Using locality-sensitive hashing to speed up tree estimation and phylogenetic placement

dan brown

dan.brown@uwaterloo.ca

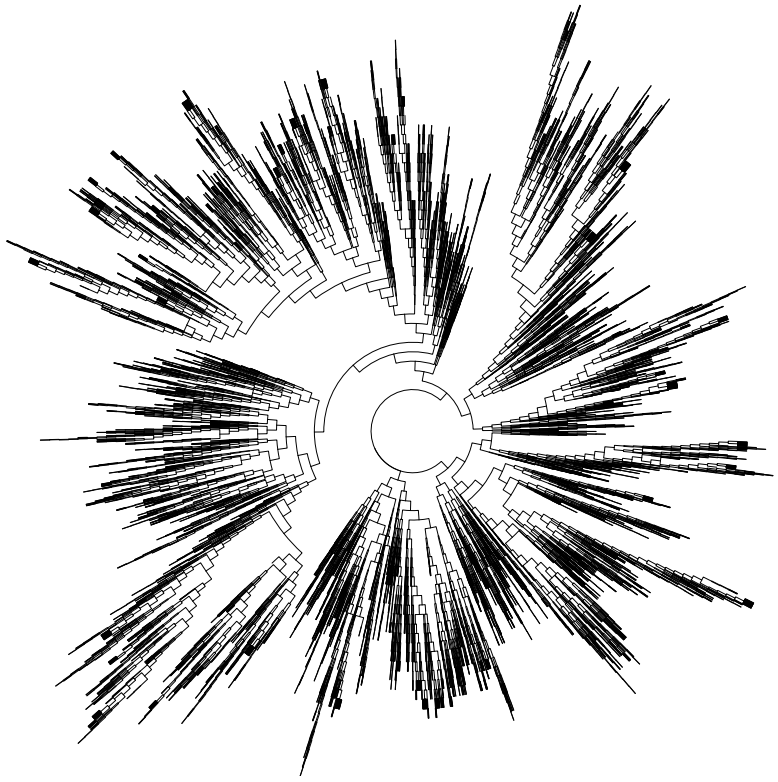
cheriton school of computer science

university of waterloo

joint work with jakub truszkowski

Core problem: speeding up phylogeny

- Trees with tens of thousands of leaves
- Too slow even for "fast" algorithms
- Can we do this in both in theory and practice?



General algorithm and important special case

- Phylogenetic inference
  - Given: Large multiple alignment, n species, Markov sequence model
  - Find: edge-weighted tree
  - Runtime faster than quadratic in n

Some possibilities:

- `FastTree` [Price *et al.*, 2009]
- Our previous program `QTree` [B + Truszkowski 2011]

No performance guarantees

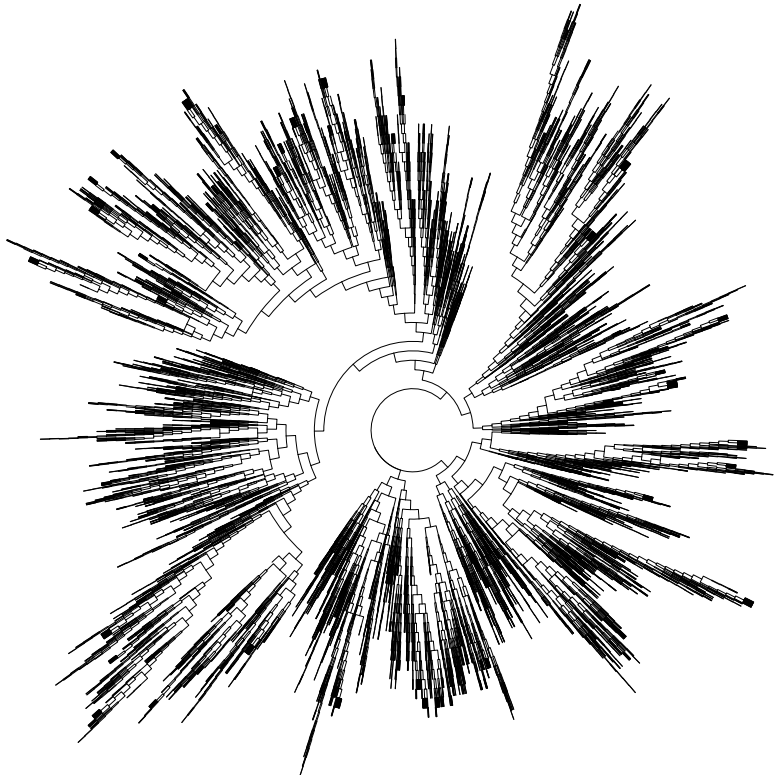
General algorithm and important special case

- Phylogenetic inference
  - Given: Large multiple alignment, n species, Markov sequence model
  - Find: edge-weighted tree
  - Runtime faster than quadratic in n

Some possibilities:

- `FastTree` [Price *et al.*, 2009]
- Our previous program `QTree` [B + Truszkowski 2011]

No performance guarantees

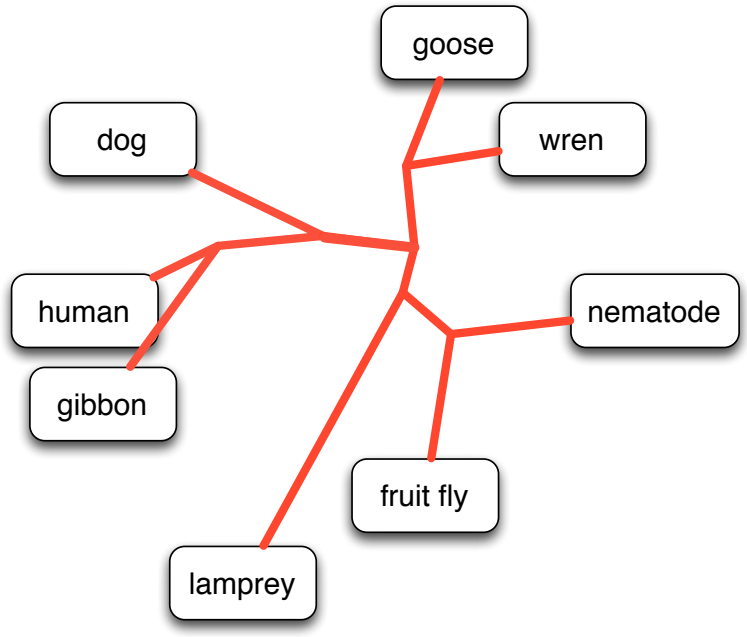


General algorithm and important special case

- Phylogenetic inference
- Phylogenetic placement
  - Given: Large multiple alignment M, tree T with n leaves
  - Plus: alignment of new sequence S to M
  - Place S into tree T
  - Faster than linear in n

Might have lots of new sequences

Example: add S = fox to this tree:



Start with special case

Why do phylogenetic placement?

- Too slow to build whole tree (lots of new sequences)
- New sequences (and alignment) less trustrworthy
- Consistency across experiments
- Real goal is something else (metagenomics, *etc.*)

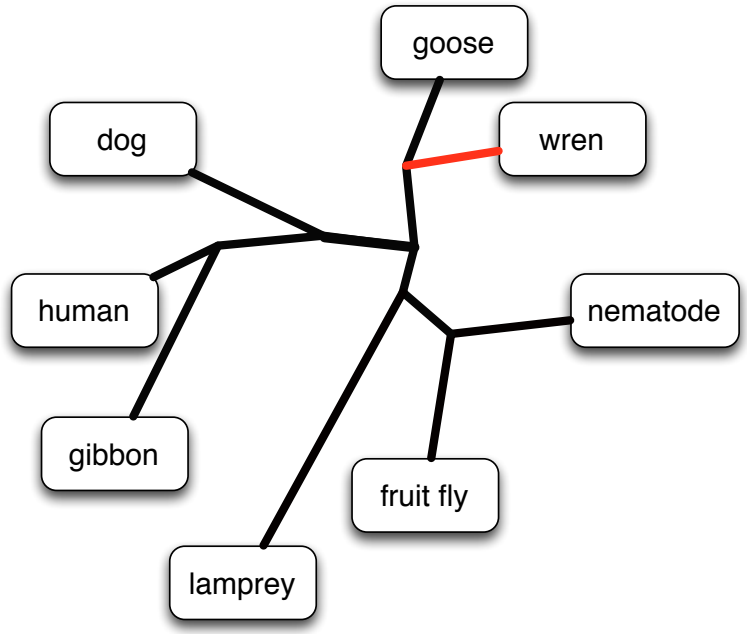
Slow way to do it

For each edge e in T:

- Find maximum likelihood if S were attached to e
- Return edge e^* of overall maximum likelihood

Linear in n: $2n - 3$ edges to examine

Example: add S = fox to this tree:



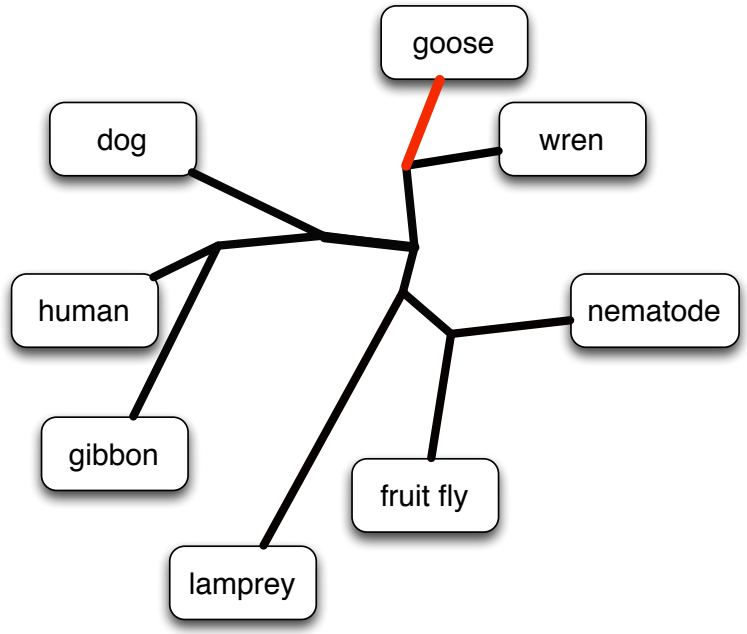
Slow way to do it

For each edge e in T:

- Find maximum likelihood if S were attached to e
- Return edge e^* of overall maximum likelihood

Linear in n: $2n - 3$ edges to examine

Example: add S = fox to this tree:



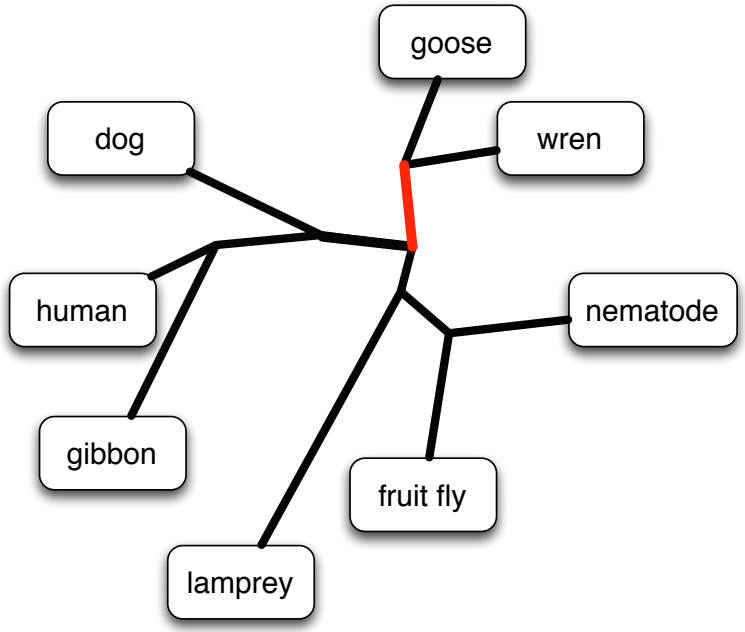
Slow way to do it

For each edge e in T:

- Find maximum likelihood if S were attached to e
- Return edge e^* of overall maximum likelihood

Linear in n: $2n - 3$ edges to examine

Example: add S = fox to this tree:



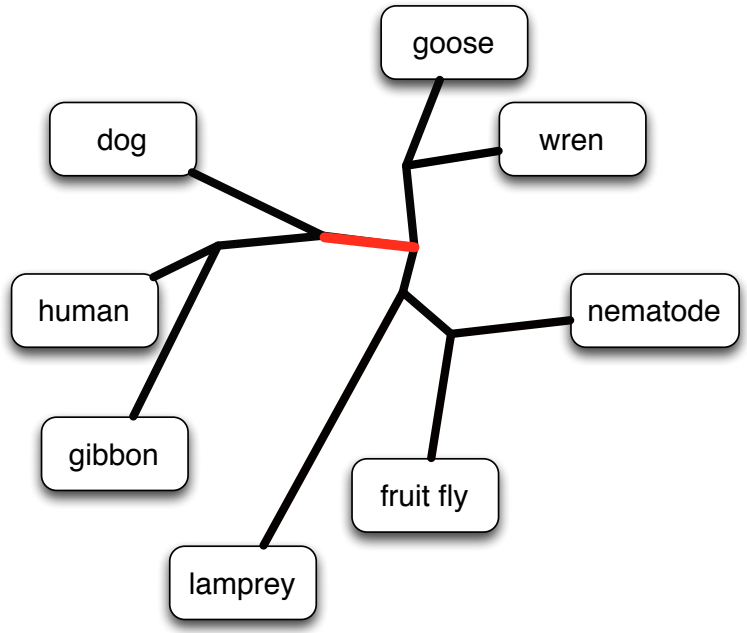
Slow way to do it

For each edge e in T:

- Find maximum likelihood if S were attached to e
- Return edge e^* of overall maximum likelihood

Linear in n: $2n - 3$ edges to examine

Example: add S = fox to this tree:



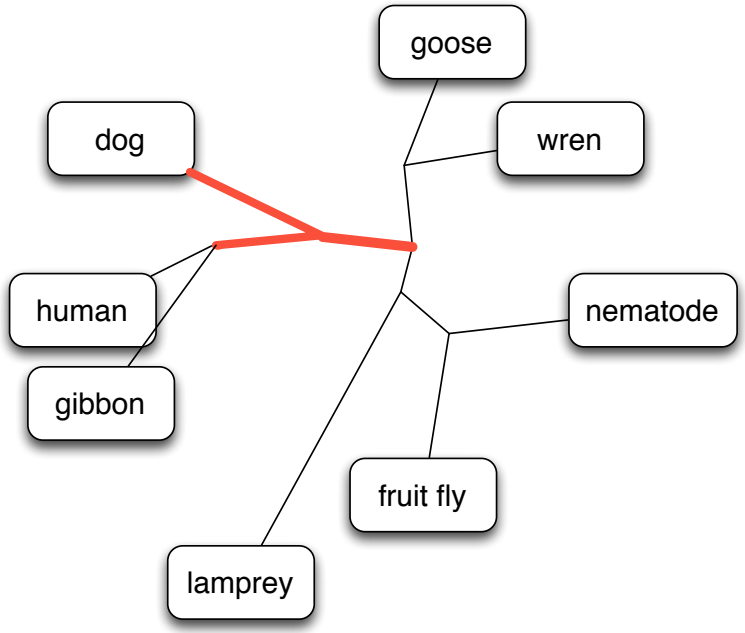
...

Idea: find a good part of the tree

Why look at the invertebrate part of the tree with fox sequence?

- Look for sequence in T similar to S
- Only explore its neighbourhood

Can we make this work?

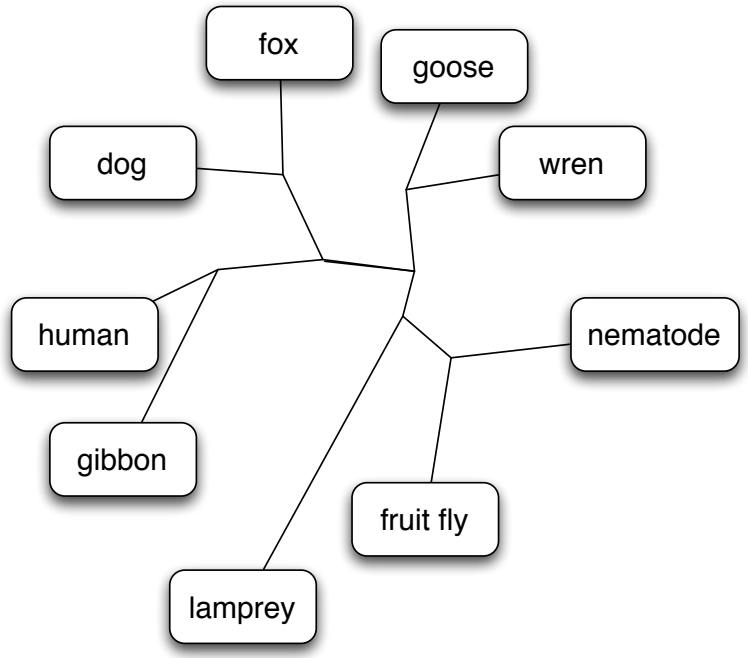


Idea: find a good part of the tree

Why look at the invertebrate part of the tree with fox sequence?

- Look for sequence in T similar to S
- Only explore its neighbourhood

Can we make this work?

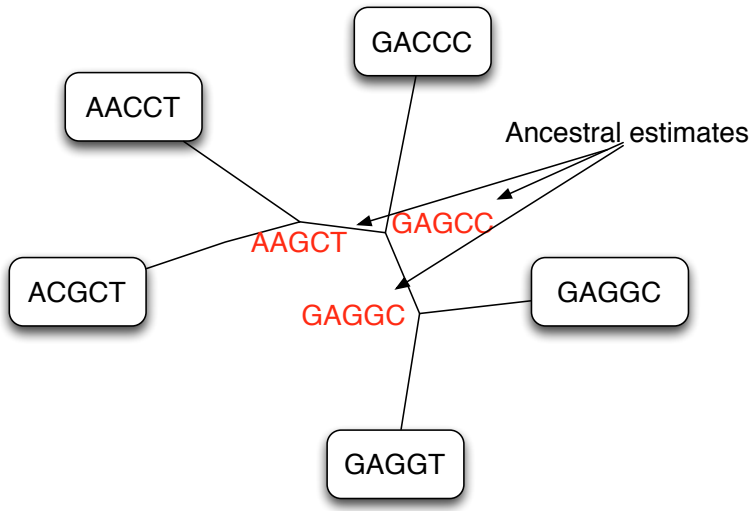


To localize new sequence S

1. Ancestral sequences
   - S may belong far from leaves
   - Would force search of whole tree

Theorem [Evans *et al.* 2000]: Can approximately infer ancestral sequences, given tree and edge lengths, regardless of n

- Can find near matches even by the root

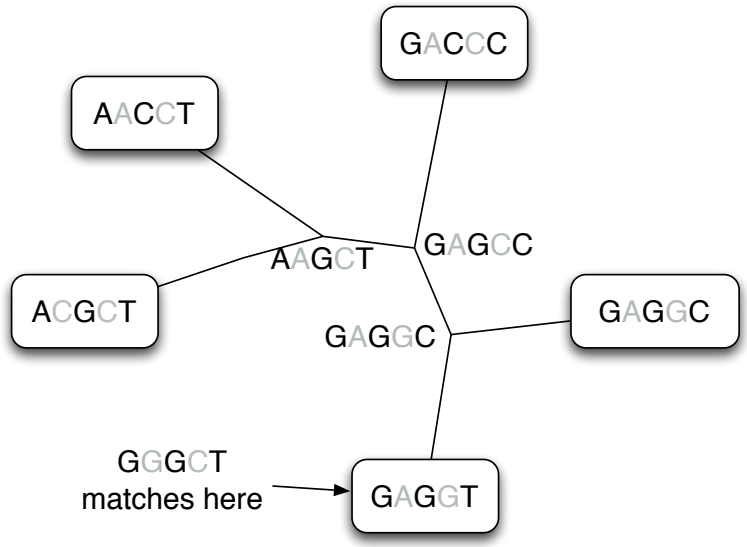


To localize new sequence S

1. Ancestral sequences
2. Search structure
   - Fast way of locating close matches to S
   - Doesn't generate lots of false positives

Theorem [Indyk + Motwani 1998]: Can find near neighbours to S in T with locality-sensitive hashing in $o(n)$ time

New sequence S = GGGCT
Consider positions $1, 3, 5$
Key is GGT



Locality-sensitive hashing

- Idea: if S is close to a sequence in the tree, they probably match in a random set of k positions.
- Build q hash tables, each from $O(\log n)$ randomly chosen positions
- Hash table hit: good neighbourhood to explore

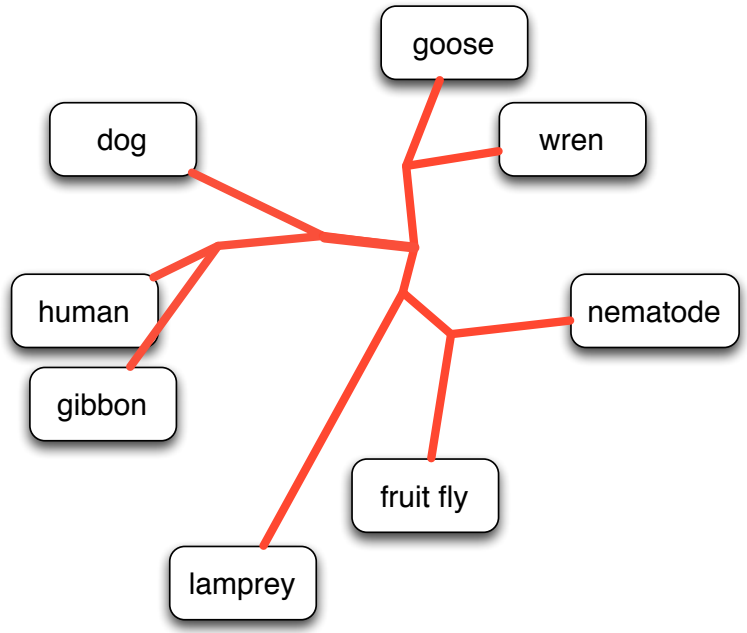
How many hash tables?

Max. length of tree edge	0.01	0.02	0.05	0.075	0.10
Number of tables	$n^{0.05}$	$n^{0.10}$	$n^{0.27}$	$n^{0.43}$	$n^{0.61}$

(In practice 4 or 8 work pretty well...)

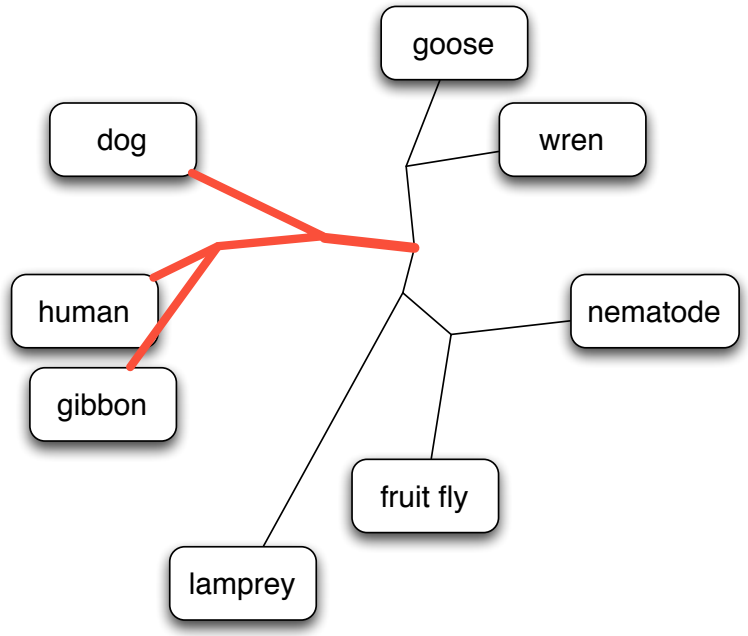
LSHplace

- Build hash tables from random columns of alignment
- For each sequence S:
  - Localize S using hash tables
  - Shrink neighbourhood with distance queries
  - In tiny region, use likelihood to find best place



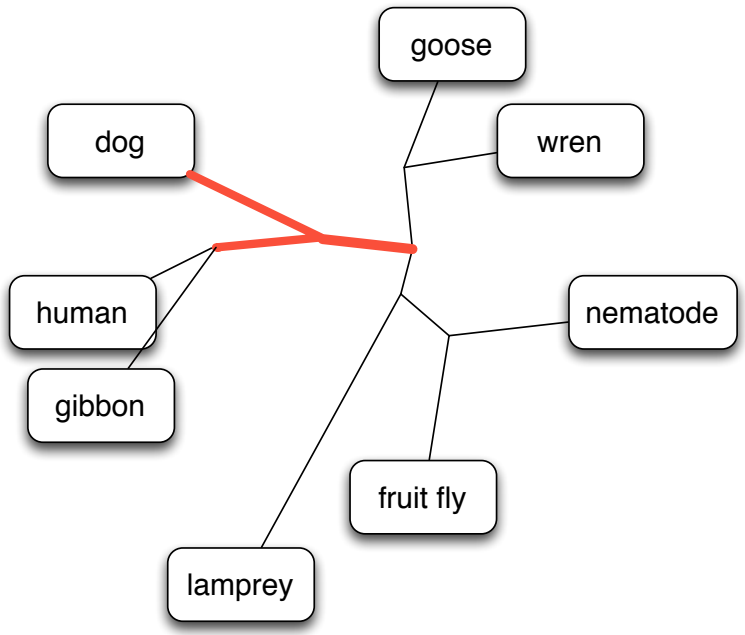
LSHplace

- Build hash tables from random columns of alignment
- For each sequence S:
  - Localize S using hash tables
  - Shrink neighbourhood with distance queries
  - In tiny region, use likelihood to find best place



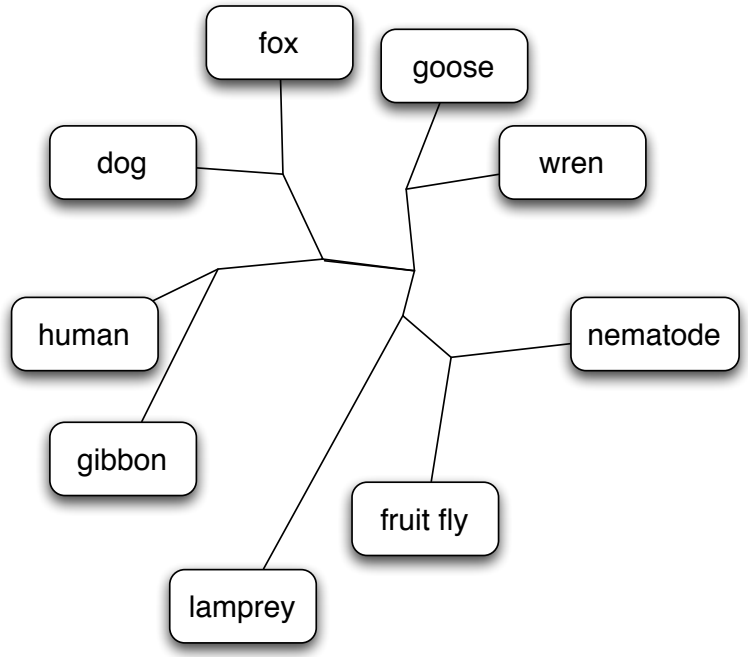
LSHplace

- Build hash tables from random columns of alignment
- For each sequence S:
  - Localize S using hash tables
  - Shrink neighbourhood with distance queries
  - In tiny region, use likelihood to find best place



LSHplace

- Build hash tables from random columns of alignment
- For each sequence S:
  - Localize S using hash tables
  - Shrink neighbourhood with distance queries
  - In tiny region, use likelihood to find best place



Results

Errors:

- Topological distance: how many edges away from the correct edge?
- Evolutionary distance: accounting for edge lengths

Simulated data (so we know the right answer), 100,000 new sequences:

Tree size	1000	5000	10,000
Run time	35 min	41 min	49 min
Correct edge	73%	68%	63%
Average topol. dist	.45 edges	.52 edges	.65 edges
Average evol. dist	.013	.008	.007

2-3 million sequences a day on a desktop; maximum tree size probably 20-40,000 sequences

Summary of results

Versus `pplacer` [Matsen *et al.* 2010]:

- 8-25 times faster
- 6% fewer reads placed exactly correctly

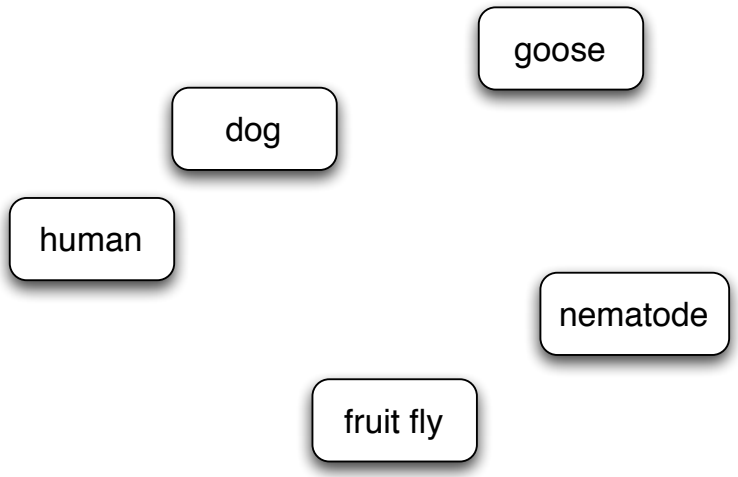
Status

- Preliminary work (`LSHplace` 1.0) published at PSB 2013
- These results are unpublished: `LSHplace` 2.0
- Current work: speed, accuracy, memory footprint

General phylogenetic algorithm

Framework

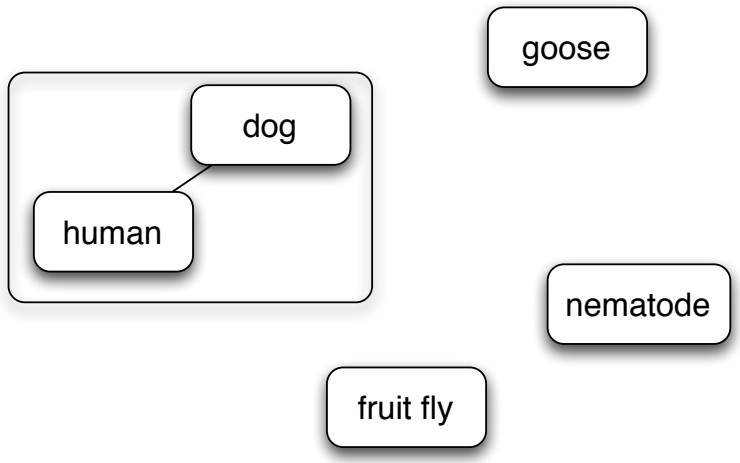
- Start with each taxon in its own tree
- Until only one tree left:
  - Find close tree edges or nodes (using LSH) and join them
  - Infer ancestral sequences at new internal nodes



General phylogenetic algorithm

Framework

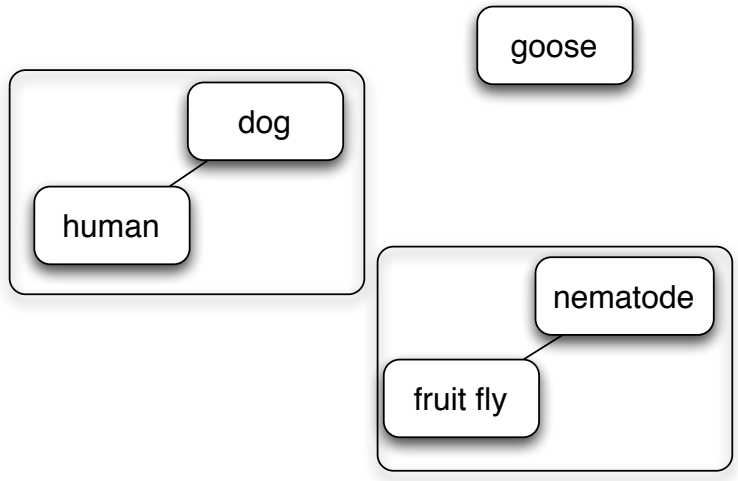
- Start with each taxon in its own tree
- Until only one tree left:
  - Find close tree edges or nodes (using LSH) and join them
  - Infer ancestral sequences at new internal nodes



General phylogenetic algorithm

Framework

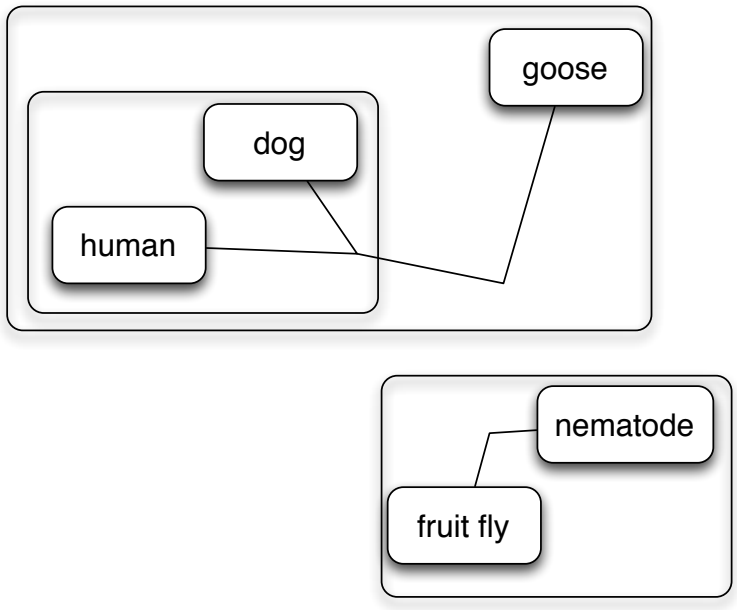
- Start with each taxon in its own tree
- Until only one tree left:
  - Find close tree edges or nodes (using LSH) and join them
  - Infer ancestral sequences at new internal nodes



General phylogenetic algorithm

Framework

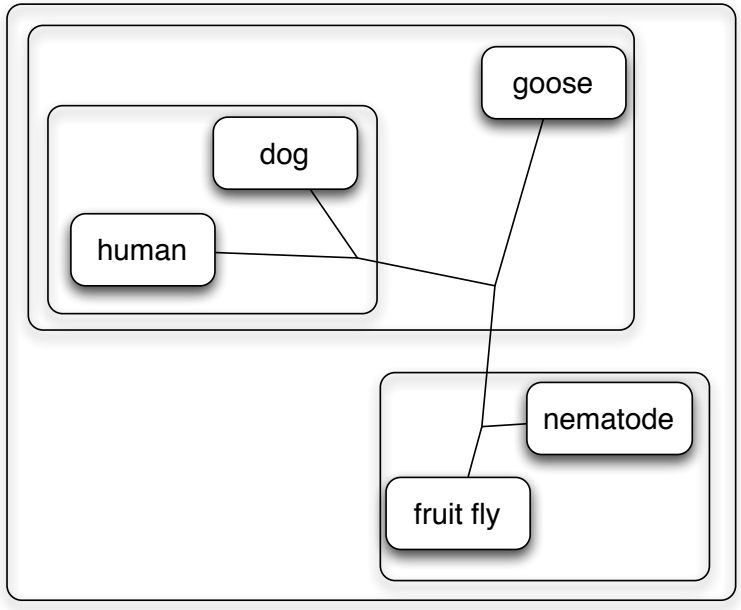
- Start with each taxon in its own tree
- Until only one tree left:
  - Find close tree edges or nodes (using LSH) and join them
  - Infer ancestral sequences at new internal nodes



General phylogenetic algorithm

Framework

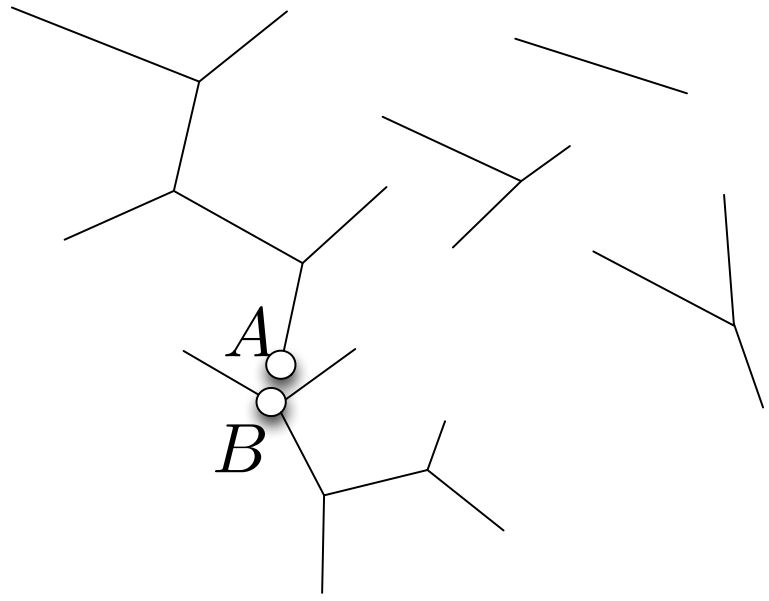
- Start with each taxon in its own tree
- Until only one tree left:
  - Find close tree edges or nodes (using LSH) and join them
  - Infer ancestral sequences at new internal nodes



A bit more detail

While forest has more than one tree:

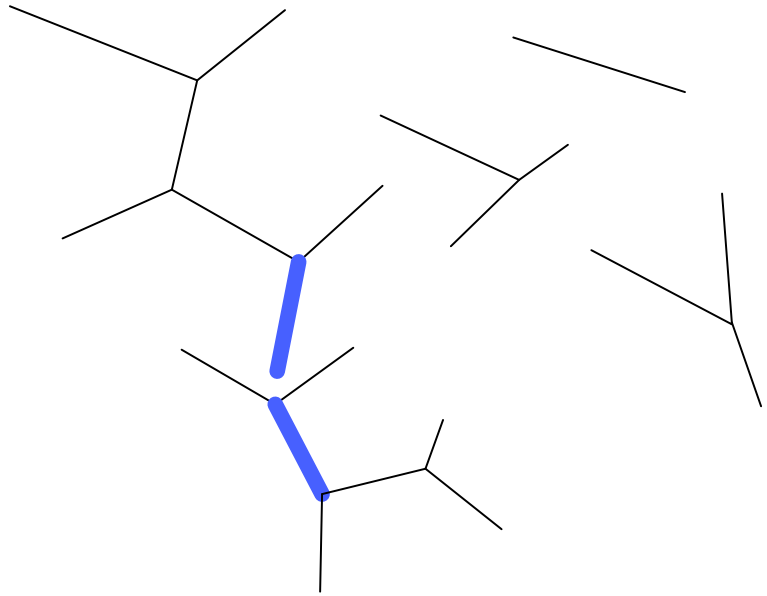
- Find close nodes A and B in different trees (using LSH)



A bit more detail

While forest has more than one tree:

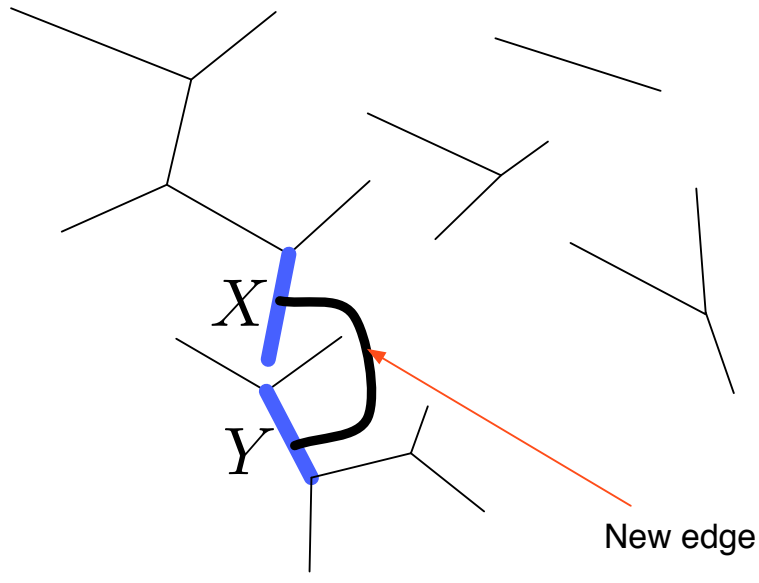
- Find close nodes A and B in different trees (using LSH)
- Identify nearby edges to join



A bit more detail

While forest has more than one tree:

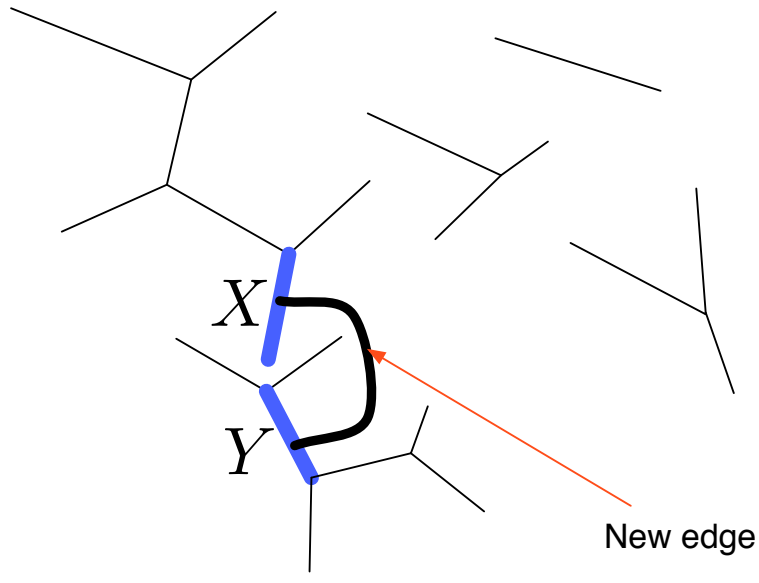
- Find close nodes A and B in different trees (using LSH)
- Identify nearby edges to join
- Create nodes X and Y in the middle of those edges, joined with a new edge



A bit more detail

While forest has more than one tree:

- Find close nodes A and B in different trees (using LSH)
- Identify nearby edges to join
- Create nodes X and Y in the middle of those edges, joined with a new edge
- Reconstruct ancestral sequences at X and Y



Sweeping under the rug

- Can we always find a pair of nodes to join?
  - Yes; the theory is frighteningly complex, but there always is a good pair to join
- What other caveats?
  - All tree lengths below constant g
  - Alignment has to be right
  - Markov model of evolution (In our theorem, sequences are binary; this restriction is not required)

After the sweeping

Theorem [B + Truszkowski 2011]: If all tree lengths less than a constant $g < .15$, our algorithm finds the correct tree, with high probability, in time $O(f(g))$ time, where $f(g) = o(n^2)$.

Upper bound g	0.01	0.02	0.05	0.075	0.10
$f(g)$	$n^{1.11}$	$n^{1.20}$	$n^{1.48}$	$n^{1.68}$	$n^{1.86}$

First $o(n^2)$-runtime algorithm with provably good theoretical performance on $O(\log n)$-length sequences.

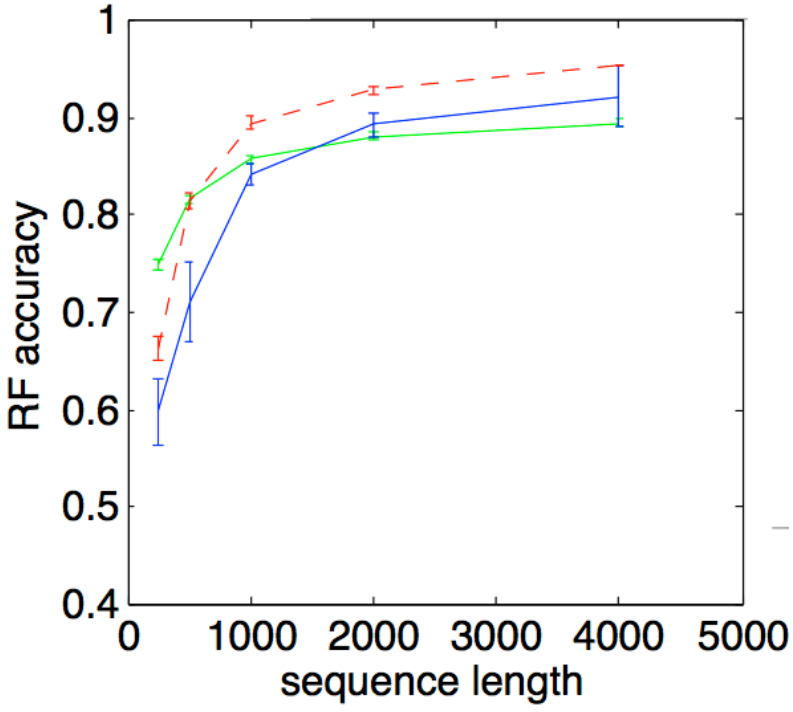
Can this be practical?

- `LSHtree`: very fast software for phylogenetic reconstruction
  - Small number of hash tables, instead of growing with upper limit on edge length (presumably unknown!)
  - Build new hash tables when not enough collisions
  - NN interchanges after each join to fix errors in pair of joined edges
  - Other simplifications of complex theoretical algorithm
- Prototype implementation, probably less stable than `LSHplace`.
- Experimental data:
  - Tree topology simulated using pure-birth process
  - Varying edge lengths: mean branch lengths range from .03 to .25
  - 2000 taxa
  - Varying sequence lengths: 500 to 4000 basepairs

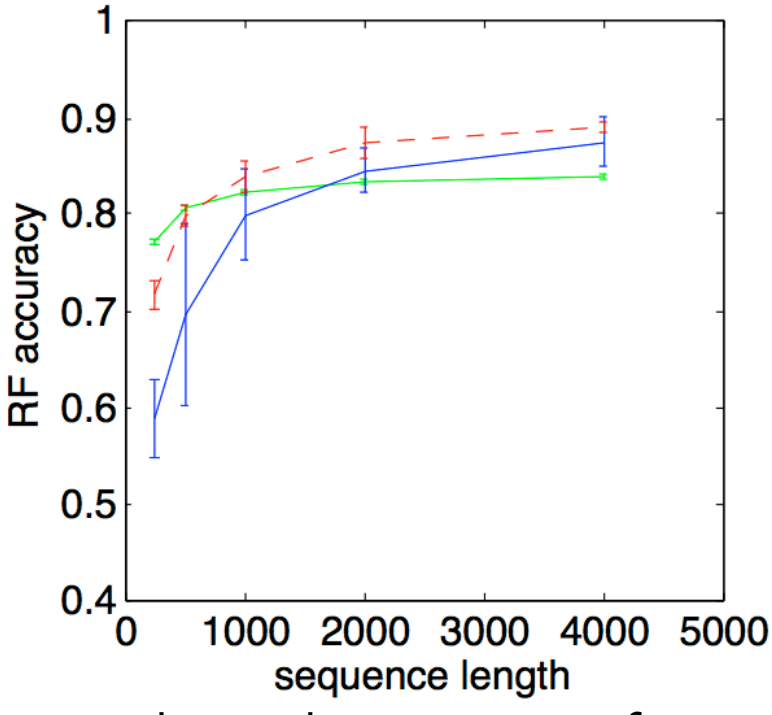
Results

LSHtree; Qtree (our previous work); NJ phase of FastTree

Short edges (mean is 0.03):



Medium edges (mean is 0.125):

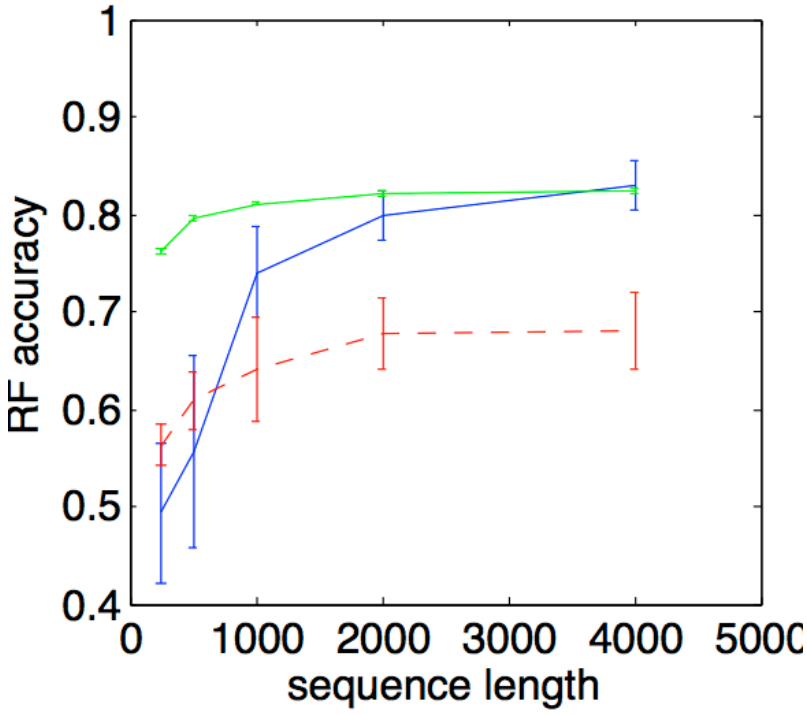


For these edge lengths, results slightly better than other programs for reasonably large sequences

Results

LSHtree; Qtree (our previous work); NJ phase of FastTree

Long edges (mean is .25):



For trees with long edges, LSHtree can't join nodes well: ancestral reconstructions are junk

Results

- Summary
  - 1.5-2.5 times faster than `FastTree`'s NJ phase
  - Accuracy comparable between both programs; `LSHtree` may be a bit better for short edges, and requires slightly longer edges to become successful
- Next phases
  - Improve speed and memory footprint
  - Make algorithm more tolerant of long edges, bad alignments, *etc.*
  - Incorporate local search

Conclusions

- `LSHplace` (PSB 2013): phylogenetic placement software 8-25 x faster than `pplacer`, not much less accurate
- `LSHtree` (WABI 2012): phylogenetic reconstruction software faster than `FastTree`, roughly as accurate
- Theoretical basis for both programs is robust and complex
- More work in progress for both systems!

Acknowledgments:

- Josh Neufeld and Andre Masella, U of Waterloo Biology
- Erick Matsen, Hutchinson Cancer Research Centre
- Funding from NSERC and Government of Ontario