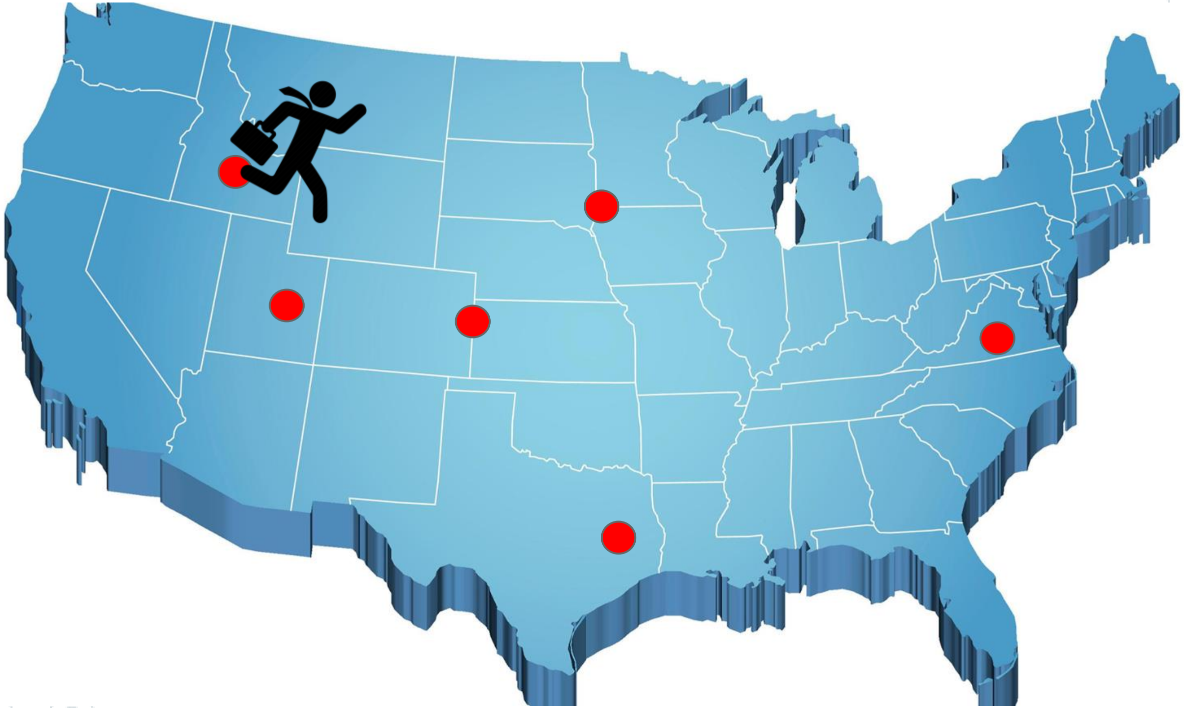
Travelling Salesman Problem Parallelisation

Group 26: Chang Kon Han, John Law, Wesley Yep

Outline

- How can the A* algorithm be used to solve TSP?
- How can the parallelising be done with ParaTask?
- What tweaks can be done to improve performance and scalability?
- How does the parallel implementation compare to the sequential version?

Travelling Salesman Problem



Greedy Approach

Optimal Solution

(6) Travel salesman problem TSP

NP-hard

Travel salesman problem (TSP) is another ~~NP-complete~~ problem.

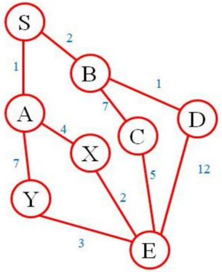
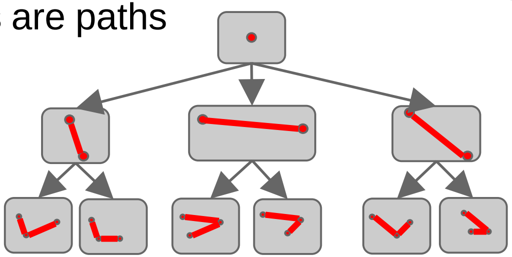
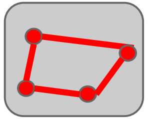
- MST can't solve TSP as the cycle has not to be completed (TSP is a cycle)
- Salesman needs to go home (whatÕs the cost going home?)
- We can use MST to approximate the solution
- As solving TSP by exhaustive approach is not a best choice

Slide borrowed from Mark Wilson, Softeng 250

A* Algorithm

- Tree Search Algorithm
- Heuristic Search
- Best First Search Algorithm
- $f(n) = g(n) + h(n)$

A* for TSP

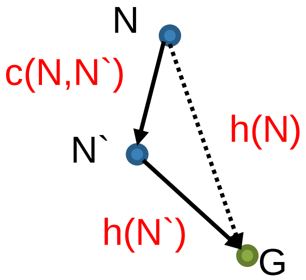
A* for Path Finding	A* for TSP
States are cities 	States are paths 
Goal state is a single city 	Goal state is a Hamiltonian Cycle 
Neighbours are all the cities that are connected to a city	Neighbours are all graphs that can be formed by the addition of one edge from an existing state

Heuristic

Admissible

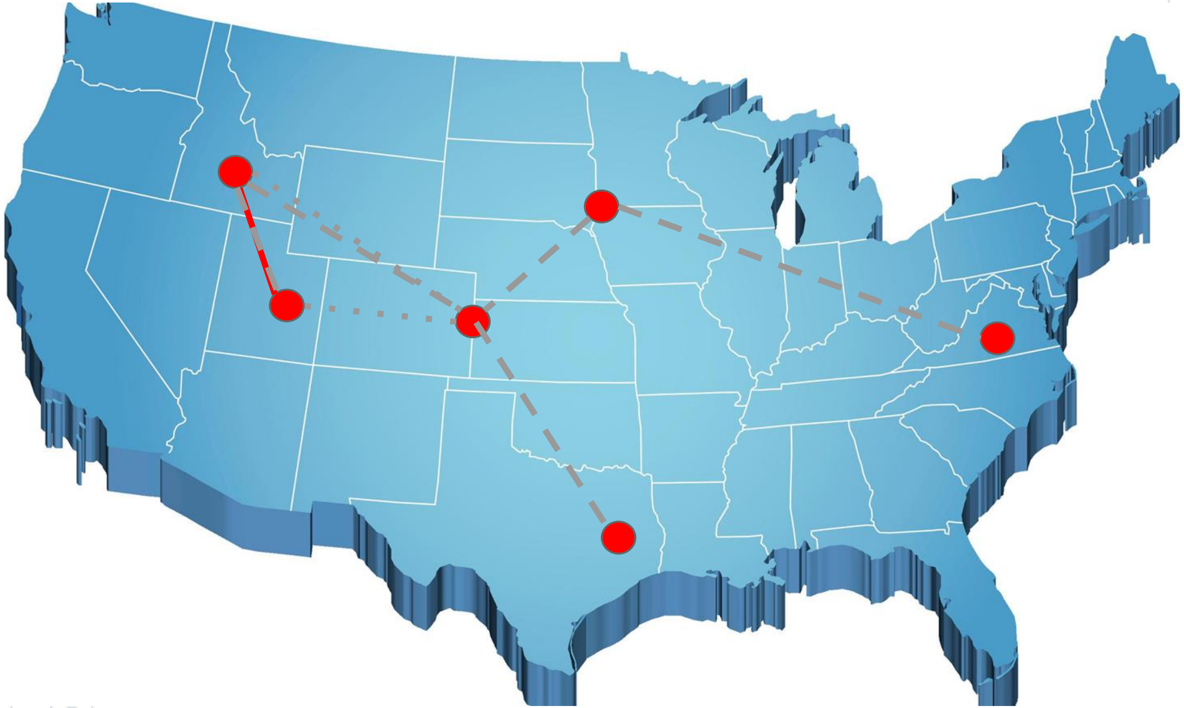
◎ Never overestimates cost to the goal

Consistent

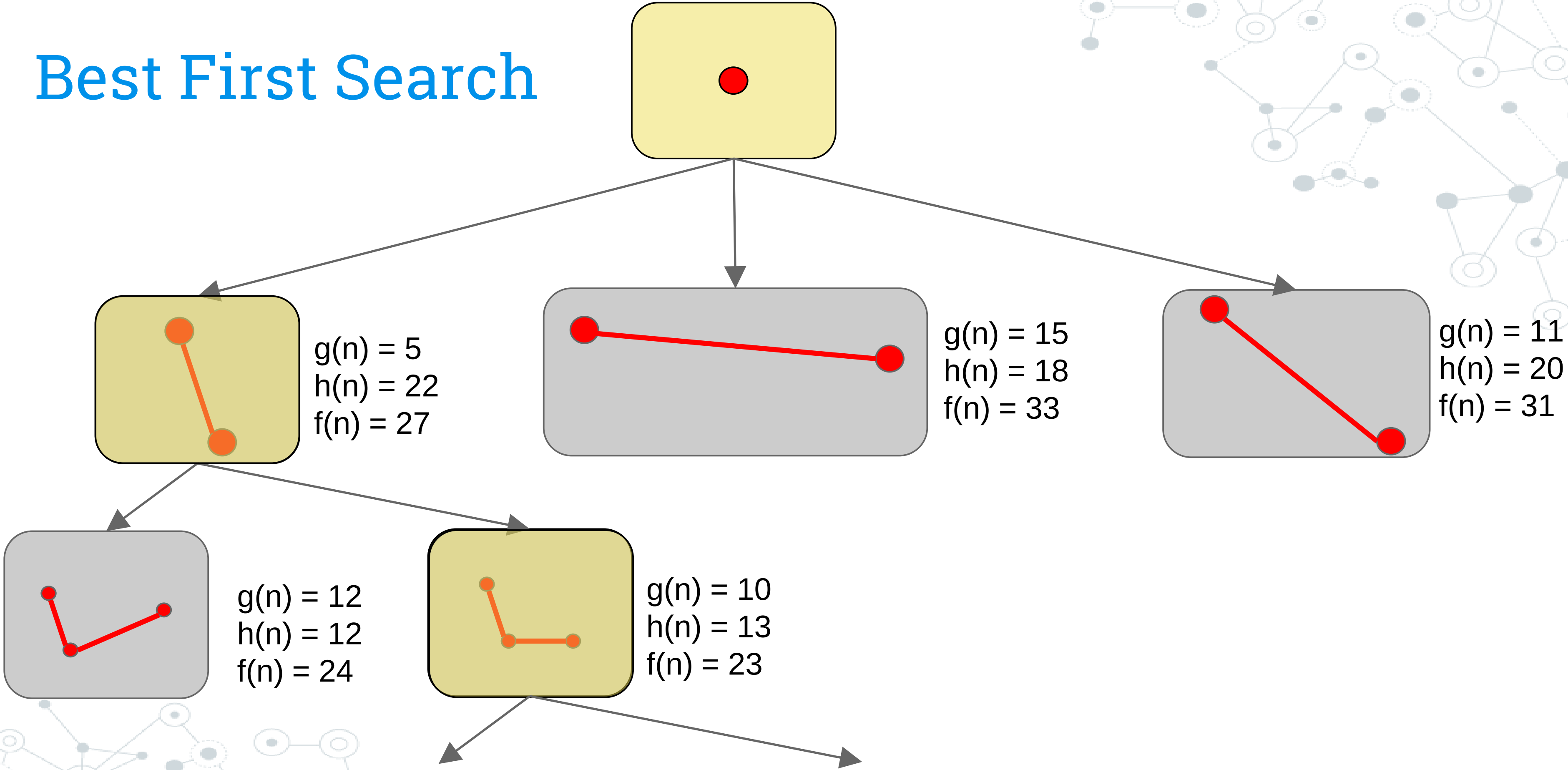


◎ h(N) <= c(N,N`) + h(N`)

Minimum Spanning Tree (heuristic)



Best First Search



A* for TSP: Pseudocode

Algorithm 1: A* for TSP

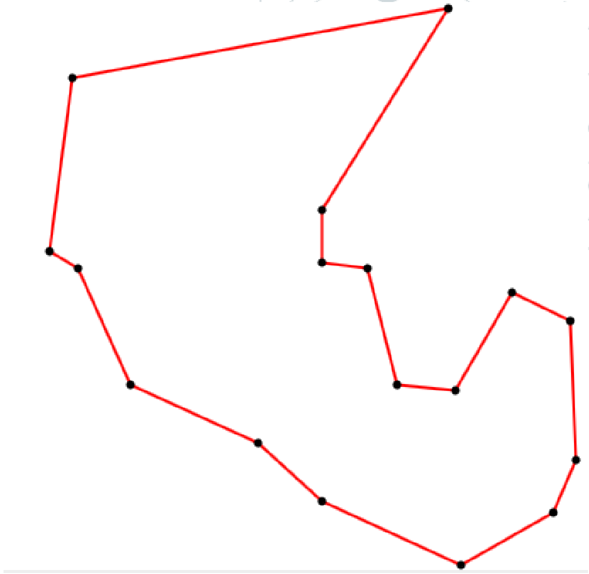
```
s0 ← state(initialCity);
frontier ← priorityqueue();
frontier.add(s0);
s ← s0;
while s not hamiltonian cycle do
    children ← s.expand();
    foreach child in children do
        frontier.add(child, childValue);
    end
    s ← frontier.poll();
end
s ← s + initialCity;
return s;
```

```
NAME : Test17
COMMENT : 17 cities
TYPE : TSP
DIMENSION : 17
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 21 1
2 54 87
3 14 45
4 34 78
5 12 5
6 50 12
7 34 32
8 54 37
9 64 45
10 55 45
11 3 21
12 45 2
13 57 92
14 24 56
15 87 88
16 33 22
17 99 23
EOF
```

Libraries

TSPLIB

TSPLIB4J



http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
https://github.com/dhadka/TSPLIB4J

Parallel A*

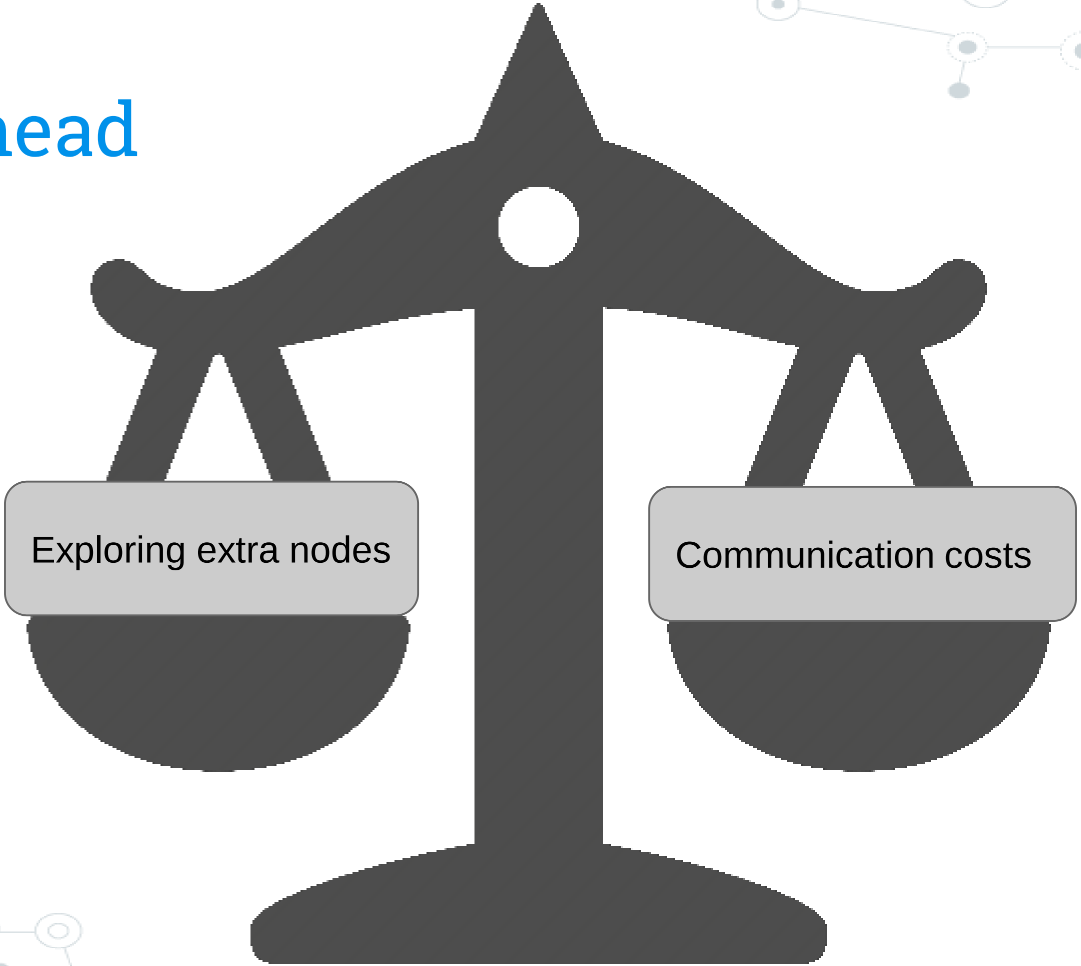
Task Parallelism

◎Difficult due to the sequential nature of expanding node

Data Parallelism

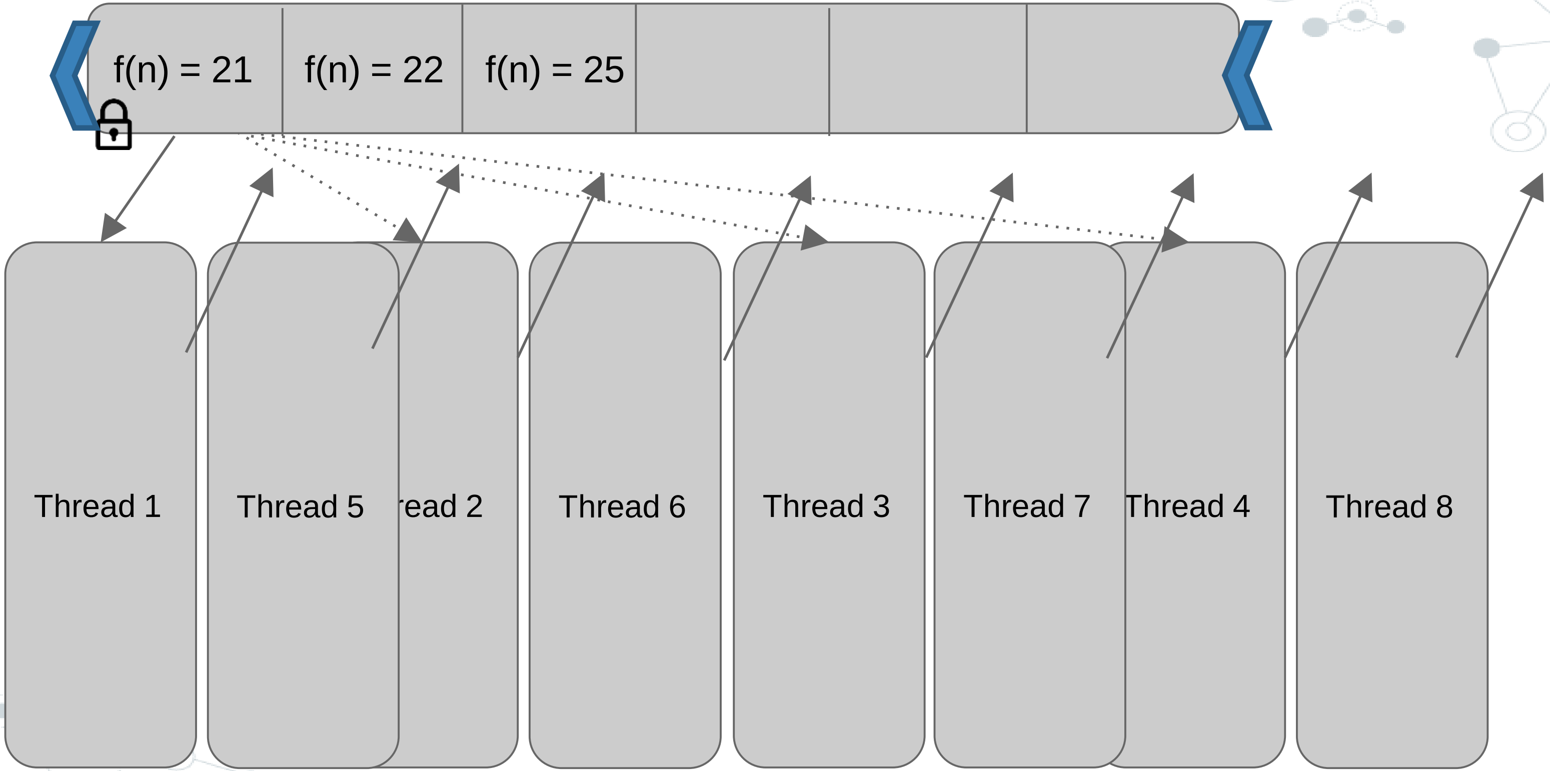
◎Centralised, using shared frontier (Centralised List A*)
◎Localised, using shared memory (Blackboard)
◎Localised, using message passing (HDA*)

Overhead

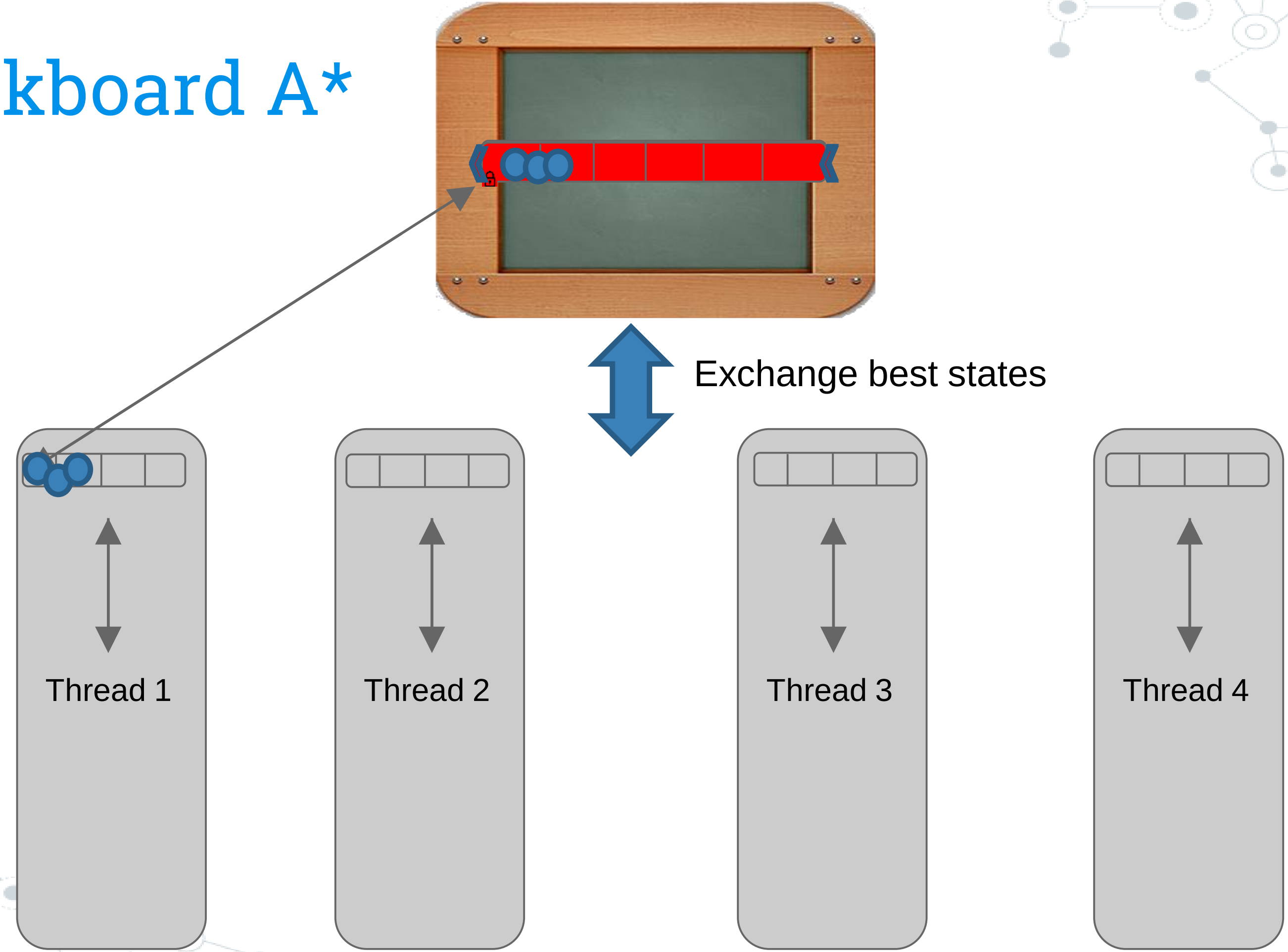


Centralised Shared Frontier A*

```
TASK(*) public void solveTSP() { …
```



Blackboard A*



Parallel Best-First Search, http://parallelcomp.uw.hu/ch11lev1sec5.html

Blackboard Distributed A*

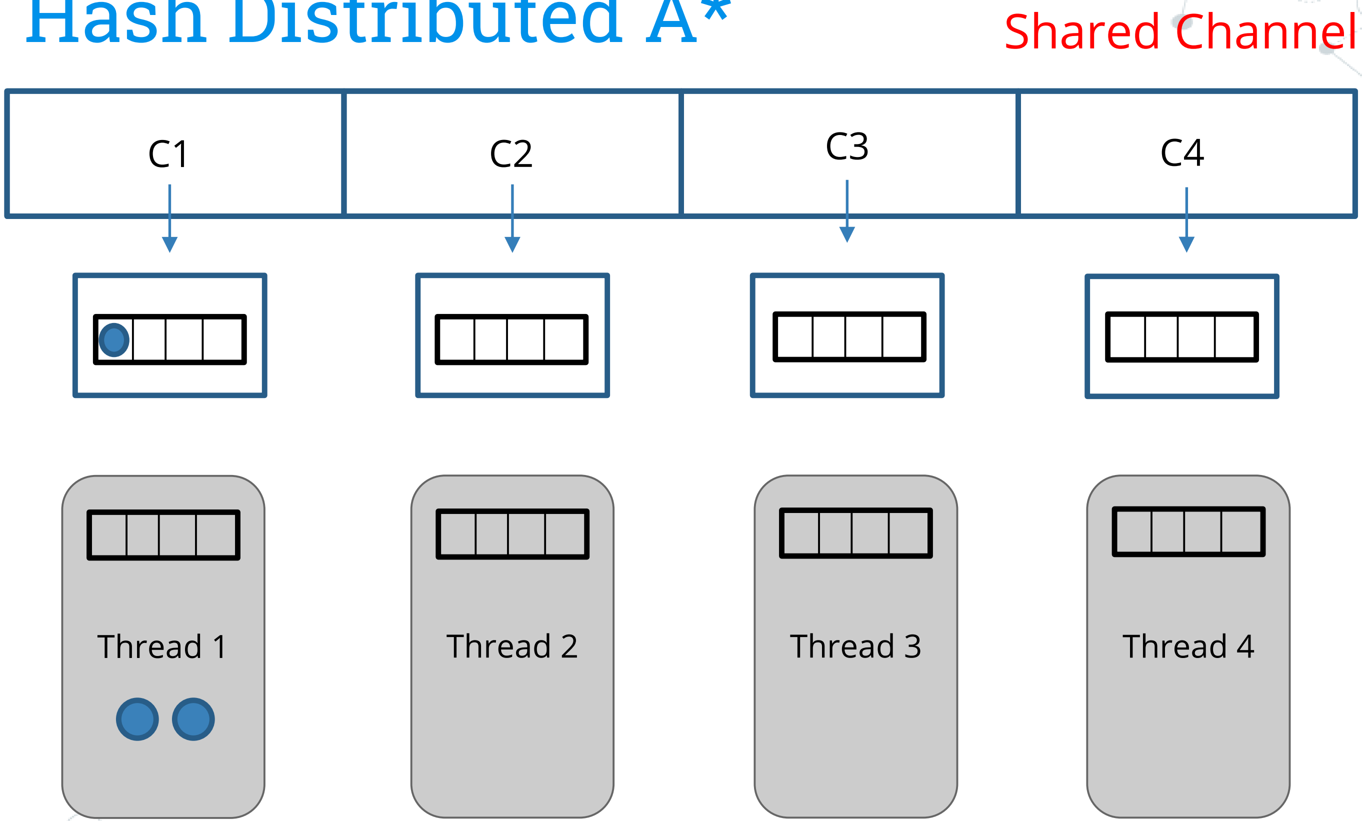
✓ Minimum access to shared data structure

✓ Reduced synchronisation cost

☹ Performance depends on parameters
(e.g. Threshold and number of states copied)

Parallel Best-First Search, http://parallelcomp.uw.hu/ch11lev1sec5.html

Hash Distributed A*



Scalable, Parallel Best-First Search for Optimal Sequential Planning
Akihiro Kishimoto, Alex Fukunaga, Adi Botea

Hash Distributed A*

✓ Reduced synchronization overhead

✓ Simple hash based work distribution

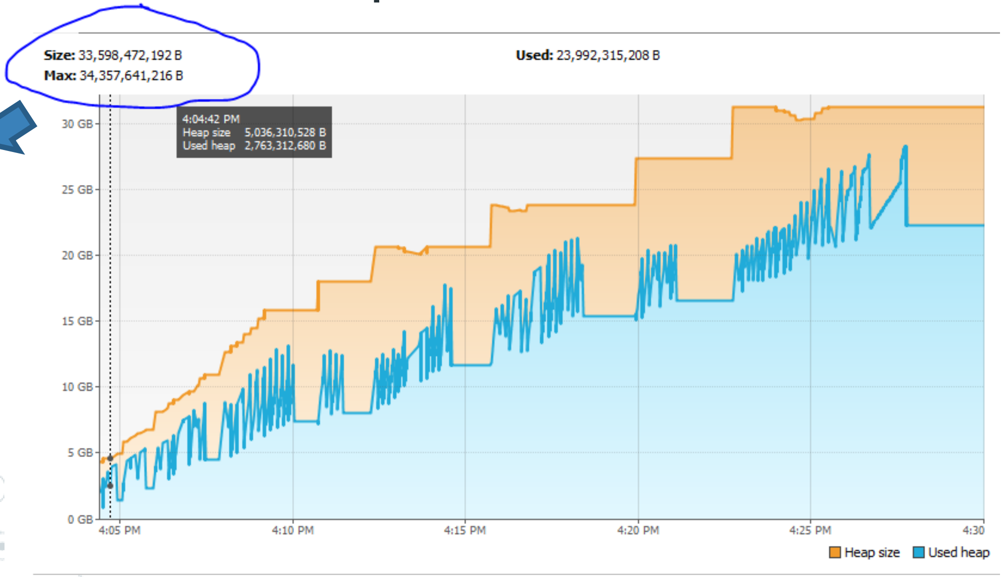
Scalable, Parallel Best-First Search for Optimal Sequential Planning
Akihiro Kishimoto, Alex Fukunaga, Adi Botea

TSP A* Demos

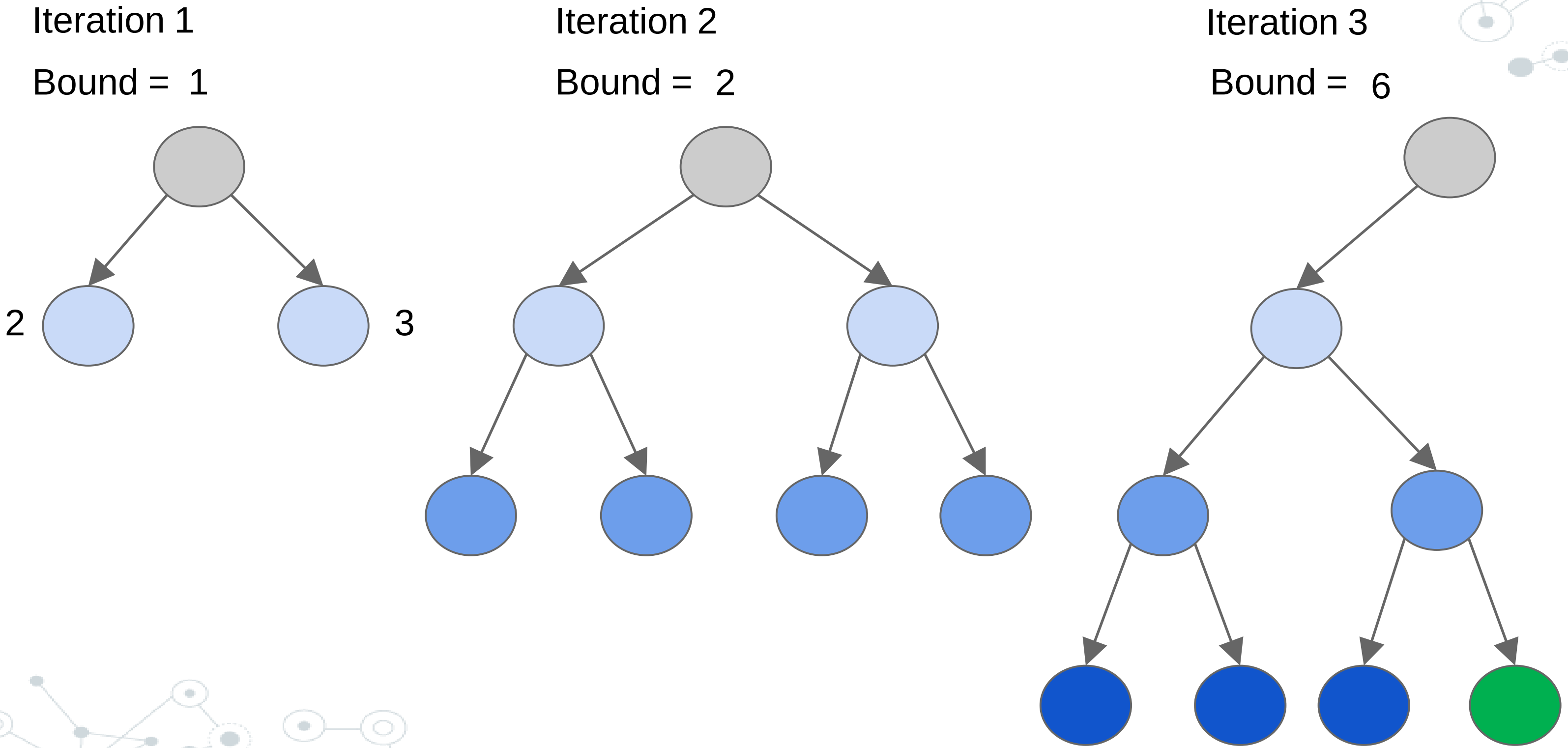
Scalability

- A* for TSP uses exponential memory in the worst case
- Problems with the Java Garbage Collector, JVM heap size, and RAM of computer

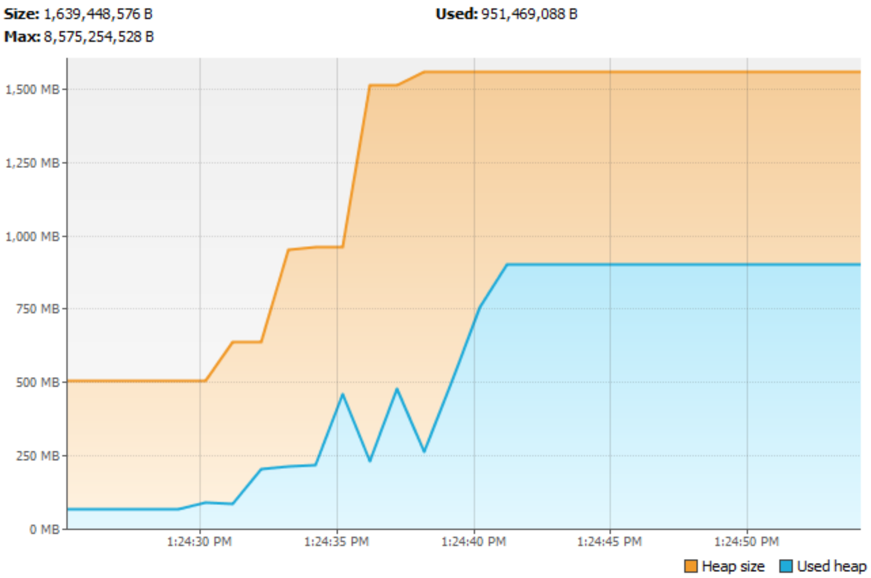
Heap
Size: 33,598,472,192 B
Max: 34,357,641,216 B
(Over 32 GB!)



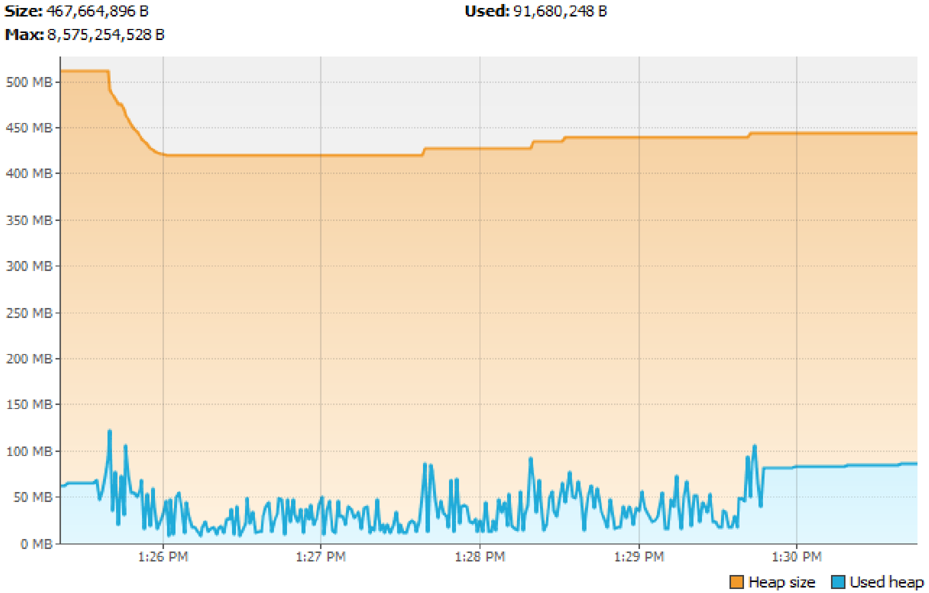
Iterative Deepening A*



Difference in Memory Usage



Sequential A* algorithm

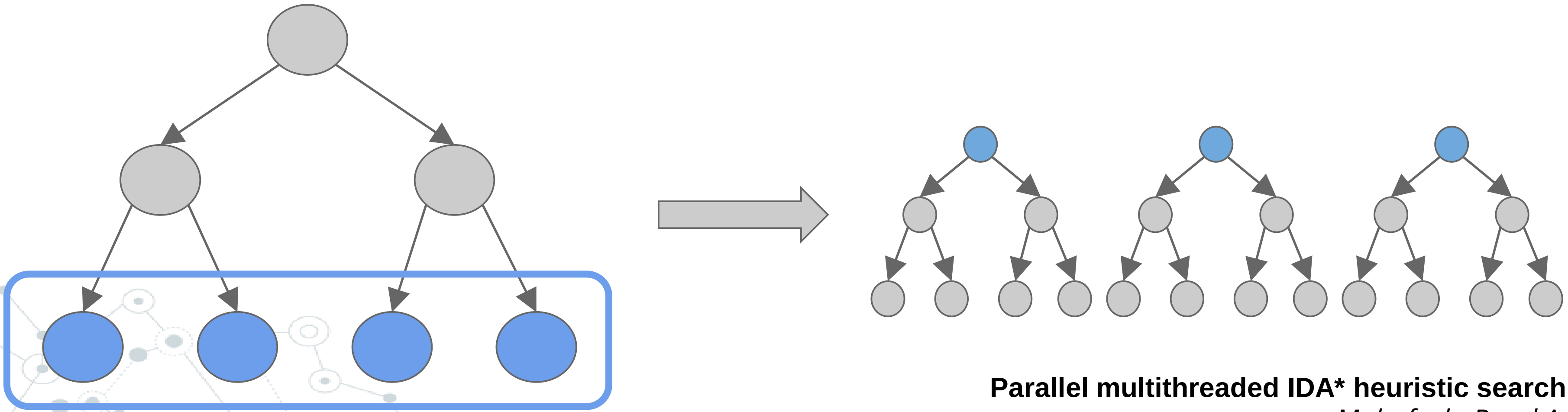


IDA* algorithm

Parallel Iterative Deepening A*

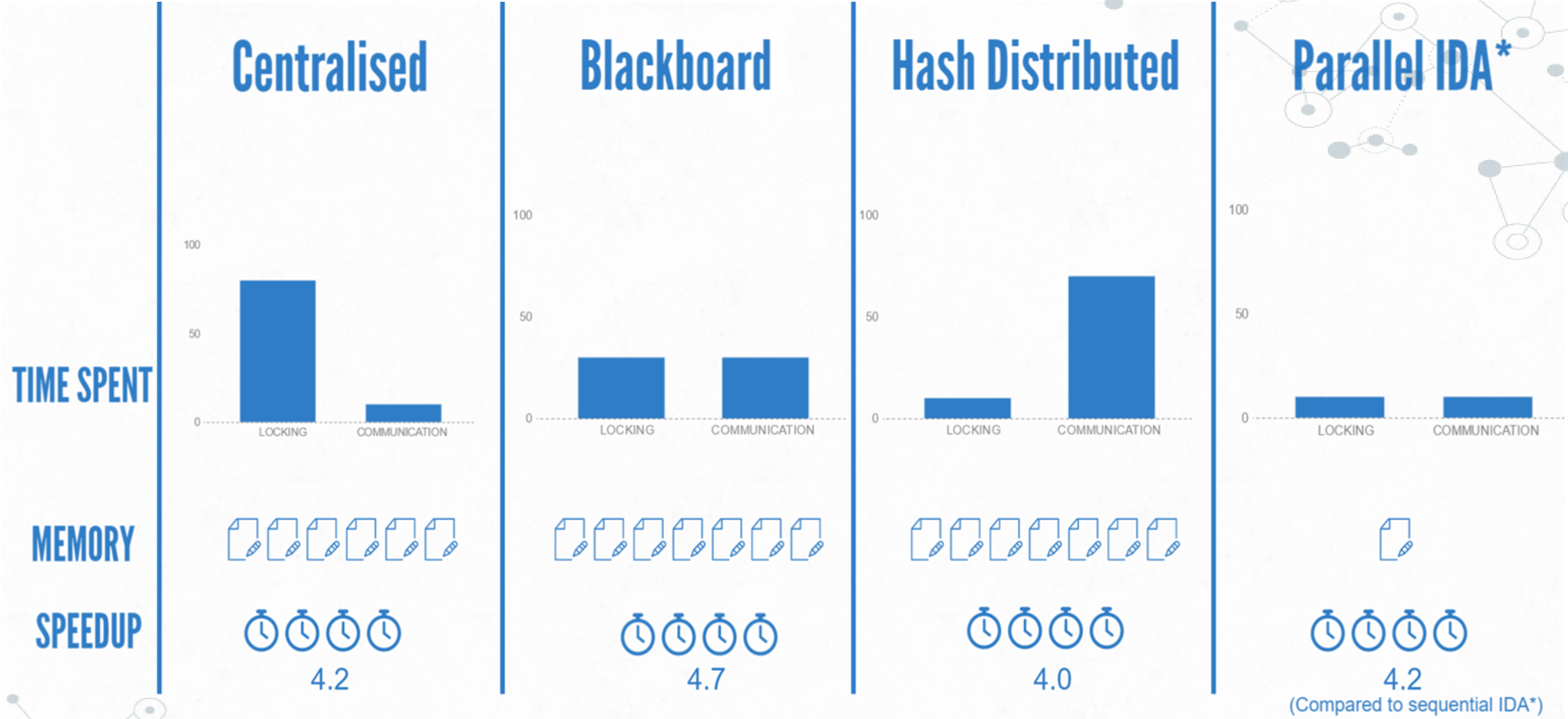
2 phase process

- Perform a depth limited breadth first search to generate a queue of states
- Perform IDA* on each state in the queue

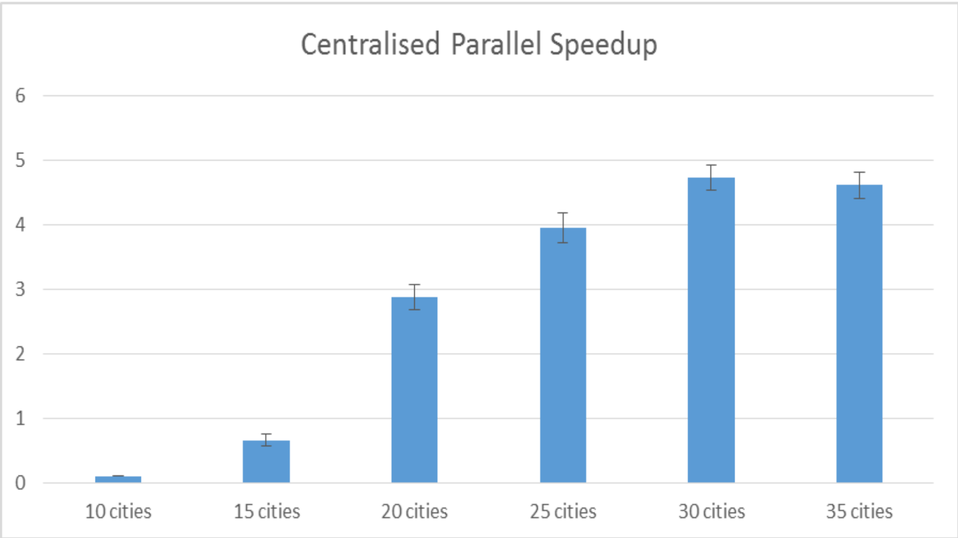


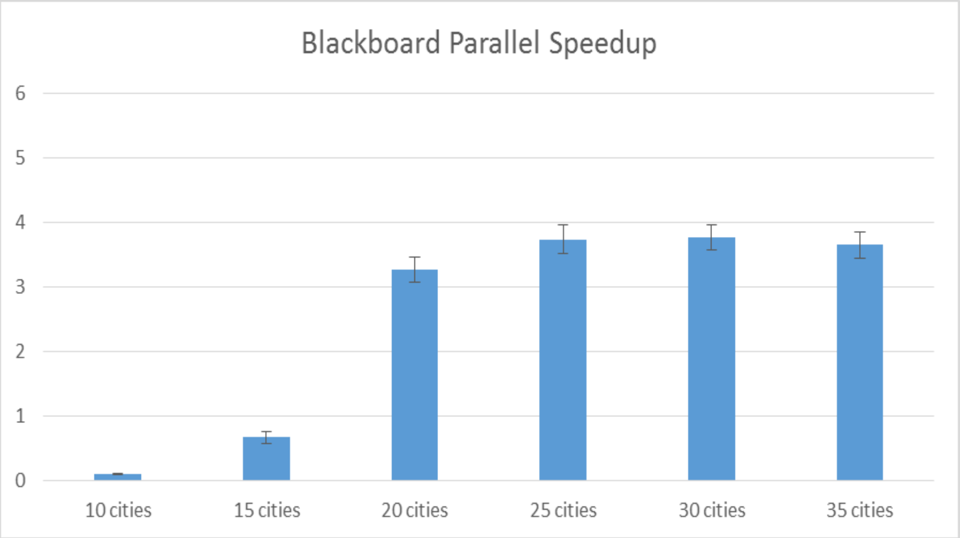
Parallel multithreaded IDA* heuristic search
Mahafzah, Basel A.

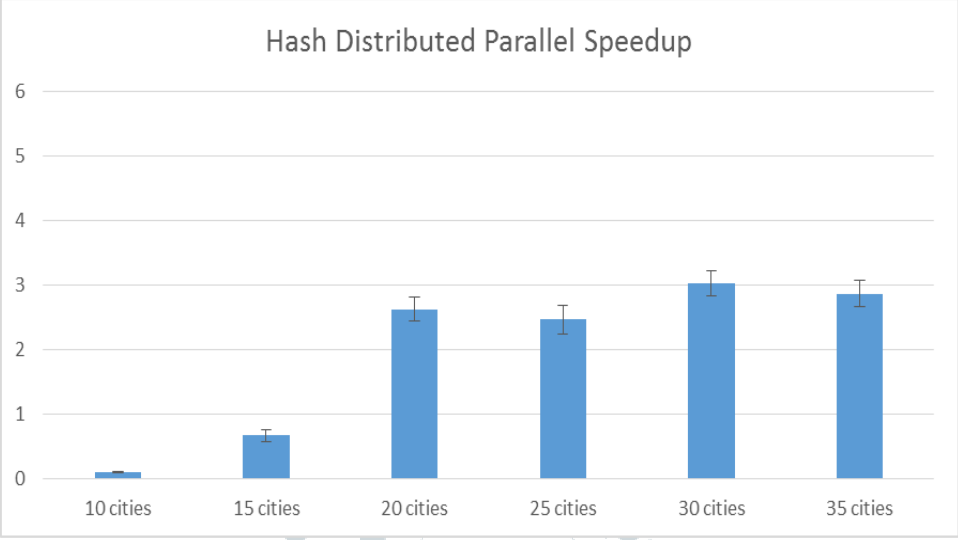
IDA* and Parallel IDA* Demo

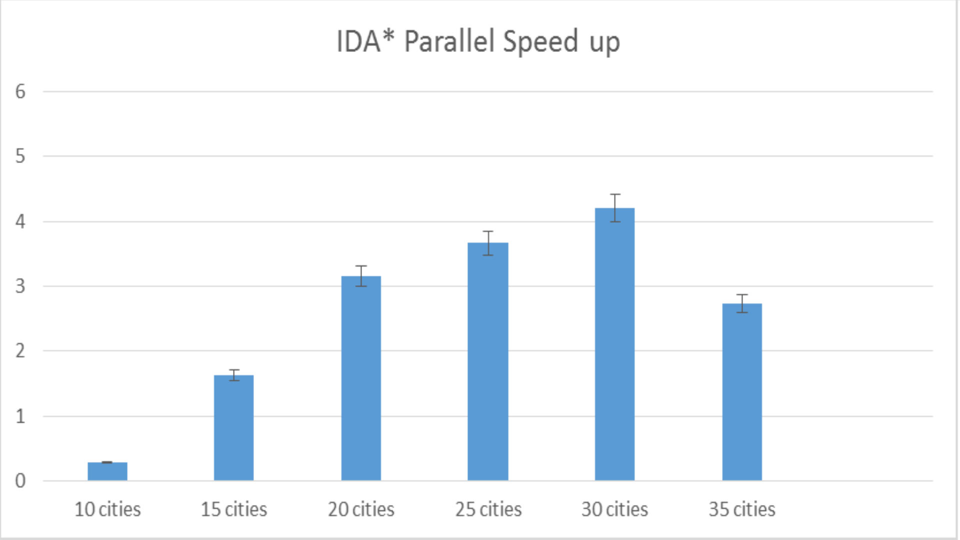


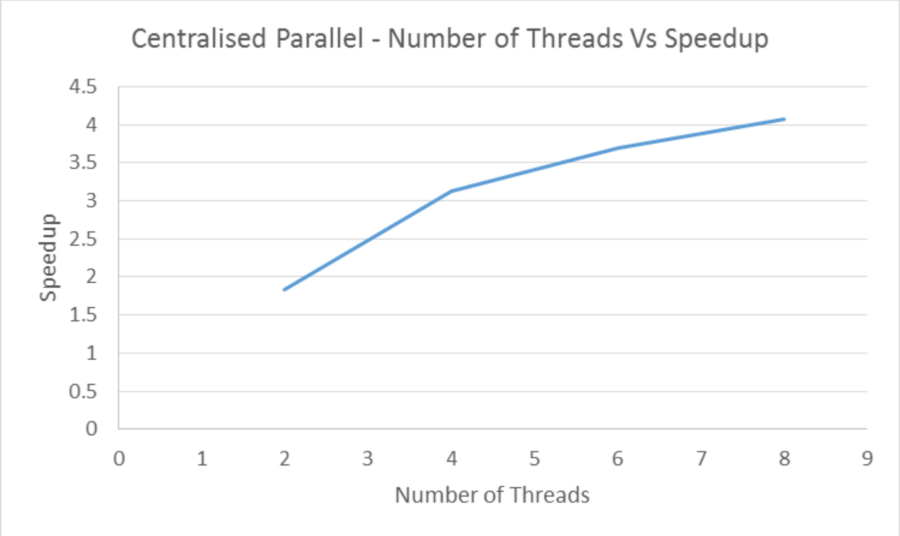
4 cores, 8 logical processors, hyperthreading, 27 city problem

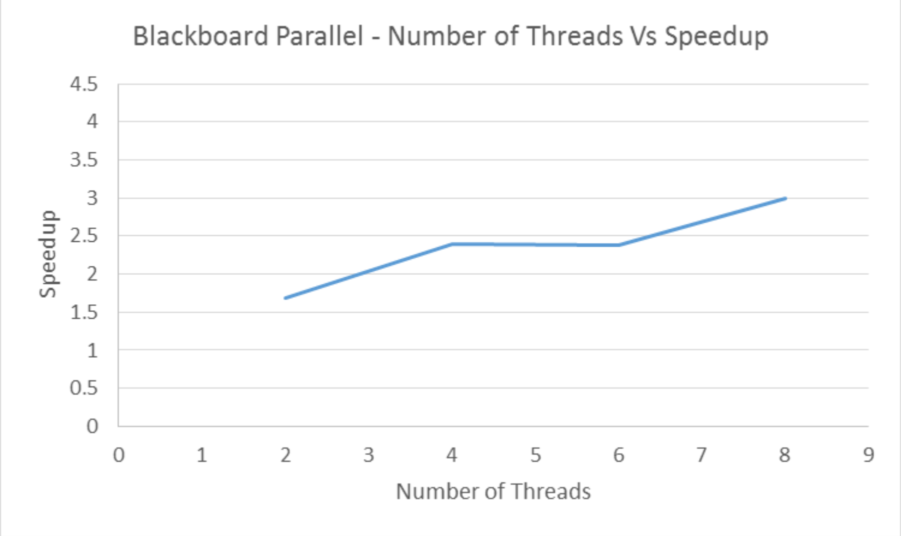

Centralised Parallel Speedup
0
1
2
3
4
5
6
10 cities
15 cities
20 cities
25 cities
30 cities
35 cities

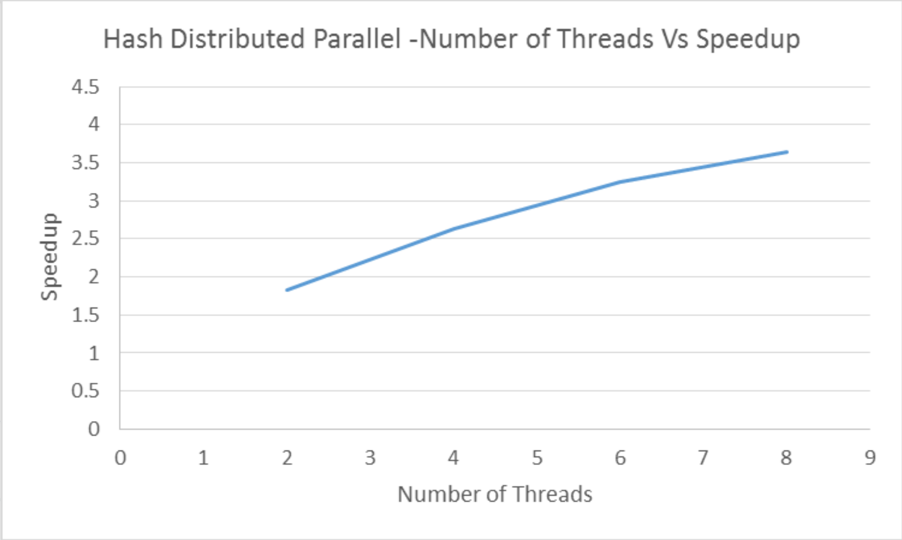

Blackboard Parallel Speedup
0
1
2
3
4
5
6
10 cities
15 cities
20 cities
25 cities
30 cities
35 cities

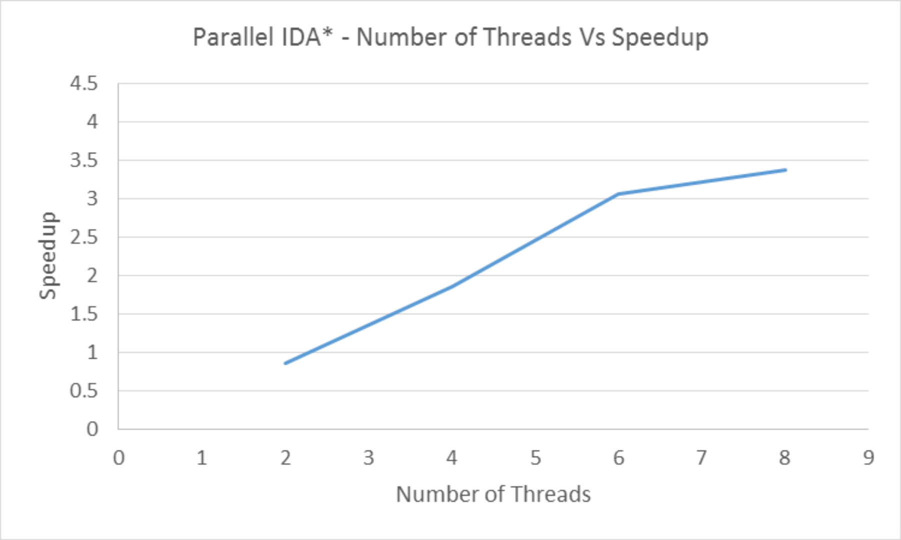

Hash Distributed Parallel Speedup
0
1
2
3
4
5
6
10 cities
15 cities
20 cities
25 cities
30 cities
35 cities


IDA* Parallel Speed up
0
1
2
3
4
5
6
10 cities
15 cities
20 cities
25 cities
30 cities
35 cities


Centralised Parallel - Number of Threads Vs Speedup
Speedup
Number of Threads
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
0 1 2 3 4 5 6 7 8 9


Blackboard Parallel - Number of Threads Vs Speedup
Speedup
Number of Threads
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
0 1 2 3 4 5 6 7 8 9


Hash Distributed Parallel -Number of Threads Vs Speedup
Speedup
Number of Threads
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
0 1 2 3 4 5 6 7 8 9


Parallel IDA* - Number of Threads Vs Speedup
Speedup
Number of Threads
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
0 1 2 3 4 5 6 7 8 9

Future work

- Try out larger problem (50+ cities)
- Implement distributed termination algorithm for HDA* and Blackboard
- Write our report

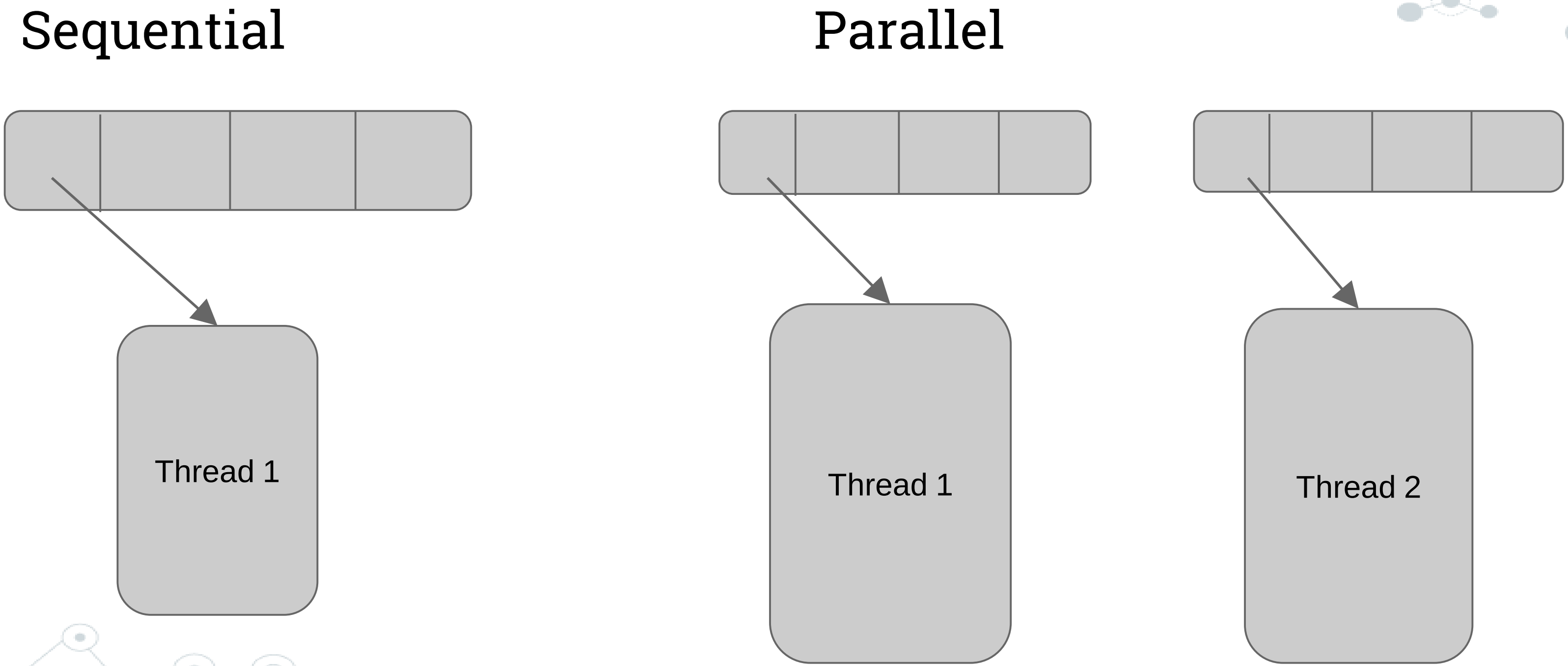
Conclusion

- A* for TSP is slightly different than traditional A* for Path Finding
- Parallelisation can be either centralised or distributed
- The parallelisation overhead is either communication cost, synchronisation cost, or exploring extra nodes
- Parallel IDA* can be used to reduce memory usage
- The different parallel implementations scale differently over the number of cities and processors

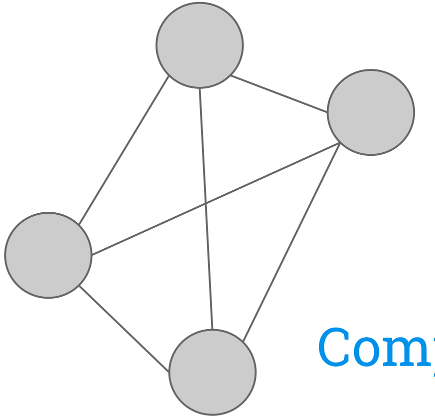
Thanks!

Any questions?

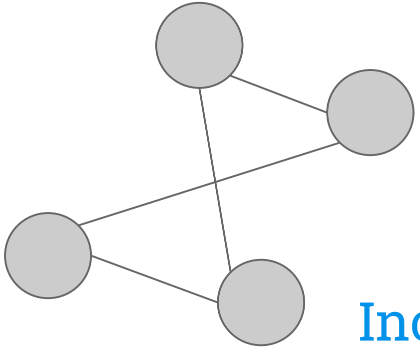
Why does parallel explore more nodes?



Complete Vs. Incomplete Graph



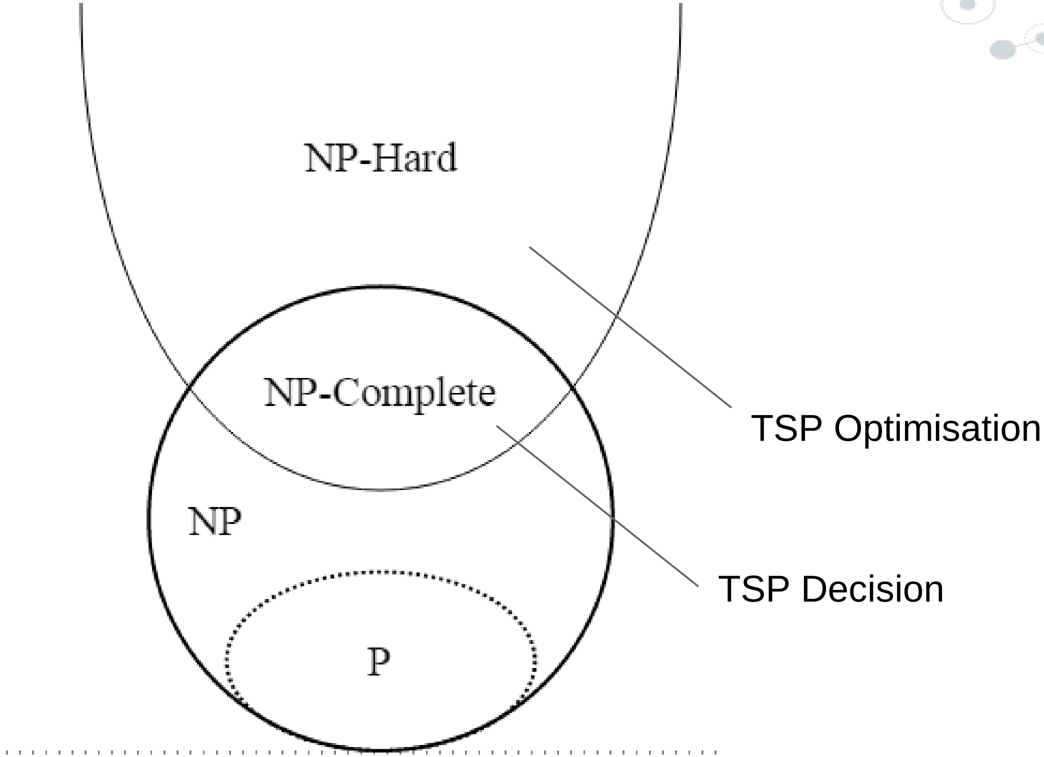
Complete



Incomplete

We could just add infinite costs between cities with no edges between them

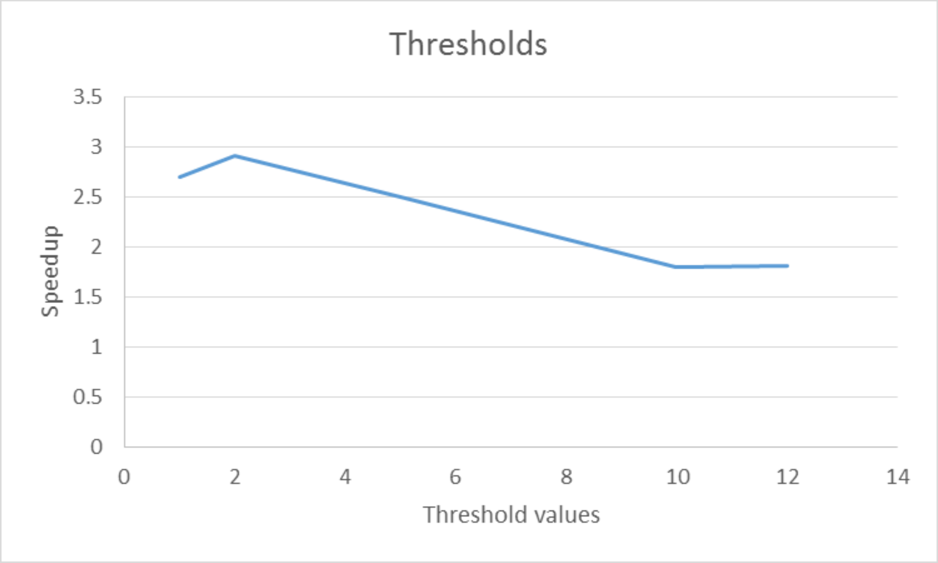
NP Complete Vs NP Hard

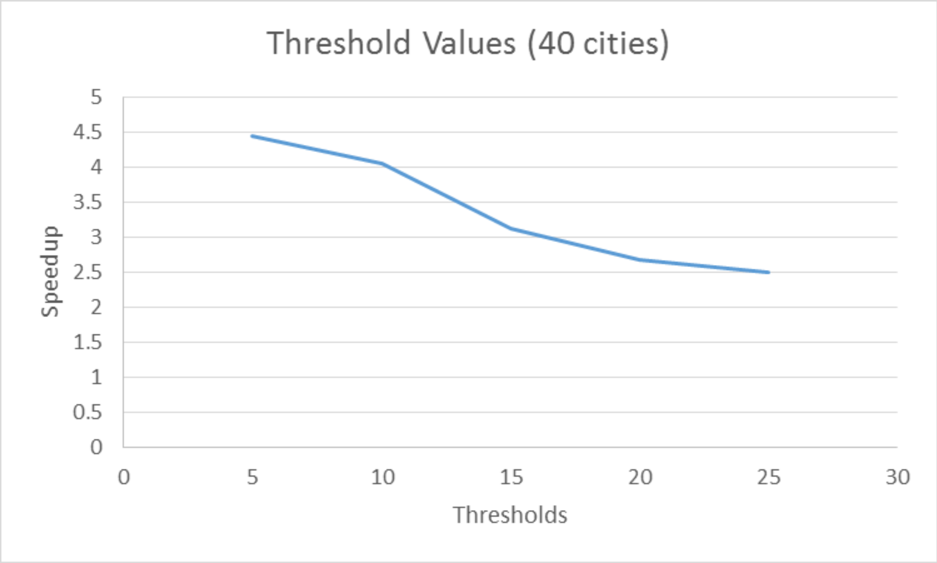


Other heuristic for A* algorithm

Heuristic	Description	Time Complexity
Greedy	Repeatedly pick the shortest edge	$O(n^2 \log_2(n))$
Insertion	Start with a subset of all cities, and insert the rest by some heuristic	$O(n^2)$
Christofides	Create a MST, MWM, and Euler Cycle	$O(n^3)$
MST	Create a MST	$O(n^2)$

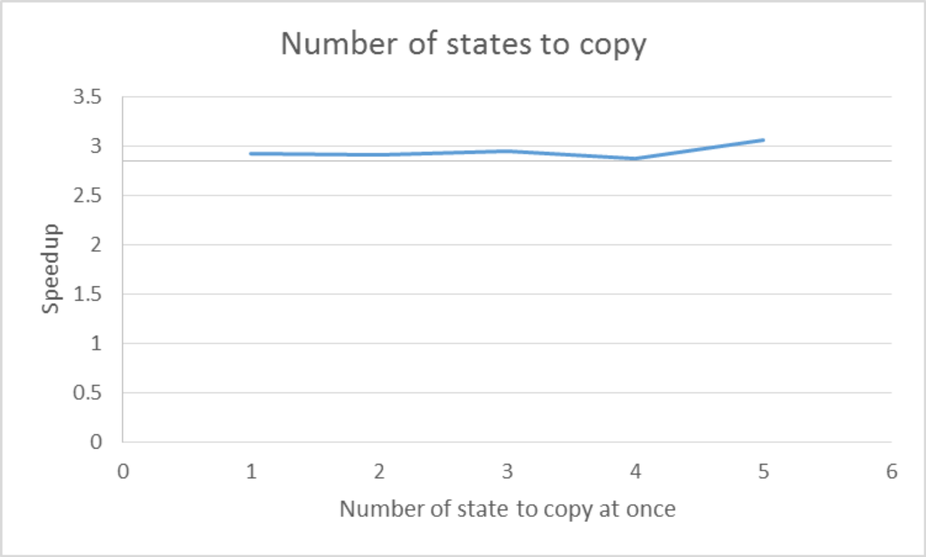
Blackboard A* Thresholds

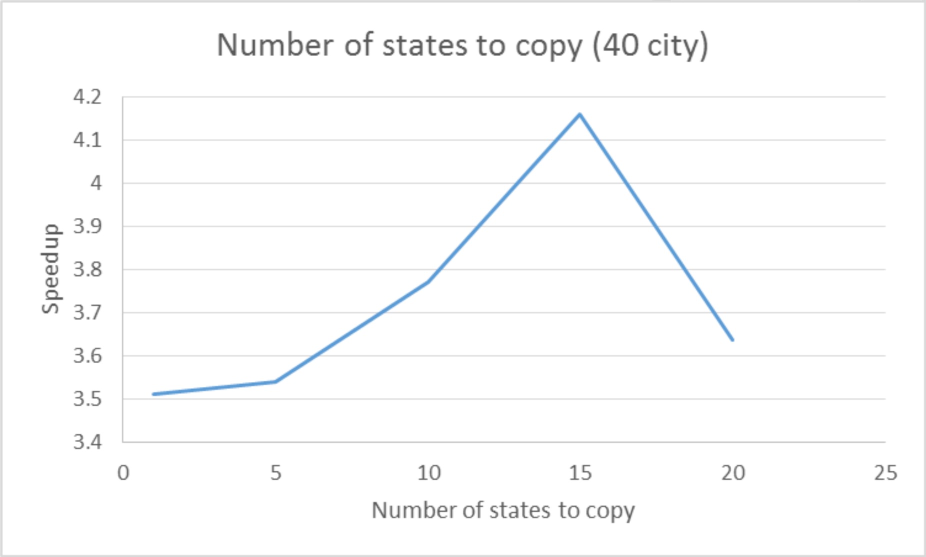




4 cores, 8 logical processors, hyperthreading

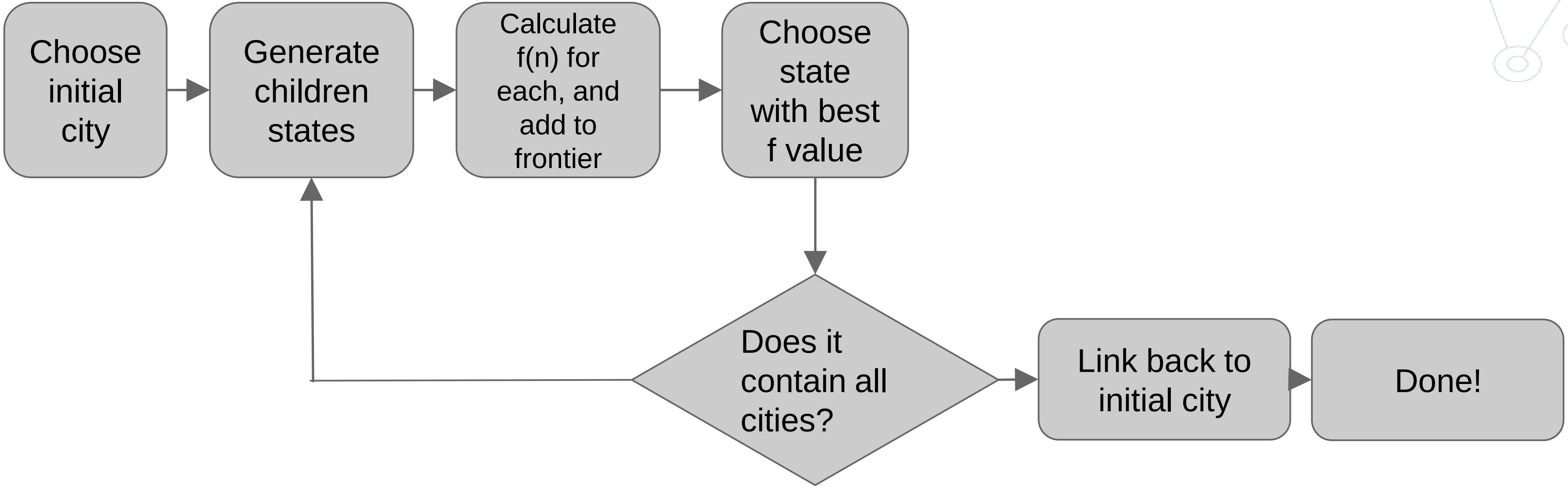
Blackboard A* Parameters





4 cores, 8 logical processors, hyperthreading

A* for TSP: Flowchart



Tree Search