Dynamic Programming for Phylogenetic Estimation

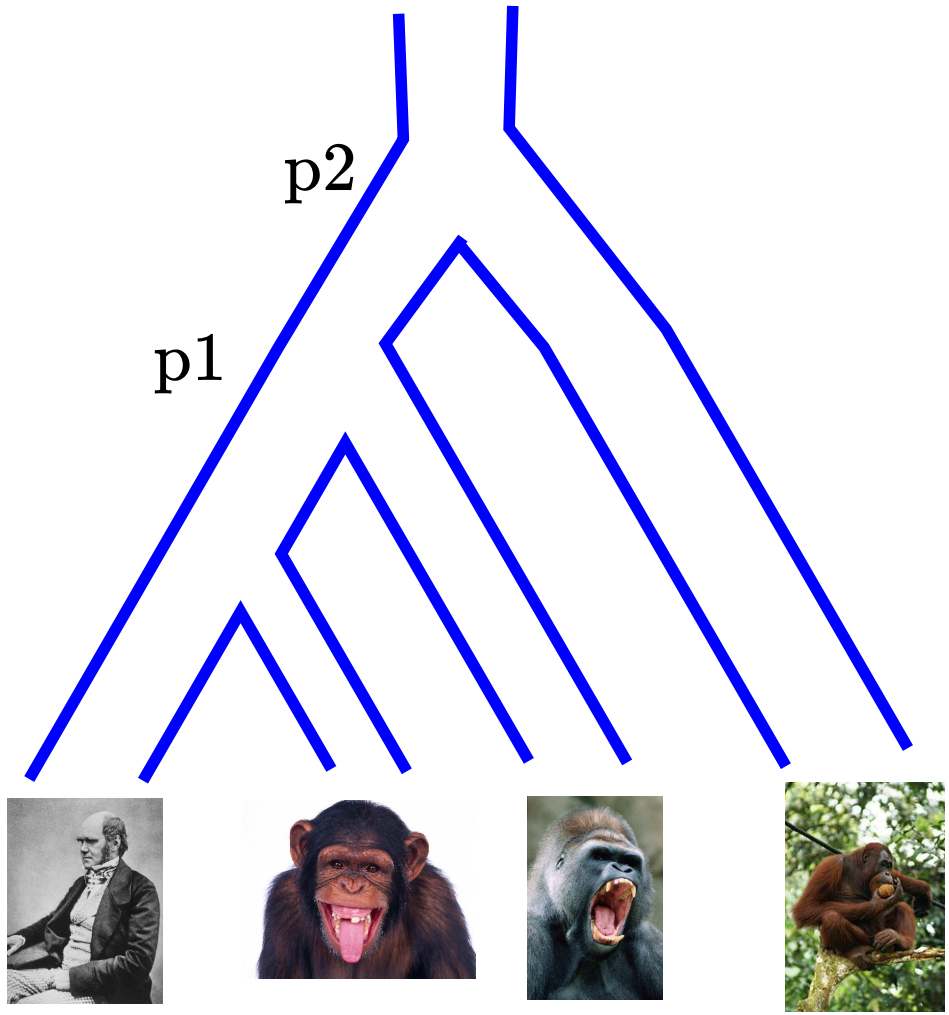
CS598AGB

Pranjal Vachaspati

University of Illinois at Urbana-Champaign

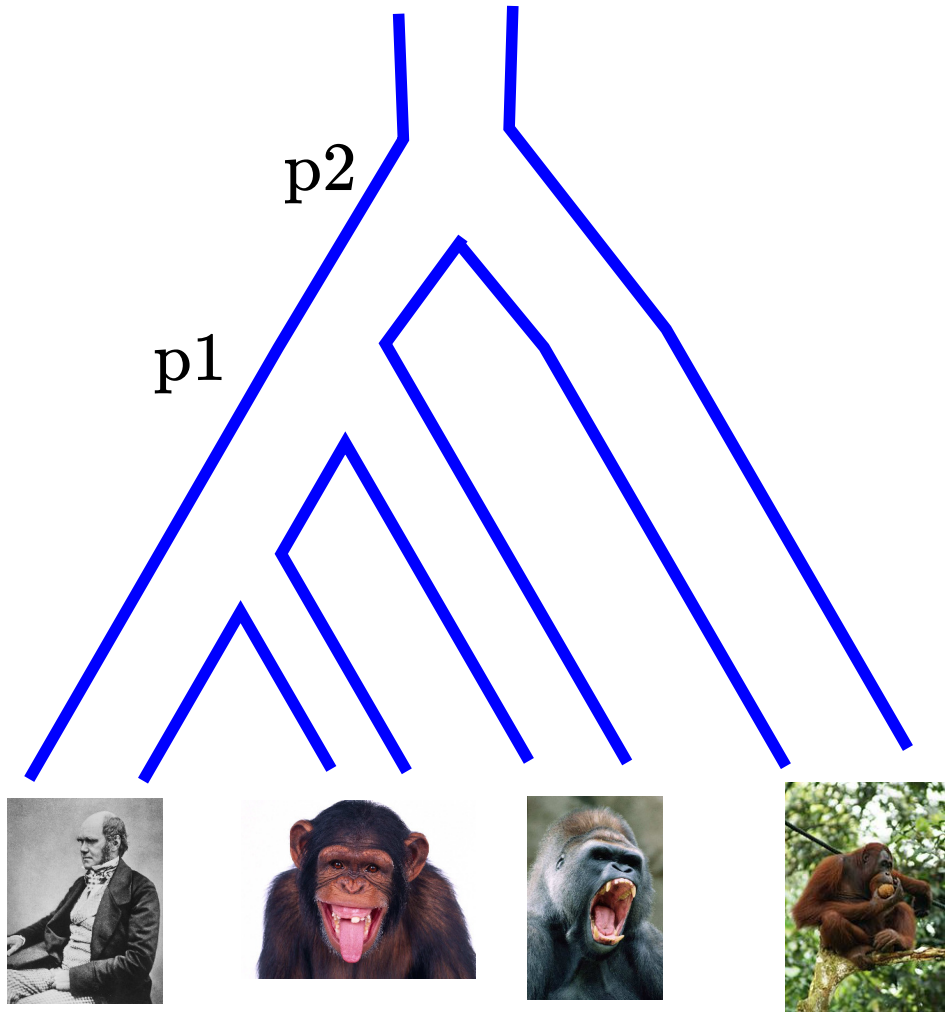
Coalescent-based Species Tree Estimation

Find evolutionary tree for species



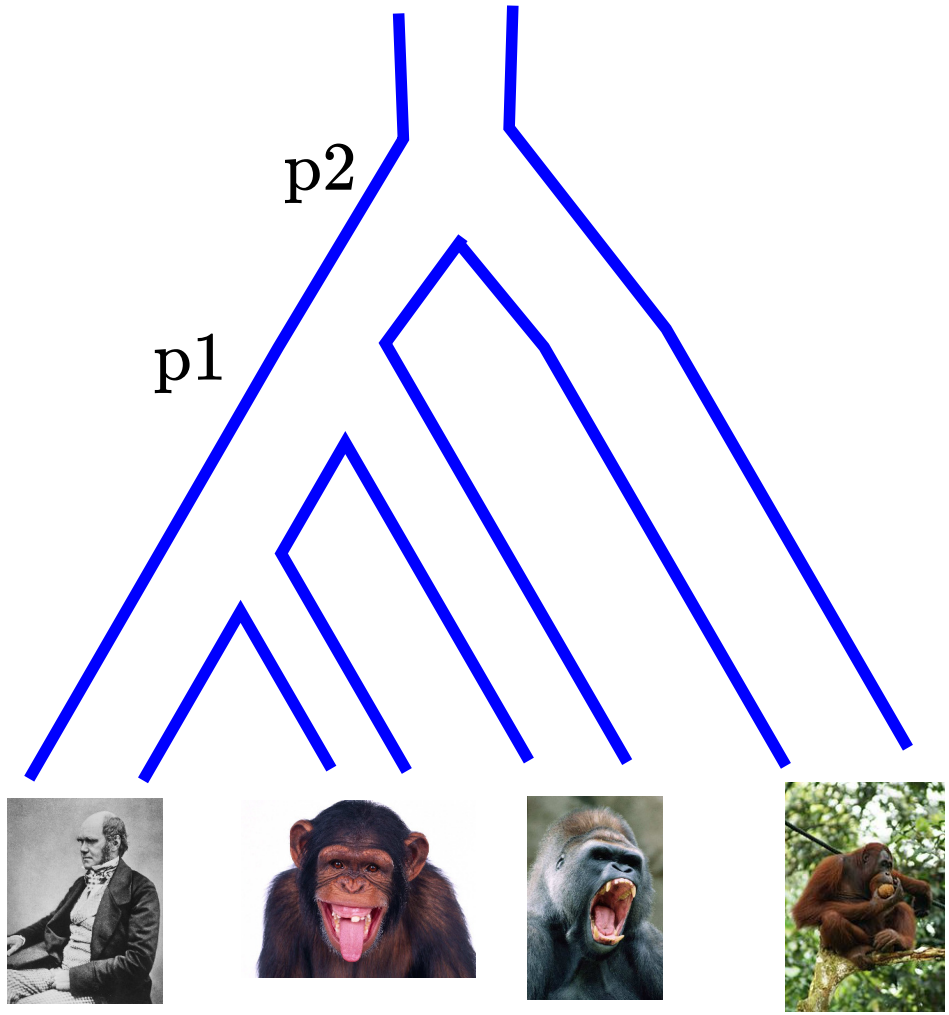
Coalescent-based Species Tree Estimation

Find evolutionary tree for species, which may differ from evolutionary trees of individual genes due to the multi-species coalescent



Coalescent-based Species Tree Estimation

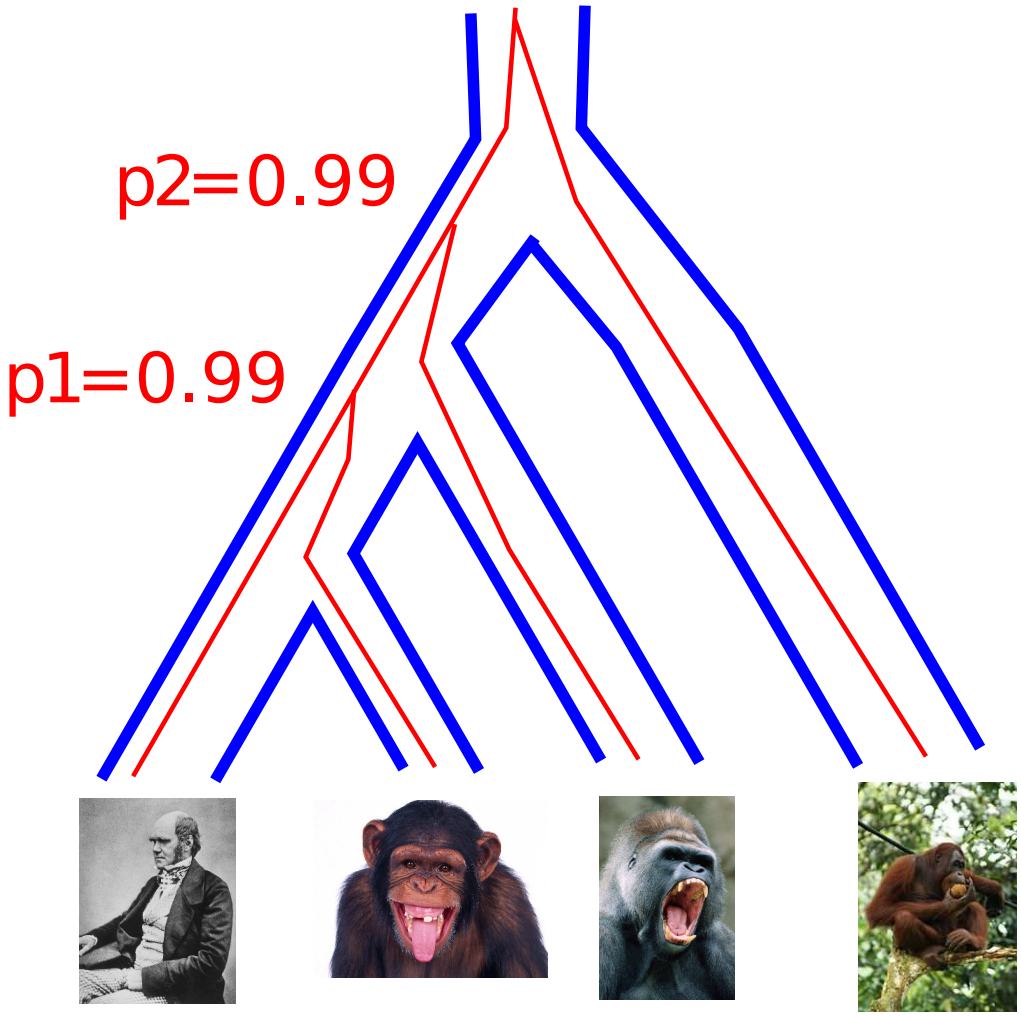
Find evolutionary tree for species, which may differ from evolutionary trees of individual genes due to the multi-species coalescent



Two separate lineages coalesce in a branch with probability p

Coalescent-based Species Tree Estimation

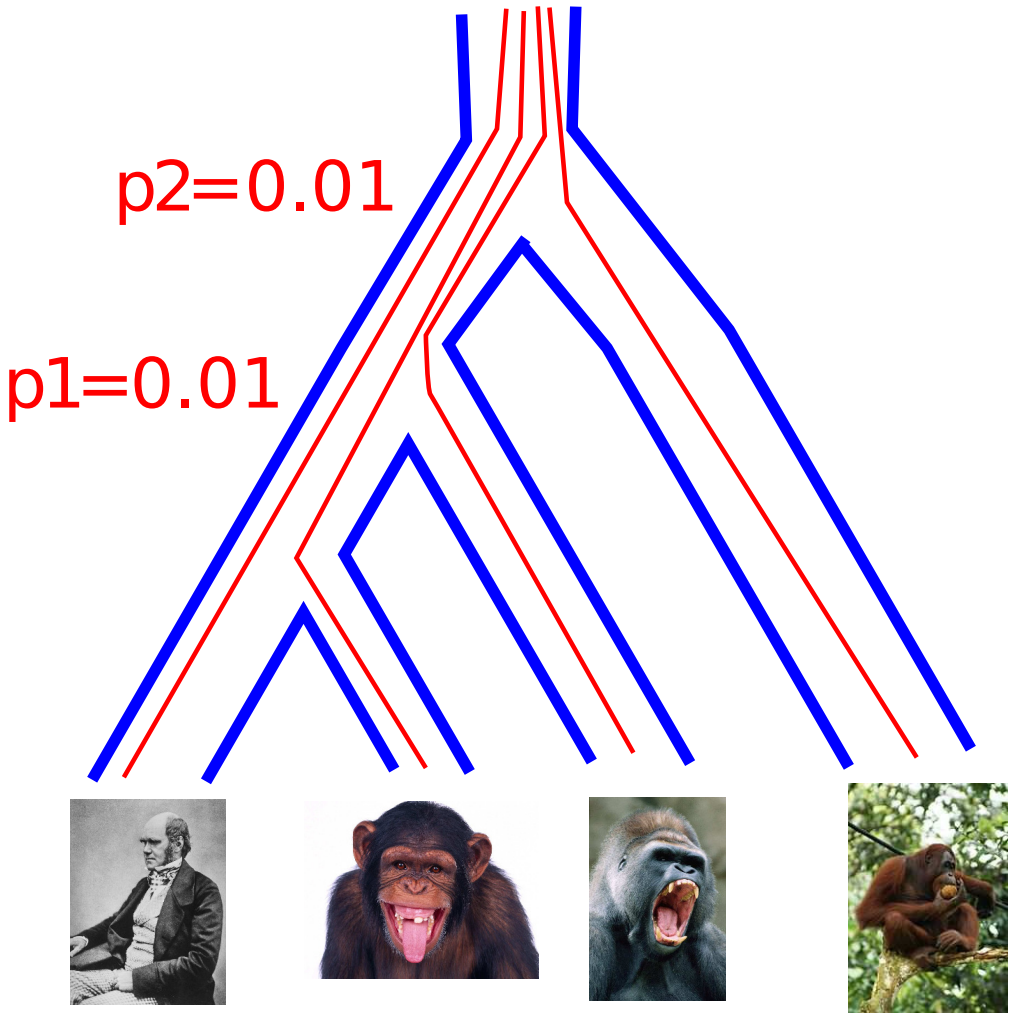
Find evolutionary tree for species, which may differ from evolutionary trees of individual genes due to the multi-species coalescent



What if evolution is *very* slow (or populations *very* small)?

Coalescent-based Species Tree Estimation

Find evolutionary tree for species, which may differ from evolutionary trees of individual genes due to the multi-species coalescent



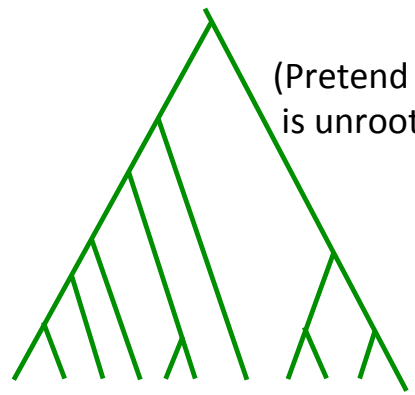
What if evolution is *very* quick (or populations *very* large)?

How to compute an unrooted species tree from unrooted gene trees?

(Pretend these are unrooted trees)

Estimate species tree for every 4 species

Combine unrooted 4-taxon trees



(Pretend this is unrooted!)

Theorem (Allman et al., 2011, and others): For every four leaves {a,b,c,d}, the most probable unrooted quartet tree on {a,b,c,d} is the true species tree.

Use the All Quartets Method to construct the species tree, based on the most frequent gene trees for each set of four species.

How do we go from gene trees to quartets?

1. Take most common quartet - fine if lots of accurate trees

How do we go from gene trees to quartets?

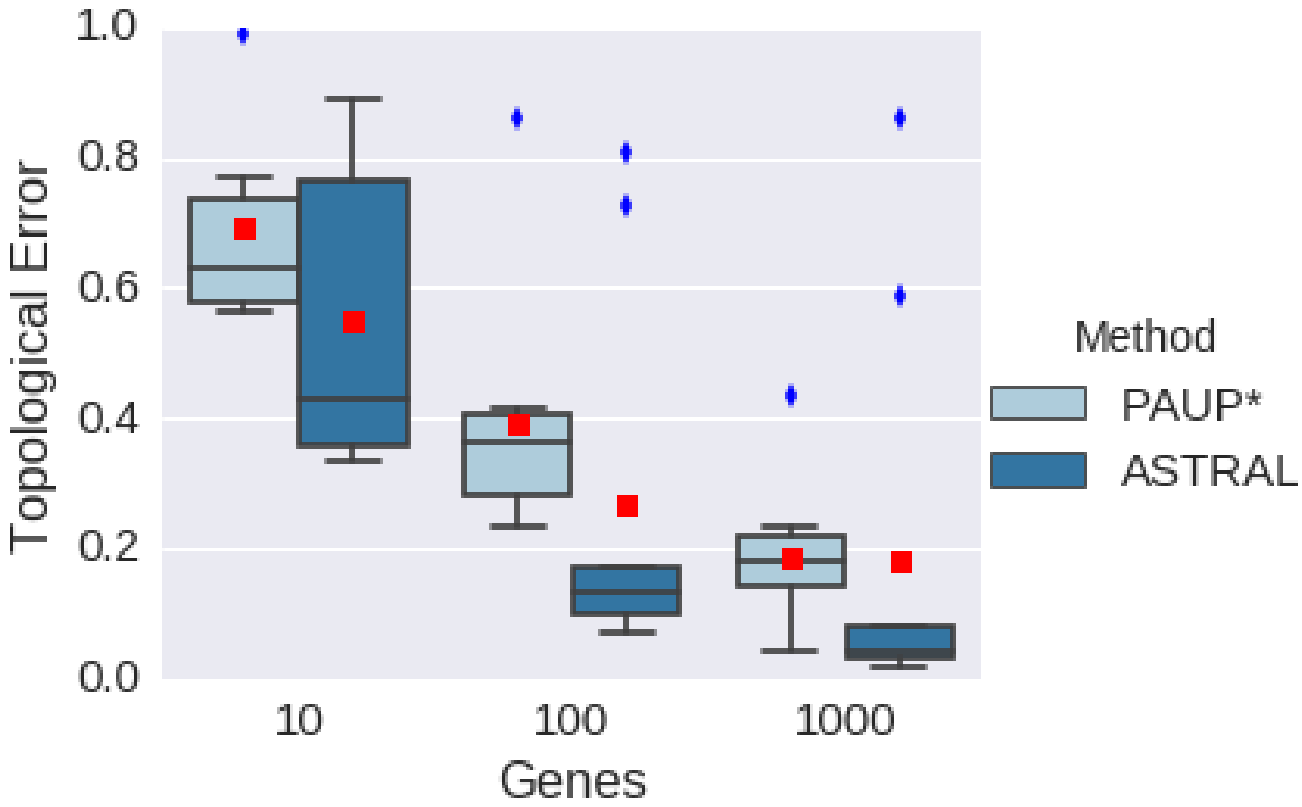
1. Take most common quartet - fine if lots of accurate trees
2. Weight quartets by number of times they appear in gene trees

How do we go from gene trees to quartets?

1. Take most common quartet - fine if lots of accurate trees
2. Weight quartets by number of times they appear in gene trees
3. SVDQuartets (Chifman and Kubatko, 2014): estimate best quartets from sequence data, avoiding effects of gene tree estimation error!

Unfortunately, SVDQuartets is frequently less accurate than ASTRAL-2

Figure: 100 taxa, low and medium levels of ILS



PAUP* is a software package containing a reference implementation of SVDQuartets

All Quartets Method doesn't really work

- Really slow
- Fails if quartets are not all compatible
- Doesn't work for anything other than picking the most common quartet

All Quartets Method doesn't really work

- Really slow
- Fails if quartets are not all compatible
- Doesn't work for anything other than picking the most common quartet

Finding tree that maximizes support over quartets is NP-hard in general

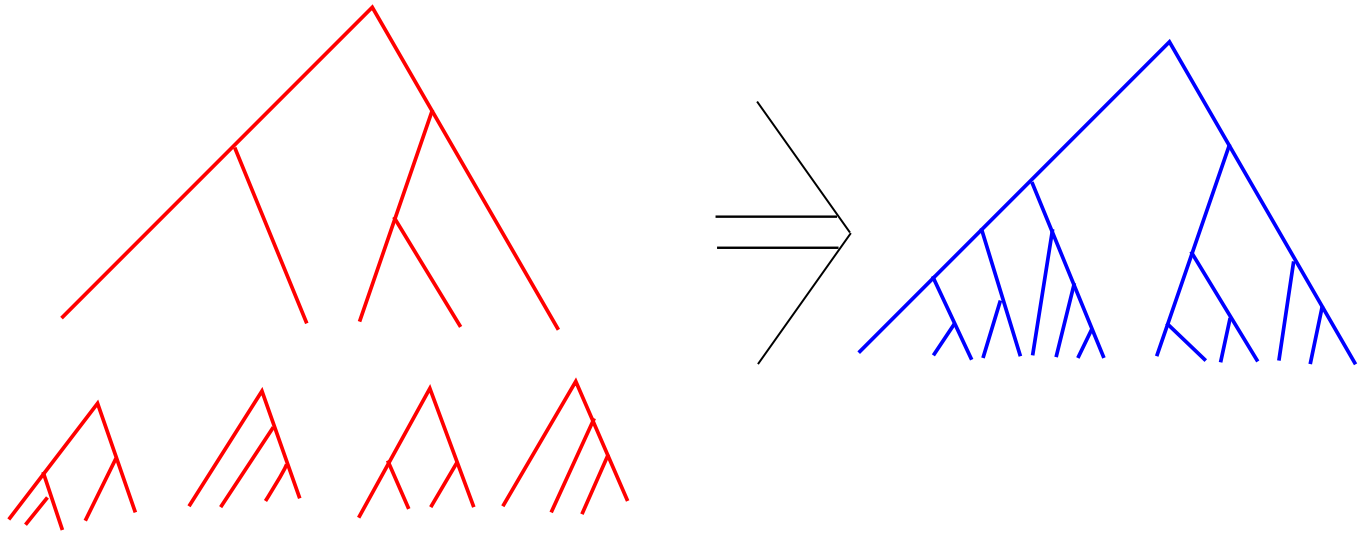
All Quartets Method doesn't really work

- Really slow
- Fails if quartets are not all compatible
- Doesn't work for anything other than picking the most common quartet

Finding tree that maximizes support over quartets is NP-hard in general
So we need some tricks.

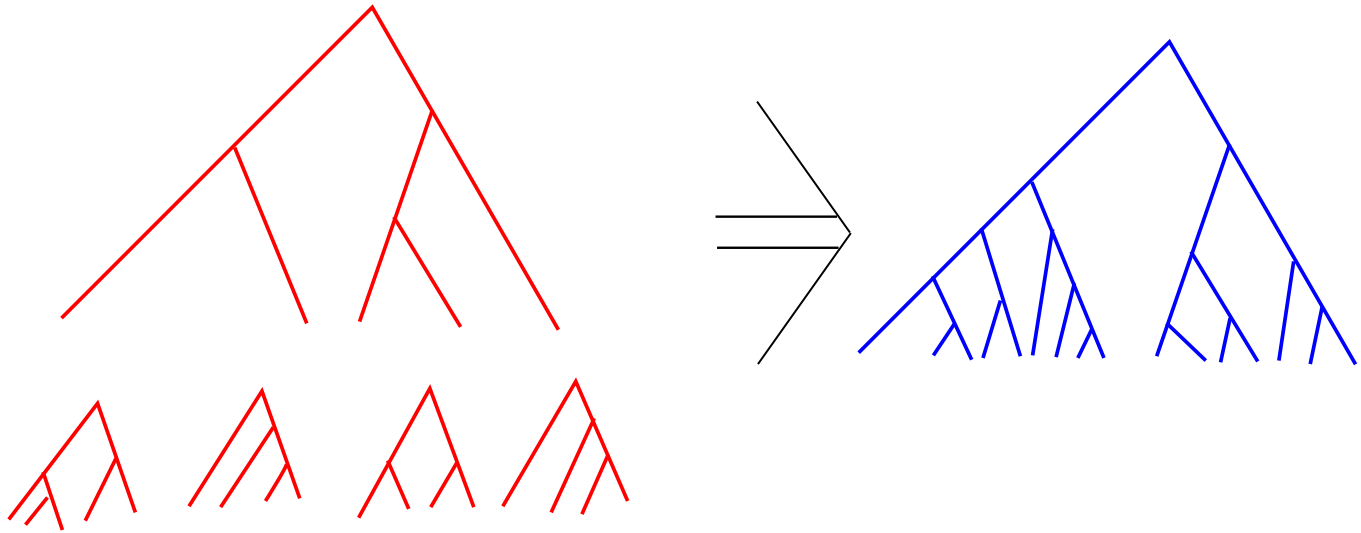
Supertree Estimation - another important phylogenetic problem

- Combine sparse scaffold + dense trees on subsets



Supertree Estimation - another important phylogenetic problem

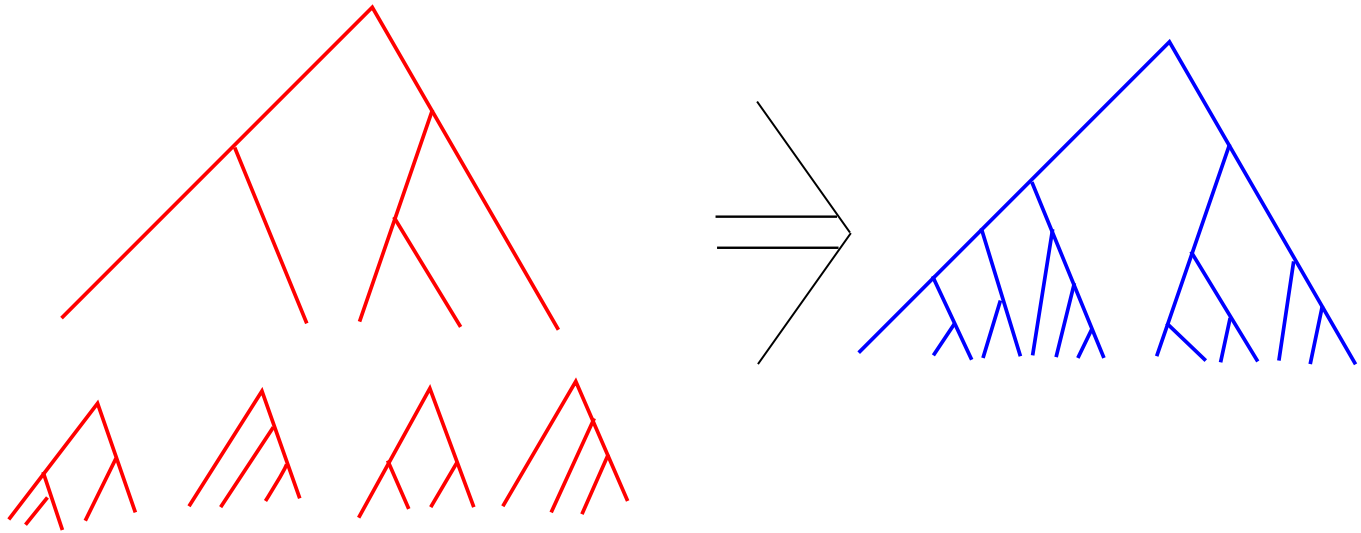
- Combine sparse scaffold + dense trees on subsets



- Like coalescent-based species tree estimation, many methods - some more accurate than others

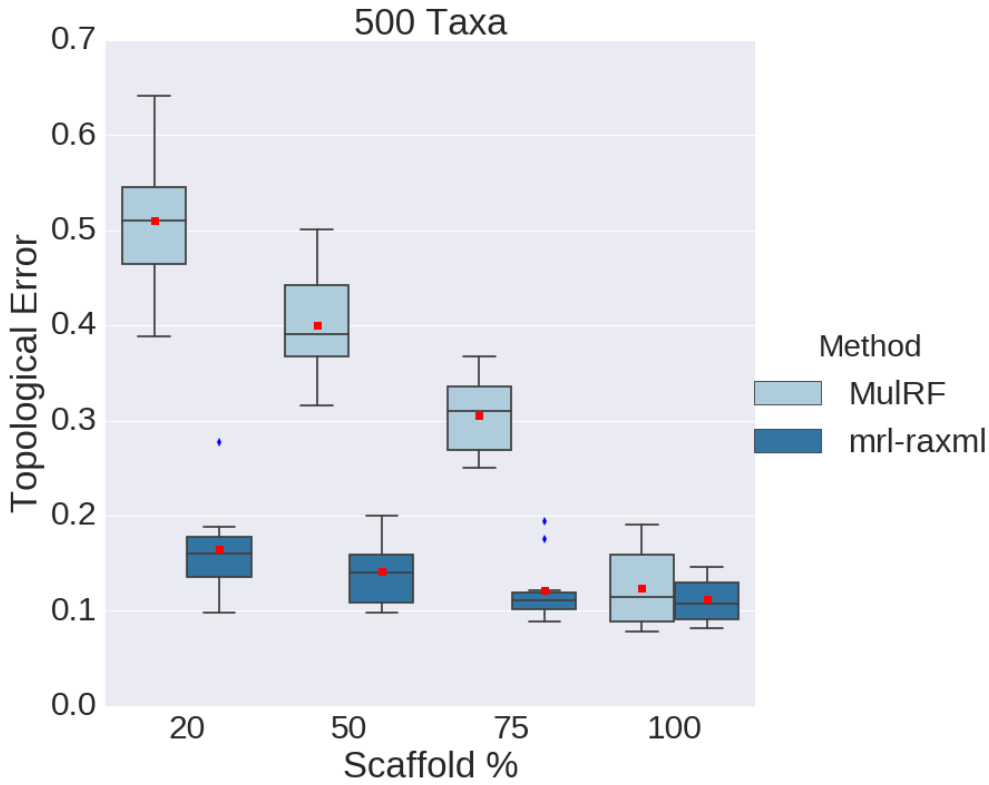
Supertree Estimation - another important phylogenetic problem

- Combine sparse scaffold + dense trees on subsets



- Like coalescent-based species tree estimation, many methods - some more accurate than others
- MRL: most accurate method in literature, finds maximum likelihood tree under a symmetric two-state model, using the "MRP" character matrix
- MulRF: less accurate method that finds a tree that minimizes the RF distance to the input trees

MulRF is frequently less accurate than MRL



What makes a method accurate?

- Optimization Criterion: we want a tree that is as close to reality as possible, but since we don't know the true tree, we have to optimize some other function
- Optimization algorithm: Optimizing these criteria is usually NP-hard, so we can't solve them exactly

What makes a method accurate?

- Optimization Criterion: we want a tree that is as close to reality as possible, but since we don't know the true tree, we have to optimize some other function
- Optimization algorithm: Optimizing these criteria is usually NP-hard, so we can't solve them exactly

We can't tell the difference between a poor criterion and a good criterion with a bad optimization algorithm!

What makes a method accurate?

- Optimization Criterion: we want a tree that is as close to reality as possible, but since we don't know the true tree, we have to optimize some other function
- Optimization algorithm: Optimizing these criteria is usually NP-hard, so we can't solve them exactly

We can't tell the difference between a poor criterion and a good criterion with a bad optimization algorithm!

We have developed a general-purpose optimization algorithm that does an exact constrained search for a variety of optimization problems

Advantages of our approach

1. Often finds better solutions than existing heuristic algorithms
2. Allows for an unbiased comparison between optimization criteria
3. Can be easily adapted to new optimization criteria

Exact Constrained Search with Dynamic Programming

- Theory introduced by Bryant and Steel (2001) for weighted quartet support
  - Input: quartet weights, constraint set X
  - Output: species tree constrained by X maximizing induced quartet weight

Exact Constrained Search with Dynamic Programming

- Theory introduced by Bryant and Steel (2001) for weighted quartet support
- Used for minimizing deep coalescences (Than and Nakleh, 2009)
  - Input: Cluster compatibility graph from gene trees, constraint set X
  - Output: species tree constrained by X minimizing deep coalescences over gene trees

Exact Constrained Search with Dynamic Programming

- Theory introduced by Bryant and Steel (2001) for weighted quartet support
- Used for minimizing deep coalescences (Than and Nakleh, 2009)
- Used by ASTRAL (Mirarab et al. 2014) and ASTRAL-II (Mirarab and Warnow, 2015) for quartet support
  - Input: gene trees, constraint set X
  - Output: species tree constrained by X maximizing support over gene trees
  - Currently most accurate coalescent-based species tree estimation method

Exact Constrained Search with Dynamic Programming

- Theory introduced by Bryant and Steel (2001) for weighted quartet support
- Used for minimizing deep coalescences (Than and Nakleh, 2009)
- Used by ASTRAL (Mirarab et al. 2014) and ASTRAL-II (Mirarab and Warnow, 2015) for quartet support

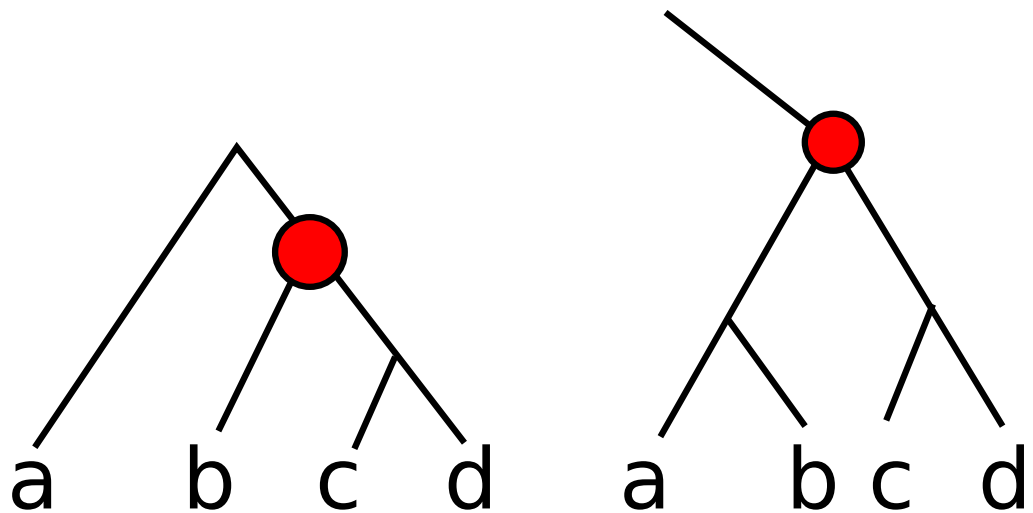
Our algorithm is a generic version of the above techniques that can solve a wide variety of problems

Weighted Quartet Support

- Input: A mapping of quartets (four-leaf trees) to real-valued weights
- Output: A tree with optimum weight (the sum of the weights of its quartets)

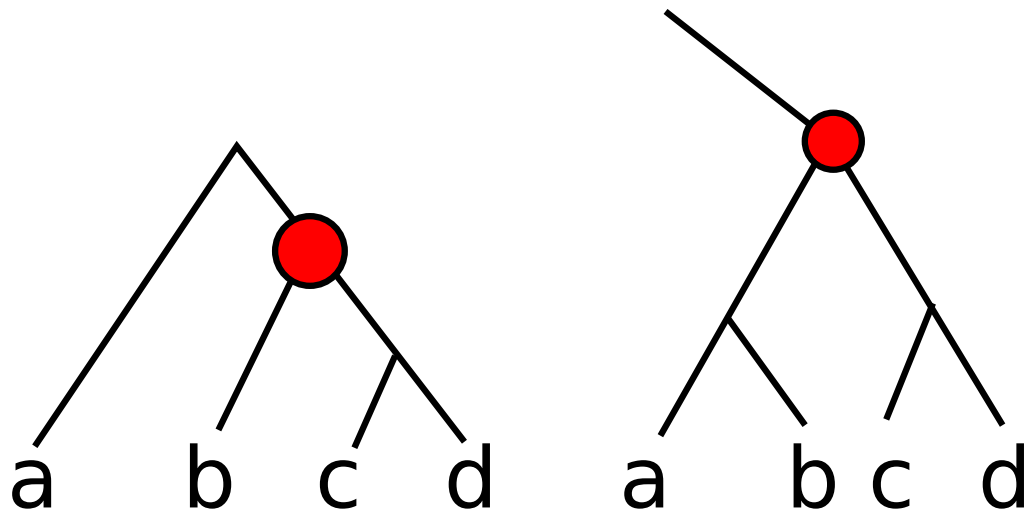
Weighted Quartet Support

- Input: A mapping of quartets (four-leaf trees) to real-valued weights
- Output: A tree with optimum weight (the sum of the weights of its quartets)
- Observation: In a rooted tree that induces quartet $((a, b), (c, d))$, there is a unique LCA for at least three of a, b, c, d



Weighted Quartet Support

- Input: A mapping of quartets (four-leaf trees) to real-valued weights
- Output: A tree with optimum weight (the sum of the weights of its quartets)
- Observation: In a rooted tree that induces quartet $((a, b), (c, d))$, there is a unique LCA for at least three of a, b, c, d



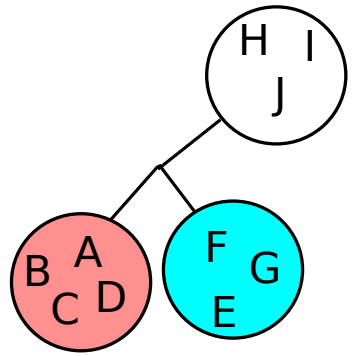
- Definition: Weight of node is sum of weights of corresponding quartets

Exact optimization scheme: Weighted Quartet Support

- Observation: The weight of the tree is the sum of the weights of each node!

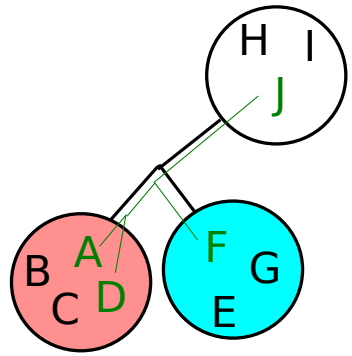
Exact optimization scheme: Weighted Quartet Support

- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



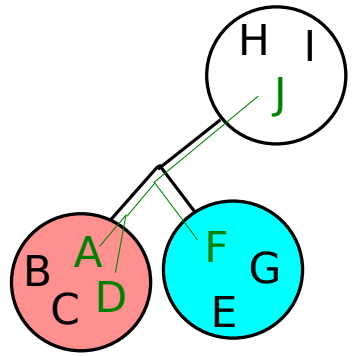
Exact optimization scheme: Weighted Quartet Support

- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



Exact optimization scheme: Weighted Quartet Support

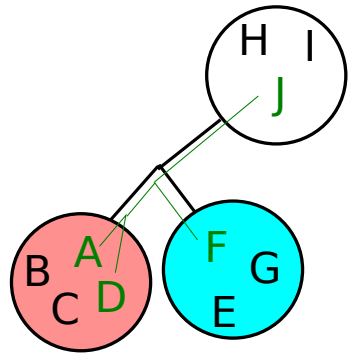
- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *EFG* split at the root is the sum of

Exact optimization scheme: Weighted Quartet Support

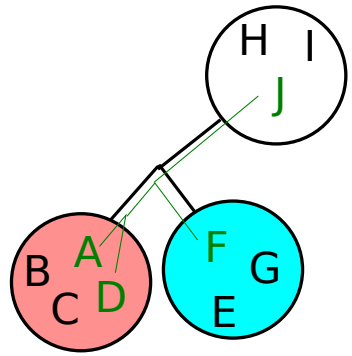
- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *EFG* split at the root is the sum of
  1. The weight of the tripartition *ABCD*, *EFG*, *HIJ*

Exact optimization scheme: Weighted Quartet Support

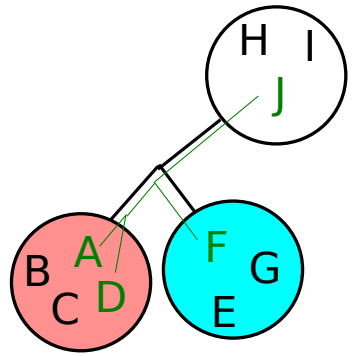
- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *EFG* split at the root is the sum of
  1. The weight of the tripartition *ABCD*, *EFG*, *HIJ*
  2. The weight of the best subtree on *ABCD*

Exact optimization scheme: Weighted Quartet Support

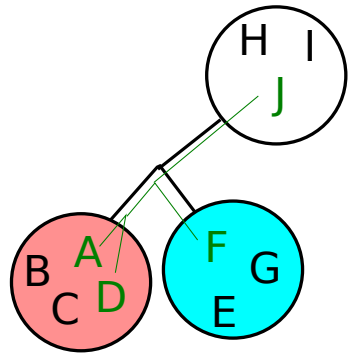
- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *EFG* split at the root is the sum of
  1. The weight of the tripartition *ABCD*, *EFG*, *HIJ*
  2. The weight of the best subtree on *ABCD*
  3. The weight of the best subtree on *EFG*

Exact optimization scheme: Weighted Quartet Support

- Observation: The weight of the tree is the sum of the weights of each node!
- Calculate the weight of a tripartition corresponding to a node without knowing the structure of the rest of the tree



- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *EFG* split at the root is the sum of
  1. The weight of the tripartition *ABCD*, *EFG*, *HIJ*
  2. The weight of the best subtree on *ABCD*
  3. The weight of the best subtree on *EFG*
- If we have the best subtrees on all subsets, we can try every combination to get the best subtree on *ABCDEFG*

Constrain the search space to avoid exponential time

- 2^n ways to divide a set of size n into two subsets

Constrain the search space to avoid exponential time

- 2^n ways to divide a set of size n into two subsets
- Instead of exploring entire search space, only allow trees to contain bipartitions from an input set X

Constrain the search space to avoid exponential time

- 2^n ways to divide a set of size n into two subsets
- Instead of exploring entire search space, only allow trees to contain bipartitions from an input set X
- In practice, we use ASTRAL-2's set X, which is based on the set of bipartitions in the input gene trees.
- Can run into problems when input trees missing taxa, poorly resolved

The Constrained DP Algorithm for quartet support

- Input: Constraint set X of bipartitions, list of quartet weights, over taxon set S
- Output: Score of species tree constrained by X maximizing weight of induced quartets

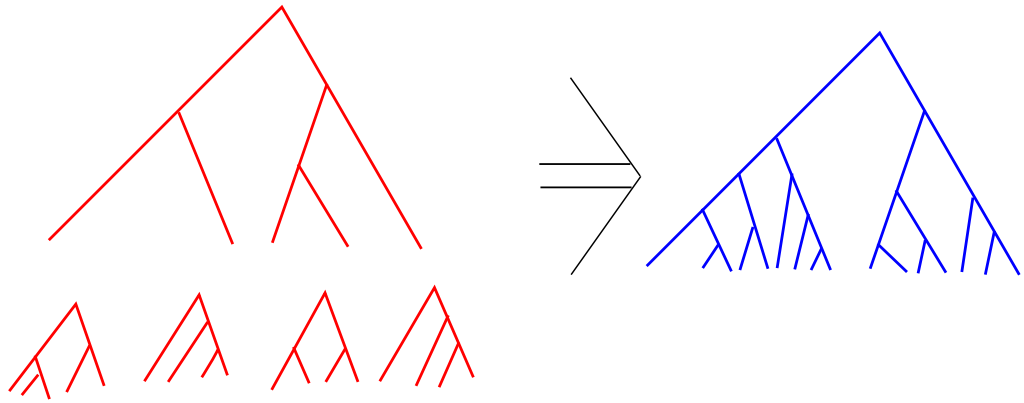
For $i \in \{1 \ldots n\}$ (number of taxa):

- For each cluster c (half of a bipartition) of size i in X:
  - Set $score(c) = 0$
  - For each cluster c' of size less than i:
    - If c contains c', and $c - c'$ is in X:
    - $score(c) = max(score(c), score(c') + score(c - c') + weight((c', c - c', S - c)))$

- $O(|X|^2)$ pairs of clusters
- $O(n^4)$ time to count the quartets matching a tripartition
- Running time: $O(|X|^2 n^4)$, can be reduced to $O(|X|^2 n^2 + n^4)$ with some precomputation (Bryant and Steel, 2001)

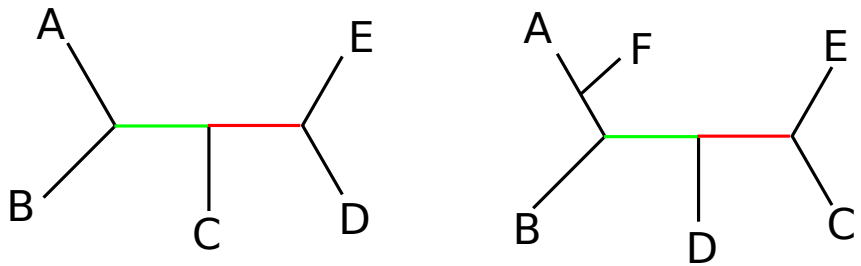
RF Supertrees

- Combine sparse scaffold + dense trees on subsets



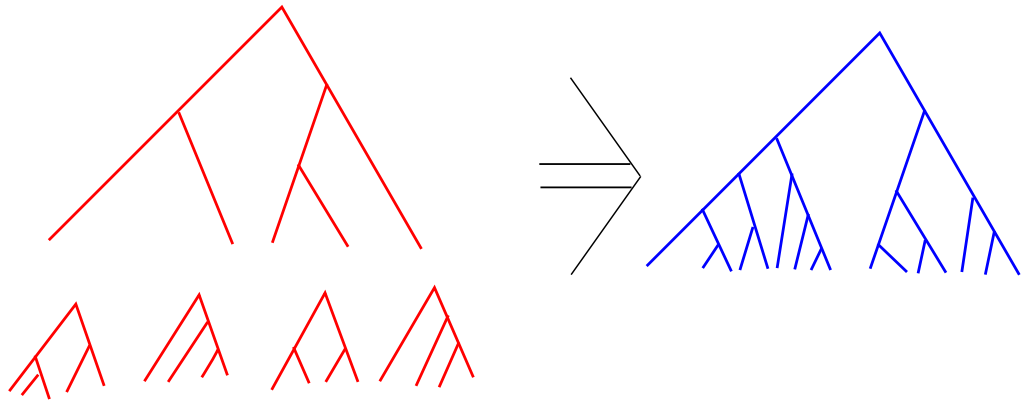
- The Robinson-Foulds (RF) distance between a binary supertree T and a binary source tree t on a taxon subset s is

$$RF(T,t) = |bipartitions(T|_s) \setminus bipartitions(t)|$$



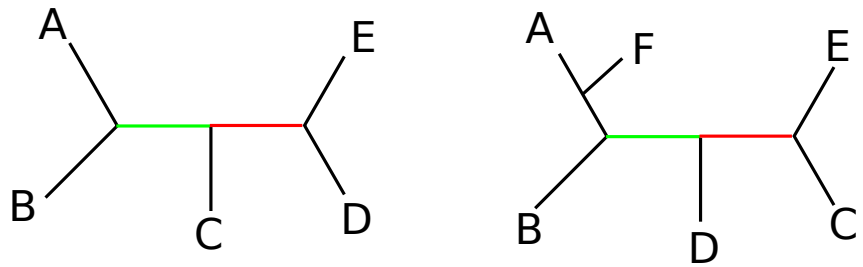
RF Supertrees

- Combine sparse scaffold + dense trees on subsets



- The Robinson-Foulds (RF) distance between a binary supertree T and a binary source tree t on a taxon subset s is

$$RF(T,t) = |bipartitions(T|_s) \setminus bipartitions(t)|$$



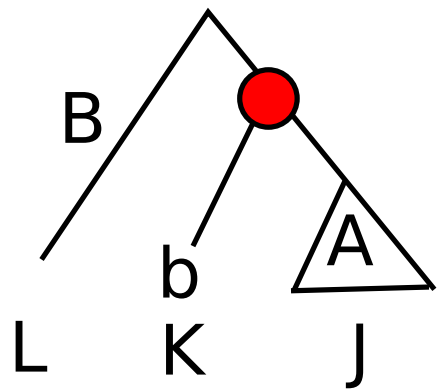
- The *RF supertree* is the tree that minimizes the total RF distance to a set of source trees

RF Supertrees

- Earlier, we mapped quartets to tripartitions. If we map bipartitions to tripartitions, we can use the same approach as our quartet support algorithm!

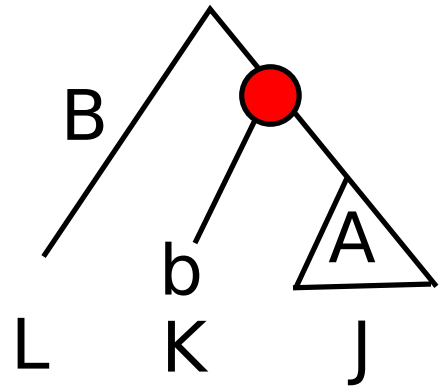
RF Supertrees

- Earlier, we mapped quartets to tripartitions. If we map bipartitions to tripartitions, we can use the same approach as our quartet support algorithm!
- We map a bipartition $\{A|B\}$ on a taxon subset to a tripartition if
  1. it has every member of A (and no members of B) below one child
  2. it has at least one member of B below the other child



RF Supertrees

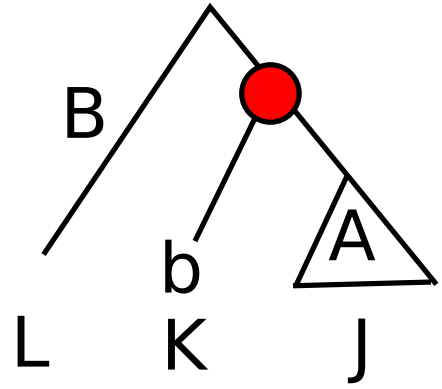
- Earlier, we mapped quartets to tripartitions. If we map bipartitions to tripartitions, we can use the same approach as our quartet support algorithm!
- We map a bipartition $\{A|B\}$ on a taxon subset to a tripartition if
  1. it has every member of A (and no members of B) below one child
  2. it has at least one member of B below the other child



- k input trees over n taxa: $O(kn)$ input bipartitions
- Checking if tripartition matches bipartition takes $O(n)$ time
- $O(|X|^2)$ pairs of clusters
- Running time:$O(|X|^2 n^2 k)$

RF Supertrees

- Earlier, we mapped quartets to tripartitions. If we map bipartitions to tripartitions, we can use the same approach as our quartet support algorithm!
- We map a bipartition $\{A|B\}$ on a taxon subset to a tripartition if
  1. it has every member of A (and no members of B) below one child
  2. it has at least one member of B below the other child



- k input trees over n taxa: $O(kn)$ input bipartitions
- Checking if tripartition matches bipartition takes $O(n)$ time
- $O(|X|^2)$ pairs of clusters
- Running time:$O(|X|^2 n^2 k)$
- A similar approach is possible for several other problems

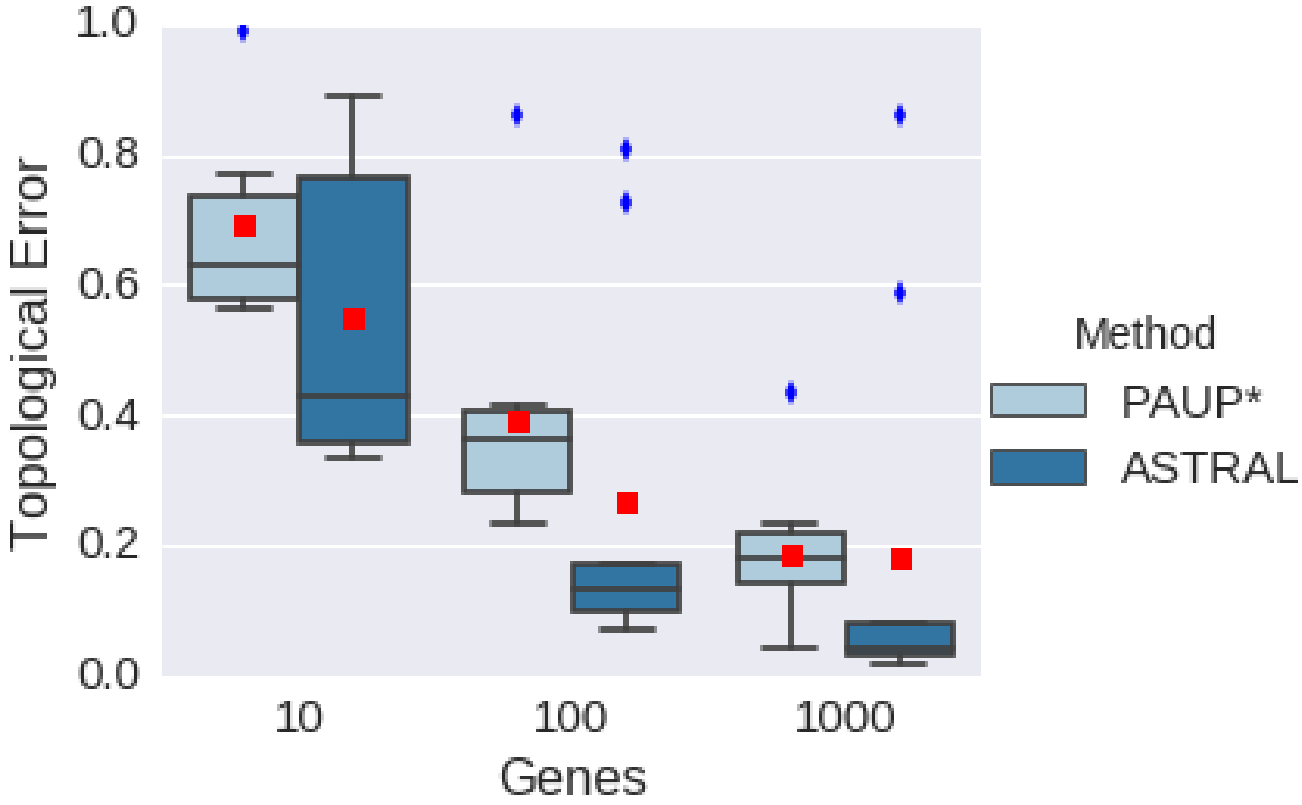
Experiment: SVDQuartets

We can use our weighted quartet support algorithm for the SVDQuartets problem
Questions:

1. Can the DP algorithm find better solutions to the SVDQuartets problem than the reference implementation in PAUP*?
2. Can the SVDQuartets criterion with the DP algorithm find topologically more accurate trees than ASTRAL?
3. What is the effect of expanding the constraint space?

Unfortunately, SVDQuartets is frequently less accurate than ASTRAL-2

Figure: 100 taxa, low and medium levels of ILS



PAUP* is a software package containing a reference implementation of SVDQuartets

Datasets: SVDQuartets

- ASTRAL-2 simulated data (based on data from Mirarab and Warnow, 2015)
  - Datasets with 10, 50, 100 taxa
  - 10, 100, 1000 genes
  - Sequence length random from 300-1500 bp
  - Also restricted to 25, 100 base pairs per gene
  - Three ILS model conditions for 100-taxon dataset: 21%, 33%, 69% average distance (AD) between true gene trees + species tree
  - 10- and 50-taxon datasets have 34% AD
  - 10 replicates per model condition

10-taxon data: all methods are capable of finding good SVDQuartets criterion scores

Genes	10			100		
Seq. Len.	25 bp	100 bp	Full	25 bp	100 bp	Full
DP-constr.	**0.300**	**0.294**	**0.124**	**0.224**	**0.127**	**0.049**
PAUP*	0.302	0.295	**0.124**	**0.224**	**0.127**	0.051
DP-exact	**0.300**	**0.294**	**0.124**	**0.224**	**0.127**	**0.049**

Genes	1000		
Seq. Len.	25 bp	100 bp	Full
DP-constr.	**0.087**	**0.049**	**0.020**
PAUP*	**0.087**	**0.049**	**0.020**
DP-exact	**0.087**	**0.049**	**0.020**

Smaller criterion scores are better

50-taxon data: DP consistently outperforms PAUP* in SVDQuartets criterion score

Genes	10			100		
Seq. Len.	25 bp	100 bp	Full	25 bp	100 bp	Full
DP	518.5	**426.0**	**210.3**	**318.8**	**174.6**	**86.0**
PAUP*	**517.3**	427.0	210.5	320.6	176.5	88.3

Genes	1000		
Seq. Len.	25 bp	100 bp	Full
DP	**117.1**	**65.3**	**33.0**
PAUP*	118.5	67.7	35.1

Smaller criterion scores are better

100-taxon data, low/mid ILS: DP almost always finds better SVDQuartets criterion scores than PAUP*

			25 bp	100 bp	Full
10 genes	Low ILS	DP	**12245.5**	**8230.0**	**3485.5**
		PAUP*	12268.3	8253.4	3510.0
	Mid ILS	DP	2992.9	**5502.5**	**2768.1**
		PAUP*	**2937.0**	5528.8	2776.4
100 genes	Low ILS	DP	**6055.3**	**3279.0**	**1352.5**
		PAUP*	6082.2	3357.0	1462.4
	Mid ILS	DP	**4062.9**	**2269.9**	**1063.8**
		PAUP*	4065.1	2292.3	1091.3
1000 genes	Low ILS	DP	**2182.8**	**1180.2**	**496.1**
		PAUP*	2200.7	1233.0	544.8
	Mid ILS	DP	**1553.5**	**873.3**	**414.9**
		PAUP*	1566.4	882.9	439.5

Smaller criterion scores are better

100-taxon data, high ILS: PAUP* finds better scores with short sequences, few genes

			25 bp	100 bp	Full
10 genes	High ILS	DP	34.4	241.2	1317.9
		PAUP*	**23.5**	**216.7**	**1305.2**
100 genes	High ILS	DP	1531.6	1436.2	**864.4**
		PAUP*	**1418.0**	**1413.0**	867.7
1000 genes	High ILS	DP	1149.4	**670.6**	**351.7**
		PAUP*	**1039.6**	677.5	355.2

Observations

- Under low and medium ILS conditions, the DP algorithm finds trees with scores as good as or better than PAUP*

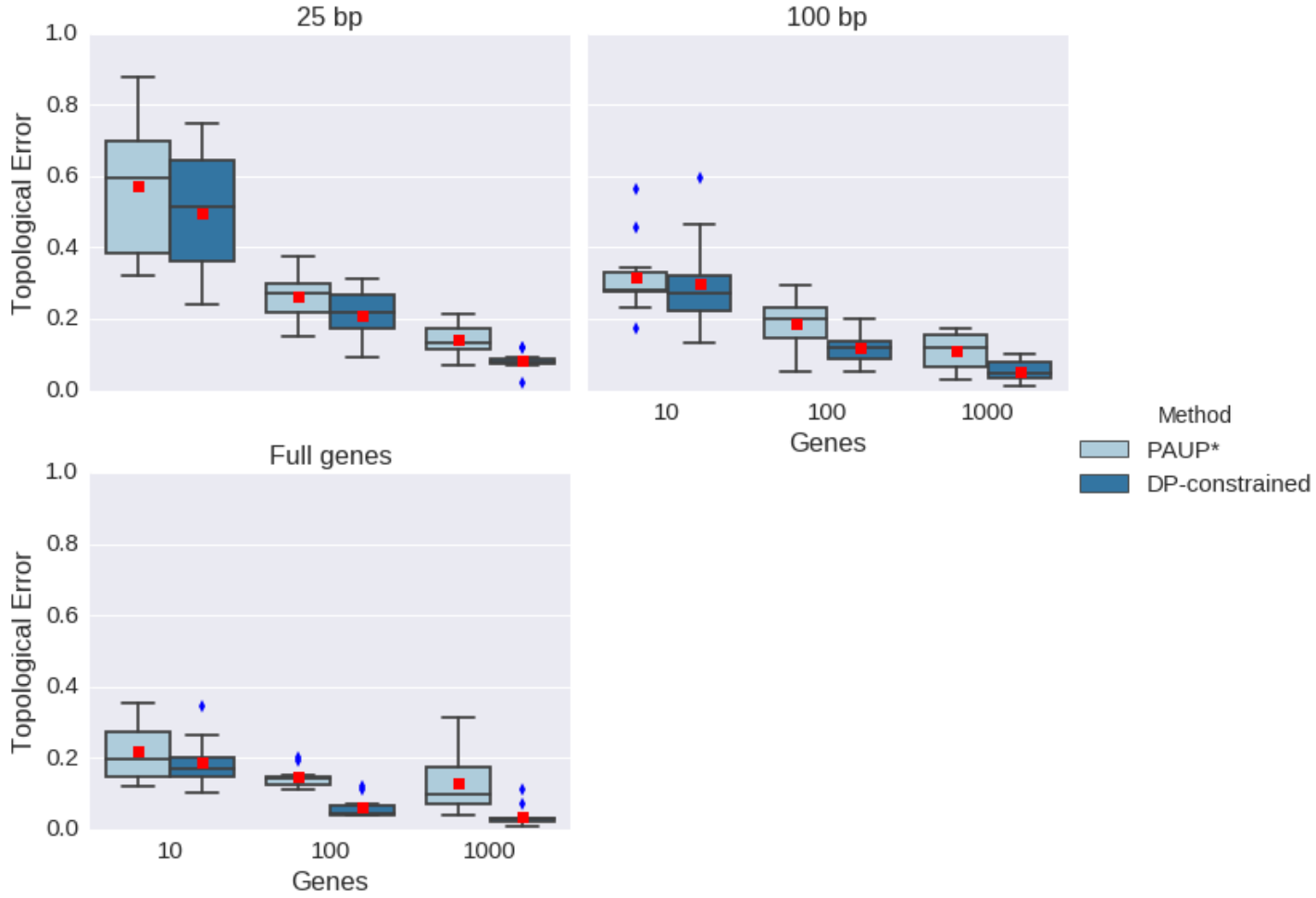
Observations

- Under low and medium ILS conditions, the DP algorithm finds trees with scores as good as or better than PAUP*
- Under high ILS conditions, especially with short sequences and few genes, PAUP* gets better scores than the DP algorithm

Observations

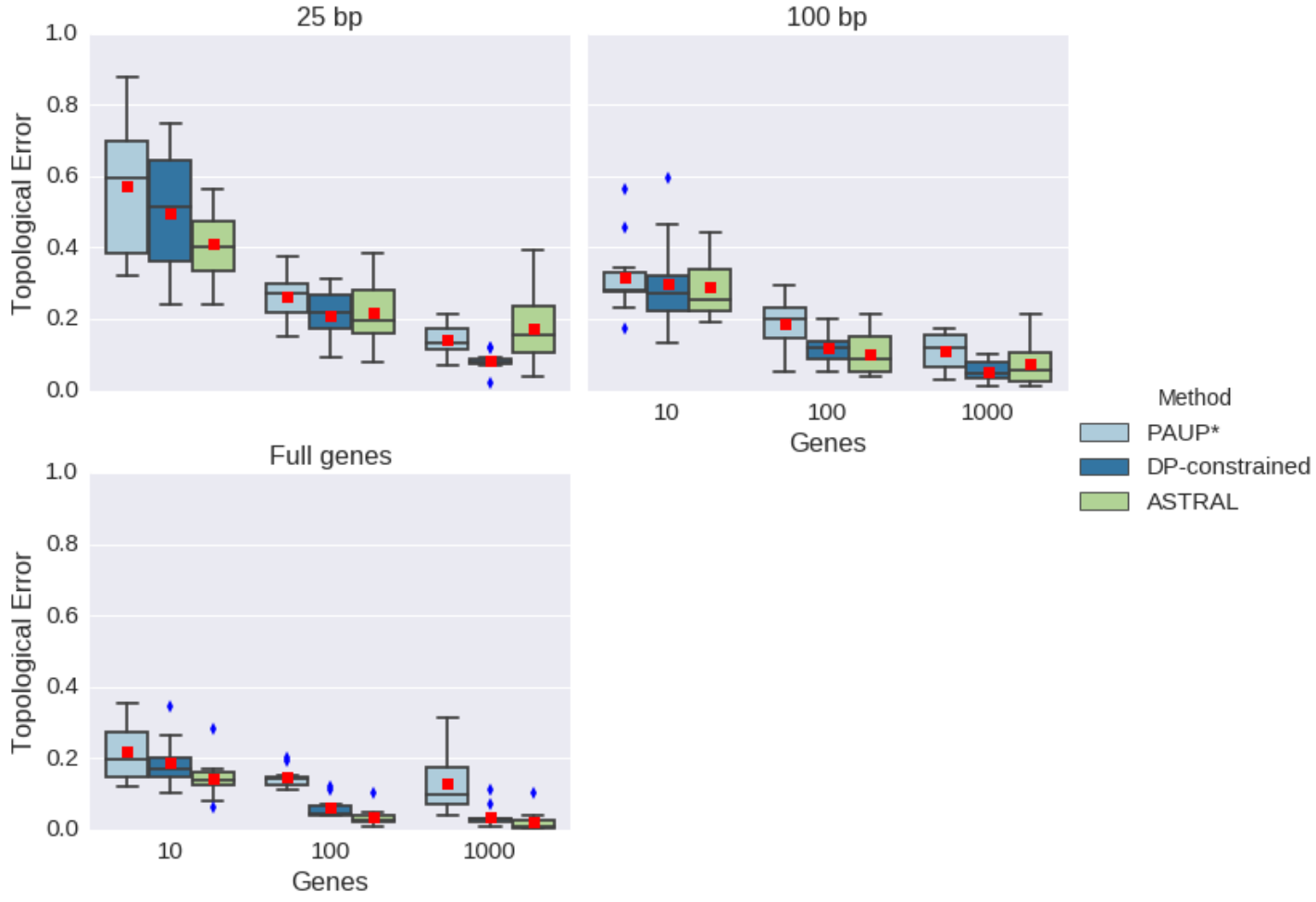
- Under low and medium ILS conditions, the DP algorithm finds trees with scores as good as or better than PAUP*
- Under high ILS conditions, especially with short sequences and few genes, PAUP* gets better scores than the DP algorithm
  - This is due to the constraint set X not containing good enough bipartitions

100-taxon data, low ILS: Improving criterion scores leads to reduced topological error



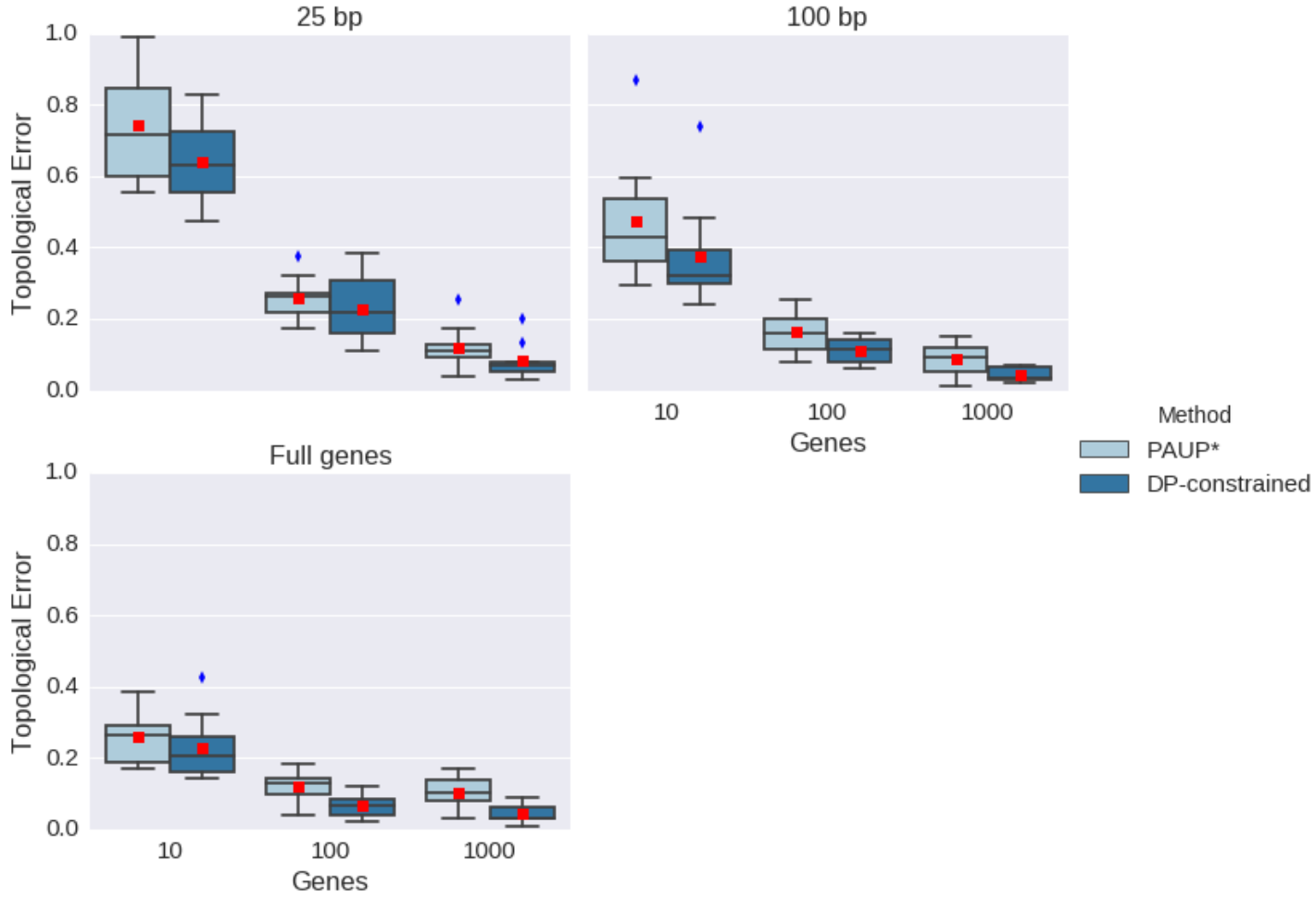
Lower topological error is better

100-taxon data, low ILS: Improving criterion scores with DP make SVDQuartets competitive with ASTRAL



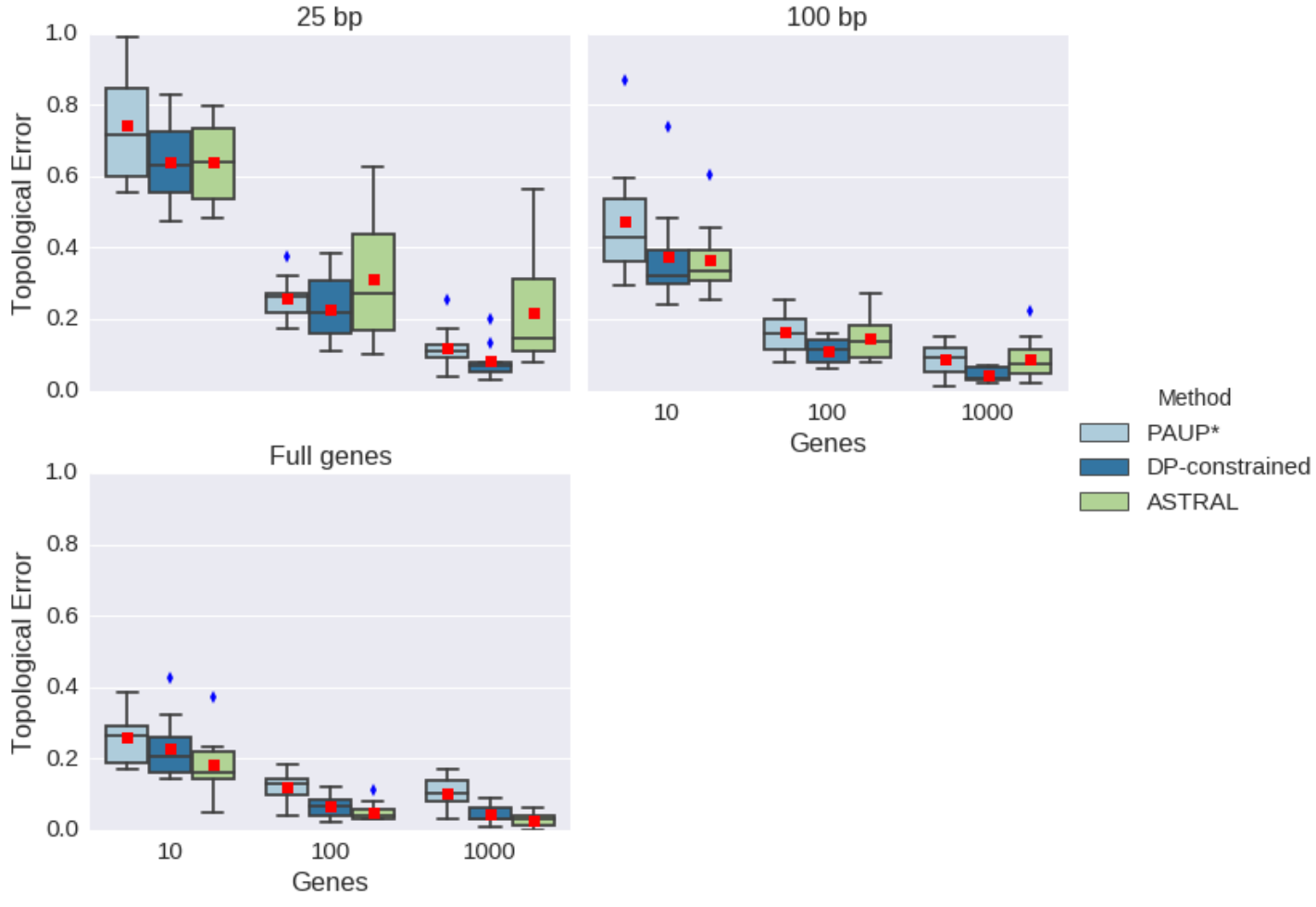
Lower topological error is better

100-taxon data, medium ILS: Improving criterion scores leads to reduced topological error



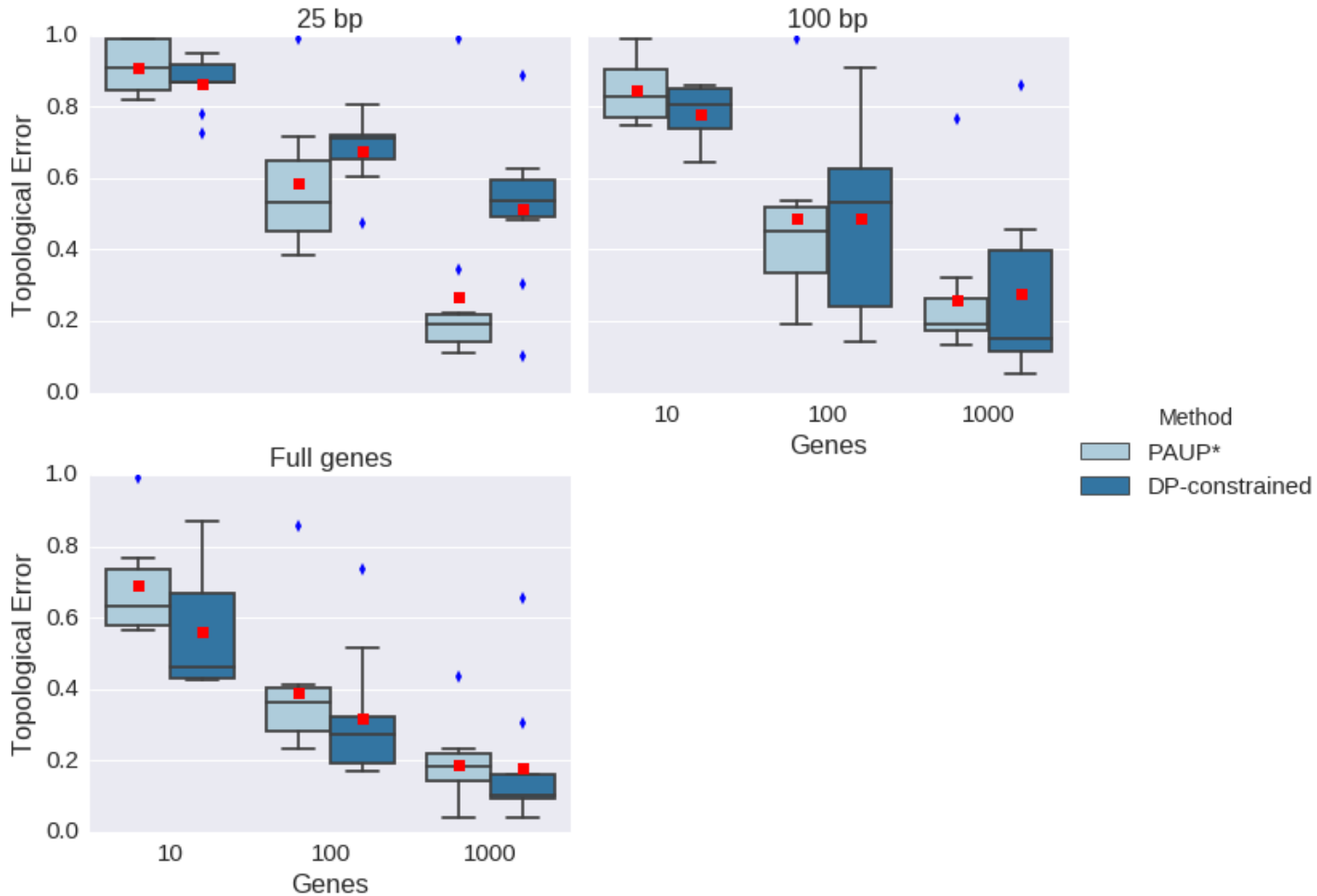
Lower topological error is better

100-taxon data, medium ILS: Improving criterion scores with DP make SVDQuartets competitive with ASTRAL



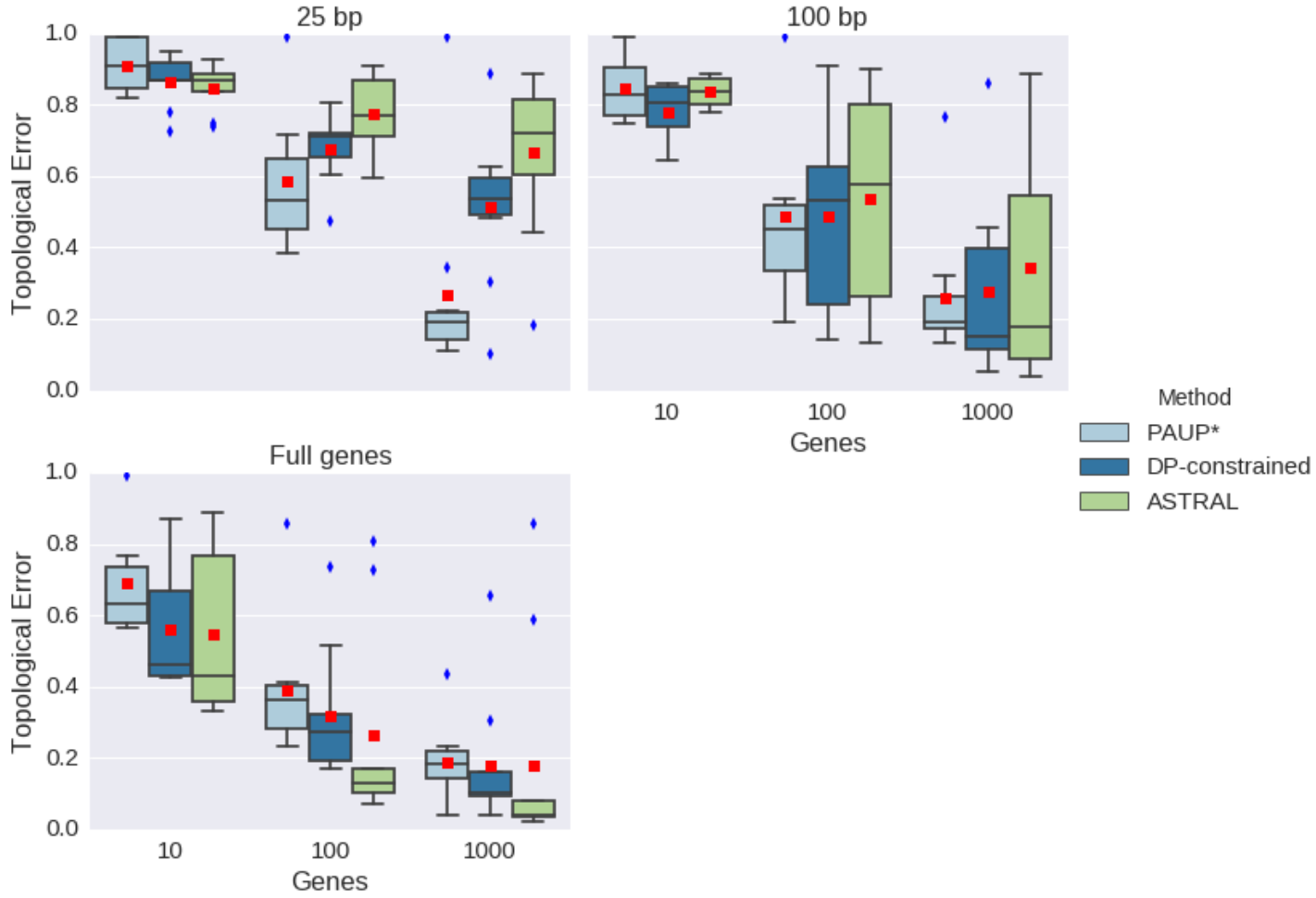
Lower topological error is better

100-taxon data, high ILS: PAUP* is better than DP on short sequences



Lower topological error is better

100-taxon data, high ILS: PAUP* is better than DP and ASTRAL on short sequences, but DP can help on longer sequences



Lower topological error is better

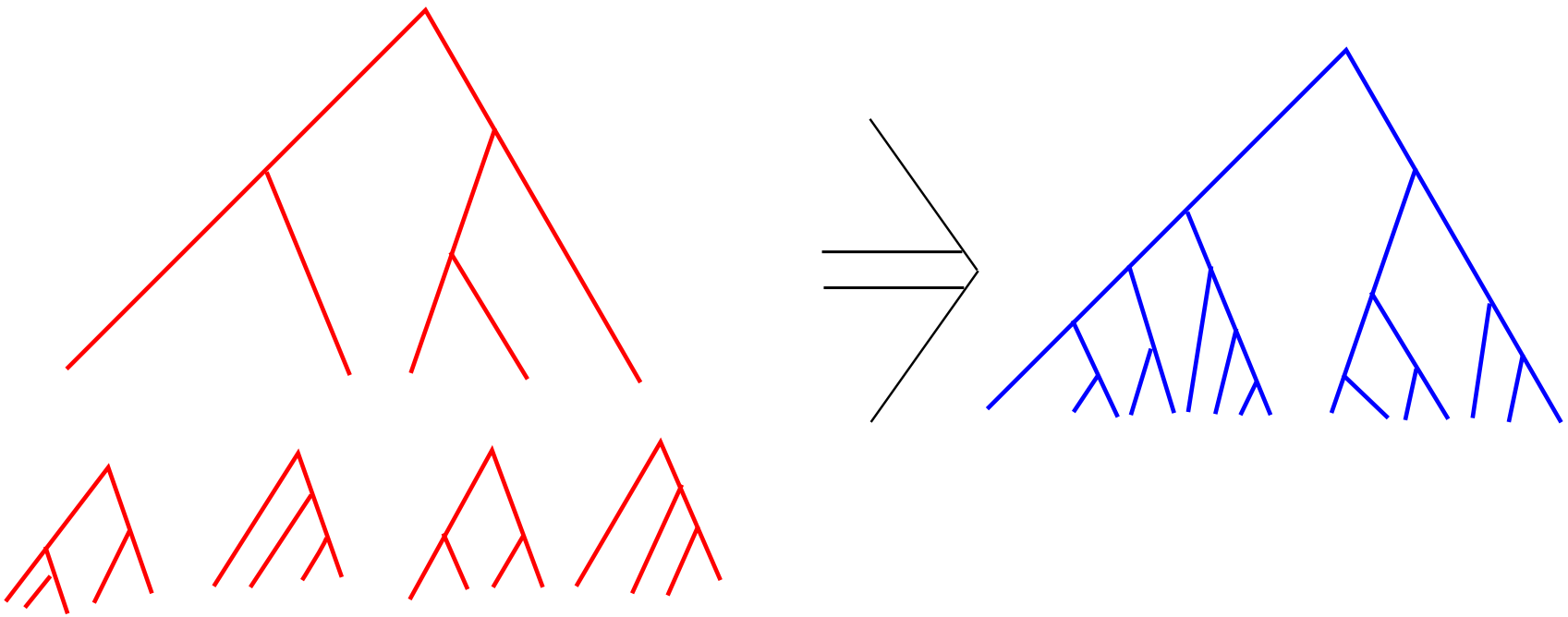
SVDQuartets: Lessons learned

- Under low and medium ILS conditions, DP finds better criterion scores and more accurate trees than PAUP*, and is competitive with ASTRAL with respect to tree accuracy.
- Under high ILS conditions, especially when a limited amount of data is available, PAUP* finds more accurate trees with better criterion scores than the DP algorithm, and more accurate trees than ASTRAL.

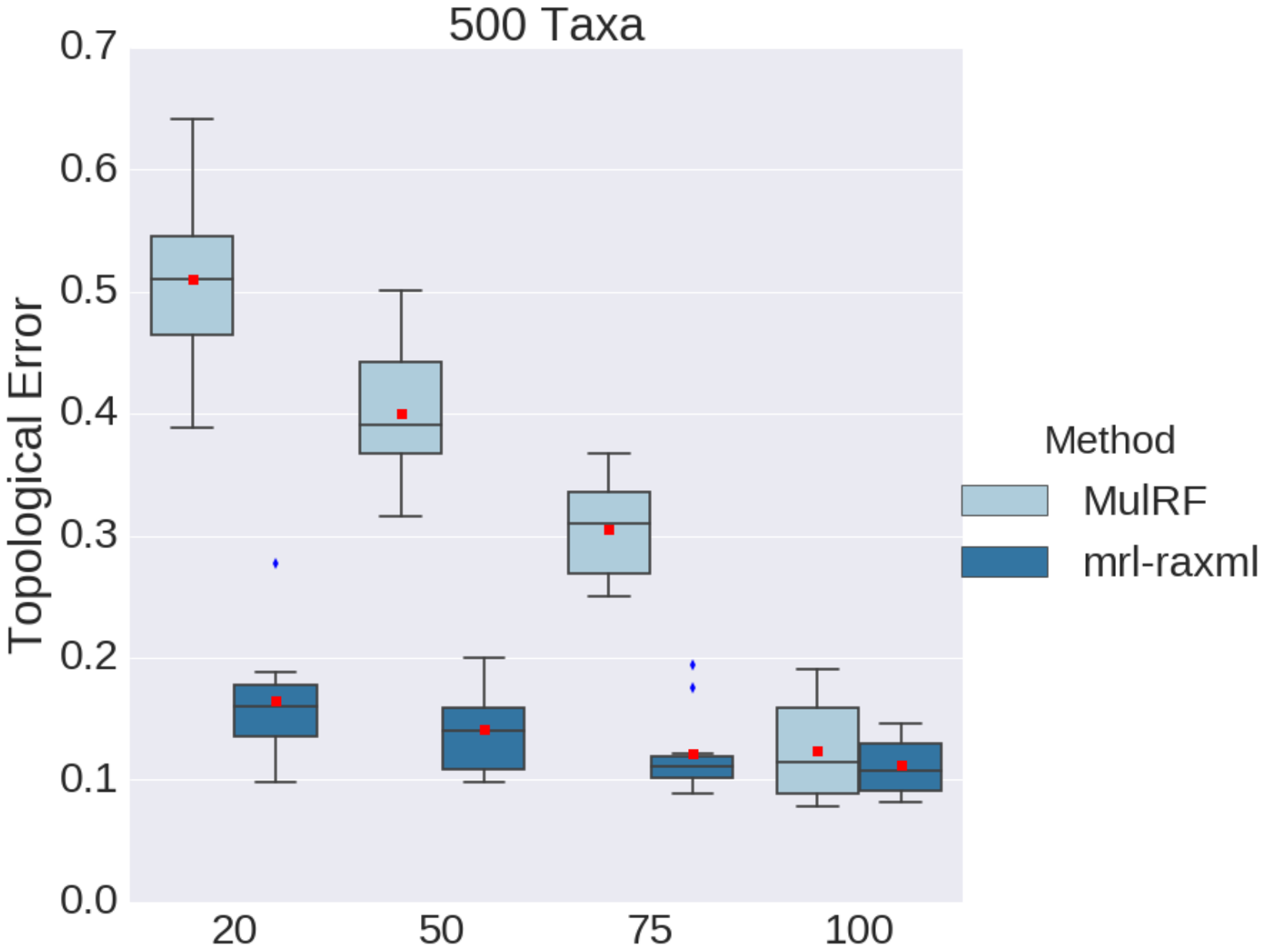
SVDQuartets: Lessons learned

- Under low and medium ILS conditions, DP finds better criterion scores and more accurate trees than PAUP*, and is competitive with ASTRAL with respect to tree accuracy.
- Under high ILS conditions, especially when a limited amount of data is available, PAUP* finds more accurate trees with better criterion scores than the DP algorithm, and more accurate trees than ASTRAL.
- High ILS conditions make gene trees very different from species tree (69% difference between true gene trees and species tree)
- Hard to find a good set X

Supertree Estimation - another important phylogenetic problem



Supertree Estimation - another important phylogenetic problem



Experiment: RF Supertrees

Questions:

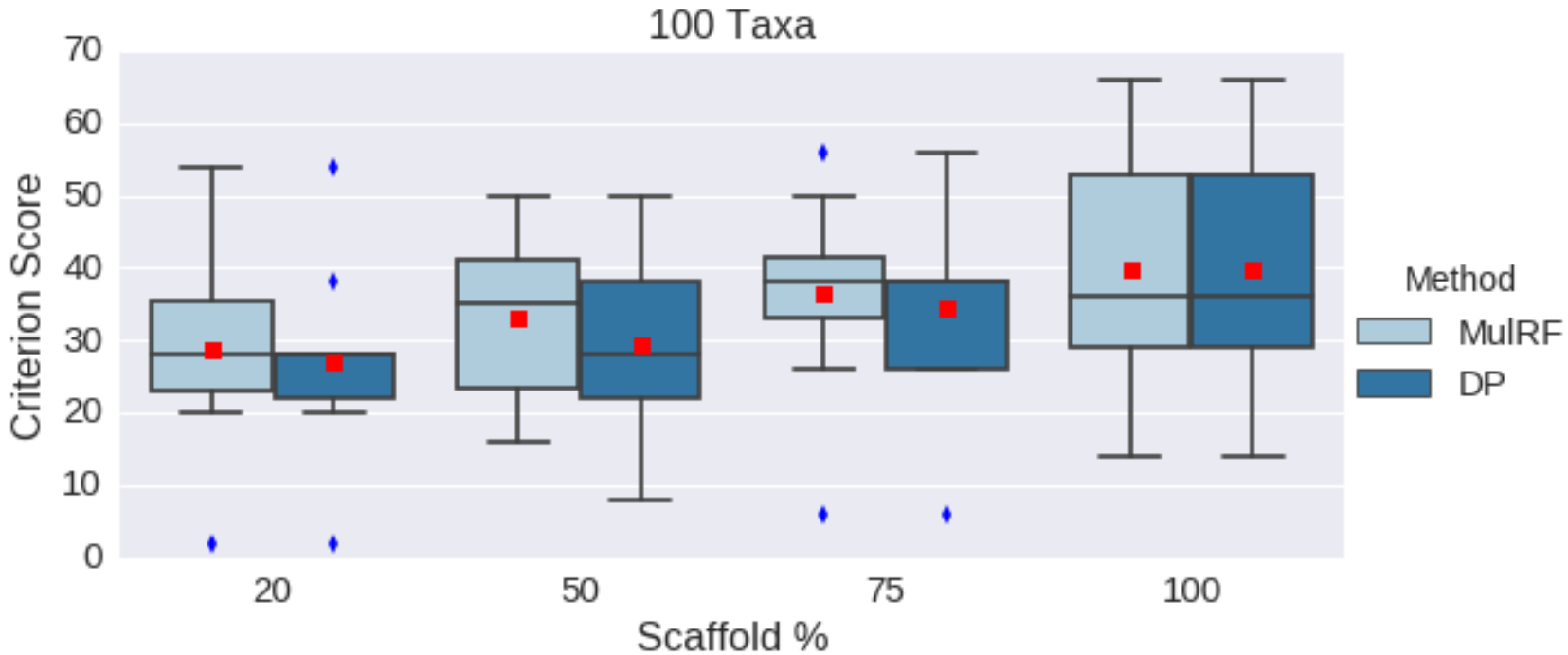
1. Can the DP algorithm find better solutions to the RF supertrees problem than a reference implementation (MulRF)?
2. Can the RF criterion with the DP algorithm find more accurate trees than other leading supertree methods (MRL)?

Datasets: RF Supertrees

- SMIDgen datasets (Swenson et al., 2010)
  - 100, 500, 1000 species (taxa)
  - Scaffold tree with 20, 50, 75, or 100% of the species
  - 5, 15, or 25 trees with clade-based subsets of species
- Biological datasets

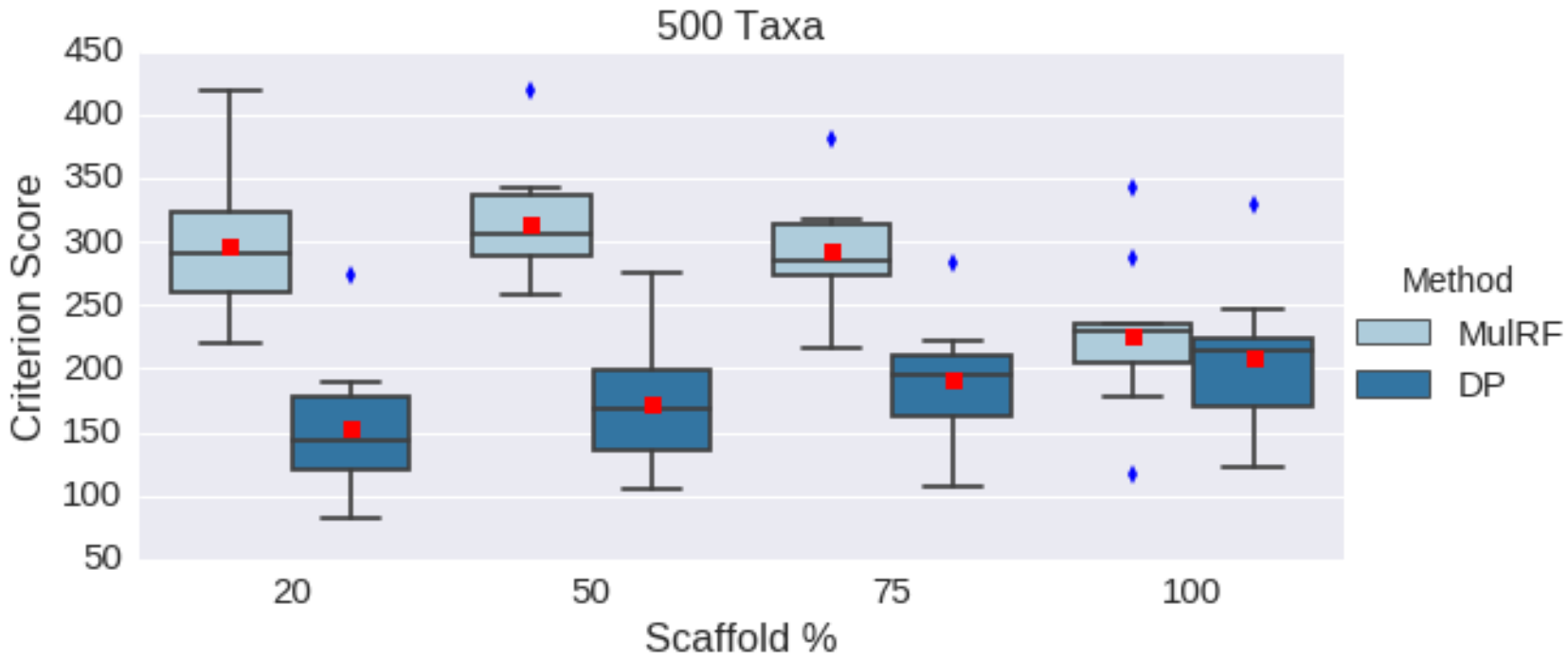
Dataset	# source trees	# taxa	Largest tree
THPL	19	558	140 (25%)
CPL	39	2228	1648 (73%)
Seabirds	7	121	90 (74%)
Marsupials	158	267	267 (100%)
Placental Mammals	726	116	116 (100%)

The DP algorithm almost always finds a better RF criterion score than MulRF



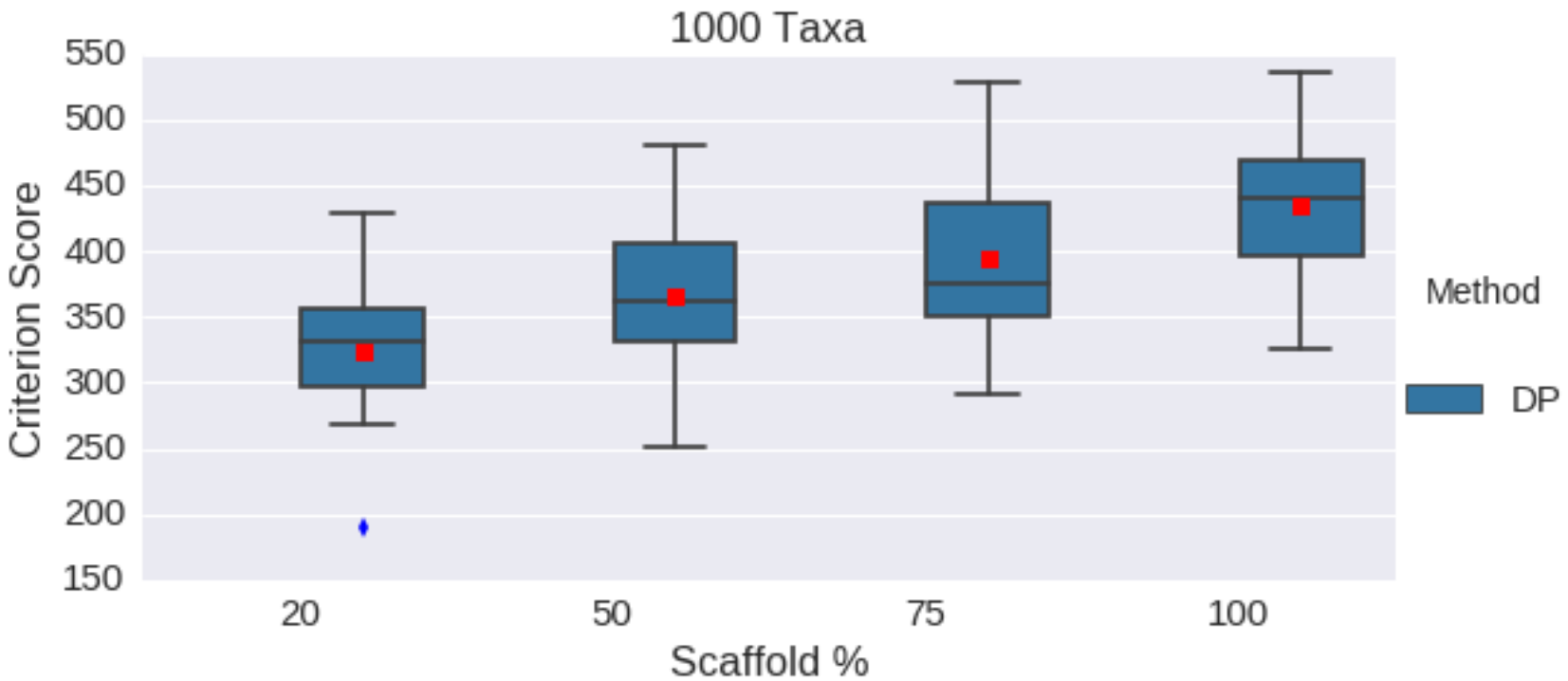
Lower criterion scores are better

The DP algorithm always finds a better RF criterion score than MulRF



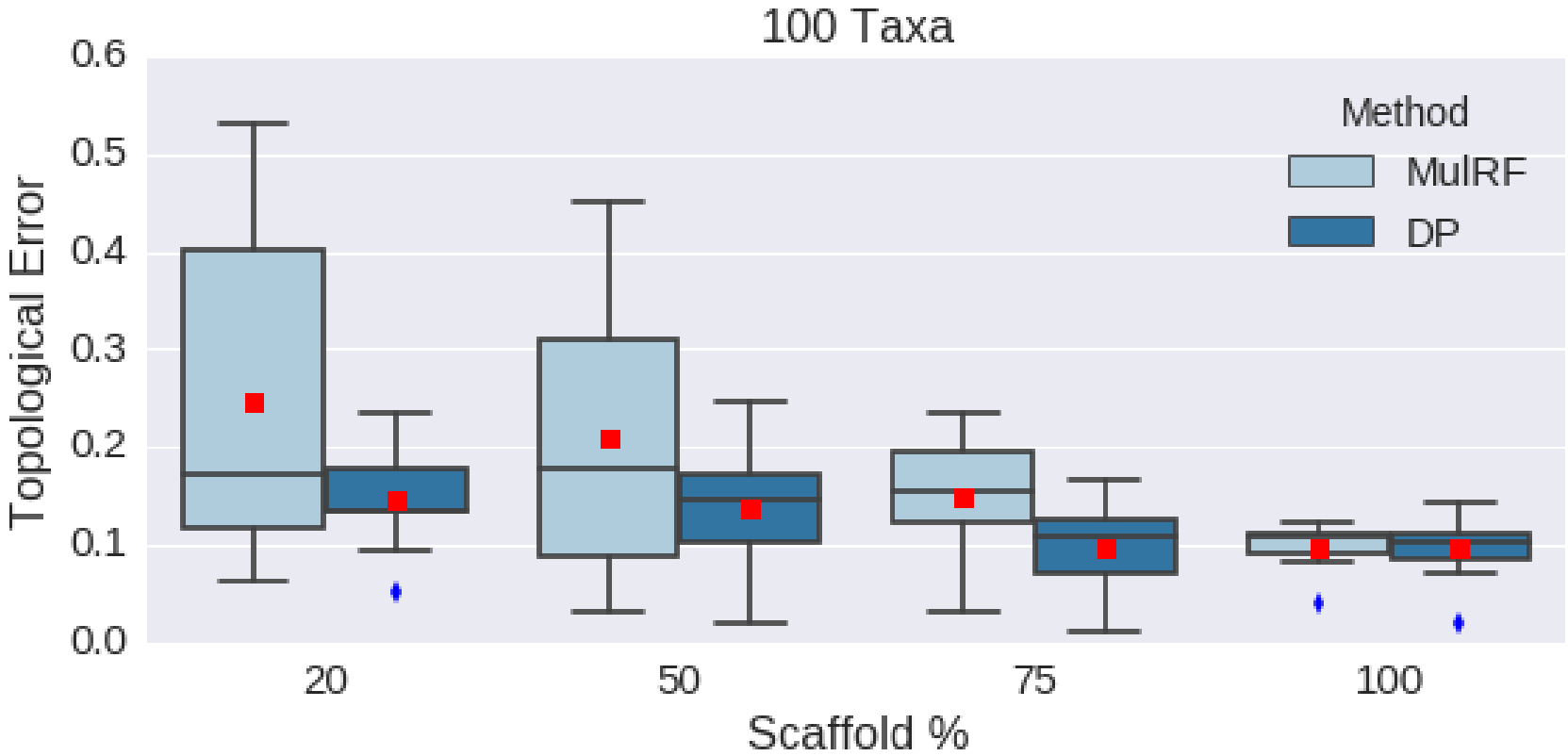
Lower criterion scores are better

MulRF is not able to run on the 1000-taxon data set



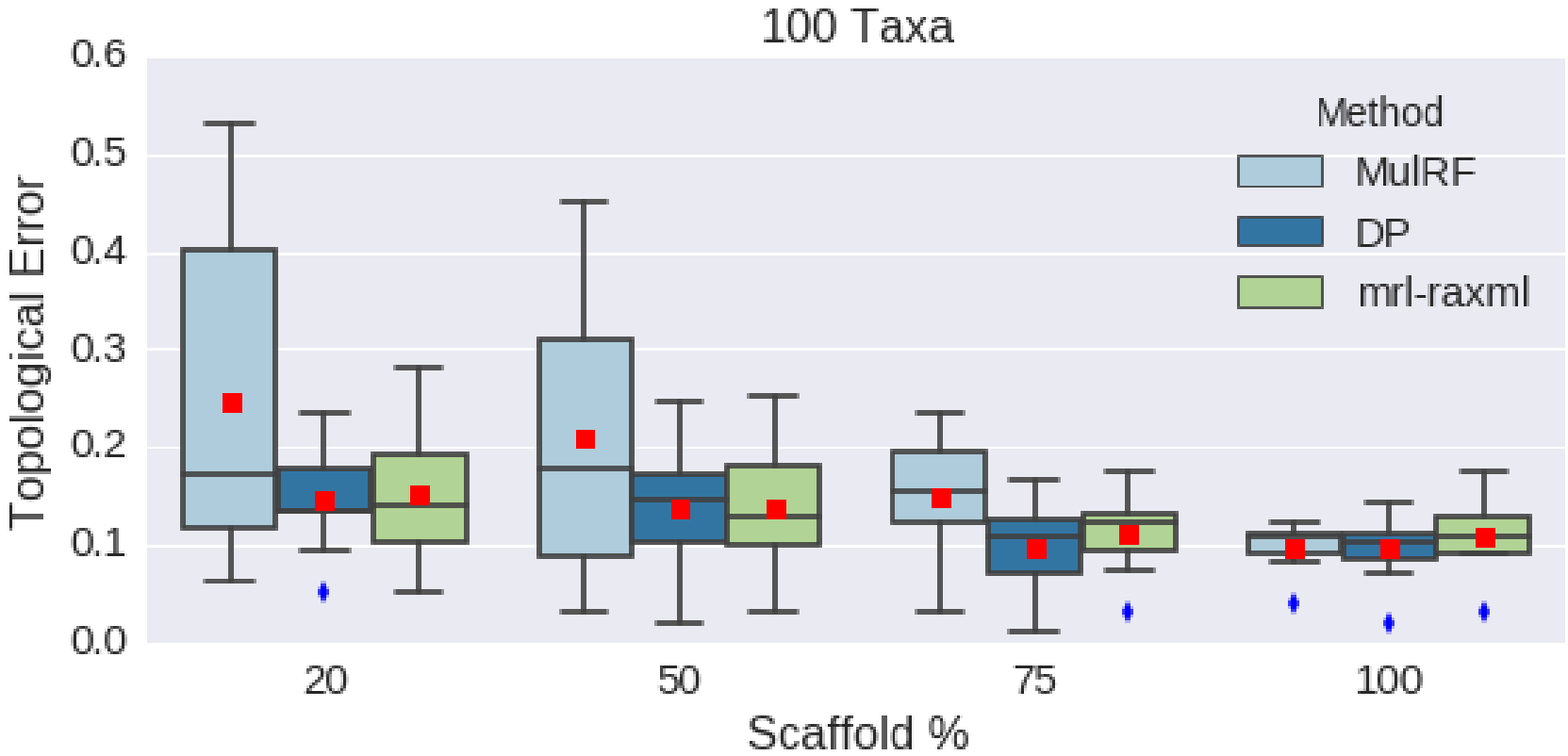
Lower criterion scores are better

This translates to much better supertrees



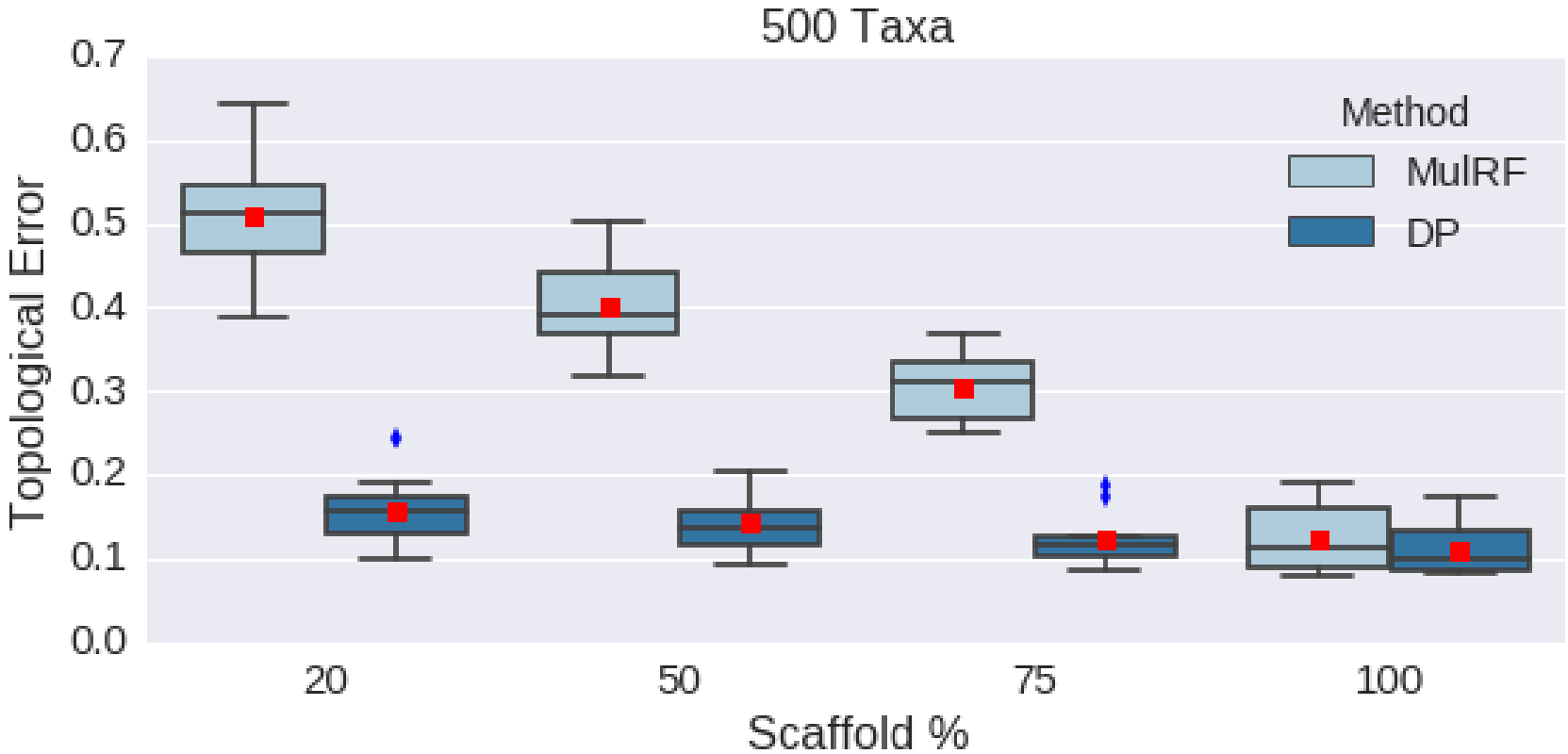
Lower topological error is better

This translates to much better supertrees, competitive with MRL



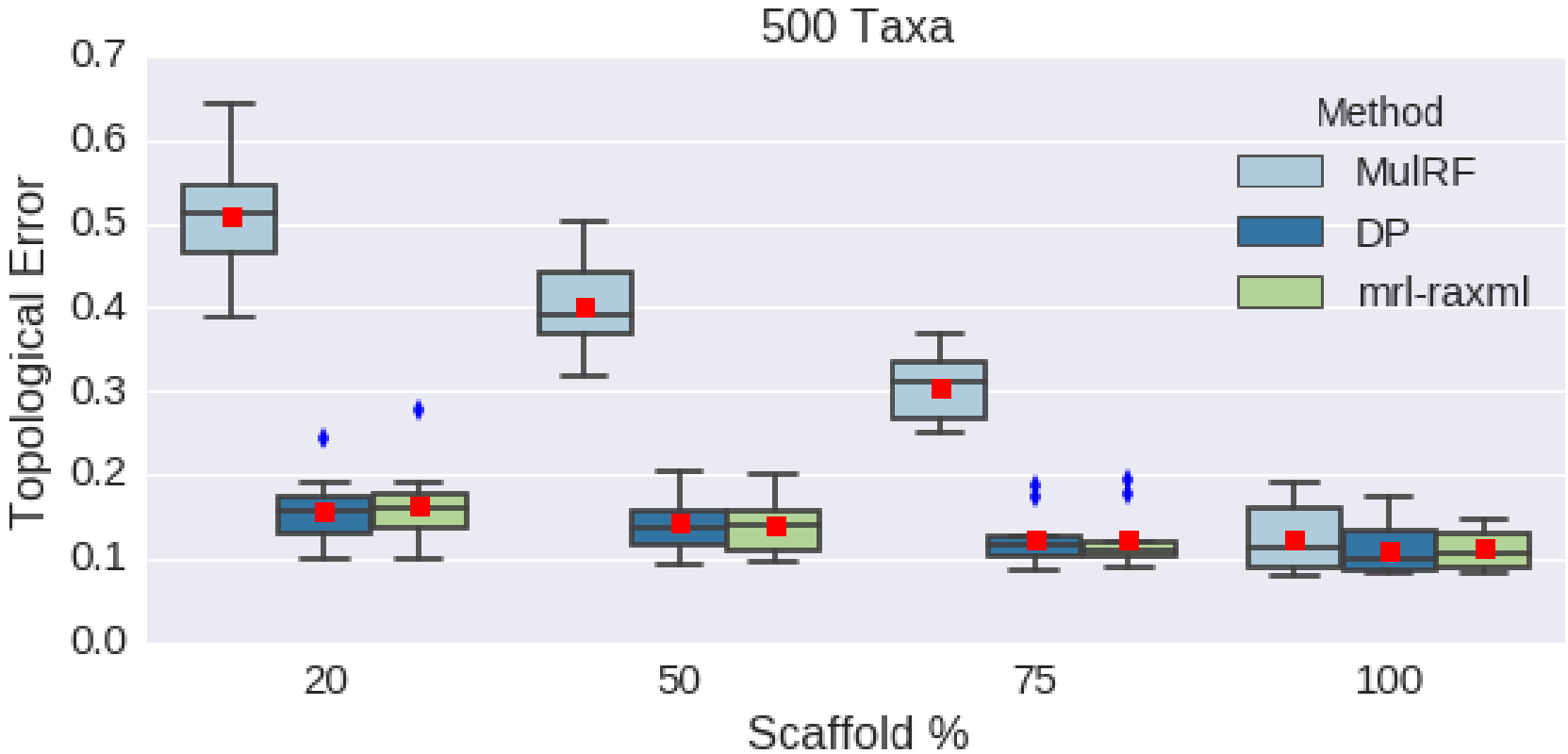
Lower topological error is better

This translates to much better supertrees



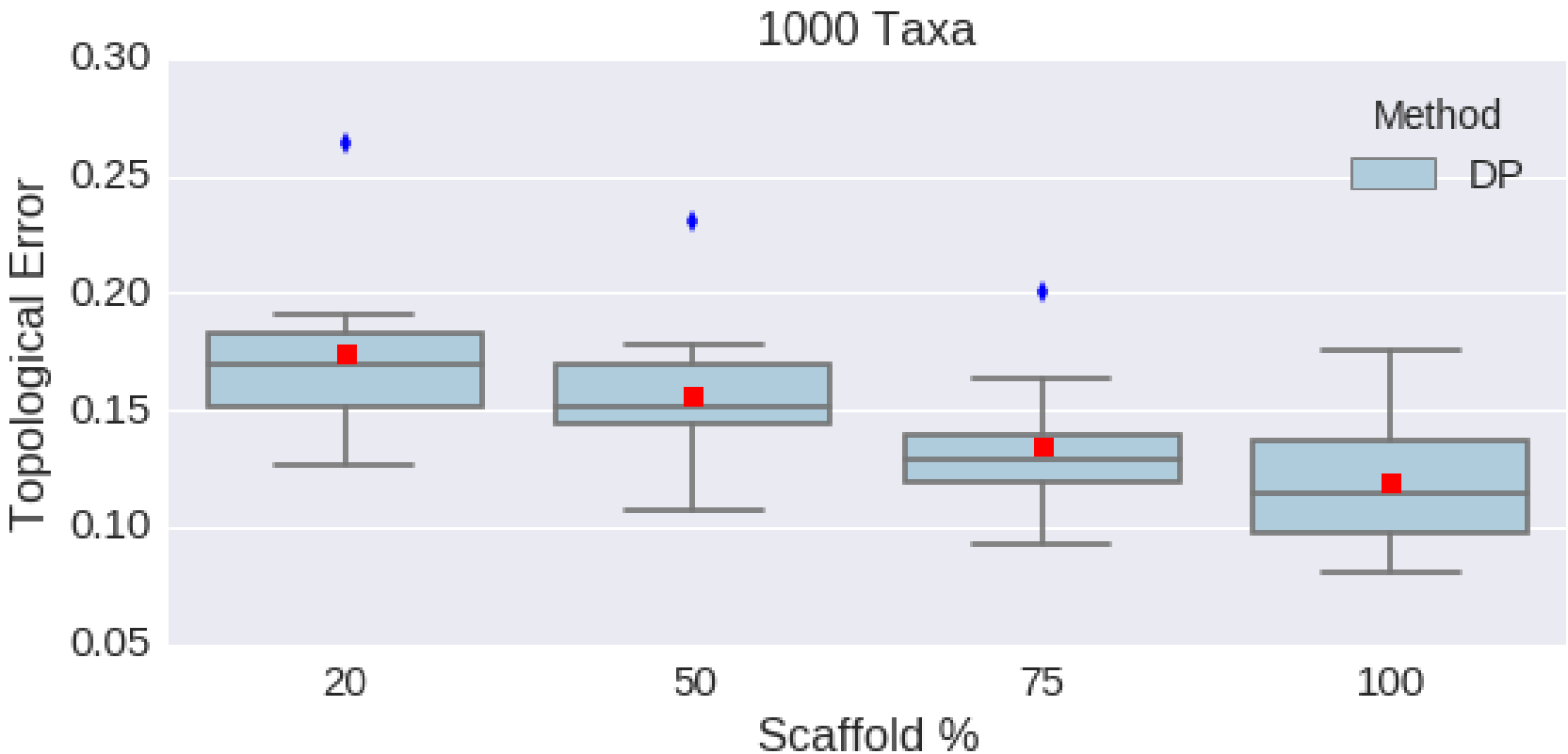
Lower topological error is better

This translates to much better supertrees, competitive with MRL



Lower topological error is better

The DP algorithm can run with good accuracy on 1000-taxon datasets



Much faster than competing methods

Taxa	Scaffold	MulRF	DP	MRL
100	20	1028.3	6.6	30.8
100	50	1144.2	6.7	35.4
100	75	1033.0	6.8	38.1
100	100	1151.7	7.2	40.2
500	20	20951	61	1837.3
500	50	25460	71	2158.5
500	75	28472	75	2049.0
500	100	32527	73	-
1000	20	-	222.9	-
1000	50	-	259.5	-
1000	75	-	289.9	-
1000	100	-	284.8	-

Table: Running times (seconds) for RF Supertree methods.

Good performance on most biological datasets

Dataset	MulRF	DP
CPL	–	**2294**
THPL	617	**475**
Marsupials	1533	**1425**
Placental Mammals	**5387**	5449
Seabirds	99	**79**

Table: RF Criterion scores

Dataset	MulRF	DP
THPL	486	3.3
CPL	–	32
Seabirds	5	0.005
Marsupials	672	1.2
Placental Mammals	691	8.8

Table: Running times (min)

Discussion of biological datasets

- Biological datasets are somewhat different from the simulated datasets
  - More source trees
  - More poorly resolved nodes (nodes with high degree)

Discussion of biological datasets

- Biological datasets are somewhat different from the simulated datasets
  - More source trees
  - More poorly resolved nodes (nodes with high degree)
- DP finds more accurate trees than MulRF for all datasets except placental mammals
  - The dataset has a 100% scaffold, which is a good condition for MulRF
  - The dataset has several very poorly resolved nodes, which makes it challenging to find a good set of bipartitions
  - This results in increased runtime and a worse criterion score

Discussion of biological datasets

- Biological datasets are somewhat different from the simulated datasets
  - More source trees
  - More poorly resolved nodes (nodes with high degree)
- DP finds more accurate trees than MulRF for all datasets except placental mammals
  - The dataset has a 100% scaffold, which is a good condition for MulRF
  - The dataset has several very poorly resolved nodes, which makes it challenging to find a good set of bipartitions
  - This results in increased runtime and a worse criterion score
- Next step: Add trees from MulRF search to DP constraint set X

Ongoing work

- Choice of bipartition set
  - Add bipartitions from best tree to DP bipartition set, especially for biological analyses
- New criteria
  - MulRF can optimize over multi-copy gene trees
  - MRP is another supertree method that uses parsimony
- Better understanding of conditions where the DP algorithm performs well or poorly
  - Missing taxa for SVDQuartets
  - Poorly resolved source trees
- Finish running time analysis for SVDQuartets, taking into account the size of the constraint set X
- In progress for ECCB - due March 29

The DP algorithm is fast, accurate and versatile

- The DP algorithm optimizes any criterion that can be expressed as a sum of tripartition scores
- Finds exact solutions over a constrained search space
- Often faster, more accurate than heuristic searches
- Need to be careful with choice of constraint space
  - DP algorithm performs poorly when very little data are available
  - Adding bipartitions from other methods substantially improves this
- Enables several exciting new research projects, including Bayesian quartet weighting (with Mike Nute), gene tree correction (with Ashu Gupta), divide-and-conquer species tree estimation with DACTAL (with Tandy Warnow), and quintet-based species tree estimation and rooting (with Ruth Davidson)