Fast Robinson-Foulds Supertrees

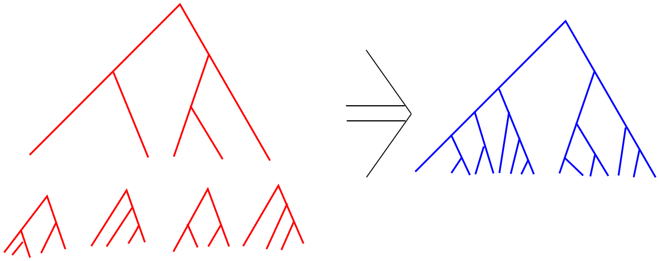
CS598AGB

Pranjal Vachaspati

University of Illinois at Urbana-Champaign

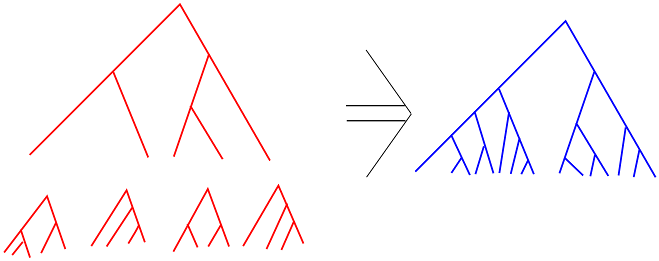
Supertree Estimation - combine trees on subsets of data into a single large tree

- Combine sparse scaffold + dense trees on subsets



Supertree Estimation - combine trees on subsets of data into a single large tree

- Combine sparse scaffold + dense trees on subsets



- No discrepancy among true source trees - all disagreement is due to estimation error
  - This means no ILS, no HGT, etc.

Models of missing data

1. Randomly pick taxa per tree

Models of missing data

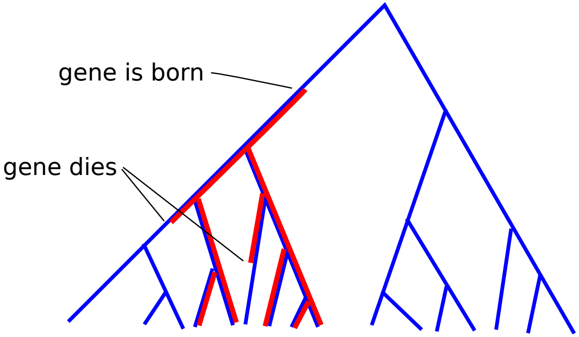
1. Randomly pick taxa per tree
    - Represents scenario where biologists have done analyses of taxa based on what was available to them
    - Usually doesn't correspond to biological reasons for missing data

Models of missing data

1. Randomly pick taxa per tree
    - Represents scenario where biologists have done analyses of taxa based on what was available to them
    - Usually doesn't correspond to biological reasons for missing data
2. Gene birth/death model - more similar to biological processes

Models of missing data

1. Randomly pick taxa per tree
   - Represents scenario where biologists have done analyses of taxa based on what was available to them
   - Usually doesn't correspond to biological reasons for missing data
2. Gene birth/death model - more similar to biological processes

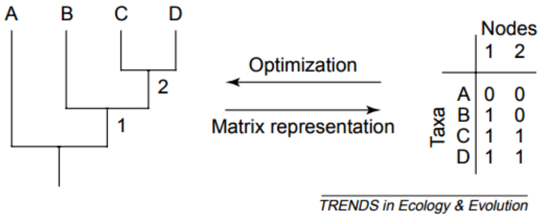


Methods

1. Use species tree estimation methods like ASTRAL, ASTRID, concatenation max-likelihood, concatenated neighbor-joining, etc.

Methods

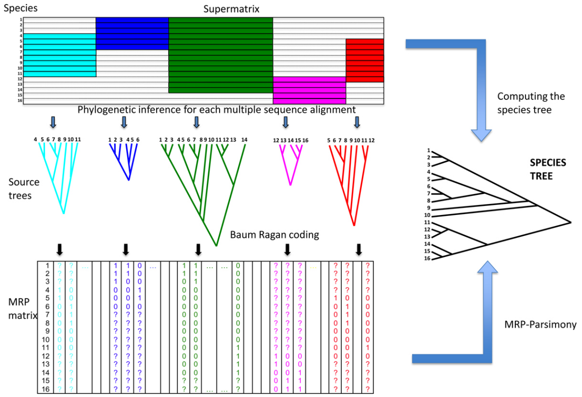
1. Use species tree estimation methods like ASTRAL, ASTRID, concatenation max-likelihood, concatenated neighbor-joining, etc.
2. Use likelihood, parsimony, etc. on matrix representation (MRP, MRL):



Taxa	Node 1	Node 2
A	0	0
B	1	0
C	1	1
D	1	1

Methods

1. Use species tree estimation methods like ASTRAL, ASTRID, concatenation max-likelihood, concatenated neighbor-joining, etc.
2. Use likelihood, parsimony, etc. on matrix representation (MRP, MRL):



Species
Supermatrix
Phylogenetic inference for each multiple sequence alignment
Source trees
Baum Ragan coding
MRP matrix
Computing the species tree
SPECIES TREE
MRP-Parsimony

Methods

1. Use species tree estimation methods like ASTRAL, ASTRID, concatenation max-likelihood, concatenated neighbor-joining, etc.
2. Use likelihood, parsimony, etc. on matrix representation (MRP, MRL):
3. Use a heuristic designed for our models of supertrees and source tree error

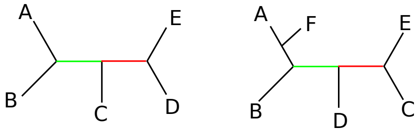
RF Supertrees

- Exponential decay model of gene tree error (Steel and Rodrigo, 2009)

$$P_{T,Y}[T'] = \alpha \exp[-\beta RF(T', T|Y)]$$

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

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



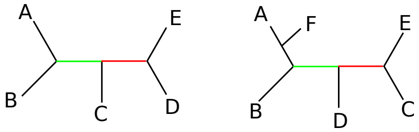
RF Supertrees

- Exponential decay model of gene tree error (Steel and Rodrigo, 2009)

$$P_{T,Y}[T'] = \alpha \exp[-\beta RF(T', T|Y)]$$

- 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

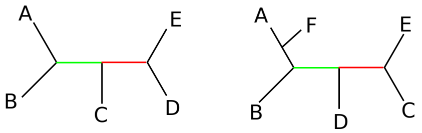
RF Supertrees

- Exponential decay model of gene tree error (Steel and Rodrigo, 2009)

$$P_{T,Y}[T'] = \alpha \exp[-\beta RF(T', T|Y)]$$

- 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
- The maximum likelihood tree is usually, but not always the RF supertree (depending on α and β parameters)

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

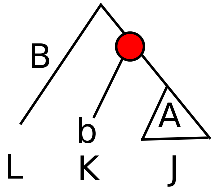
We use an analogous approach for the RF supertree problem

Calculating RF support of a supertree

- Suppose a source tree induces the bipartition (on a taxon subset) $\{A|B\}$

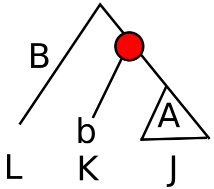
Calculating RF support of a supertree

- Suppose a source tree induces the bipartition (on a taxon subset) $\{A|B\}$
- A supertree will induce this bipartition if there is some *tripartition* in the supertree that
  1. has every member of A (and no members of B) below one child
  2. has at least one member of B below the other child



Calculating RF support of a supertree

- Suppose a source tree induces the bipartition (on a taxon subset) $\{A|B\}$
- A supertree will induce this bipartition if there is some *tripartition* in the supertree that
  1. has every member of A (and no members of B) below one child
  2. has at least one member of B below the other child



- In other words, the single highest edge in the tree that induces $\{A|B\}$

Calculating RF support of a supertree

- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it

Calculating RF support of a supertree

- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it
- Then get the RF support of an entire supertree by adding the RF supports of each tripartition in the tree

Calculating RF support of a supertree

- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it
- Then get the RF support of an entire supertree by adding the RF supports of each tripartition in the tree
- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *EFG* split at the root is the sum of

Calculating RF support of a supertree

- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it
- Then get the RF support of an entire supertree by adding the RF supports of each tripartition in 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*

Calculating RF support of a supertree

- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it
- Then get the RF support of an entire supertree by adding the RF supports of each tripartition in 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*

Calculating RF support of a supertree

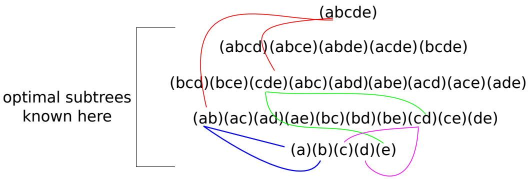
- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it
- Then get the RF support of an entire supertree by adding the RF supports of each tripartition in the tree
- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *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*

Calculating RF support of a supertree

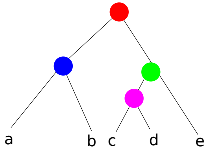
- So we get the RF support of a tripartition in the supertree by counting bipartitions in the input set that correspond to it
- Then get the RF support of an entire supertree by adding the RF supports of each tripartition in the tree
- The weight of the best subtree on *ABCDEFG* that has the *ABCD* vs *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*

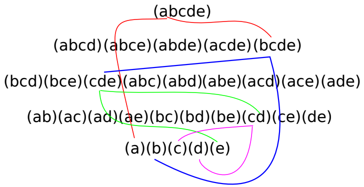
Calculating an optimal supertree

For example, to find the optimal subtree on (*abcde*), we have to try, (among others):

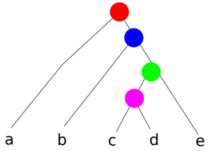


(abcde)
(abcd)(abce)(abde)(acde)(bcde)
(bcd)(bce)(cde)(abc)(abd)(abe)(acd)(ace)(ade)
(ab)(ac)(ad)(ae)(bc)(bd)(be)(cd)(ce)(de)
(a)(b)(c)(d)(e)
optimal subtrees known here





(abcde)
(abcd)(abce)(abde)(acde)(bcde)
(bcd)(bce)(cde)(abc)(abd)(abe)(acd)(ace)(ade)
(ab)(ac)(ad)(ae)(bc)(bd)(be)(cd)(ce)(de)
(a)(b)(c)(d)(e)



Running time analysis

- There are $O(2^n)$ possible clades on n taxa

Running time analysis

- There are $O(2^n)$ possible clades on n taxa
- This algorithm takes exponential time

Constrain the search space to avoid exponential time

- 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

- Instead of exploring entire search space, only allow trees to contain bipartitions from an input set X
- This doesn't work well when all the trees are incomplete - some clades in the true tree appear in the source trees with very low probability

Constrain the search space to avoid exponential time

- Instead of exploring entire search space, only allow trees to contain bipartitions from an input set X
- This doesn't work well when all the trees are incomplete - some clades in the true tree appear in the source trees with very low probability
- ASTRAL-2 has heuristics to complete trees, then get clades from completed trees

Constrain the search space to avoid exponential time

- Instead of exploring entire search space, only allow trees to contain bipartitions from an input set X
- This doesn't work well when all the trees are incomplete - some clades in the true tree appear in the source trees with very low probability
- ASTRAL-2 has heuristics to complete trees, then get clades from completed trees
- FastRFS-basic: use ASTRAL-2 clades with DP algorithm

Running time analysis

- k input trees over n taxa: $O(kn)$ input bipartitions
- Checking if tripartition matches bipartition takes $O(n)$ time

Running time analysis

- 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:$O(|X|^2 n^2 k)$

Running time analysis

- 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:$O(|X|^2 n^2 k)$

Speed up with precomputation

- Comparing a tripartition against a bipartition takes $O(n)$ time

Speed up with precomputation

- Comparing a tripartition against a bipartition takes $O(n)$ time
- We end up checking if a particular clade contains a particular bipartition multiple times!

Speed up with precomputation

- Comparing a tripartition against a bipartition takes $O(n)$ time
- We end up checking if a particular clade contains a particular bipartition multiple times!
- For each clade c, we compute four sets:
  1. c_1: bipartitions where c contains at least one member of the first half
  2. c_2: bipartitions where c contains all of the first half
  3. c_3: bipartitions where c contains at least one member of the second half
  4. c_4: bipartitions where c contains all of the second half

Speed up with precomputation

- Comparing a tripartition against a bipartition takes $O(n)$ time
- We end up checking if a particular clade contains a particular bipartition multiple times!
- For each clade c, we compute four sets:
    1. c_1: bipartitions where c contains at least one member of the first half
    2. c_2: bipartitions where c contains all of the first half
    3. c_3: bipartitions where c contains at least one member of the second half
    4. c_4: bipartitions where c contains all of the second half
- Then the bipartitions that correspond to a tripartition with children $A1, A2$ are the union of $A1_1 \cap A2_4$, $A1_2 \cap A2_3$, $A1_3 \cap A2_2$, and $A1_4 \cap A2_1$

Speed up with precomputation

- Comparing a tripartition against a bipartition takes $O(n)$ time
- We end up checking if a particular clade contains a particular bipartition multiple times!
- For each clade c, we compute four sets:
  1. c_1: bipartitions where c contains at least one member of the first half
  2. c_2: bipartitions where c contains all of the first half
  3. c_3: bipartitions where c contains at least one member of the second half
  4. c_4: bipartitions where c contains all of the second half
- Then the bipartitions that correspond to a tripartition with children $A1, A2$ are the union of $A1_1 \cap A2_4$, $A1_2 \cap A2_3$, $A1_3 \cap A2_2$, and $A1_4 \cap A2_1$
- Precomputation takes $O(|X|n^2k)$ time; the main phase of the algorithm takes $O(|X|^2nk)$ time.
- Since $|X| \geq n$, the total time for the algorithm is $O(|X|^2nk)$ (as opposed to $O(|X|^2n^2k)$ before)

Improve accuracy of FastRFS

- Sometimes other methods find trees with better RF scores than FastRFS
- If we add that tree to the FastRFS constraint set X, we are guaranteed to find a tree at least that good!

Improve accuracy of FastRFS

- Sometimes other methods find trees with better RF scores than FastRFS
- If we add that tree to the FastRFS constraint set X, we are guaranteed to find a tree at least that good!
- FastRFS-enhanced: Add MRL tree to set X, and add ASTRID tree if ASTRID is fast

Experiment

Questions:

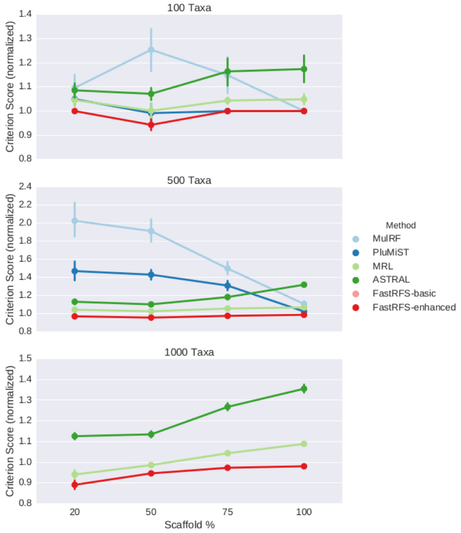
1. Can FastRFS find better solutions to the RF supertree problem than other RF supertree heuristics?
2. Does FastRFS-enhanced find better trees than the extra trees we give it?
3. Does FastRFS find trees with good topological accuracy?
4. Is FastRFS faster than heuristic methods?
5. How does FastRFS perform on biological datasets?

Datasets: RF Supertrees

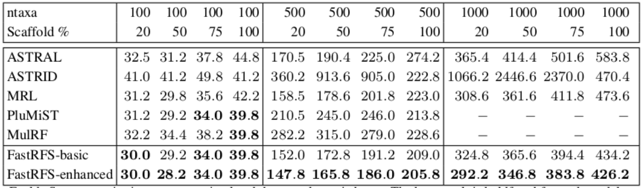
- SMIDgen datasets (Swenson et al., 2010)
  - 100, 500, 1000 species (taxa)
  - Scaffold tree with 20, 50, 75, or 100% of the species
  - 5, 15, or 25 trees with clade-based subsets of species
- Biological datasets

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%)

FastRFS-basic finds trees with good criterion scores, and FastRFS-enhanced improves them



FastRFS-basic finds trees with good criterion scores, and FastRFS-enhanced improves them



ntaxa	100	100	100	100	500	500	500	500	1000	1000	1000	1000
Scaffold %	20	50	75	100	20	50	75	100	20	50	75	100
ASTRAL	32.5	31.2	37.8	44.8	170.5	190.4	225.0	274.2	365.4	414.4	501.6	583.8
ASTRID	41.0	41.2	49.8	41.2	360.2	913.6	905.0	222.8	1066.2	2446.6	2370.0	470.4
MRL	31.2	29.8	35.6	42.2	158.5	178.6	201.8	223.0	308.6	361.6	411.8	473.6
PluMiST	31.2	29.2	**34.0**	**39.8**	210.5	245.0	246.0	213.8	–	–	–	–
MulRF	32.2	34.4	38.2	**39.8**	282.2	315.0	279.0	228.6	–	–	–	–
FastRFS-basic	**30.0**	29.2	**34.0**	**39.8**	152.0	172.8	191.2	209.0	324.8	365.6	394.4	434.2
FastRFS-enhanced	**30.0**	**28.2**	**34.0**	**39.8**	**147.8**	**165.8**	**186.0**	**205.8**	**292.2**	**346.8**	**383.8**	**426.2**

Table 2. Robinson-Foulds Supertree criterion scores on simulated datasets; lower is better. The best result is boldfaced for each model condition. No results are shown for PluMiST and MulRF for the 1000-taxon datasets due to running time limitations for these methods. Results are averaged over ten replicates.

FastRFS finds trees with good criterion scores quickly

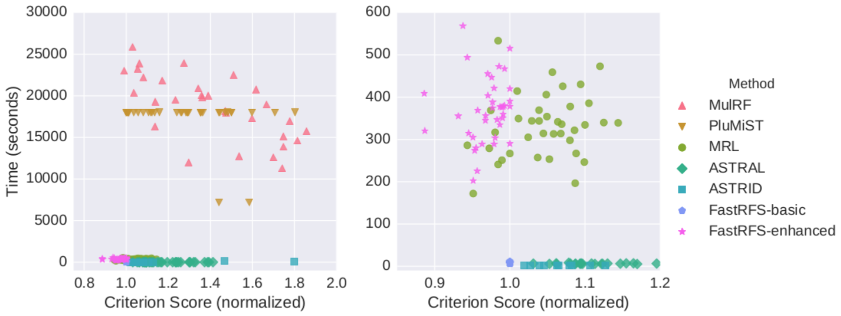
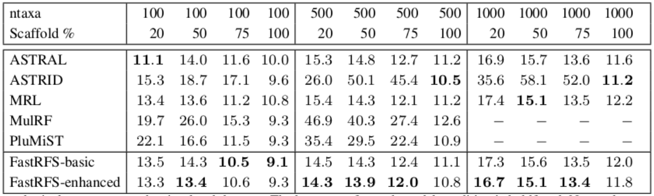


Fig. 4. Scatterplot of normalized Robinson-Foulds Supertree criterion scores (x-axis) and running time (in seconds, y-axis) on the 500-taxon simulated datasets. LEFT: Results shown for all the methods. RIGHT: Results shown only for those methods that obtain good normalized criterion scores (i.e., below 1.2). RF-supertree criterion scores are normalized by the score found by FastMLS-basic, so that values below 1.0 indicate better scores than FastMLS-basic, and values above 1.0 indicate worse scores. The fastest methods are the ones close to the x-axis, but the methods with the best criterion scores are the ones close to the y-axis.

FastRFS almost always finds trees with the lowest topological error



ntaxa	100	100	100	100	500	500	500	500	1000	1000	1000	1000
Scaffold %	20	50	75	100	20	50	75	100	20	50	75	100
ASTRAL	**11.1**	14.0	11.6	10.0	15.3	14.8	12.7	11.2	16.9	15.7	13.6	11.6
ASTRID	15.3	18.7	17.1	9.6	26.0	50.1	45.4	**10.5**	35.6	58.1	52.0	**11.2**
MRL	13.4	13.6	11.2	10.8	15.4	14.3	12.1	11.2	17.4	**15.1**	13.5	12.2
MulRF	19.7	26.0	15.3	9.3	46.9	40.3	27.4	12.6	–	–	–	–
PluMiST	22.1	16.6	11.5	9.3	35.4	29.5	22.4	10.9	–	–	–	–
FastRFS-basic	13.5	14.3	**10.5**	**9.1**	14.5	14.3	12.4	11.1	17.3	15.6	13.5	12.0
FastRFS-enhanced	13.3	**13.4**	10.6	9.3	**14.3**	**13.9**	**12.0**	10.8	**16.7**	**15.1**	**13.4**	11.8

Table 3. Supertree topological error rates on the simulated datasets. The best score for each model condition is boldfaced. No results are shown for PluMiST and MulRF on the 1000-taxon simulated datasets due to running time limitations for these methods. Results are averaged over ten replicates.

FastRFS (quickly) finds good criterion scores on biological data

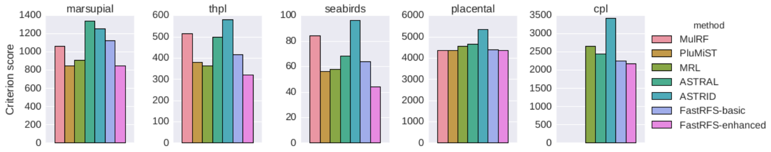


Fig. 2. Robinson-Foulds Supertree criterion scores on biological data of supertree methods. MulRF and PluMiST could not be run on the CPL dataset, due to its large size; hence no values are shown for those methods on that dataset.

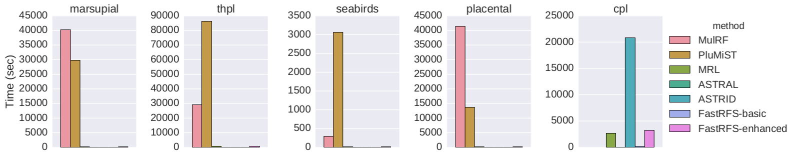


Fig. 3. Sequential running times (in seconds) on biological data of supertree methods. MulRF and PluMiST could not be run on the CPL dataset, due to its large size; hence no values are shown for those methods on that dataset.

FastRFS is fast and accurate

- Finds exact solutions to the RFS problem over a constrained search space
- Faster, more accurate than heuristic searches
- Choice of search space matters!
    - Adding bipartitions from other methods allows finding trees with better criterion score, and frequently better topological error