Construction, enumeration, and optimization of perfect phylogenies on multi-state data.

Michael Coulombe[1] Kristian Stevens[2] Dan Gusfield[2]
mcoulomb@mit.edu {kastevens,gusfield}@ucdavis.edu

[1]Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

[2]Department of Computer Science
University of California, Davis

5th IEEE International Conference on Computational Advances in Bio and Medical Sciences

Outline

Outline

Phylogeny Problem and Approaches

The Phylogeny Problem

Given extant taxa with observed traits, reconstruct an evolutionary tree which best explains their ancestral relationships.

1. Distance-Based Algorithms
    - Must know or estimate evolutionary distances between taxa.
    - Must choose a metric and clustering strategy.
2. Maximum Parsimony and Maximum Likelihood
    - Must model and give costs to evolutionary events.
    - Must efficiently prune the search-space to find the optimal tree.

Phylogeny Problem and Approaches

The Phylogeny Problem

Given extant taxa with observed traits, reconstruct an evolutionary tree which best explains their ancestral relationships.

1. Distance-Based Algorithms
    - Must know or estimate evolutionary distances between taxa.
    - Must choose a metric and clustering strategy.
2. Maximum Parsimony and Maximum Likelihood
    - Must model and give costs to evolutionary events.
    - Must efficiently prune the search-space to find the optimal tree.

Multi-state Perfect Phylogeny Problem

Input

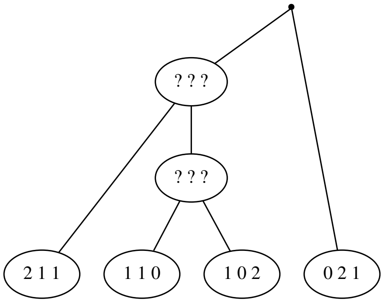
A set S of n **taxa** for which there are m **characters**, or observed traits. Each character takes on at most k distinct **states**.

Output

A **perfect phylogeny** of S: a tree T with leaves labeled by the taxa and ancestors labeled such that each character-state is **convex** with respect to T.

Characters

$$\begin{array}{c|ccc|}
 & 0 & 2 & 1 \\
 & 1 & 0 & 2 \\
\text{Taxa} & 1 & 1 & 0 \\
 & 2 & 1 & 1 \\
\end{array}$$



Multi-state Perfect Phylogeny Problem

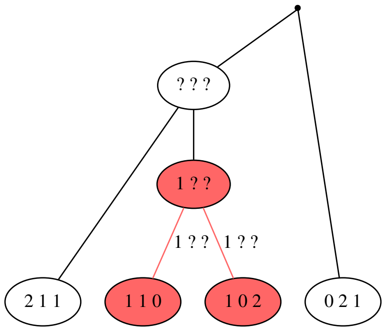
Input

A set S of n **taxa** for which there are m **characters**, or observed traits. Each character takes on at most k distinct **states**.

Output

A **perfect phylogeny** of S: a tree T with leaves labeled by the taxa and ancestors labeled such that each character-state is **convex** with respect to T.

$$\begin{array}{c|ccc} & \multicolumn{3}{c}{\text{Characters}} \\ \hline \text{Taxa} & 0 & 2 & 1 \\ & 1 & 0 & 2 \\ & 1 & 1 & 0 \\ & 2 & 1 & 1 \end{array}$$



Multi-state Perfect Phylogeny Problem

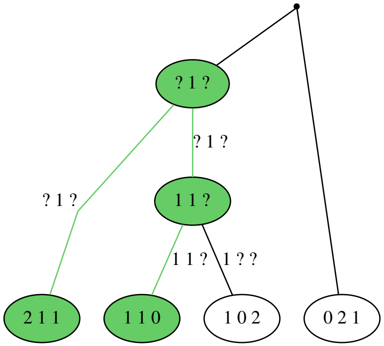
Input

A set S of n **taxa** for which there are m **characters**, or observed traits. Each character takes on at most k distinct **states**.

Output

A **perfect phylogeny** of S: a tree T with leaves labeled by the taxa and ancestors labeled such that each character-state is **convex** with respect to T.

$$\begin{array}{c|ccc}
 & \multicolumn{3}{c}{\text{Characters}} \\
\text{Taxa} & 0 & 2 & 1 \\
 & 1 & 0 & 2 \\
 & 1 & 1 & 0 \\
 & 2 & 1 & 1
\end{array}$$



Multi-state Perfect Phylogeny Problem

Input

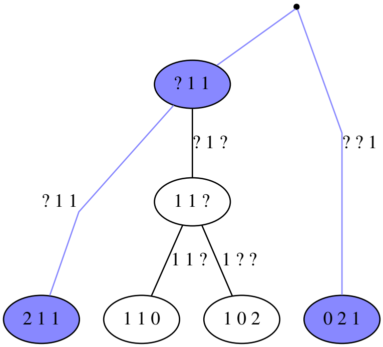
A set S of n **taxa** for which there are m **characters**, or observed traits. Each character takes on at most k distinct **states**.

Output

A **perfect phylogeny** of S: a tree T with leaves labeled by the taxa and ancestors labeled such that each character-state is **convex** with respect to T.

Characters

$$\begin{array}{c|ccc|}
 & 0 & 2 & 1 \\
 & 1 & 0 & 2 \\
\text{Taxa} & 1 & 1 & 0 \\
 & 2 & 1 & 1 \\
\end{array}$$



Multi-state Perfect Phylogeny Problem

Input

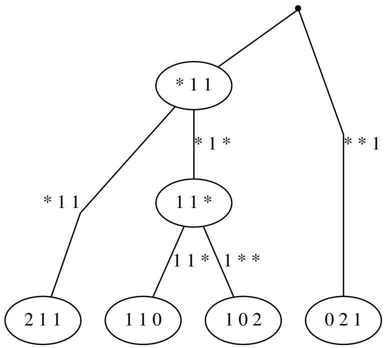
A set S of n **taxa** for which there are m **characters**, or observed traits. Each character takes on at most k distinct **states**.

Output

A **perfect phylogeny** of S: a tree T with leaves labeled by the taxa and ancestors labeled such that each character-state is convex with respect to T.

Characters

$$\begin{array}{c|ccc|}
 & 0 & 2 & 1 \\
 & 1 & 0 & 2 \\
\text{Taxa} & 1 & 1 & 0 \\
 & 2 & 1 & 1 \\
\end{array}$$



Perfect Phylogeny with Bounded Number of States

Problem known to be NP-Hard for arbitrary n, m, k [**?**].

Result	**Construction Time**	**Notes**
[**?**]	$O(nm)$	Binary data ($k = 2$)
[**?**]	$O(nm^2)$	3-State data ($k \leq 3$)
[**?**]	$O(n^2 m)$	4-State data ($k \leq 4$)
[**?**]	$O(2^{3k}(nm^3 + m^4))$	Fixed Parameter Tractable in k
[**?**]	$O(2^{2k} nm^2)$	Improvement on [**?**]

Outline

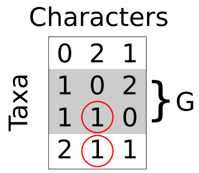
Proper Clusters [?]

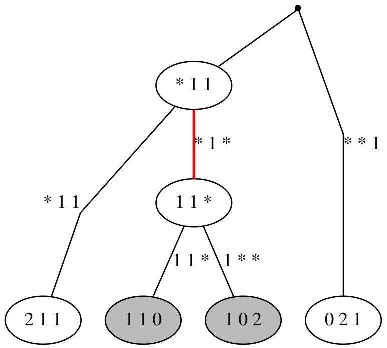
Definition

$G \subset S$ is a **proper cluster** if each character shares at most one state between G and $S - G$, and some character shares none.

Definition

The **splitting vector** $Sv(G) = v$ of proper cluster G is the vector where $\alpha(v)$ is the unique shared state of character α between some $x \in G$ and $y \in S - G$, else $\alpha(v) = *$ if no state is shared.


Characters
Taxa
0 2 1
1 0 2
1 1 0
2 1 1
G



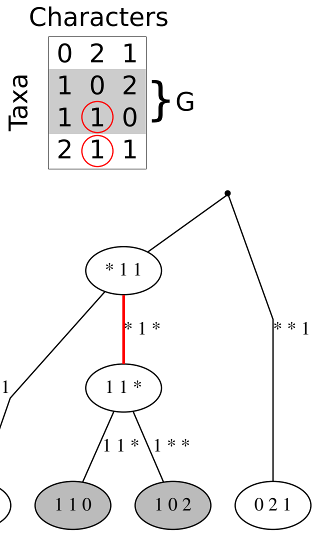
Proper Clusters [?]

Lemma

If S has a perfect phylogeny, then S has a perfect phylogeny where the leaf set of every subtree is a proper cluster.

Intuition:

1. Each edge must share at most one character due to convexity.
2. If all characters share a state over an edge, then the edge can be contracted.



Preprocessing: $S/Sv(G)$ [?]

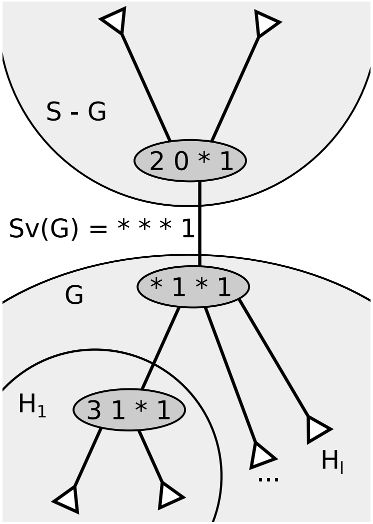
Definition

Given $G \subseteq S$ and $v \in \{*, 1, \ldots, k\}^n$, G/v groups taxa which share a character-state not present in v.

Example

$$S/Sv(G) = \{\{a \mid \alpha_4(a) = i\} \mid i \neq 1\} \cup \{\{a\} \mid \alpha_4(a) = 1\}$$

If $a \sim b$ and $Sv(G)$ labels an edge, then a and b **must** be on the same side of the edge due to convexity.



Recursive Formulation of [?] and [?]

$\textsc{PerfectPhylogeny}(S)$

```
1 if $T \leftarrow \textsc{SubPhylogeny}(S - \{t_{out}\})$ returns failure  then return failure
2 else return the tree created by attaching $t_{out}$ to the root of $T$
```

$\textsc{SubPhylogeny}(G)$

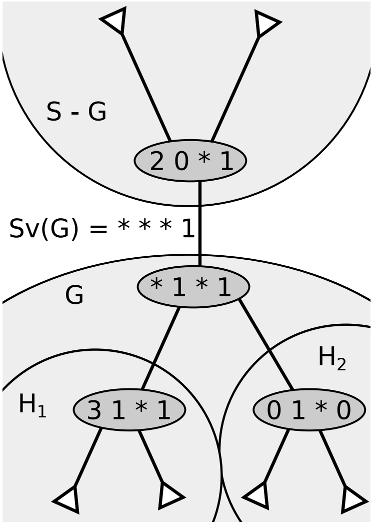
```
1 initialize root $r$ labeled with $Sv(G)$
2 if $G$ is a single taxon $t$  then return the taxon $t$ attached to $r$
3 foreach subset $H_1$ of $G$ where
      $T_{H_1} \leftarrow \textsc{SubPhylogeny}(H_1)$ exists and can be attached to $r$
4   if $H_2 \leftarrow G - H_1$ is a proper cluster
5      if $T_{H_2} \leftarrow \textsc{SubPhylogeny}(H_2)$ exists and can be attached to $r$
6         return the tree created by attaching $T_{H_1}$ and $T_{H_2}$ to $r$
7   elsif $G$ can be partitioned into $l > 2$ proper clusters $H_1, \ldots, H_l$
          with subphylogenies $T_{H_1}, \ldots, T_{H_l}$ that can be attached to $r$
8      return the tree created by attaching $T_{H_1}, \ldots, T_{H_l}$ to $r$
9  return failure if no $H_1$ worked
```

Recursive Formulation of [?] and [?]

1. If $H_2 \leftarrow G - H_1$ is a proper cluster:

SUBPHYLOGENY(G)

```
    ...
3   foreach subset $H_1$ of $G$ where
        $T_{H_1} \leftarrow$ SUBPHYLOGENY($H_1$) exists
        and can be attached to $r$
      ...
5       if $T_{H_2} \leftarrow$ SUBPHYLOGENY($H_2$) exists
           and can be attached to $r$
6          return the tree created by attaching
                $T_{H_1}$ and $T_{H_2}$ to $r$
      ...
```

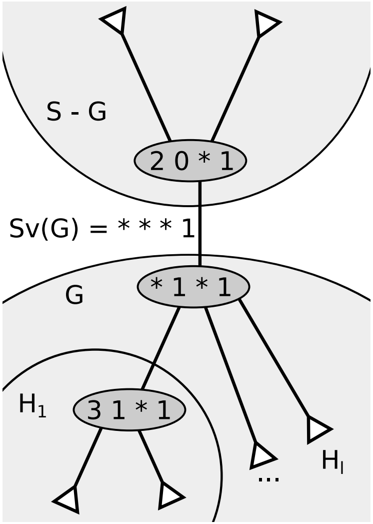


Recursive Formulation of [?] and [?]

2. If $G - H_1$ is **not** a proper cluster:

SUBPHYLOGENY(G)

```
    ...
3 foreach subset $H_1$ of $G$ where
      $T_{H_1} \leftarrow$ SUBPHYLOGENY$(H_1)$ exists
      and can be attached to $r$
    ...
7   elsif $G$ can be partitioned into $l > 2$
          proper clusters $H_1, \ldots, H_l$
          with subphylogenies $T_{H_1}, \ldots, T_{H_l}$
          that can be attached to $r$
8     return the tree created by attaching
              $T_{H_1}, \ldots, T_{H_l}$ to $r$
```



Whole Algorithm Pipeline of [?]

1. Compute all proper clusters $G \subset S$ and their splitting vectors.
   - $O(2^k m)$ possible proper clusters G, $O(nm)$ to verify and compute $Sv(G)$, thus $O(2^k m^2 n)$ total time.
2. Build proper cluster dictionary data structure.
3. Compute $S/Sv(G)$ for each proper cluster G.
4. Run $\textsc{PerfectPhylogeny}(S)$ and output answer.

Whole Algorithm Pipeline of [?]

1. Compute all proper clusters $G \subset S$ and their splitting vectors.
   - $O(2^k m)$ possible proper clusters G, $O(nm)$ to verify and compute $Sv(G)$, thus $O(2^k m^2 n)$ total time.
2. Build proper cluster dictionary data structure.
   - $O(n^2)$ per proper cluster to build a trie.
     Our improvement: $O(n)$ per proper cluster to build pointer table. $O(2^k mn)$ total time.
3. Compute $S/Sv(G)$ for each proper cluster G.
4. Run $\textsc{PerfectPhylogeny}(S)$ and output answer.

Whole Algorithm Pipeline of [?]

1. Compute all proper clusters $G \subset S$ and their splitting vectors.
   - $O(2^k m)$ possible proper clusters G, $O(nm)$ to verify and compute $Sv(G)$, thus $O(2^k m^2 n)$ total time.
2. Build proper cluster dictionary data structure.
   - $O(n^2)$ per proper cluster to build a trie.
     Our improvement: $O(n)$ per proper cluster to build pointer table.
     $O(2^k mn)$ total time.
3. Compute $S/Sv(G)$ for each proper cluster G.
   - $O(2^k m)$ possible proper clusters G, $O(nm)$ to compute $S/Sv(G)$, $O(2^k m^2 n)$ total time.
4. Run $\textsc{PerfectPhylogeny}(S)$ and output answer.

Whole Algorithm Pipeline of [?]

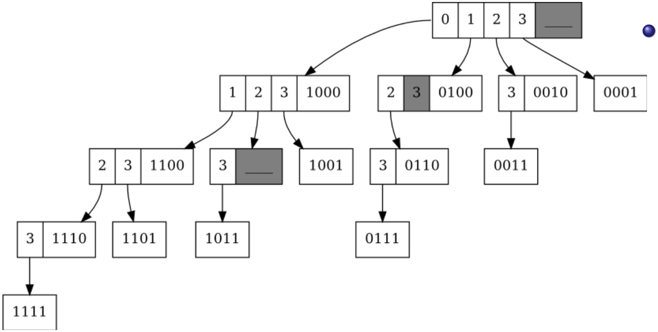
1. Compute all proper clusters $G \subset S$ and their splitting vectors.
   - $O(2^k m)$ possible proper clusters G, $O(nm)$ to verify and compute $Sv(G)$, thus $O(2^k m^2 n)$ total time.
2. Build proper cluster dictionary data structure.
   - $O(n^2)$ per proper cluster to build a trie.
     Our improvement: $O(n)$ per proper cluster to build pointer table.
     $O(2^k mn)$ total time.
3. Compute $S/Sv(G)$ for each proper cluster G.
   - $O(2^k m)$ possible proper clusters G, $O(nm)$ to compute $S/Sv(G)$, $O(2^k m^2 n)$ total time.
4. Run $\textsc{PerfectPhylogeny}(S)$ and output answer.
   - Using dynamic programming, $O(2^k m)$ subphylogeny calls which iterate over $O(2^k m)$ subsets performing $O(n)$ work each, thus $O(2^{2k} m^2 n)$ total time.

Preprocessing: Proper Cluster Dictionary

- The proper cluster dictionary is used to test whether or not an arbitrary $G \subset S$ is a proper cluster, and if so to get an index p_G for use in other data structures, in time $O(|G|) = O(n)$.
  - We represent G as a bit-vector $\{0, 1\}^n$.
- More specifically, given a partition $H_1, ..., H_\ell$ of $G \subset S$, it must be able to verify and output $p_{H_1}, ..., p_{H_\ell}$ in time $O(|H_1| + ... + |H_\ell|) = O(|G|) = O(n)$.
  - We represent $H_1, .., H_\ell$ as a vector over $\{1, ..., \ell\}^n$.
- Proposal of [**?**]: build a trie

Proper Cluster Dictionary: Trie issues

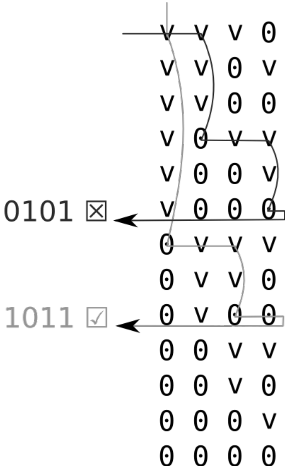
- The paths down a 0-1 binary trie is necessarily $O(n)$, thus looking up $H_1, \ldots, H_\ell$ simultaneously cannot be done with $O(\ell)$ independent lookups within $O(n)$ time.



- By expanding the nodes of the trie to support multiple children, the space requirement increases to $O(n^2)$ per proper cluster.

The Pointer Table, a Smaller Proper Cluster Dictionary

$Q[p][t]$ = the smallest $p' \geq p$ where $t \in G_{p'}$ (lexicographical order)



LOOKUP(Q, G)

```
1 $p \leftarrow 0$
2 foreach taxa $t \in G$ in order
3     $p \leftarrow Q[p][t]$
4 if $|G| = |G_p|$ and $\forall t \in G.\ p = Q[p][t]$
5     return $p$
6 else
7     return $NULL$
```

Representing $H_1, .., H_\ell$ as a vector over $\{1, ..., \ell\}^n$ allows simultaneous LOOKUP in $O(n)$ time.

Trie Slowdown vs Pointer Table

Average full program runtime and dictionary size increase when using the Trie **instead** of the Pointer Table, over 80 trials.

n,m	k = 4	k = 10	k = 20
50,50	3.07% / 165%	2.75% / 189%	1.55% / 174%
100,100	2.80% / 386%	2.95% / 516%	1.60% / 490%
500,500	1.27% / 1886%	2.67% / 2749%	1.18% / 2957%
1000,1000	1.15% / 3775%	2.89% / 5522%	1.12% / 6525%

Construction Algorithm Runtime

Average execution times (using pointer table) over 30 trials:

n, m	4 state (nucleotide)	10 state	20 state (amino acid)	Scaling (n, m)
10,10	0.001s	0.001s	0.003s	
50,50	0.005s	0.024s	0.303s	$\times 125$
100,100	0.028s	0.113s	1.55s	$\times 8$
500,500	3.21s	17.6s	239s	$\times 125$
1000,1000	51.9s	271s	2,320s	$\times 8$
2000,2000	529s	2,590s	19,300s	$\times 8$
Scaling (k)		$\times 2^{12}$	$\times 2^{20}$	

In practice, scales much better than asymptotic complexity predicts with respect to k, scales as predicted with respect to n and m
$O(2^{2k} m^2 n)$

Outline

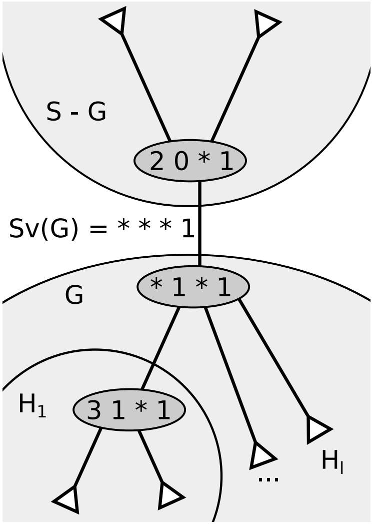
Enumeration of Minimal Perfect Phylogenies

Definition

A **minimal perfect phylogeny** is a perfect phylogeny T in which no edge can be **contracted** to make a smaller perfect phylogeny.

For each $(x, y) \in T$, there exists a character α such that:

$\alpha(x) \neq \alpha(y)$
$\alpha(x) \neq *$
$\alpha(y) \neq *$



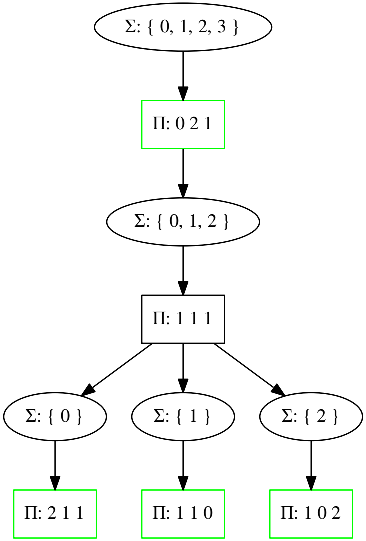
The DAG: Compact Representation of MPPs [?]

Definition

A **sum node** $\sum(H; y)$ represents a subphylogeny for proper cluster H with its root connected to a node y in $S - H$. The children of $\sum(H; y)$ are possible choices of product nodes.

Definition

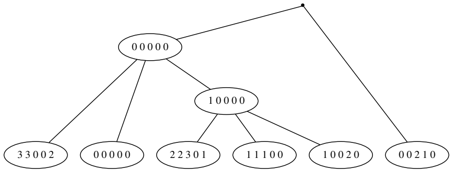
A **product node** $\prod(G_1, \ldots, G_t; x)$ represents a root x of a subphylogeny partitioned into subtrees that are sum nodes for $G_1, \ldots, G_t$.

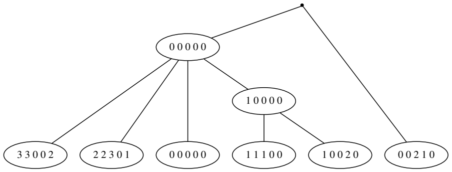


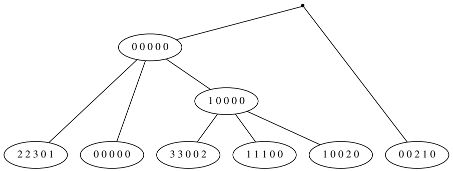
The DAG: Compact Representation of MPPs [?]

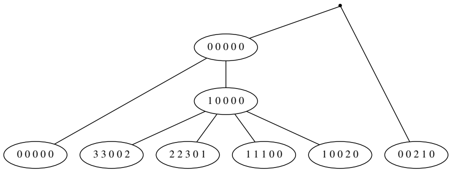
Example

6 Taxa, 5 Characters, 4 States
⇒ Found 4 Minimal Perfect Phylogenies





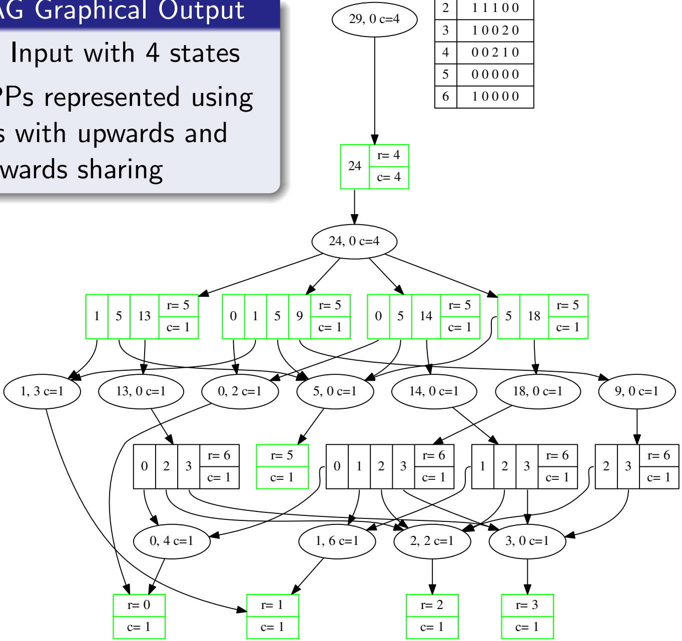




Actual DAG Graphical Output

- 6×5 Input with 4 states
- 4 MPPs represented using nodes with upwards and downwards sharing

r	Node Label
0	3 3 0 0 2
1	2 2 3 0 1
2	1 1 1 0 0
3	1 0 0 2 0
4	0 0 2 1 0
5	0 0 0 0 0
6	1 0 0 0 0



DAG Construction Optimizations

First step of enumeration algorithm computes sets $Ext(H, G)$.

Algorithm, adapted from [?]

... we consider all partitions of $H - G$ into at most $k - 1$ proper clusters $G_1, G_2, ..., G_t$, and consider the (possible) perfect phylogeny for H which has root x with subtrees perfect phylogenies for $G, G_1, G_2, ..., G_t$. The canonical labeling for x is then an element of $Ext(H, G)$.

Implementation choices:

1. Brute force checking, the naïve interpretation of [?]
2. Maximal Independent Set generating algorithms using $(G_1, G_2) \in E$ if $Sv(G_1)$ and $Sv(G_2)$ are incompatible.
3. MaxIS algorithms optimized for a known maximum size k.

DAG Analysis

- Number of MPPs t (bottom-up dynamic programming):
$$\textsc{count}(Sn) = \sum_{Pn \in \textsc{children}(Sn)} \textsc{count}(Pn)$$
$$\textsc{count}(Pn) = \prod_{Sn \in \textsc{children}(Pn)} \textsc{count}(Sn)$$
- Find tree with fewest nodes:
$$\textsc{nodeCount}(Sn) = \min_{Pn \in \textsc{children}(Sn)} \textsc{nodeCount}(Pn)$$
$$\textsc{nodeCount}(Pn) = 1 + \sum_{Sn \in \textsc{children}(Pn)} \textsc{nodeCount}(Sn)$$
- Access i^{th} MPP in $O(n+p)$ time (p is $\#$ product nodes)
- Iterate over MPPs in $O(n)$ time per tree
- All-pairs Robinson-Foulds Distance: $O(nt^2)$ time, $O(nt)$ space
$RF(T_1, T_2) = \frac{|P(T_1) \Delta P(T_2)|}{2}$ where $P(T) = T$'s proper clusters

DAG Analysis Algorithm: Support

Definition

The **support** of a proper cluster H is the number of MPPs in which H is the leaf set on one side of an edge.

$$\textsc{support}(H) = \textsc{support}(S - H) = \sum_{y} \textsc{support}(\textstyle\sum(H; y))$$

$$\textsc{support}(Sn_G) = \sum_{\substack{Sn_G \in \textsc{children}(Pn) \\ Pn \in \textsc{children}(Sn_H)}} \textsc{support}(Sn_H) \times \frac{\textsc{count}(Pn)}{\textsc{count}(Sn_H)}$$

- Top-down dynamic program computes $\textsc{support}(Sn_G)$, values used to find tree with maximum proper cluster support.
- We observed that these trees were usually distinct objects, not just trees with the most edges.

DAG Construction Runtime

Average time to construct DAG using $Ext(H, G)$ algorithms (k-MaxIS / MaxIS / brute force) enumeration over 60 trials.

n, m	$k = 4$	$k = 10$
50,50	14ms / 12ms / 49ms	41ms / 48ms / 421s*
100,100	48ms / 46ms / 1.11s	160ms / 176ms / 261s
500,500	3.99s / 3.95s / 4.38s	13.3s / 13.4s / 44.7s*
1000,1000	30.8s / 31.4s / 33.6s	127s / 124s / 142s*

* Actual average execution time is higher because some trials timed out at 20min.

Number of Minimal Perfect Phylogenies

Average number minimal perfect phylogenies and runtime (using k-MaxIS) over 80 trials.

n,m	$k = 4$	$k = 10$	$k = 20$
50,50	3.40 (0.00998 s)	237 (0.0539s)	120,000 (3.68s)
100,100	2.48 (0.0473s)	495 (0.195s)	1,710,000 (3.30s)
500,500	1.66 (4.38s)	118 (15s)	292,000 (184s)
1000,1000	1.91 (33.5s)	11.0 (124s)	207,000 (1,000s)

DAG Size

Average DAG output filesize in kilobytes over 50 trials

n,m	$k = 4$	$k = 10$	$k = 20$
50,50	6.13	11.2	33.6
100,100	19.2	32.6	89.5
500,500	355	572	881
1000,1000	1,310	2,170	2,980

Outline

PerfectPhy Software Package

- No dependencies C++ command line application

Example

```
$ ./perfectphy -f mydataset -newick
1 The data DOES have a perfect phylogeny
('3 3 0 0 2','0 0 2 1 0',('1 1 1 0 0','1 0 0 2 0')'1
0 0 0 0','2 2 3 0 1','0 0 0 0 0')
```

- Includes source code for main program, helpful tools, and scripts to run the experiments.
- Available at `http://wwwcsif.cs.ucdavis.edu/~gusfield` and linked on my website `http://www.mit.edu/~mcoulomb`

PerfectPhy Software Package: Tools and Extensions

- Character Removal (Wrapper)
- Missing Data (Wrapper)
- Phylip [**?**] sequence format (de)conversion adapters
- Newick tree format to Graphviz Dot format [**?**] for visualizing phylogenies.
- Experimental extension to construction algorithm to efficiently check if multiple MPPs exist without enumeration.

Efficient Unique Minimal Perfect Phylogeny Testing

- If there is only one tree, then there is no need to run the expensive enumeration algorithm, just minimize the tree constructed by the dynamic program.
- Given one perfect phylogeny on S, it is NP-Hard to decide if another exists for S. [**?**]
- Ideas?

Efficient Unique Minimal Perfect Phylogeny Testing

- If there is only one tree, then there is no need to run the expensive enumeration algorithm, just minimize the tree constructed by the dynamic program.
- Given one perfect phylogeny on S, it is NP-Hard to decide if another exists for S. [**?**]
- We can leverage the computation of the dynamic program to try to output two trees instead of one.
- Our Result: $O(n + m)$ additional time per inner loop iteration, thus $O(2^{2k} m^2 (n + m))$ total time.

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

SUBPHYLOGENY(G)

```
1 initialize root $r$ labeled with $Sv(G)$
1̂ unique($G$) $\leftarrow$ true, $T_G \leftarrow$ null
2 if $G$ is a single taxon $t$ then return the taxon $t$ attached to $r$
3 foreach subset $H_1$ of $G$ where
      $T_{H_1} \leftarrow$ SUBPHYLOGENY($H_1$) exists and can be attached to $r$
4   if $H_2 \leftarrow G - H_1$ is a proper cluster
5     if $T_{H_2} \leftarrow$ SUBPHYLOGENY($H_2$) exists and can be attached to $r$
6       $T'_G \leftarrow$ the tree created by attaching $T_{H_1}$ and $T_{H_2}$ to $r$
6̂       $T_G \leftarrow$ MINIMIZESUBTREES($T_G$, $T'_G$)
7   elsif $G$ can be partitioned into $l > 2$ proper clusters $H_1, \ldots, H_l$
        with subphylogenies $T_{H_1}, \ldots, T_{H_l}$ that can be attached to $r$
8     $T'_G \leftarrow$ the tree created by attaching $T_{H_1}, \ldots, T_{H_l}$ to $r$
8̂     $T_G \leftarrow$ MINIMIZESUBTREES($T_G, T'_G$)
9  return $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

```
MINIMIZESUBTREES($T_G$, $T'_G$)
1  **If** unique($G$)
2    **If** any $H_i$ subtree of the root has $\neg$ unique($H_i$) **then** unique($G$) $\leftarrow$ false
3    $cl_G$ = canonical labeling of root of $T'_G$
4    **If** the root of $T'_G$ has two subtrees for $H_1$,$H_2$
5      **If** COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
           but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
6        unique($G$) $\leftarrow$ false
7      Contract the subtrees of $T'_G$ arbitrarily until none can be
8    **else if** the root has over two subtrees
9      Contract the $H_1$ subtree of $T'_G$ if possible
10   **If** $T_G$ doesn't exist yet **then** $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
11   **else if** SetEqChecker decides $T_G \neq T'_G$ **then** unique($G$) $\leftarrow$ false
12 **return** $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

```
MINIMIZESUBTREES($T_G$, $T'_G$)
1  If unique($G$)
2    If any $H_i$ subtree of the root has $\neg$ unique($H_i$) then unique($G$) $\leftarrow$ false
3    $cl_G$ = canonical labeling of root of $T'_G$
4    If the root of $T'_G$ has two subtrees for $H_1$,$H_2$
5      If COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
           but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
6        unique($G$) $\leftarrow$ false
7      Contract the subtrees of $T'_G$ arbitrarily until none can be
8    else if the root has over two subtrees
9      Contract the $H_1$ subtree of $T'_G$ if possible
10   If $T_G$ doesn't exist yet then $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
11   else if SetEqChecker decides $T_G \neq T'_G$ then unique($G$) $\leftarrow$ false
12 return $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

MINIMIZESUBTREES(T_G, T'_G)

```
1 If unique($G$)
2   If any $H_i$ subtree of the root has $\neg$ unique($H_i$) then unique($G$) $\leftarrow$ false
3   $cl_G$ = canonical labeling of root of $T'_G$
4   If the root of $T'_G$ has two subtrees for $H_1$,$H_2$
5     If COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
          but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
6       unique($G$) $\leftarrow$ false
7     Contract the subtrees of $T'_G$ arbitrarily until none can be
8   else if the root has over two subtrees
9     Contract the $H_1$ subtree of $T'_G$ if possible
10  If $T_G$ doesn't exist yet then $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
11  else if SetEqChecker decides $T_G \neq T'_G$ then unique($G$) $\leftarrow$ false
12 return $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

```
MINIMIZESUBTREES($T_G$, $T'_G$)
1  If unique($G$)
2    If any $H_i$ subtree of the root has $\neg$ unique($H_i$) then unique($G$) $\leftarrow$ false
3    $cl_G$ = canonical labeling of root of $T'_G$
4    If the root of $T'_G$ has two subtrees for $H_1$,$H_2$
5      If COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
           but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
6        unique($G$) $\leftarrow$ false
7      Contract the subtrees of $T'_G$ arbitrarily until none can be
8    else if the root has over two subtrees
9      Contract the $H_1$ subtree of $T'_G$ if possible
10   If $T_G$ doesn't exist yet then $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
11   else if SetEqChecker decides $T_G \neq T'_G$ then unique($G$) $\leftarrow$ false
12 return $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

```
MINIMIZESUBTREES($T_G$, $T'_G$)
1  If unique($G$)
2    If any $H_i$ subtree of the root has $\neg$ unique($H_i$) then unique($G$) $\leftarrow$ false
3    $cl_G$ = canonical labeling of root of $T'_G$
4    If the root of $T'_G$ has two subtrees for $H_1$,$H_2$
5      If COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
          but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
6        unique($G$) $\leftarrow$ false
7      Contract the subtrees of $T'_G$ arbitrarily until none can be
8    else if the root has over two subtrees
9      Contract the $H_1$ subtree of $T'_G$ if possible
10   If $T_G$ doesn't exist yet then $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
11   else if SetEqChecker decides $T_G \neq T'_G$ then unique($G$) $\leftarrow$ false
12 return $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

```
MINIMIZESUBTREES($T_G$, $T'_G$)
If unique($G$)
  If any $H_i$ subtree of the root has $\neg$ unique($H_i$) then unique($G$) $\leftarrow$ false
  $cl_G$ = canonical labeling of root of $T'_G$
  If the root of $T'_G$ has two subtrees for $H_1$,$H_2$
    If COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
        but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
      unique($G$) $\leftarrow$ false
    Contract the subtrees of $T'_G$ arbitrarily until none can be
  else if the root has over two subtrees
    Contract the $H_1$ subtree of $T'_G$ if possible
  If $T_G$ doesn't exist yet then $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
  else if SetEqChecker decides $T_G \neq T'_G$ then unique($G$) $\leftarrow$ false
return $T_G$
```

Efficient Unique Minimal Perfect Phylogeny Testing

Global tables unique(G), rootlabels(G), and a SetEqChecker

```
MINIMIZESUBTREES($T_G$, $T'_G$)
1  If unique($G$)
2    If any $H_i$ subtree of the root has $\neg$ unique($H_i$) then unique($G$) $\leftarrow$ false
3    $cl_G$ = canonical labeling of root of $T'_G$
4    If the root of $T'_G$ has two subtrees for $H_1$,$H_2$
5      If COMPATIBLE($cl_G$, rootlabels($H_1$)) and COMPATIBLE($cl_G$, rootlabels($H_2$))
          but $\neg$ COMPATIBLE(rootlabels($H_1$), rootlabels($H_2$))
6        unique($G$) $\leftarrow$ false
7      Contract the subtrees of $T'_G$ arbitrarily until none can be
8    else if the root has over two subtrees
9      Contract the $H_1$ subtree of $T'_G$ if possible
10   If $T_G$ doesn't exist yet then $T_G \leftarrow T'_G$ and rootlabels($G$) $\leftarrow cl_G$
11   else if SetEqChecker decides $T_G \neq T'_G$ then unique($G$) $\leftarrow$ false
12 return $T_G$
```

Thanks!

Questions?