Faster Shortest Path Computation for Traffic Assignment

Boshen Chen
Supervised by: Dr. Andrea Raith, Olga Perederieieva

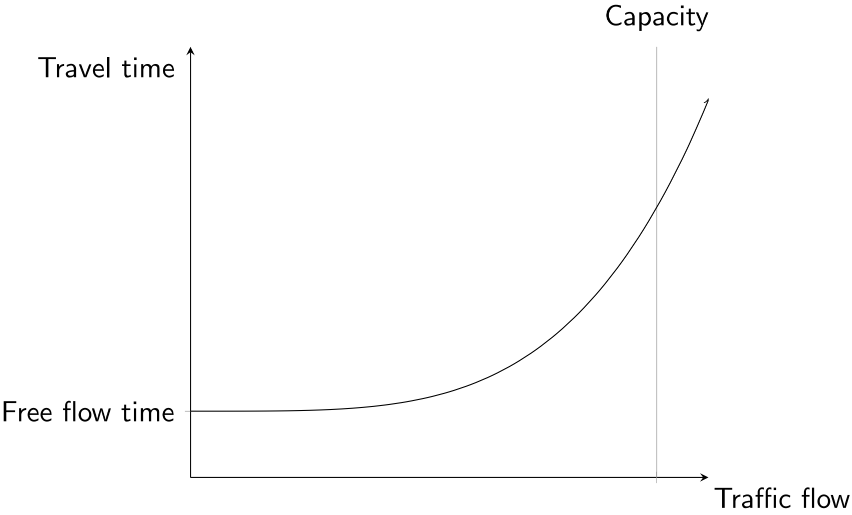
Department of Engineering Science
University of Auckland

Contents

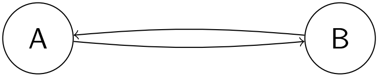
The traffic assignment problem

- Assigns traffic to a transportation network
- Used to determine areas of high congestion
- Deals with selecting the best route for vehicles to minimise their travel times (Wardrop equilibrium)
- (add animation showing 2 node 1 arc, then lots of nodes and arcs to emph lots of travellers and paths)
- Arcs have non-linear travel times for capturing congestion effects

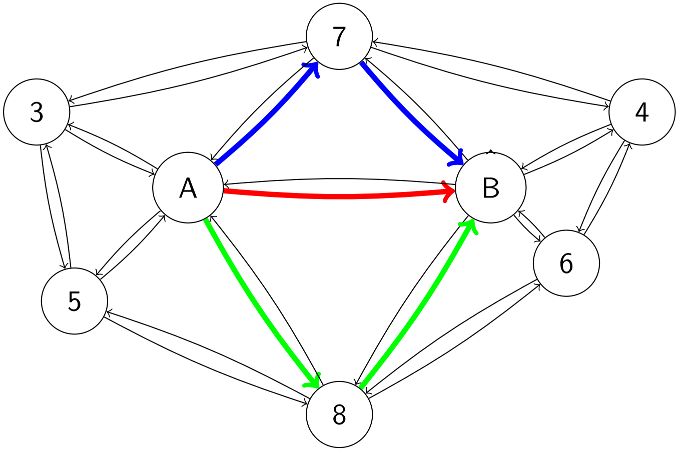
Non-linear travel time function



Traffic assignment illustration



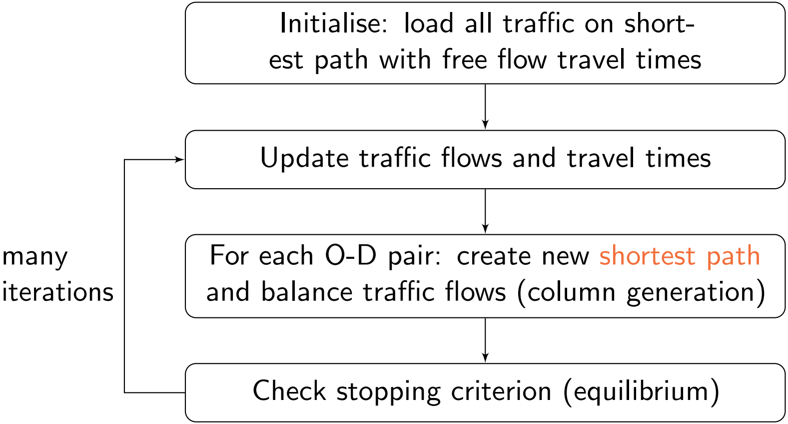
Traffic assignment illustration



Path equilibration algorithm

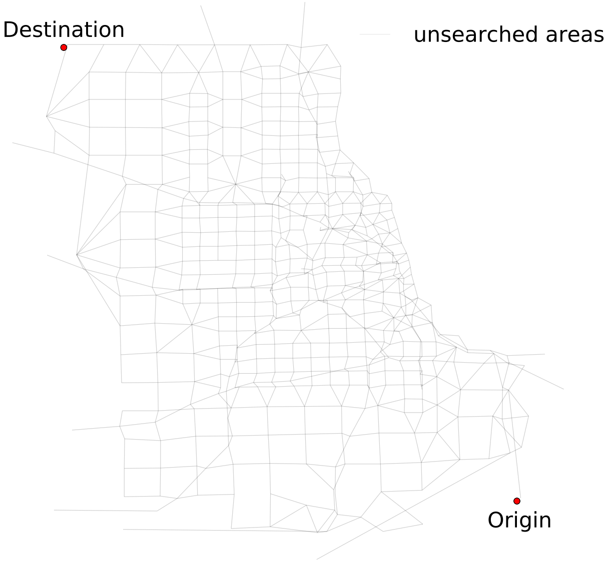
- Path-based: solutions are represented by traffic flows on path between origin and destination pairs
- Computational memory was an issue, not studied heavily before
- people do not emphasize shortest path calculations in traffic assignment
- can extend results to other algorithms that solve the traffic assignment problem

Path equilibration algorithm



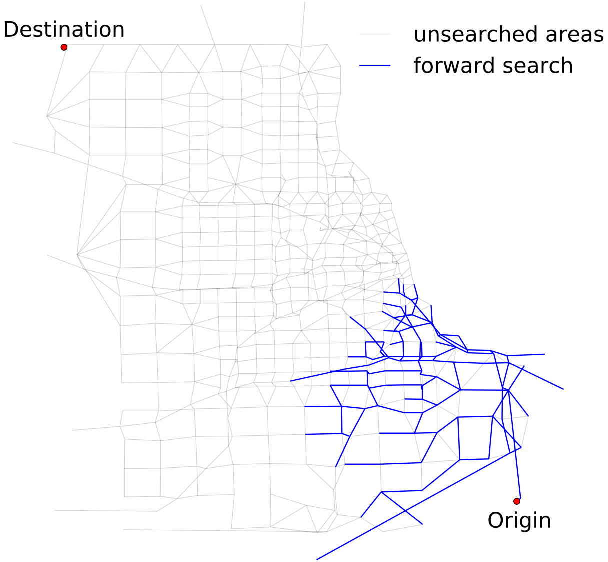
- Requires millions of shortest paths to be found

Dijkstra's algorithm



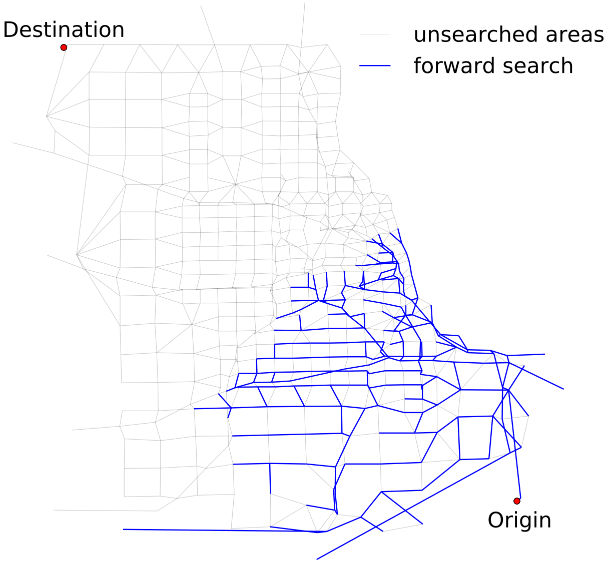
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



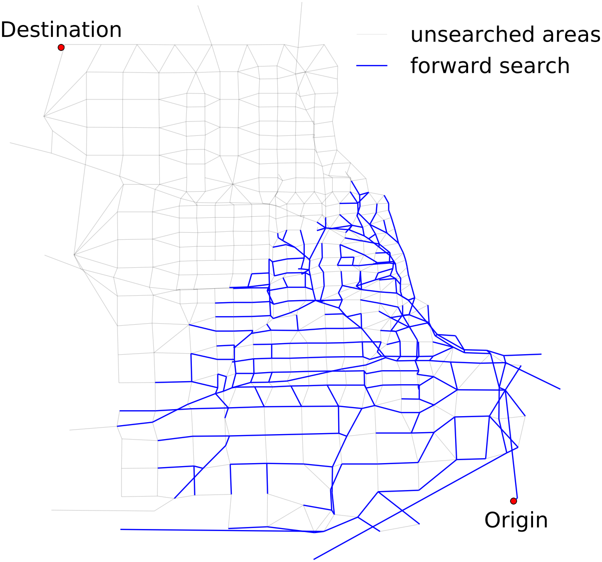
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



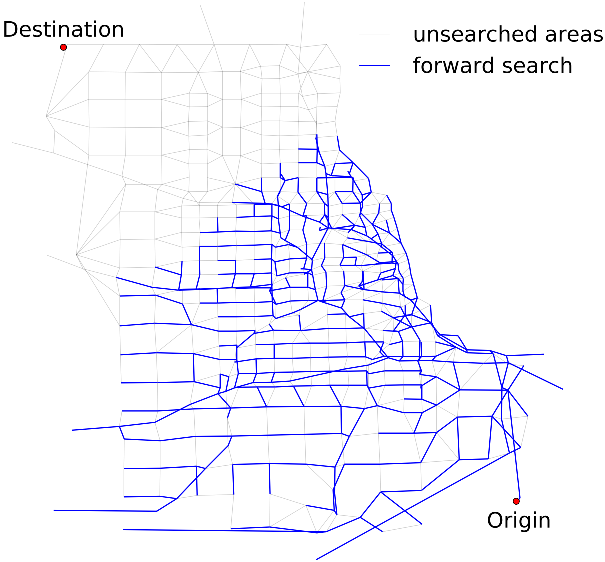
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



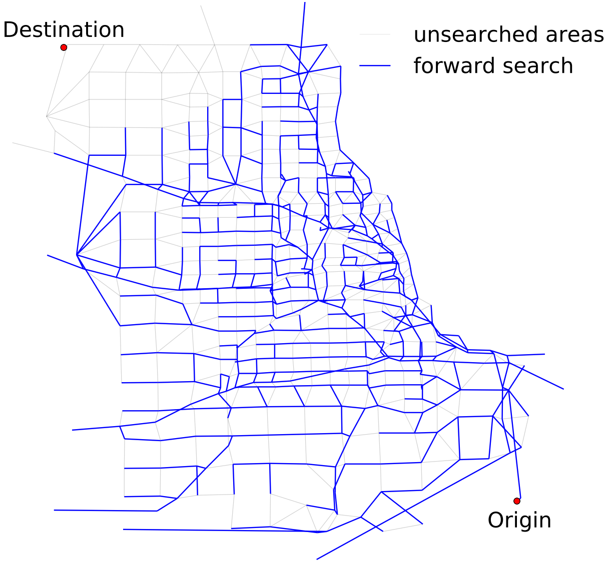
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



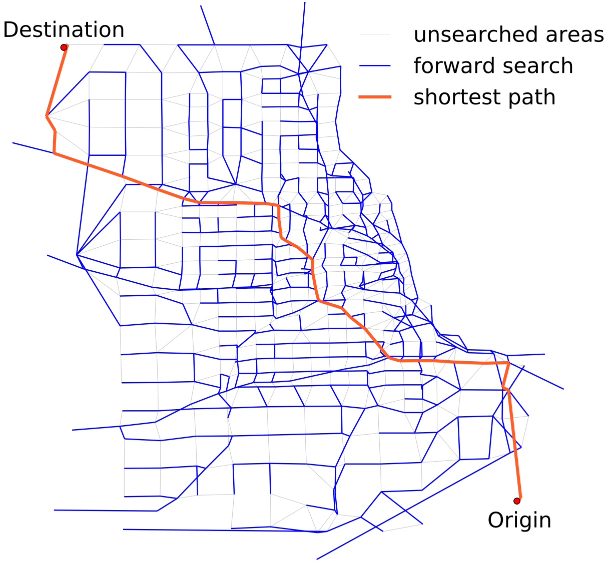
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

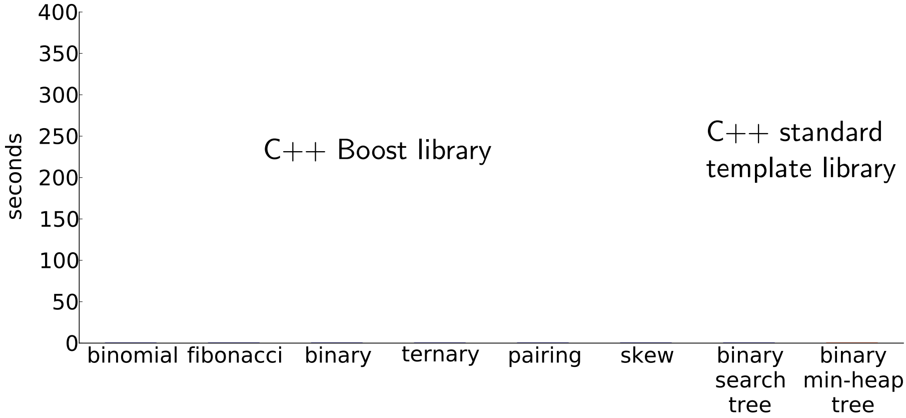
Dijkstra's algorithm



- $\min_{u \in \mathcal{Q}} [d_u]$
- d_u: shortest path from origin to u

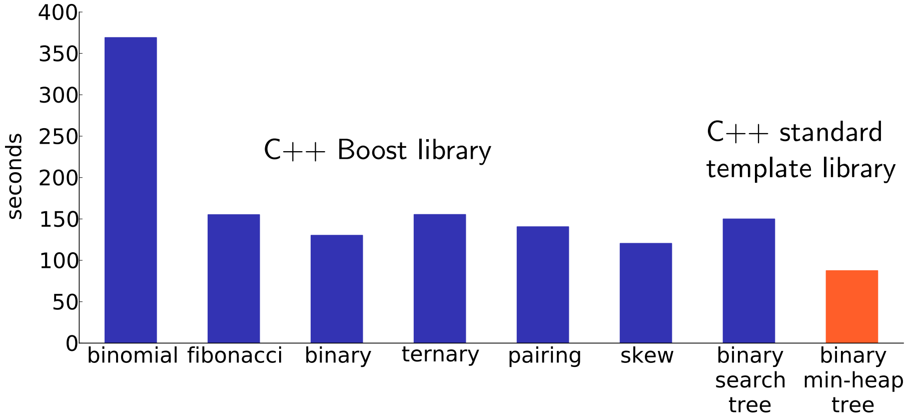
Priority queues

- Priority queue - a data structure for storing the searched nodes in some order so the next location to search can be found easily

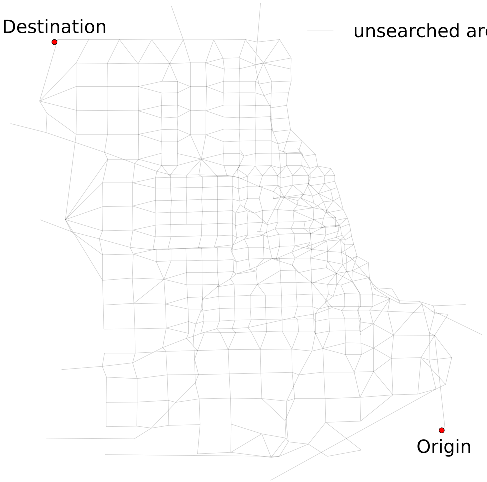


Priority queues

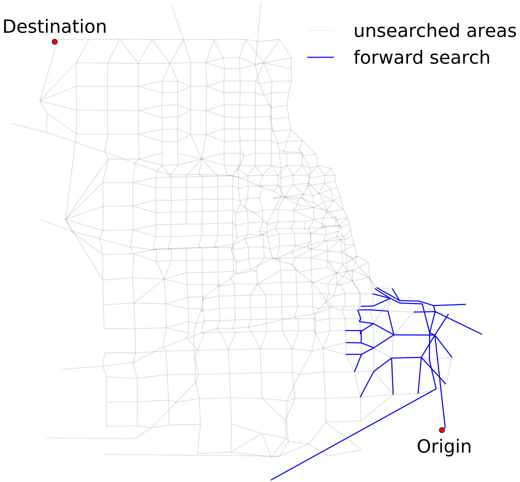
- Priority queue - a data structure for storing the searched nodes in some order so the next location to search can be found easily



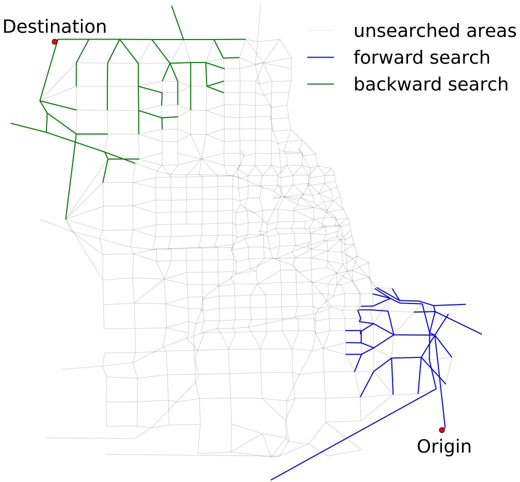
Bidirectional Dijkstra's algorithm



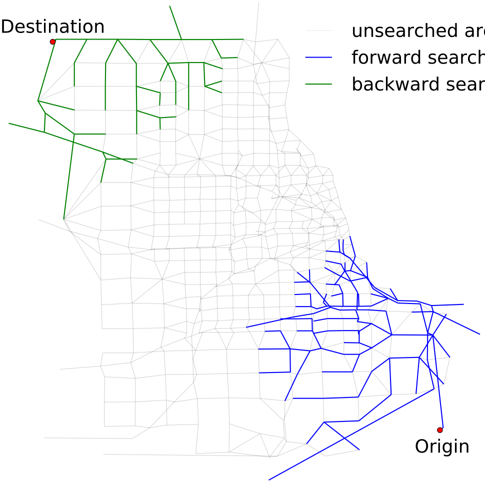
Bidirectional Dijkstra's algorithm



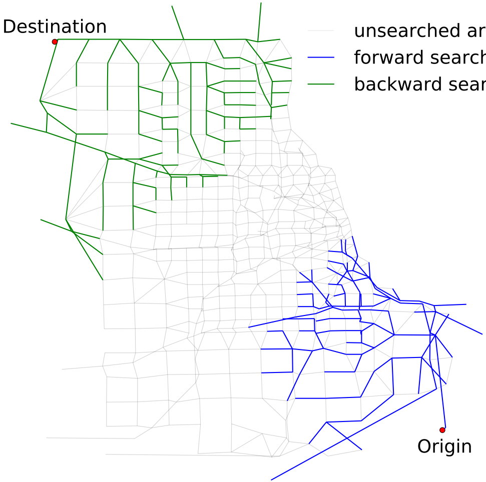
Bidirectional Dijkstra's algorithm



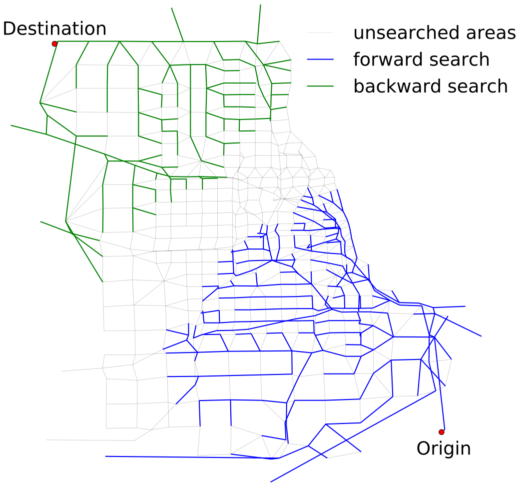
Bidirectional Dijkstra's algorithm



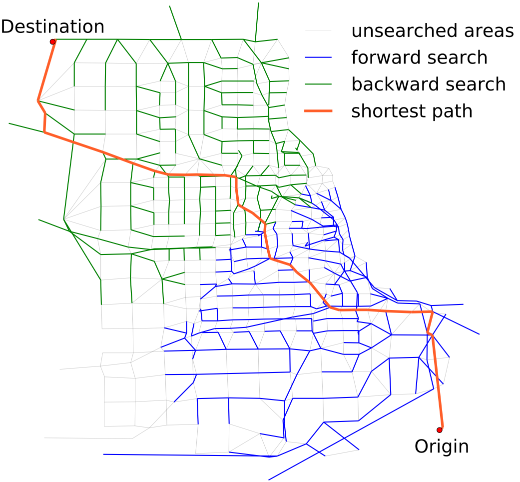
Bidirectional Dijkstra's algorithm



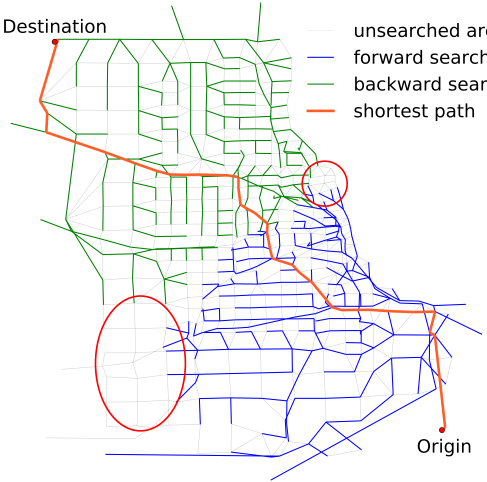
Bidirectional Dijkstra's algorithm



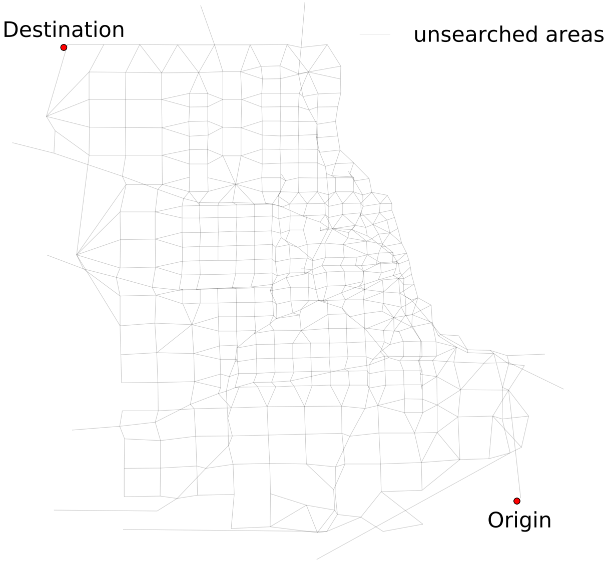
Bidirectional Dijkstra's algorithm



Bidirectional Dijkstra's algorithm

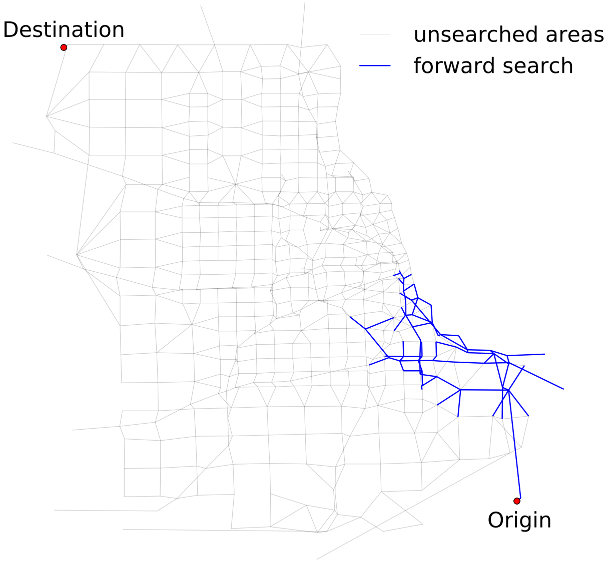


A* search (goal-directed search)



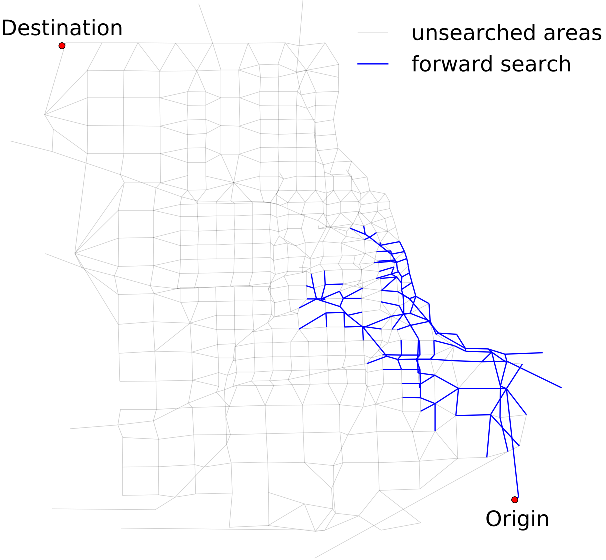
- $\min\limits_{u \in \mathcal{Q}} [d_u + h_u]$
- h_u: shortest path estimate from u to destination

A* search (goal-directed search)



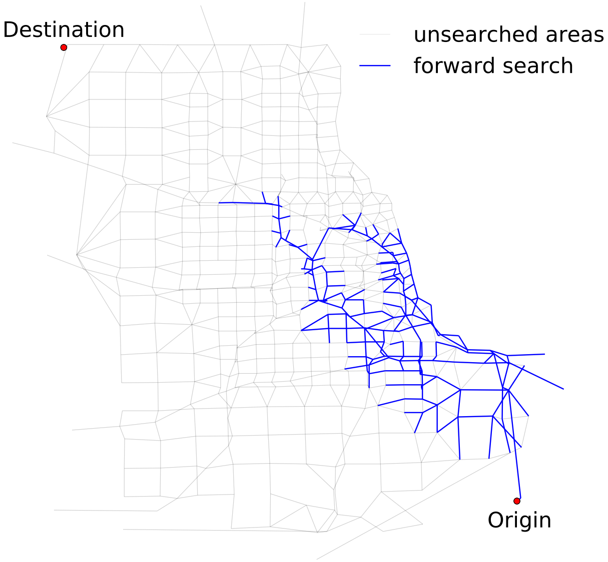
- $\min\limits_{u \in \mathcal{Q}} [d_u + h_u]$
- h_u: shortest path estimate from u to destination

A* search (goal-directed search)



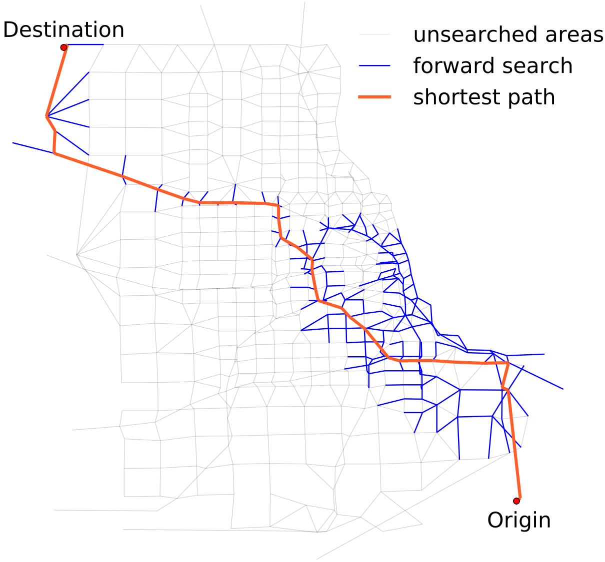
- $\min_{u \in \mathcal{Q}} [d_u + h_u]$
- h_u: shortest path estimate from u to destination

A* search (goal-directed search)



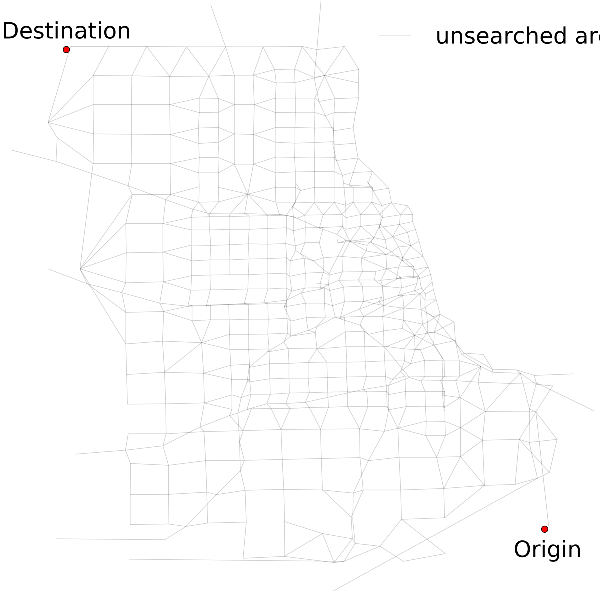
- $\min_{u \in \mathcal{Q}} [d_u + h_u]$
- h_u: shortest path estimate from u to destination

A* search (goal-directed search)

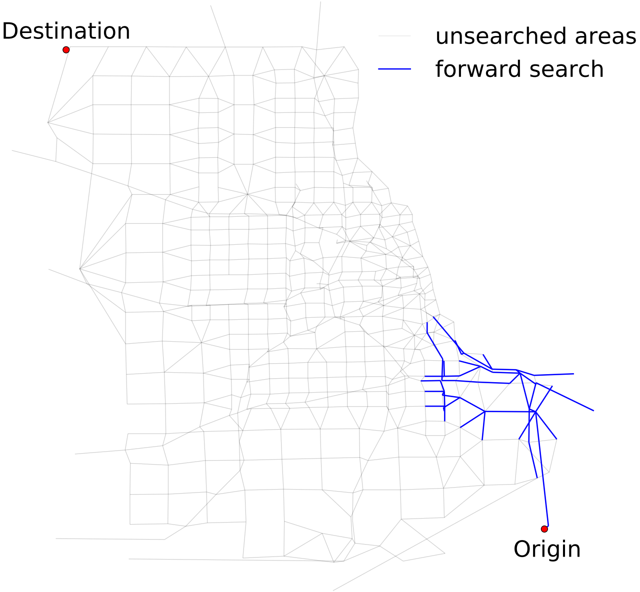


- $\min_{u \in \mathcal{Q}} [d_u + h_u]$
- h_u: shortest path estimate from u to destination

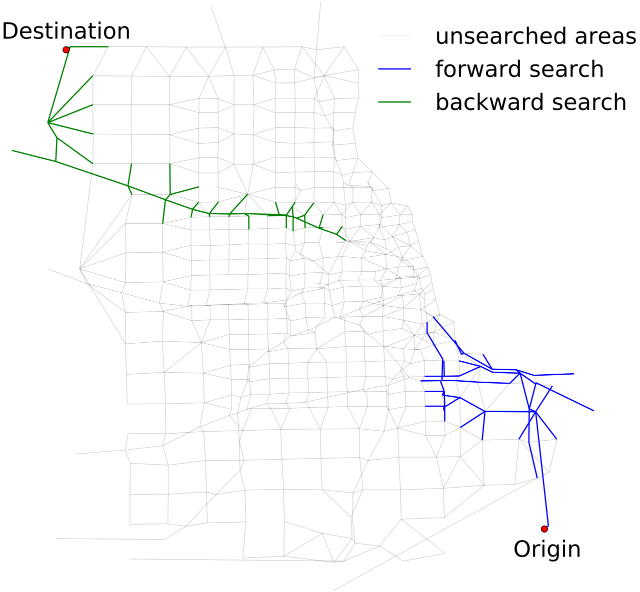
Bidirectional A* search



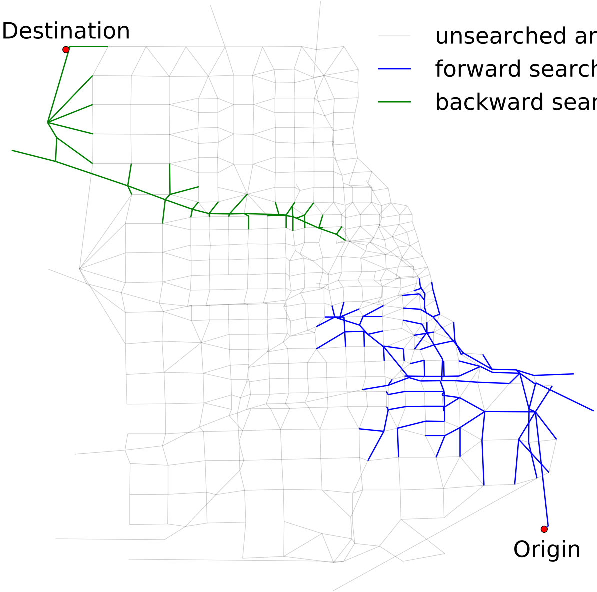
Bidirectional A* search



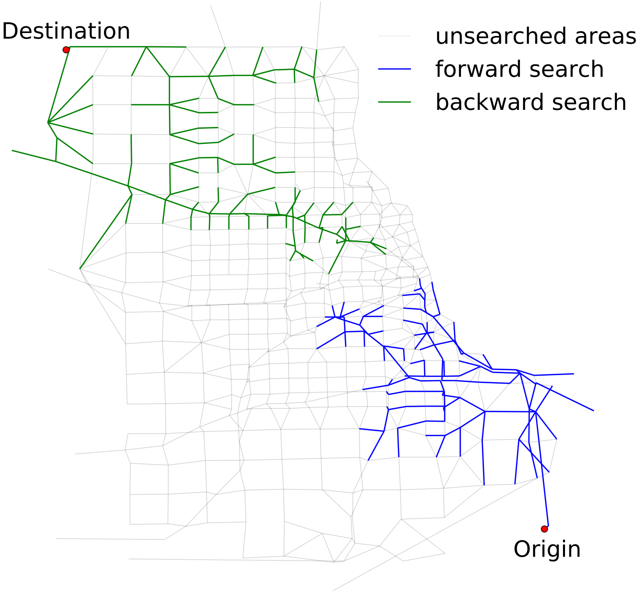
Bidirectional A* search



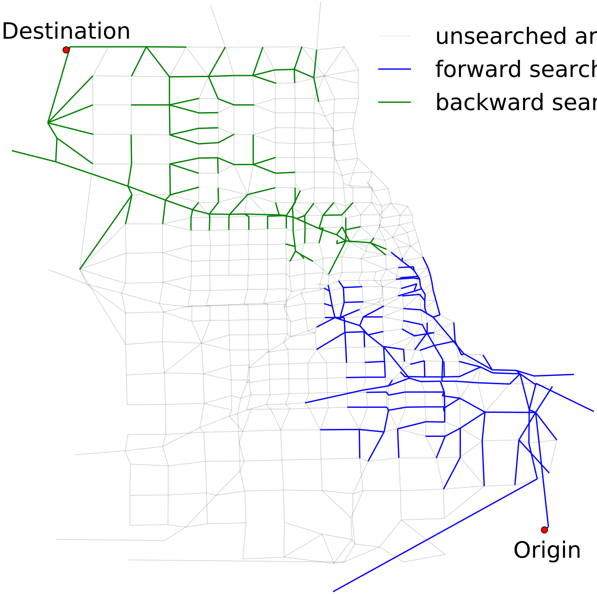
Bidirectional A* search



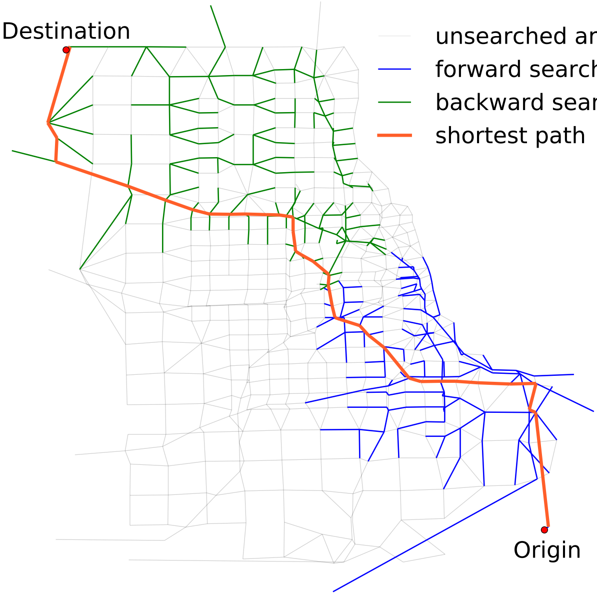
Bidirectional A* search



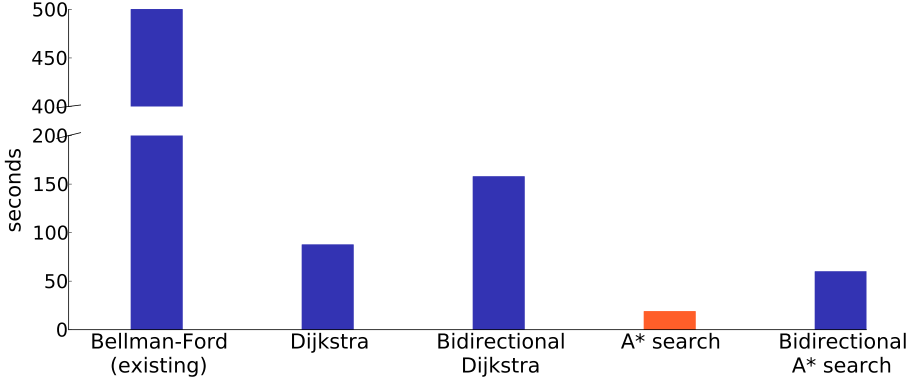
Bidirectional A* search



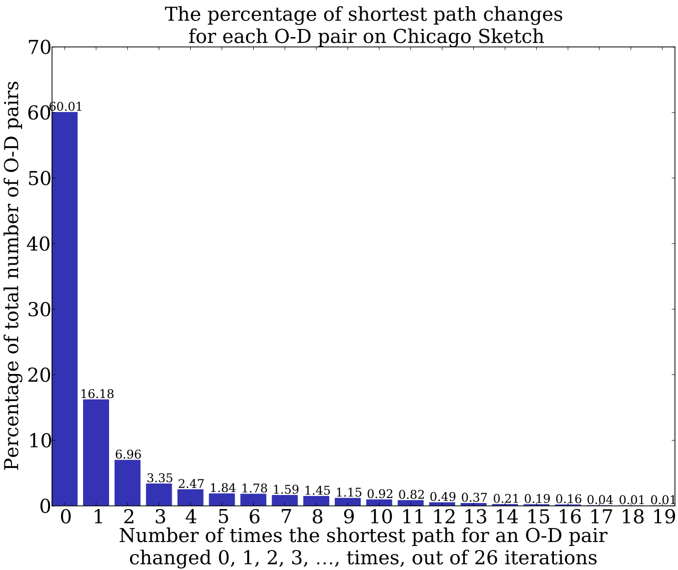
Bidirectional A* search



Shortest path algorithm results

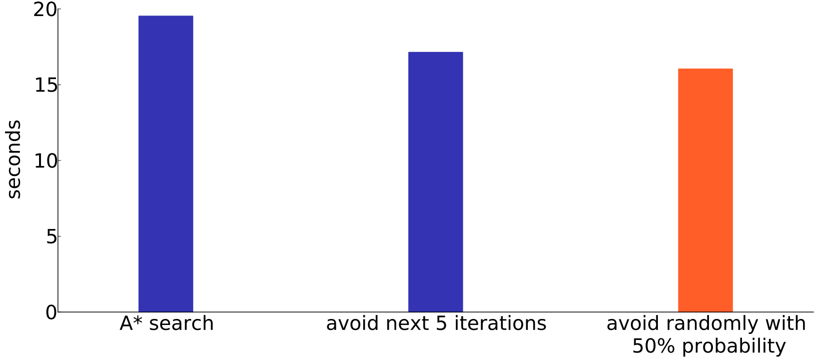


Shortest path change between iterations



Avoid shortest path calculations

1. avoid the next few iterations if the shortest paths of the previous two iterations are identical
2. randomly avoid the next shortest path calculation in the hope that the shortest path of previous and current iteration are identical
3. (emph it will converge because convex problem, if sp skipped, may just result no improve, then new sp will be calculated, then become bettter



Conclusion and future work

Conclusion

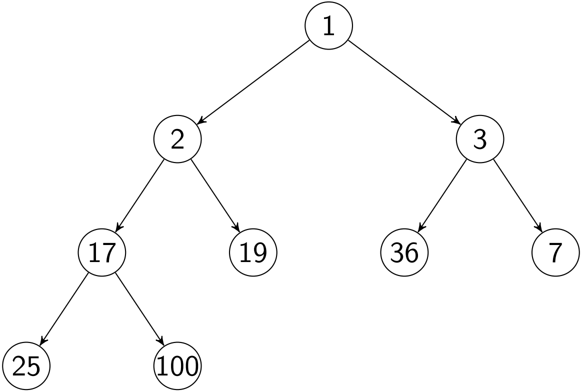
- Best performance: A* search algorithm using min-heap tree with random avoiding strategy
- 30 times faster than the existing implemented Bellman-Ford algorithm
- Bidirectional algorithms are worse compared to the unidirectional ones

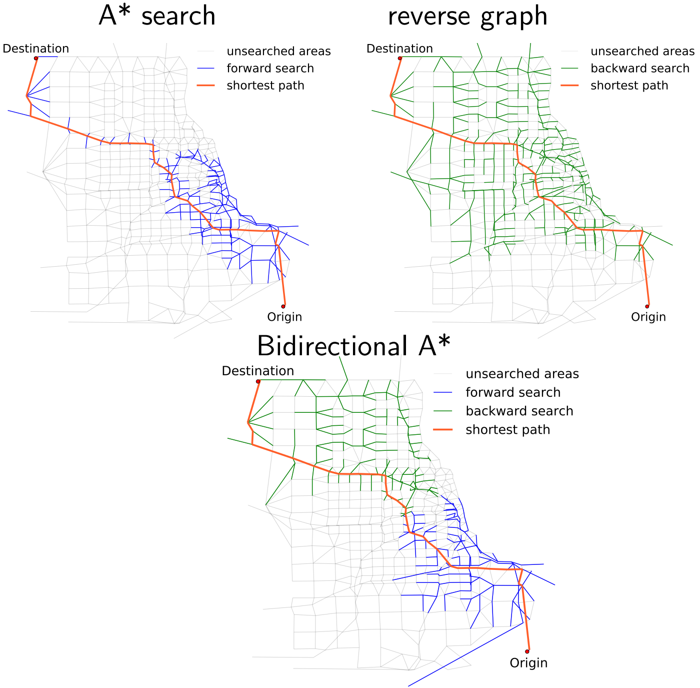
Future work

- Pre-processing: A* search with landmarks
- Multi-thread on GPU
- Test the avoiding strategies on other algorithms that solve the traffic assignment problem

Appendix
show binary min heap tree

Min-priority heap tree




A* search
Destination
unsearched areas
forward search
shortest path
Origin
reverse graph
Destination
unsearched areas
backward search
shortest path
Origin
Bidirectional A*
Destination
unsearched areas
forward search
backward search
shortest path
Origin

Shortest path algorithm results on different networks