concepts and definitions, deferring the actual algorithm until the next section.

HC* on input set S and its inferred witness quartet set W, is guaranteed to return the set of edges best supported by W. We define an edge e, and the distance (i.e. lack of support) of an edge e by W as follows:

Definition 2 - *Edge (bipartition) of a phylogeny*
An edge e in an evolutionary tree T is defined by the bipartition (X,Y) where X,Y denotes the leaves set of the two disjoint sub-trees of T resulting from removing e. Note that $X \cup Y = S$, and $X \bigcap Y = \{\}$. The left part of Figure(2) illustrates.

Definition 3 - *distance function of an edge e*
Given an edge $e = (X,Y)$, its distance or **quartet error** with respect to W, is given by

$$\sigma(e,W) = \frac{\sum_{ab|cd \in W(X,Y)} w(ac \mid bd) + w(ad \mid bc)}{\binom{|X|}{2}\binom{|Y|}{2}} \quad (1)$$

where $W(X,Y)$ are the topologies of quartet entries in W induced by the edge $e = (X,Y)$. The denominator is the normalizing factor, since note that there are $\binom{|X|}{2}\binom{|Y|}{2}$ quartet topologies induced by edge e. Figure (2) illustrates.

The edge recovery component [1] of HC* , denoted HC*$_E$, has the following interface:

$$Best(m,W) \longleftarrow \text{HC*}_E(m,W)$$

whereby HC*$_E$ takes in a witness quartet set W on the input taxa set S, and input parameter $m \in \aleph$ and returns the set of best supported edges $Best(m,W)$, defined as follows:

Definition 4- *Best supported edge set* $Best(m,W)$
Given input parameter m, and the witness quartet set W, the set of best supported edges is given by:

$$Best(m,W) = \{(X,Y) :: \sigma((X,Y),W) < \frac{2m}{\mid X \mid\mid Y \mid}\} \quad (2)$$

Similar to the HC algorithm, the HC* component for constructing the set $Best(m,W)$ has a parameterized polynomial upper bound of $O(n^3 f(2m))$ (see [2] for proof), where

$$f(m) = 4m^2(1+2m)^{4m} \quad (3)$$

On lower bounded values of m, this yields a polynomial time algorithm. But on practical datasets, the value of m required such that set $Best(m,W)$ contains enough compatible edges for constructing a fully resolved tree [2] renders HC computationally intractable.

[1] This also holds in HC.

[2] Given the input set S of size n, the necessary condition for HC* to return a fully resolved (unrooted) tree, is that set $Best(m,W)$ contains at least $n-3$ compatible edges.

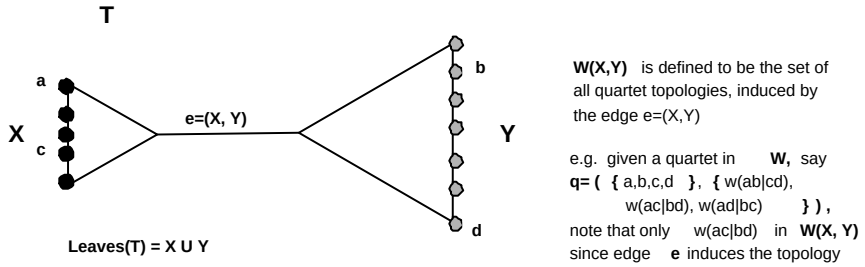

Figure 2: An edge e of an unrooted binary tree T, and the quartet topologies in the witness quartet set W induced by e.

Here we say a set of edges are compatible if they can all exist in the same binary tree.

The HC* method addresses the aforementioned efficiency problem through the use of a time/memory tradeoff collapsing mechanism, denoted HC*$_{COLLAPSE}$. In essence, the collapsing mechanism enables HC* to run at a low value of m, returning the set $Best(m, W)$ of edges. A subset of compatible edges from such a set will induce an unresolved tree, or a cluster tree. The collapsing mechanism will then collapse all but one of the clusters in this unresolved tree. The resulting single cluster can be viewed as simply a star topology on a set of leaves and collapsed nodes. Moreover this cluster set has fewer vertices than the original input set S. We can then run HC* on this smaller cluster set, and attempt to resolve edges that lie in that cluster by using a higher value of m. In other words, we reduce the input size n, such that we can raise the input parameter m, allowing more edges to be resolved without accruing the high cost of added computational time. Figure (3) shows the idea.

The collapsing mechanism guarantees no information loss during the collapse, such that any edges recovered by HC* under a collapsed cluster, will have the same score with respect to W as under no collapsing with the original input set S, by raising the value of m to the necessary level. This will be formally defined in Property 1 later on.

Definition 5 - *Cluster Tree*
A cluster tree T is a tree consisting of edges and vertices, where each vertex is either a leaf, an internal vertex (i.e. non-leaf), or a supernode. Note that a regular tree is a cluster tree whose vertices are either leaves or internal vertices. $\odot$

Definition 6 - *Cluster*
Given a cluster tree T, a cluster $C_i \in T$ is the set of vertices consisting of exactly one internal vertex, along with its degree-1 neighboring vertices (i.e. leaves and supernodes). The leaves of a cluster C_i are denoted $Leaves(C_i)$, and the supernodes (see following definition) in C_i are denoted $SuperNodes(C_i)$. Thus we have $C_i =$ an internal node $\cup\, SuperNodes(C_i) \cup Leaves(C_i)$. An internal vertex with no adjacent leaves or supernodes is a *trivial cluster*. Let $Clu(T)$ be the set of all clusters in T. $\odot$

Definition 7 - *Supernode*
A supernode $\mathbf{c}$ of a cluster tree T, is a degree-1 non-leaf vertex, denoting some collapsed cluster C_i. $\odot$

Definition 8 - *Cluster Collapse*
Given a cluster tree T and cluster $C_i \in T$, the cluster can be collapsed into a single node $\mathbf{c_i}$, only if the resulting node $\mathbf{c_i}$ is a degree-1 vertex in the remaining tree. After the collapse, the tree remains a connected tree, but has one more supernode and one less cluster. $\odot$

Figure (5) illustrates a valid and an invalid cluster collapse.

Definition 9 - *Full Collapse of a tree : Full* $\bullet\, (T \mid C_i)$
Given a (cluster) tree T, its full collapse with respect to cluster C_i, denoted $Full \bullet (T \mid C_i)$ is the ordering of all the clusters $C_j \neq C_i \in Clu(T)$, and the subsequent collapse in turn of these ordered clusters. The ordering must be valid, in the sense that each cluster, on its turn to collapse, must be collapsible as in previous definition, with respect to T. $\odot$

The result of the sequence of collapses is a single cluster $C_i^* = C_i \cup \{\mathbf{s_1}, ..., \mathbf{s_k}\}$, $k <\mid S \mid$, and the corresponding tree topology on C_i^* is simply the star topology.

Definition 10 - *Size and Cardinality of Vertices*
Given a vertex s, we define its size recursively as

$$Size(s) = \begin{cases} 1, & s \text{ is a leaf or intern. vertex} \\ \sum_{v \in C_s} Size(v) & s \text{ is a supernode, } C_s \\ & \text{is the cluster collapsed to } s \end{cases}$$

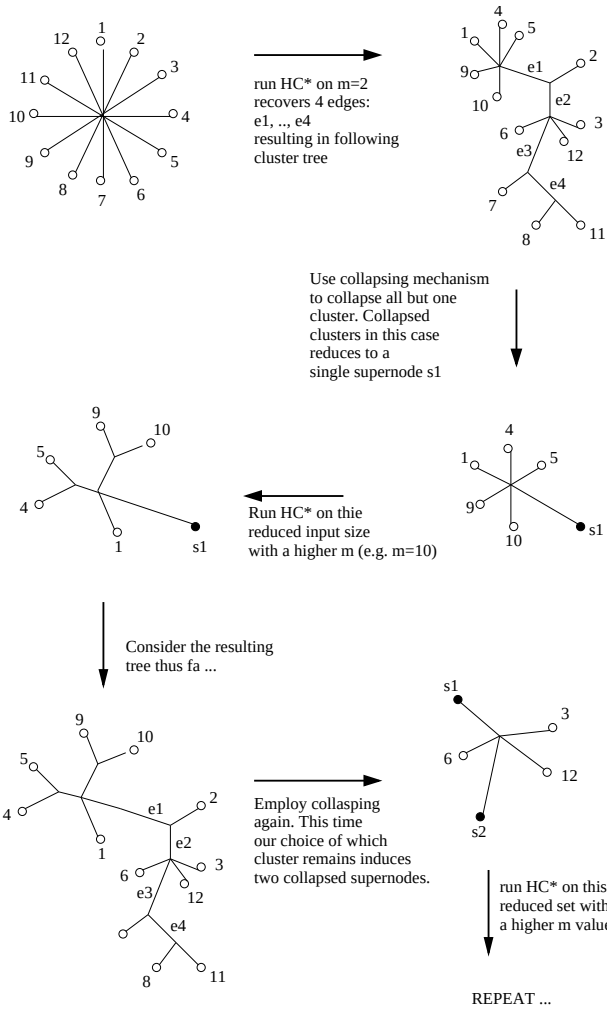

Figure 3: Running HC* with low values of m and further resolution of the unresolved tree by recursively collapsing the problem into smaller instances and then running HC* with higher values of m on these smaller problems. Here a solid dot represents a collapsed sub-tree and an open circle represents an original input leaf.

The cardinality of a vertex s, denoted $\mid s \mid$ is 1 if s is a leaf or an internal vertex, and the number of elements in its corresponding cluster if s is a supernode. $\odot$

The following lemma states that is always possible to correctly collapse the clusters of a cluster tree until only one (pre-determined)cluster remains.

Lemma - *Generating a valid collapse ordering for* $Full \bullet (T \mid C_i)$
Given tree T with k clusters, a valid ordering of the clusters can be obtained by sorting all clusters $\neq C_i$ by decreasing distance from C_i, where the $dist(C, D)$ between two clusters is simply the number of edges between their closest vertices. Ties in the ordering can be settled arbitrarily.

proof - Omitted

Definition 11 - *Expanding a supernode: exp(**s**)*
Consider a supernode **s** resulted from collapsing a cluster C_s. The expansion of **s**, denoted $exp(\mathbf{s})$ is a recursive procedure, which returns a set of leaves as follows:

$$exp(\mathbf{s}) = \left\{ Leaves(C_s) \bigcup_{\forall s' \in SuperNode(C_s)} exp(s') \right\}$$

$\odot$

In the context of collapsing, an edge $e = (X, Y)$ defined by its bipartition of vertices also becomes more general. In a normal tree T, an edge $e = (X, Y)$ consists of two disjoint but pairwise complete sets on the leaves S. In a cluster tree T_C, an edge $e' = (X', Y')$ also consists of two disjoint, pairwise complete sets on all degree-1 vertices in T_C (i.e. leaves and supernodes).

Definition 12 -*Expansion of an edge e in a cluster tree*
Given an edge $e = (X, Y)$ in cluster tree T_C, its full expansion, denoted $Exp(e = (X, Y))$ returns the edge $e' = (X', Y')$ where X', Y' is a bipartition on the original leaves set S. Formally, the $Exp(e = (X, Y))$ is recursively defined as follows:

$$Exp(e = (X, Y)) = (X', Y') =$$
$$\Bigg(Leaves(X) \bigcup_{s \in SuperNode(X)} exp(s) \ ,$$
$$Leaves(Y) \bigcup_{s \in SuperNode(Y)} exp(s) \Bigg)$$

Having the above definitions on a cluster tree, collapsing, and expanding under a cluster tree, we are ready to formally state the definition of what it means for HC* to achieve information loss-less collapsing of the cluster tree into a single cluster, and subsequent resolution of edges on that cluster.

Property 1 - *Information loss-less edge recovering under collapsing*
Given input leaves set S and its witness quartet set W, assume that HC* is at some stage of collapsing some semi-resolved cluster tree into a star topology tree, whose leaves are on the cluster set:

$$C = \{l_1, l_2, ..l_k, \mathbf{s1}, .., \mathbf{sp}\} \tag{4}$$

consisting of leaves and supernodes. We wish to construct the quartet set W^* on the vertices of C, such that running HC* on input C using W^*, will return the edge set $Best(m, W^*)$ with the following accuracy guarantee:

$$\forall \text{ edges } e \in Best(m, W^*) :: \sigma(e, W^*) = \sigma(Exp(e), W) \tag{5}$$

Such a quartet set W^* then satisfies the **information lossless property** of the cluster C. $\odot$

3 Algorithms

The HC* algorithm has two main components, i) the edge recovery component, denoted HC*$_E$ and a col-

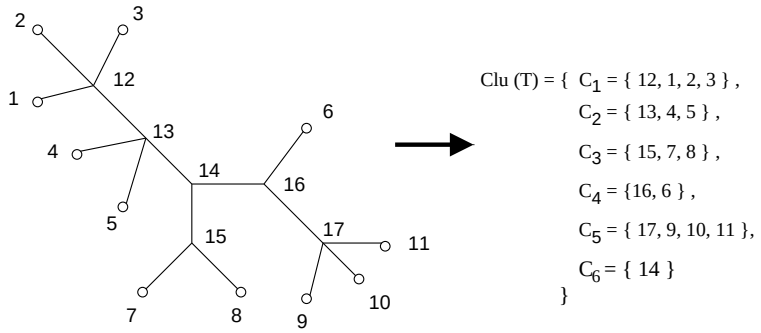

Figure 4: An unresolved cluster tree, with clusters $C_1, .., C_6$. C_6 is a trivial cluster

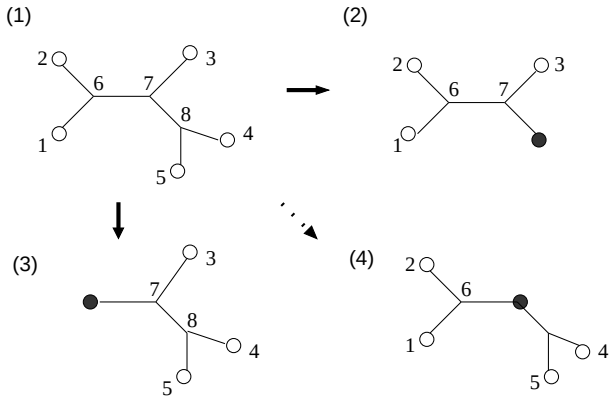

Figure 5: The cluster tree (1) has 3 clusters: $C_6 = \{6, 1, 2\}$, $C_7 = \{7, 3\}$, and $C_8 = \{8, 4, 5\}$. Tree (2)results from the valid collapse of cluster C_8 in tree (1), since the resulting vertex satis£es the de£nition of a supernode. (3) is the valid collapse of cluster C_6 from (1). In tree (4), the collapse of cluster C_7 is not valid since the resulting vertex does not satisfy the de£nition of a supernode.

lapsing component HC*$_{COLLAPSE}$. Figure(6) shows the high level flow of HC*.

3.1 The HC*$_E$ Component

This section describes the algorithmic construction of the HC*$_E$ component. HC*$_E$ on input vertex set C of size n, and integer parameter m produces the edge set $Best(m, W)$ which we redefine as follows:

$$Best(m, W) = \{(X, Y) \mid \sigma(W, (X, Y)) < \frac{2m}{Size(X)Size(Y)}\} \quad (6)$$

Notice that the above definition is a slight modification of our earlier definition (2), since in this more general setting, our input node set C might have a mix of leaves and supernodes.

The HC*$_E$ produces the set $Best(m, W)$ by first constructing for all pairs $x, y \in C$, the set:

$$Best_{xy}(m, W_k) = \Big\{(X, Y) :: \quad (7)$$

$$\sum_{ax|by \in W(X,Y)} \frac{w(ay|bx) + w(ab|xy)}{Size(x)Size(y)} < m \Big\} \quad (8)$$

where W_k is the subset of W induced by the sequence of vertices: $S_k = \{x, y, v_1, v_2, ..., v_{k-2}\} \subseteq C$, $k < n$ and $Size(x)$ as defined in the previous section. The construction of this set is iterative on $k = 1, ..., n$. Moreover any trivial edges with either zero or one element in either of its two partitions belongs to $Best_{xy}(m, W_k)$, although trivial edges induces no quartets. Note that $Best_{xy}(m, W) = Best_{xy}(m, W_k)$ when $k = n$.

Procedure 1 - Constructing $Best_{xy}(m, W_k)$ for $k = 1, .., n$

If $k = 1$ then $Best_{xy}\{m, W_k\} = \emptyset$, since W_1 does not contain any bipartitions.
Else If $k = 2$, then $Best_{xy}\{m, W_k\} = \{(\{x\}, \{y\})\}$.
Else If $k \geq 3$, then

$Best_{xy}(m, W_k) = \forall$ edges $e \in L_{xy} \cup R_{xy}$ satisfying Eqn (8)

where

$$L_{xy} = \{(X \cup \{s_k\}, Y) \mid (X, Y) \in Best_{xy}(W_{k-1}, m)\}$$
$$R_{xy} = \{(X, Y \cup \{s_k\}) \mid (X, Y) \in Best_{xy}(W_{k-1}, m)\}$$

and the vertex $\{s_k\}$ is the kth element drawn from the sequence
$S_k = \{x, y, v_1, v_2, ..., v_{k-2}\} \subseteq C$. $\odot$

Theorem 1 - Given input S and W, the set constructed under the above procedure, $Best_{xy}(m, W_k)$, for $k = n$, satisfies the definition of $Best_{xy}(m, W)$ as given in Equation (8) .

proof− Omitted

Procedure 2 - Constructing the set $Best(m, W_k)$

On input set S of n leaves, the set $Best(m, W_k)$, is defined iteratively from $Best(m, W_{k-1})$, for $2 \leq k \leq n$, where $Best(m, W) = Best(m, W_n)$. W_k is the subset of quartets of W induced by the subset of leaves $S_k = \{v_1, v_2, ..., v_k\} \subseteq S$.

If $k = 1$ then $Best(m, W_1) = \emptyset$.
If $k \geq 2$ then

$Best(m, W_k) = \forall$ edges $e \in L \cup R \cup M$ satisfying Eqn(6)

HC* (S : input nodes ,T^* : star tree, W : witness set)
OUTPUT: Resolved tree: T

0 $k = 0$;
1 $T^k = T^*$;
2 $V^k = V(T) = S$; $W^k = W$;
3 WHILE (T^k not fully resolved)
{ Alternatively, we could demand that T^k is resolved to some user specified threshold, and then apply other methods to resolve the few remaining edges. Some applications might only require that a subset of edges of the underlying phylogeny be estimated. }

4 FOR some cluster $C_i \in Clu(T^{(k)})$
5 $W^{k+1}, V^{k+1} \leftarrow HC^*_{COLLAPSE}\left(Full \bullet (T^{(k)} \mid C_i), V^k, W^k\right)$
6 $Best(m, W^{k+1}) = HC^*_E(V^{k+1}, m, W^{k+1})$
7 $T^{k+1} \leftarrow$ insert compatible edges $e \in Best(m, W^{k+1})$ into T^k
8 $k = k + 1$
9 $V^k = V(T^k)$

Figure 6: High level view of the HC* algorithm

where the sets L, R, M are constructed as follows:

$$L = \{(X \cup \{s_k\}, Y) :: (X, Y) \in Best(m, W_{k-1})\}$$
$$R = \{(X, Y \cup \{s_k\}) :: (X, Y) \in Best(m, W_{k-1})\}$$
$$M = \bigcup_{x \in S_{k-1}} Best_{xs_k}(m, W_k)$$

Theorem 2 - Given input S and W, the resulting set from Procedure 2, $Best(m, W_k)$ for $k = n$, defines the set $Best(m, W)$ as given by Equation (6).

proof - Omitted.

Procedures 1 and 2 in essence describe the iterative construction of the set $Best(m, W)$. The efficiency is that at each stage for $1 \leq k \leq n$, the cardinality of the set $Best(m, W_k)$ is inherently constrained by the size of the sets $Best(m, W_{k-1})$ and $Best_{xy}(m, W_k)$, with the guarantee that $Best(m, W)$ does not miss any edges. This in essence constrains the edge space to be searched through, whereas the naive search on n leaves involves searching through 2^n edges.

3.2 The HC$_{COLLAPSE}$ Component

Suppose we run HC* where we have already done k number of collapses (i.e. the WHILE loop in Figure 6), resulting in the current cluster tree $T_C^{(k)}$, on vertex set $V^{(k)}$ and the quartet set $W^{(k)}$ on V^k. Assume that the quartet set W^k satisfies the loss-less condition. Consequently for any edge $e = (X, Y)$ on V^k, we have:

$$\sigma(e = (X, Y), W^k) = \sigma(Exp(e), W)$$

where W is the quartet set on the original input set S. Now HC* will choose a cluster $C_i \in T_C^k$ and perform $Full \bullet (T_C^k \mid C_i)$, effectively creating a single cluster. The following procedure describes how one produces the witness quartet set W^{k+1} on the resulting single cluster V^{k+1} such that the loss-less condition is preserved.

Procedure 3
$W^{k+1}, V^{k+1} \leftarrow$ HC*$_{COLLAPSE}(Full \bullet (T_C^k | C_i), V^k, W^k)$
Given some unresolved cluster tree T_C^k on vertex set V^k with witness set W^k, we wish to collapse all the clusters of T_C^k except C_i. The collapse ordering is defined by $Full \bullet (T_C^k \mid C_i)$. Without loss of generality, suppose the collapse ordering is given by:

$$C_1, C_2, ..., C_{i-1}, C_{i+1}, ..., C_q$$

The procedure is governed by the aforementioned collapse ordering and can be characterized by a sequence

of corresponding function calls to HC*$_{COLLAPSE}$:

$$W^k_{(1)}, V^k_{(1)} \leftarrow \text{HC*}_{COLLAPSE_1}(V^k, W^k, C_1)$$
$$W^k_{(2)}, V^k_{(2)} \leftarrow \text{HC*}_{COLLAPSE_2}(V^k_{(1)}, W^k_{(1)}, C_2)$$
$$\cdots$$
$$W^{k+1}, V^{k+1} \leftarrow \text{HC*}_{COLLAPSE_q}(V^k_{(q-1)}, W^k_{(q-1)}, C_q)$$

The next procedure defines the actual algorithm for:

$$\text{HC*}_{COLLAPSE_{j+1}}(V^k_{(j)}, W^k_{(j)}, C_{j+1})$$

for $j = 0, \ldots, q-1$. Note that V^0, W^0 corresponds to the original input vertex set and its quartet set V, W.

Procedure 4

$$W_{(j+1)}, V_{(j+1)} \leftarrow \text{HC*}_{COLLAPSE_{(j+1)}}\left(V^k_{(j)}, W^k_{(j)}, C_{j+1}\right)$$

Assume we are currently in the k-th iteration of $Full \bullet (T_C \mid C_i)$, and have performed q cluster collapses thus far, resulting in vertex sets $V_{(j)}$, and quartet sets $W_{(j)}$, for $j = 0, ..., q-1$. Moreover we assume that $W_{(j)}$ on $V_{(j)}$ satisfies the information loss-less precondition. Assume we want to collapse a cluster $C \neq C_i \subseteq V_{(j)}$ into supernode $\mathbf{s}$, such that we have $V_{(j+1)} = V_{(j)} - C + \mathbf{s}$. Thus, we construct an updated quartet set $W_{(j+1)}$ on $V_{(j+1)}$ as follows, to satisfy the loss-less precondition:

Consider the following combinations of quartet: $(a, b, c, d) \in V_{(j+1)} = V_{(j)} - C + \mathbf{s}$, where these vertices do not have to be all distinct [3]

C1 (a, b, c, d), where $a \neq b \neq c \neq d \in V_{(j+1)} - \mathbf{s}$:
For all such quartets, assign:

$$w_{(j+1)}(ab \mid cd) = w_{(j)}(ab \mid cd)$$
$$w_{(j+1)}(ac \mid bd) = w_{(j)}(ac \mid bd)$$
$$w_{(j+1)}(ad \mid bc) = w_{(j)}(ad \mid bc)$$

C2 (a, a, c, d), where $a \neq c \neq d \in V_{(j+1)} - \mathbf{s}$:
For quartets of this form, assign:

$$w_{(j+1)}(ac \mid ad) = w_{(j)}(ac \mid ad)$$

C3 (a, b, c, c), where $a \neq b \neq c \in V_{(j+1)} - \mathbf{s}$:
For quartets of this form, assign:

$$w_{(j+1)}(ac \mid bc) = w_{(j)}(ac \mid bc)$$

C4 (a, a, b, b), where $a \neq b \in V_{(j+1)} - \mathbf{s}$:
For quartets of this form, assign:

$$w_{(j+1)}(ab \mid ab) = w_{(j)}(ab \mid ab)$$

C5 $(a, a, \mathbf{s}, \mathbf{s})$, where $a \in V_{(j+1)} - \mathbf{s}$
For these quartets, assign:

$$w_{(j+1)}(ab \mid ab) = \sum_{c,d \in C} w_{(j)}(ac \mid ad) + \sum_{c \in C} w_{(j)}(ac \mid ac)$$

C6 $(a, b, c, \mathbf{s})$, where $a \neq b \neq c$, $a, b, c \in V_{(j+1)} - \mathbf{s}$:
For quartets of this form, assign:

$$w_{(j+1)}(ab \mid c\mathbf{s}) = \sum_{d \in C} w_{(j)}(ab \mid cd)$$

$$w_{(j+1)}(ac \mid b\mathbf{s}) = \sum_{d \in C} w_{(j)}(ac \mid bd)$$

$$w_{(j+1)}(bc \mid a\mathbf{s}) = \sum_{d \in C} w_{(j)}(bc \mid ad)$$

C7 $(a, b, \mathbf{s}, \mathbf{s})$, where $a \neq b \in V_{(j+1)} - \mathbf{s}$
For these quartets, assign:

$$w_{(j+1)}(a\mathbf{s} \mid b\mathbf{s}) = \sum_{c \in C} w_{(j)}(ac \mid bc) +$$
$$\frac{1}{2} \sum_{c,d \in C} \left(w_{(j)}(ac \mid bd) + w_{(j)}(ad \mid bc) \right)$$

C8 $(a, a, b, \mathbf{s})$, where $a \neq b \in V_{(j+1)} - \mathbf{s}$
For these quartets, assign:

$$w_{(j+1)}(ab \mid a\mathbf{s}) = \sum_{d \in C} w_{(j)}(ab \mid ad)$$

$\odot$

Theorem 3

Given procedures 3 and 4 construction of HC*$_{COLLAPSE_{(j+1)}}(V^k_{(j)}, W^k_{(j)}, C_{j+1})$ on the $(j+1)$-th cluster collapse in the sequence of cluster collapses as given by $Full \bullet (T_C \mid C_i)$, the resulting quartet set $W_{(j+1)}$ on the updated vertex set $V_{(j+1)}$ satisfies the information loss-less precondition. Note this is performed in the kth iteration of the outer WHILE loop of the HC* algorithm (see Figure(6)).

proof- Omitted

[3] Quartets can be of the form (a, a, b, c) since a might be a supernode which semantically represents collapsed leaves and or other supernodes. As such when we consider the distance score for a quartet topology $a, a \mid b, c$, we take into account the following alternative: $a, b \mid a, c$ which represents all quartets of the form $a_1, b \mid a_2, c$, where a_1, a_2 are two collapsed vertices in supernode a. A quartet of the form (a, a, a, b) is impossible since no supernode can span across an edge.

	SSU_30	SSU_30b	SSU_50	SSU_50b	Euth51	SSU_75
avg accuracy %	65.6/74.5	66.1/70.8	56.6/64.3	46.3/46.4	68.1/77.5	62.8/68.5
p-value	4.722e-004	0.028	4.151e-004	0.454	4.767e-005	9.766e-004

Figure 7: Average accuracy of treepuzzle / HC* and the p value on all sampled trees.

4 Results

We designed an experiment as a preliminary gauge of the utility of HC* for approximating the Maximum Likelihood (ML) method under realistic circumstances (i.e. when our assumptions about the model of evolution is only partially correct). In particular we compared HC* with treepuzzle [16] (the original implementation of the quartet puzzling algorithm, considered by many to be one of the better ML approximation methods to date). We generated the input sequence sets using simulation on known (ML analyzed) tree topologies, The simulated datasets were generated by first choosing several tree topologies with varying branch length composition, on input leaves sizes: 30, 50, 75. We than generated sequence datasets by 'evolving' sequences along these topologies using the HYK model of evolution with perturbations to its various parameters. The sequences were generated using Seq-Gen [11].
For the tree topologies we chose:

- two 30-taxon tree topologies randomly sampled from the tree of the set of 218 representative prokaryotic sequences from the RDP database: denoted SSU_30, SSU_30b.
- two 50-taxon tree topologies randomly sampled from the RDP prokaryotic representative tree from the RDP database, denoted SSU_50, SSU_50b, as well as the 51-taxon Eutherian tree, denoted $Euth51$ from [13].
- one 75-taxon tree, denoted SSU_75, sampled from the same source tree.

We then simulated sequences along the topology sets using the HYK model of evolution by perturbing 4 parameters as follows: i) sequence lengths (2000, 4000), ii)gamma heterogeneity (0.2, 0.5) , iii) branch lengths scaling factor (1, 4, 10), and iv) transversion/transition ratio (2, 4), effectively generating 24 sequence sets per tree topology.[4] Using these simulated sequences as input, we ran both treepuzzle [16] and HC* where the witness quartet set [5] was inferred using fastDNAml [10]. With both fastDNAml and treepuzzle, we ran the software on default settings, such that the assumptions on the model of evolution is 'incorrect' with respect to the generated sequences. We then proceed to test the accuracy and robustness of the inference methods under such a circumstance when our model of evolution does not match the actual evolutionary process. Figure 6 shows the average accuracy (given as the percentage of edges shared with the true underlying tree), and the p-value for testing the statistical significance of accepting the null hypothesis (that the accuracy datapoints on treepuzzle and HC* came from the same distribution). This was performed using the Wilcoxon paired-sample test (i.e. non-parametric paired t-test) in MATLAB. From table(7) we see that HC* outperforms treepuzzle, in a statistically significant way, on all sequences except those on topology SSU_50b. This preliminary study shows that HC* tends to be more robust than treepuzzle on inferring phylogenies when the assumptions on the model of evolution are broken, although more detailed studies need to be conducted to determine the effect of the various parameters on the model of evolution and their effects on the performance of HC*.

On average HC* runs on the order of 2 times slower than treepuzzle, although most (90%+) of the CPU time was spent on inferring the witness quartet set through time-consuming process calls to fastDNAml. Currently we are adopting fastDNAml (as well as other ML techniques) directly into the HC* source code, which should significantly speed up the run time of HC*. This enables HC* to be practical on larger datasets (input set $n > 100$). Moreover the witness quartet set lends itself naturally to be parallel computed should the need arise.

5 Conclusion & Future Work

This paper describes the HC* method as an efficient method for recovery phylogenetic tree edges best supported by the witness quartet set. The HC* method makes interesting accuracy guarantees on the edges recovered with respect to the witness quartet set. This focuses a larger phylogenetic inference problem into numerous, smaller subproblems on input size four. As more sophisticated and powerful ML methods arise on learning models of evolution from sequences (given fixed topologies or on small topology search spaces) [17], [14], the accuracy on inferring quartet topologies will continue to improve. Thus HC* becomes a potentially powerful technique for approximing ML based phylogenetic tree

[4] On the 75 taxon tree we simulated only 16 sequences, by omitting values 4 on branch scaling parameter due to heavy computations.

[5] The quartets were unweighted, where a score of 1 was assigned to the highest scoring topology and 0 to the two alternative topologies on a given quartet.

inference.

Due to the efficient and optimal nature of HC*, one area of further investigation would be using HC* to evaluate the effectiveness of weighted quartet methods. This stems of the motivation that some early studies [15] suggests that unweighted quartet methods display poor scalability properties. In particular, it would be interesting to combine sophisticated inference techniques such as in [14] for inferring quartet topologies along with various schemes for constructing weighted quartets (e.g. ensemble learning) to further probe the scalability properties of weighted quartet methods using HC* as a baseline study.

References

[1] Blanchette, M. Algorithms for phylogenetic footprinting. In *Proceedings of the 5th Conference on Research in Computational Molecular Biology* (2001), ACM Press, pp. 49–58.

[2] Bryant, D., Berry, V., Kearney, P., Li, M., Jiang, T., and Wareham, T. A practical algorithm for recovering the best supported edges of an evolutionary tree. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2000), ACM Press, pp. 287–296.

[3] Cao, Y., Adachi, J., and Hasegawa, M. Comments on the quartet puzzling method for finding maximum likelihood tree topologies. *Mol. Bio. Evol. 15* (1998), 87–89.

[4] Erdos, P., Rice, K., Steel, M., Szekely, L., and Warnow, T. The short quartet method. *International Congress on Automata, Languages and Programming* (1997).

[5] Friedman, N., Ninio, M., and Pe'er, I. A structural EM algorithm for phylogenetic inference. In *Proceedings of the 5th Conference on Computational Molecular Biology* (2001), ACM Press, pp. 132–140.

[6] Jiang, T., Kearney, P., and Li, M. A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM Journal on Computing 30* (2001), 1942–1961.

[7] Jiang, T., and Zhang, M., Eds. *Current Topics in Computational Molecular Biology.* MIT Press, 2002.

[8] Korber, B., Muldon, M., Theiler, J., Gao, F., Gupta, R., Lapedes, A., Hahn, B., Wolinsky, S., and Bhattacharya, T. Timing the ancestor of the {HIV-1} pandemic strains. *Science 288* (2000), 1789–96.

[9] Kuhner, M., and Felsenstein, J. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution 11* (1994), 459–468.

[10] Olsen, G., Matsuda, H., Hagstrom, R., and Overbeek, R. Fastdnaml: a tool for construction of phylogenetic trees of dna sequences using maximum likelihood. *Current Applications in Biosciences 10* (1994), 41–48.

[11] Rambaut, A., and Grassly, N. Seq-gen: An application for the monte carlo simulation of dna sequence evolution along phylogenetic trees. *Comp. Appl. Biosci. 13* (1997), 235–238.

[12] Rogers, J., and Swofford, D. A fast method for approximating maximum likelihoods of phylogenetic trees from nucleotide sequences. *Systematic Biology 47* (1998), 77–89.

[13] Rosenberg, M., and Kumar, S. Incomplete taxon sampling is not a problem for phylogenetic inference. *Proc Natl Acad Sci, USA 98(19)* (2001), 10751–6.

[14] Schadt, E., Sinsheimer, J., and Lange, K. Computational advances in maximum liklihood methods for molecular phylogeny. *Genome Research* (1998), 222–233.

[15] St.John, K., Warnow, T., Moret, M., and Vawter, L. Performance study of phylogenetic methods: (unweighted) quartet methods and neighbour-joining. In *Symposium on Discrete Algorithms* (2001), ACM Press, pp. 196–205.

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

[17] Yang, Z. Maximum likelihood analysis of adaptive evolution in hiv-1 gp120 env gene. In *Pacific Symposium on BioComputing* (2001), pp. 226–237.