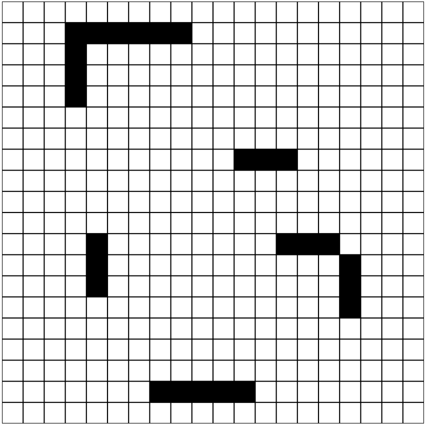
Pathfinding with Trees

Samuel Bader <s.bader@unibas.ch>

DMI, University of Basel

13. Sep. 2016

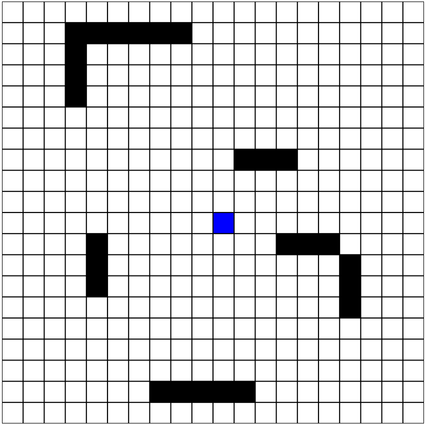
Pathfinding



We are given

- a map

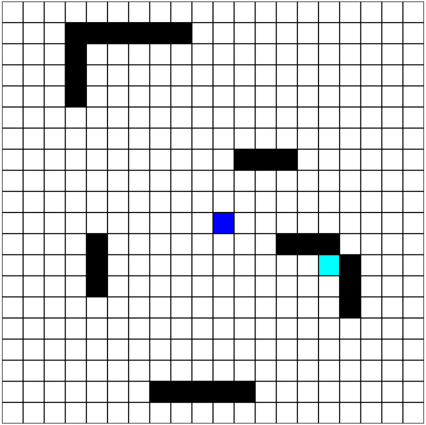
Pathfinding



We are given

- a map
- a start point

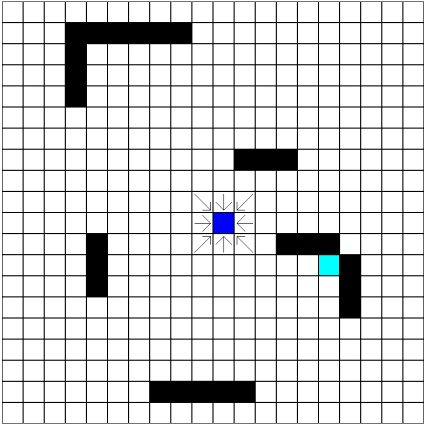
Pathfinding



We are given

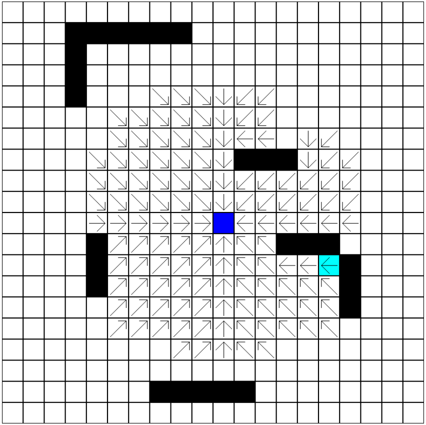
- a map
- a start point
- a goal point

Dijkstra's Algorithm



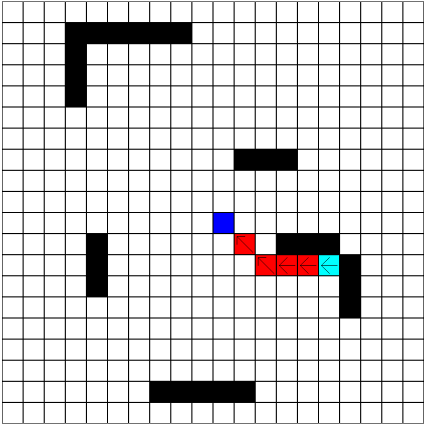
A simple solution:
Look at all points
neighbouring the start point

Dijkstra's Algorithm



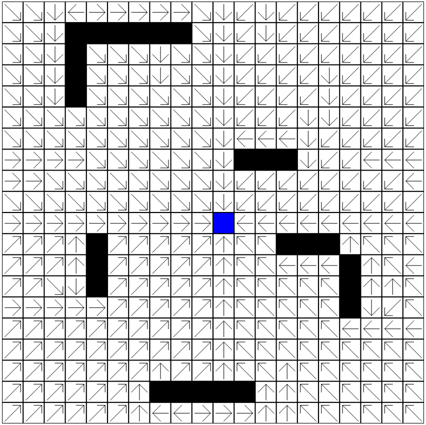
A simple solution:
Look at all points neighbouring the start point
Continue until the goal is found

Dijkstra's Algorithm



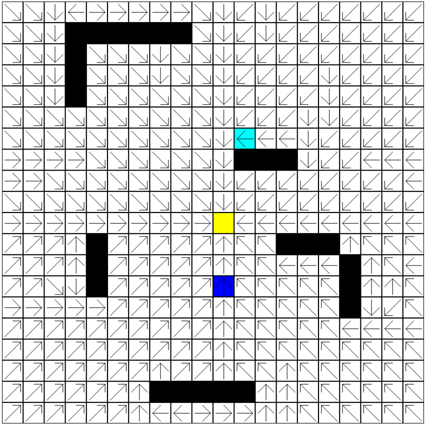
A simple solution:
Look at all points neighbouring the start point
Continue until the goal is found
Follow the arrows back to the start

Dijkstra's Algorithm -> Tree Cache



Save the search tree for a single root

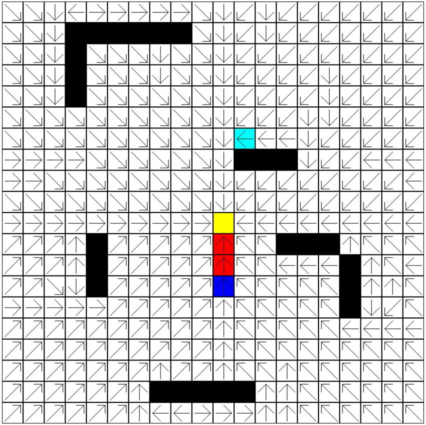
Dijkstra's Algorithm -> Tree Cache



Save the search tree for a single root

When searching:

Dijkstra's Algorithm -> Tree Cache

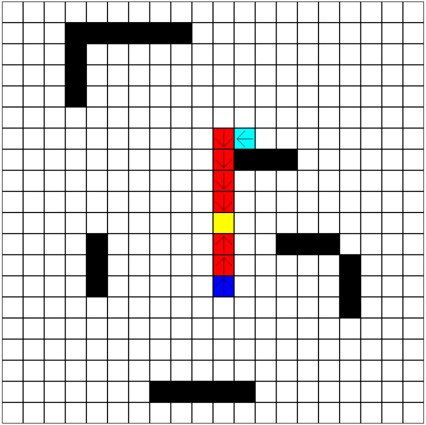


Save the search tree for a single root

When searching:

- traverse the tree from start to root

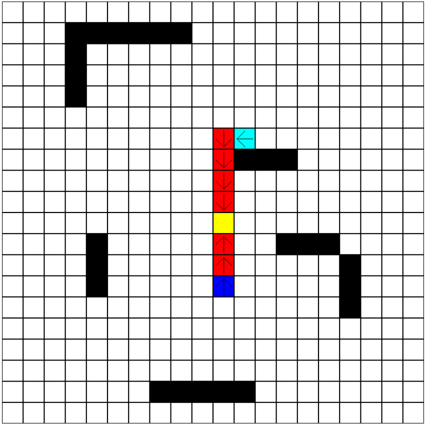
Dijkstra's Algorithm -> Tree Cache



Save the search tree for a single root
When searching:

- traverse the tree from start to root
- traverse the tree from goal to root

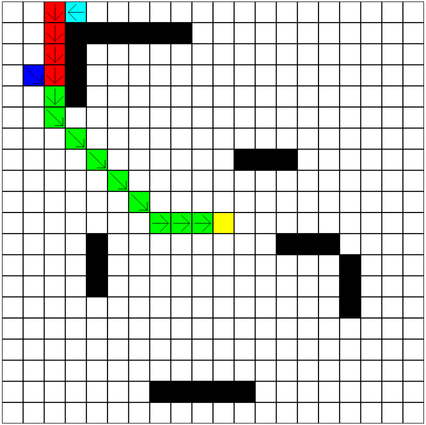
Dijkstra's Algorithm -> Tree Cache



Save the search tree for a single root
When searching:

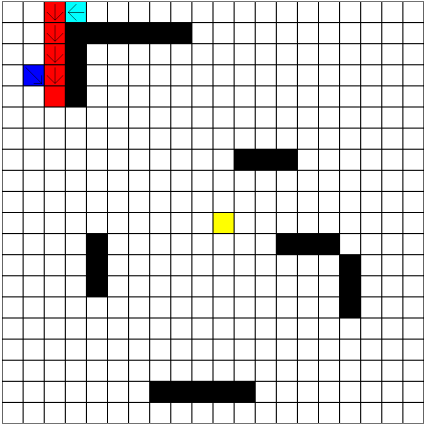
- traverse the tree from start to root
- traverse the tree from goal to root
- invert the second part
- concatenate parts at the root

Tree Cache: Example 2



Paths can have redundant parts

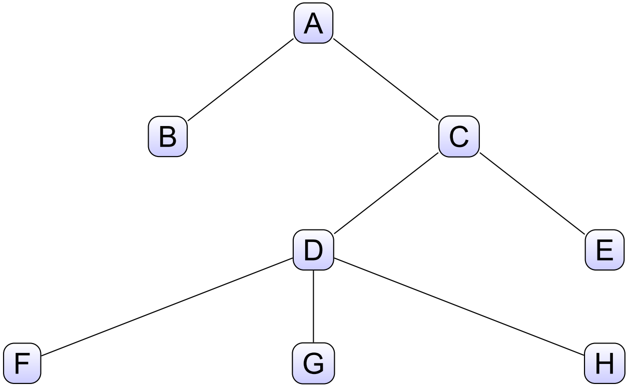
Tree Cache: Example 2



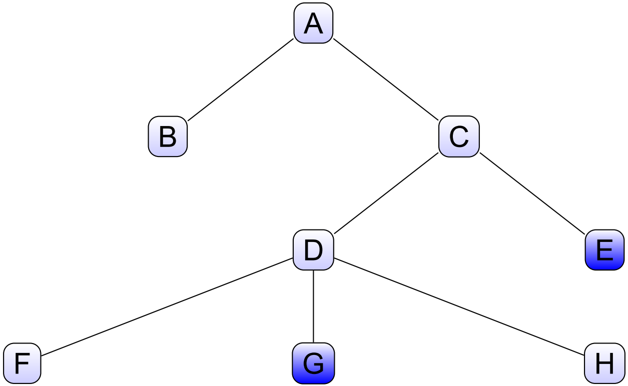
Paths can have redundant parts

The redundant part can be easily removed by looking for the lowest common ancestor (LCA) of the two nodes

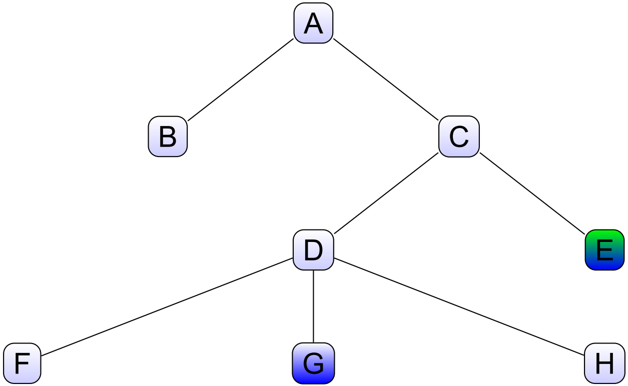
LCA: Naive implementation



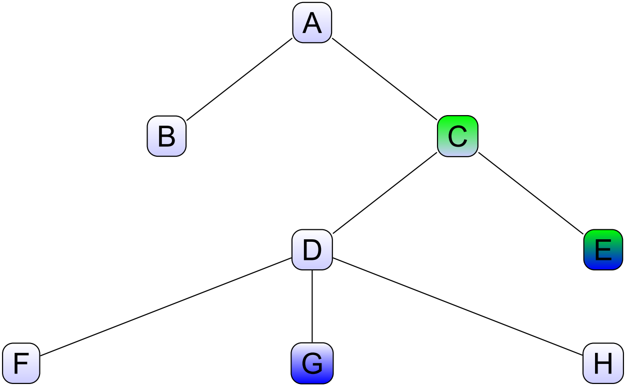
LCA: Naive implementation



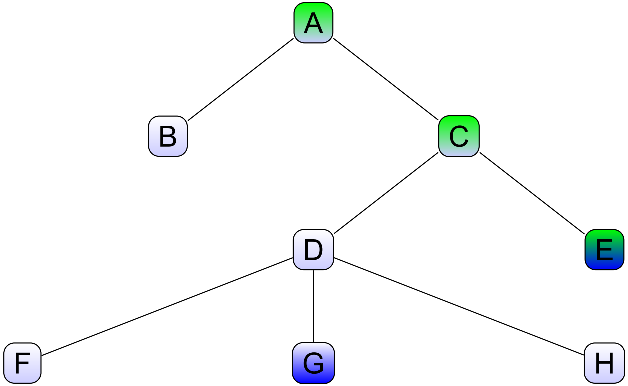
LCA: Naive implementation



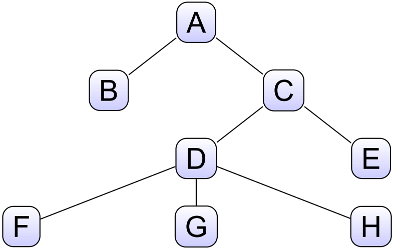
LCA: Naive implementation



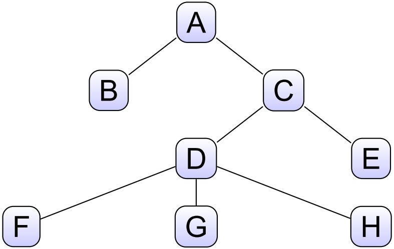
LCA: Naive implementation



LCA: Faster implementation



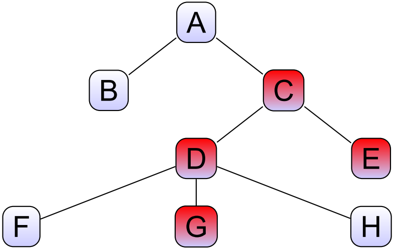
LCA: Faster implementation



A depth-first traversal and corresponding levels:

A	B	A	C	D	F	D	G	D	H	D	C	E	C	A
0	1	0	1	2	3	2	3	2	3	2	1	2	1	0

LCA: Faster implementation



A depth-first traversal and corresponding levels:

A	B	A	C	D	F	D	G	D	H	D	C	E	C	A
0	1	0	1	2	3	2	3	2	3	2	1	2	1	0

Range minimum query implementation

A	B	A	C	D	F	D	G	D	H	D	C	E	C	A
0	1	0	1	2	3	2	3	2	3	2	1	2	1	0

Range minimum query implementation

A	B	A	C	D	F	D	G	D	H	D	C	E	C	A
0	1	0	1	2	3	2	3	2	3	2	1	2	1	0

The range minimum query can be solved with a few lookups by pre-calculating the minimum for all sub-ranges of length 2^i:

A	A	A	C	D	D	D	D	D	D	C	C	C	A	2
A	A	A	C	D	D	D	D	C	C	C	A			4
A	A	A	C	C	C	C	A							8

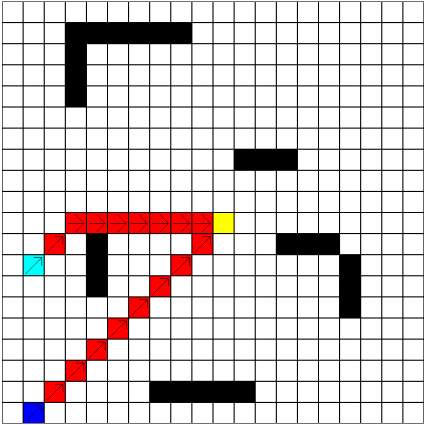
Range minimum query implementation

A	B	A	C	D	F	D	G	D	H	D	C	E	C	A
0	1	0	1	2	3	2	3	2	3	2	1	2	1	0

The range minimum query can be solved with a few lookups by pre-calculating the minimum for all sub-ranges of length 2^i:

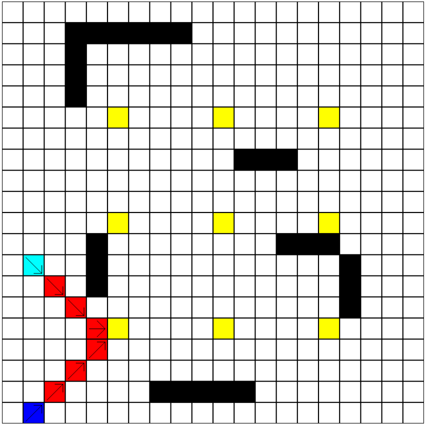
A	A	A	C	D	D	D	D	D	D	C	C	C	A	2
A	A	A	C	D	D	D	D	C	C	C	A			4
A	A	A	C	C	C	C	A							8

Tree Cache: Example 3



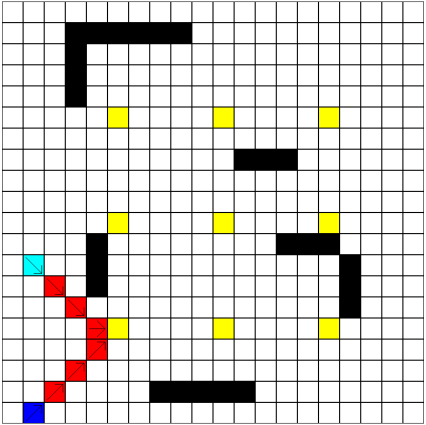
Paths can be bad without redundant parts

Tree Cache improvement: More than one tree



Generate more than one tree, store distance as well as parent

Tree Cache improvement: More than one tree

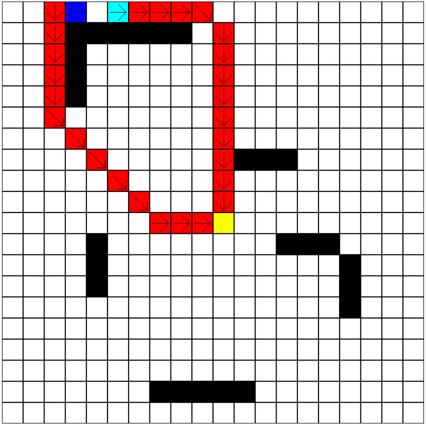


Generate more than one tree, store distance as well as parent

When searching:

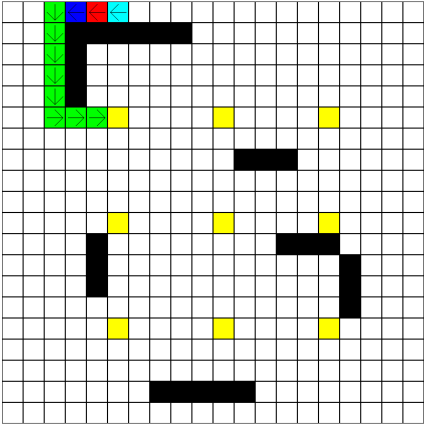
- calculate path distance for each tree
- choose tree with shortest path
- construct path

Tree Cache: Example 4



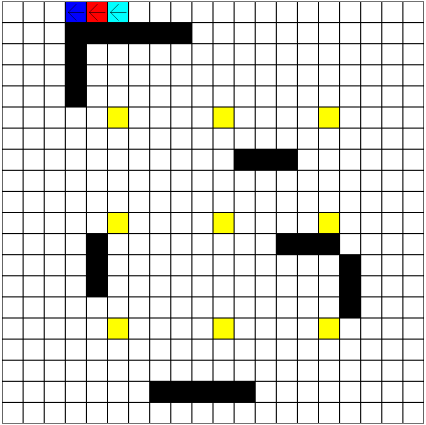
Tree Cache generated path far too long without redundancy

Using Multiple Trees



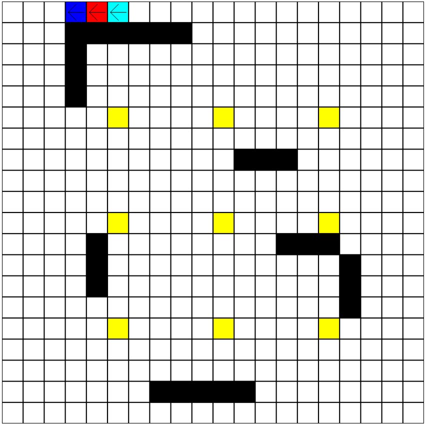
Using multiple trees generates better path with redundancy

Both improvements combined



Generate more than one tree, store distance as well as parent, calculate LCA information for each tree

Both improvements combined

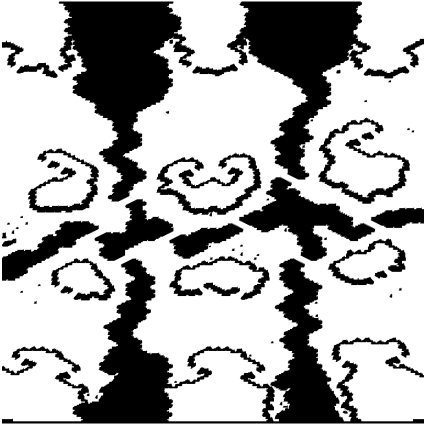


Generate more than one tree, store distance as well as parent, calculate LCA information for each tree

When searching:

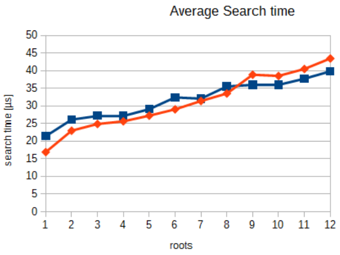
- look up LCA and calculate path distance for each tree
- choose tree with shortest path
- construct path

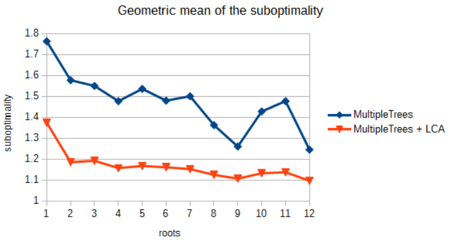
Experiments



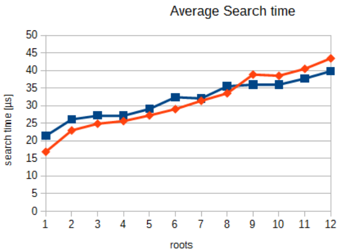
Maps from the Gridbased Path Planning Competition
Most maps 512x512 tiles, the largest 768x768

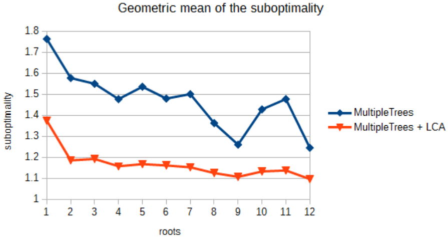
Results


Average Search time
search time [µs]
50
45
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12
roots


Geometric mean of the suboptimality
suboptimality
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
1 2 3 4 5 6 7 8 9 10 11 12
roots
MultipleTrees
MultipleTrees + LCA

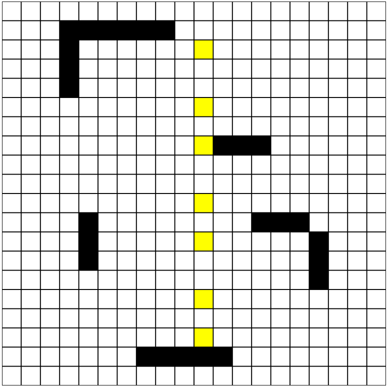
Results

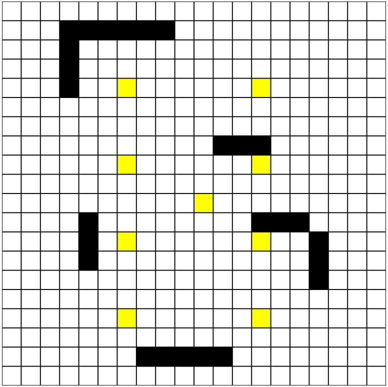




map size:768x768	time per tree	memory per tree
Trees	230 ms	10 MB
Trees + LCA	400 ms	110 MB

Results: Root placement





Conclusions

- Tree Cache finds potentially bad paths very fast

Conclusions

- Tree Cache finds potentially bad paths very fast
- Path quality can be improved significantly

Conclusions

- Tree Cache finds potentially bad paths very fast
- Path quality can be improved significantly
- Combination of improvements big amounts of memory

Conclusions

- Tree Cache finds potentially bad paths very fast
- Path quality can be improved significantly
- Combination of improvements big amounts of memory
- Good results using only a few trees

Future Work

- advanced root placement strategies

Future Work

- advanced root placement strategies
- compress tree information

Future Work

- advanced root placement strategies
- compress tree information
- adapt to directed graphs