Faster Shortest Path Computation for Traffic Assignment

Boshen Chen

Supervised by: Dr. Andrea Raith, Olga Perederieieva

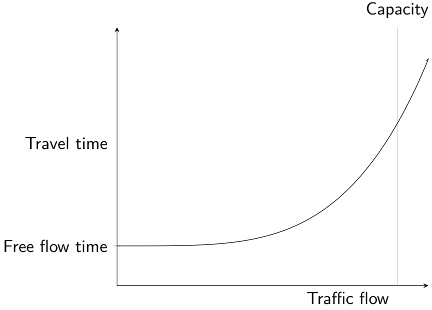
Department of Engineering Science
University of Auckland

Contents

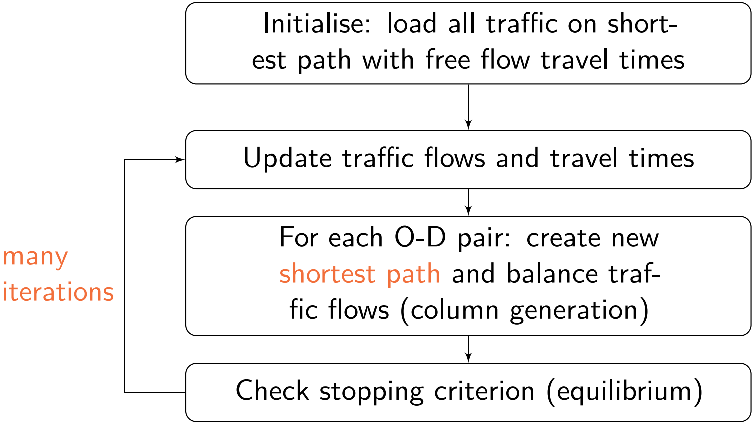
1. The traffic assignment problem
2. Find faster shortest path algorithms
3. Avoiding shortest path calculations
4. Conclusion and future work

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)
- Path distance is measured by non-linear travel times for capturing congestion effects



Path equilibration algorithm



Initialise: load all traffic on shortest path with free flow travel times

Update traffic flows and travel times

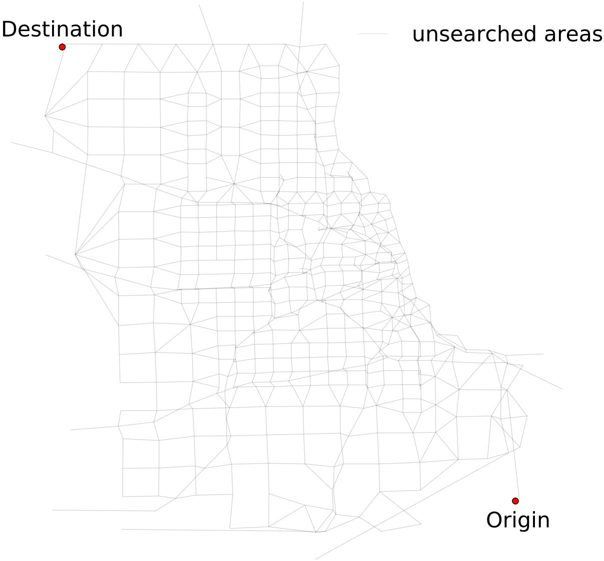
For each O-D pair: create new shortest path and balance traffic flows (column generation)

Check stopping criterion (equilibrium)

many iterations

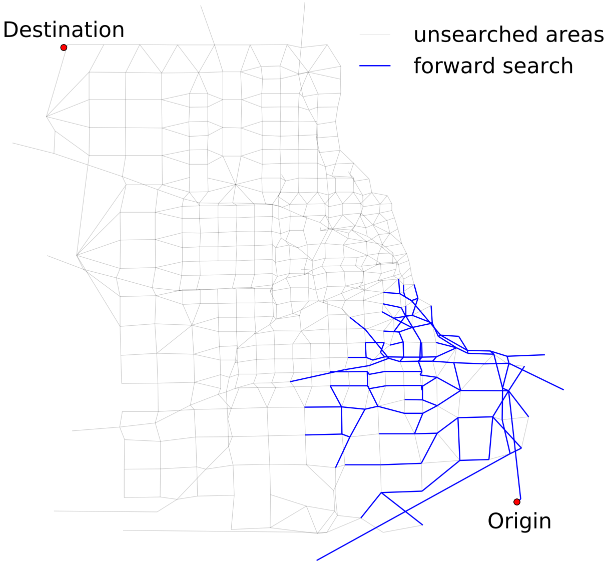
- Requires millions of shortest paths to be found

Dijkstra's algorithm



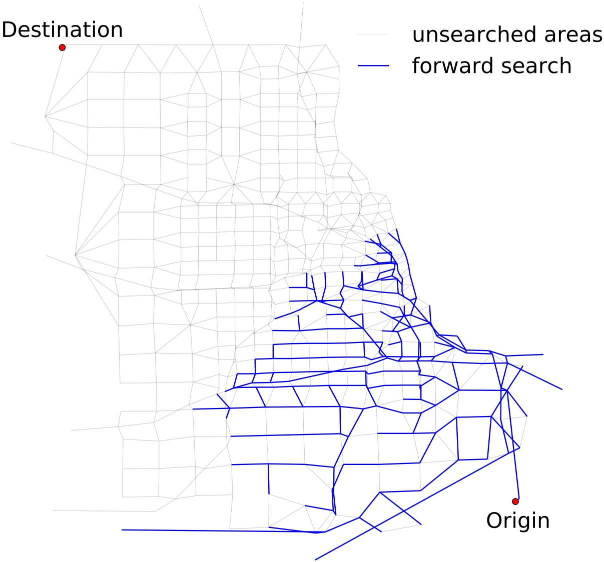
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



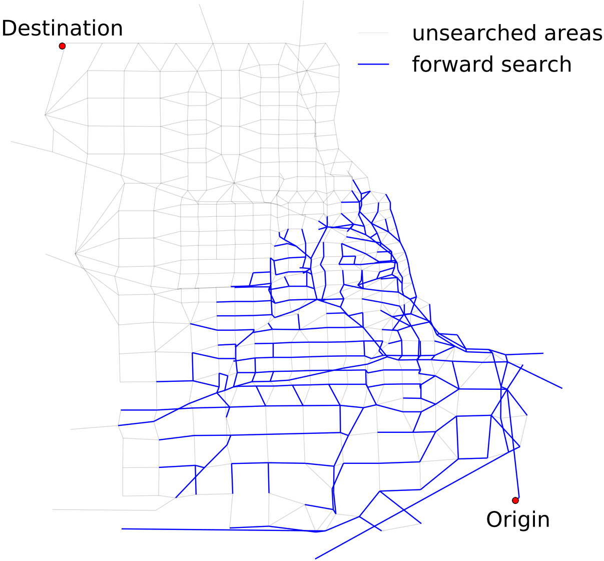
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



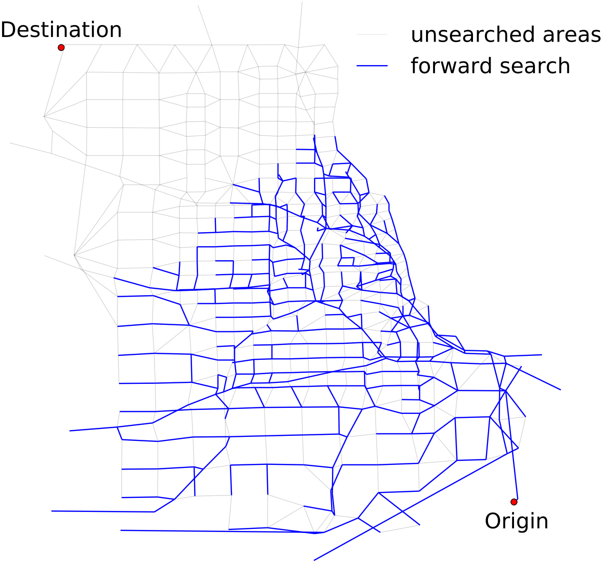
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



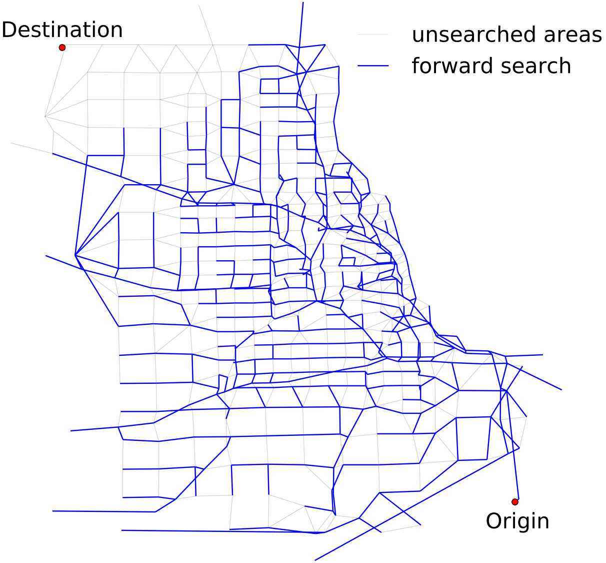
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra’s algorithm



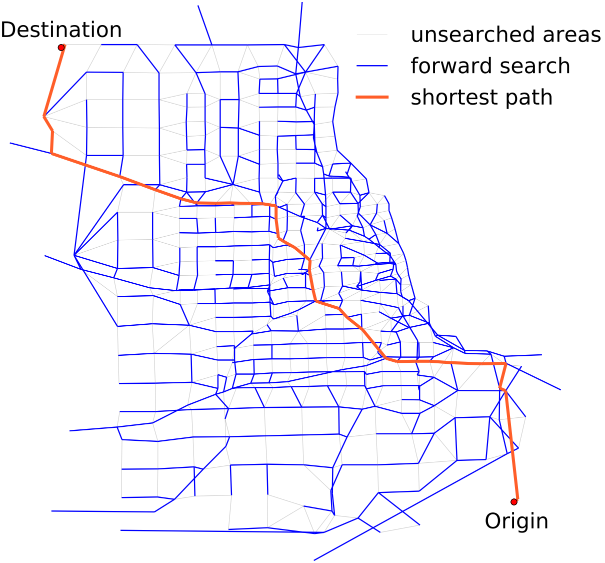
- Chicago Sketch network
- 93,135 O-D pairs
- 546 nodes
- 2,950 arcs

Dijkstra's algorithm



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

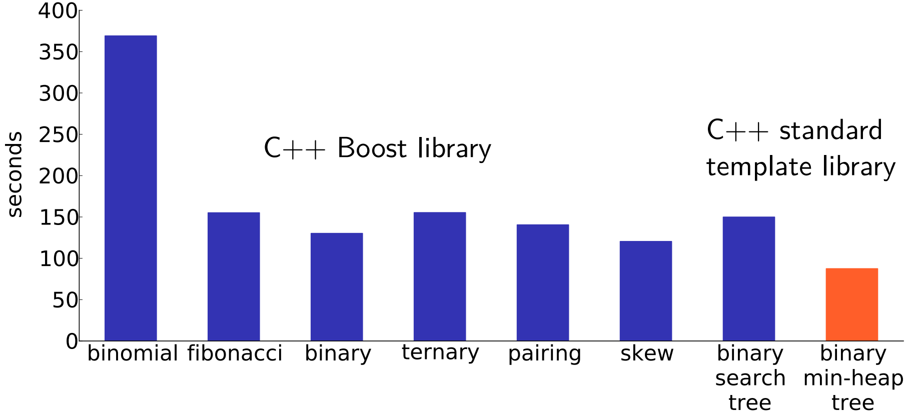
Dijkstra's algorithm



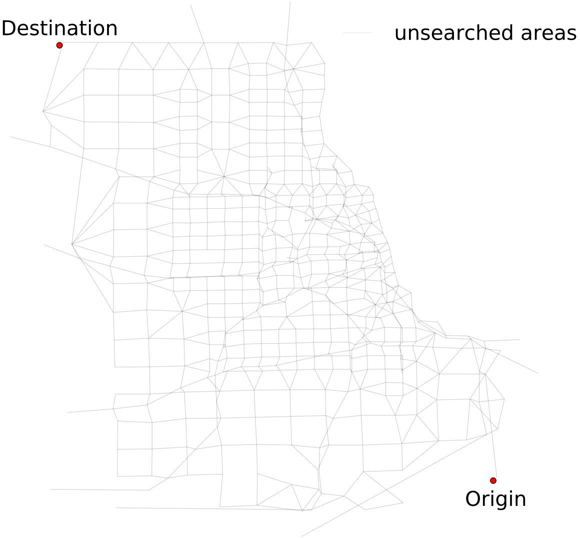
- $\min_{u \in \mathcal{Q}} [d_u]$
- d_u: shortest path from origin to u
- $\mathcal{Q}$: set of labelled nodes

Priority queues

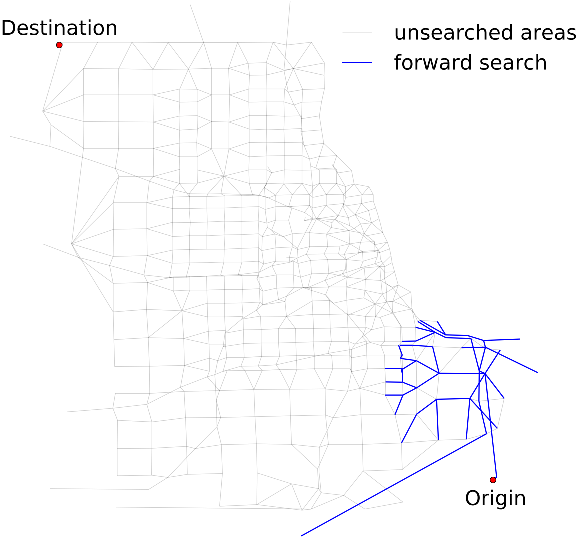
- Priority queue - a data structure for storing the searched nodes in priority order so the next location to search can be found easily
- $O(1)$ extract minimum, $O(\log(n))$ insert



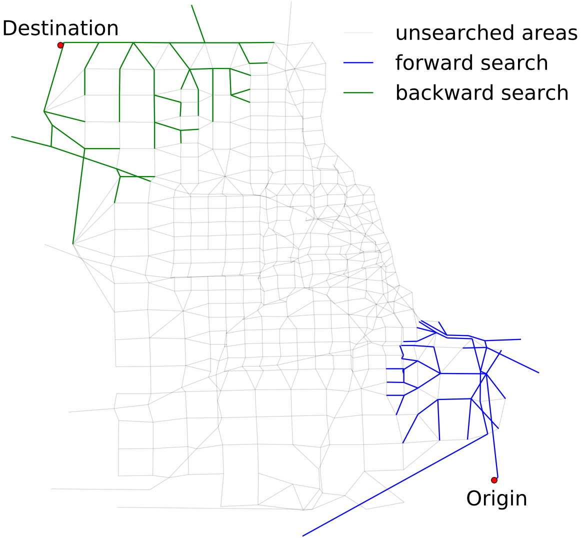
Bidirectional Dijkstra's algorithm



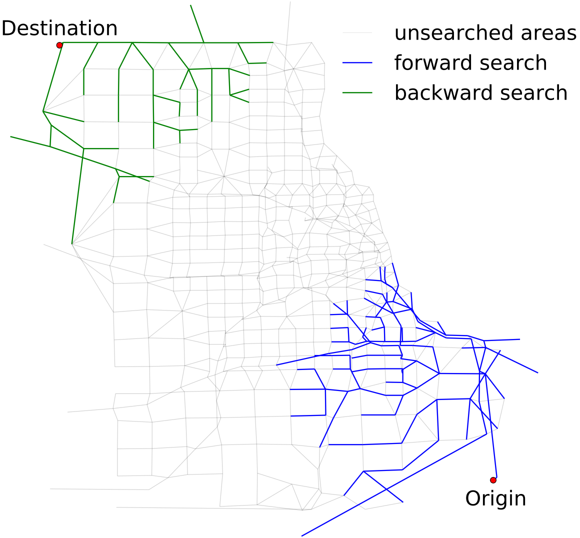
Bidirectional Dijkstra's algorithm



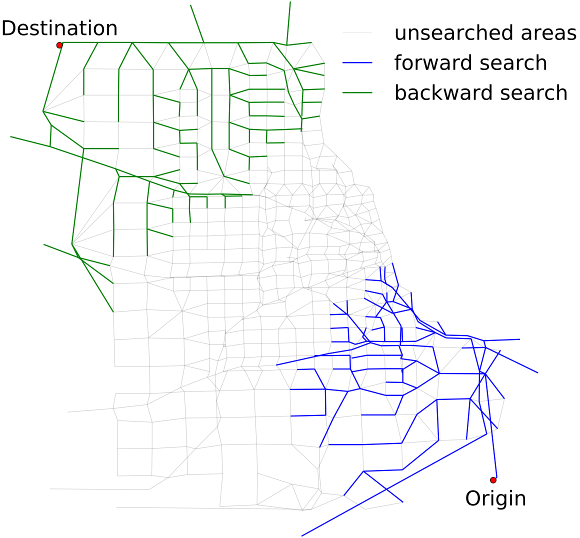
Bidirectional Dijkstra's algorithm



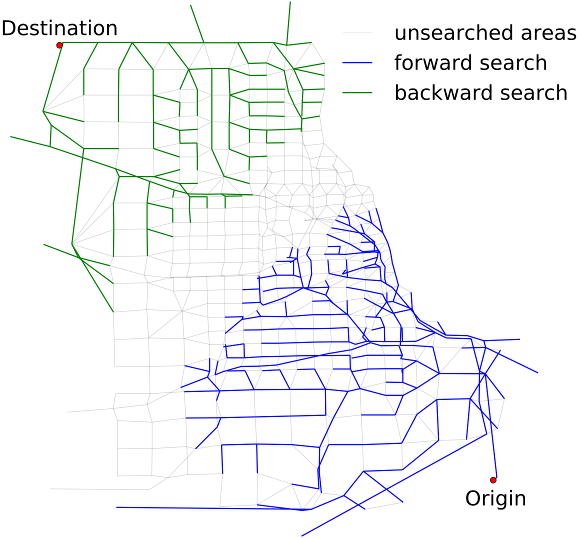
Bidirectional Dijkstra's algorithm



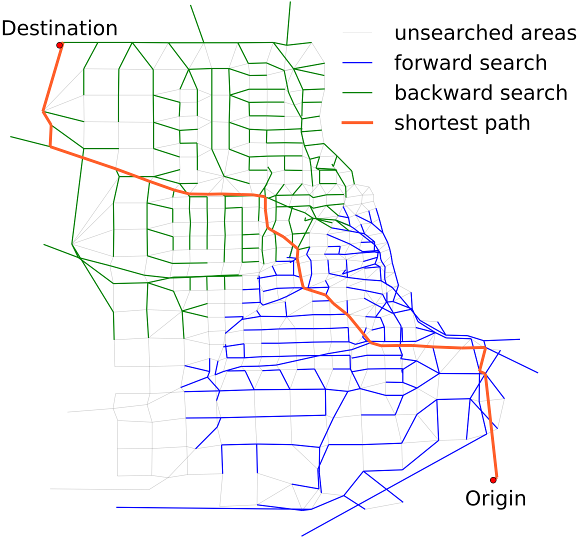
Bidirectional Dijkstra's algorithm



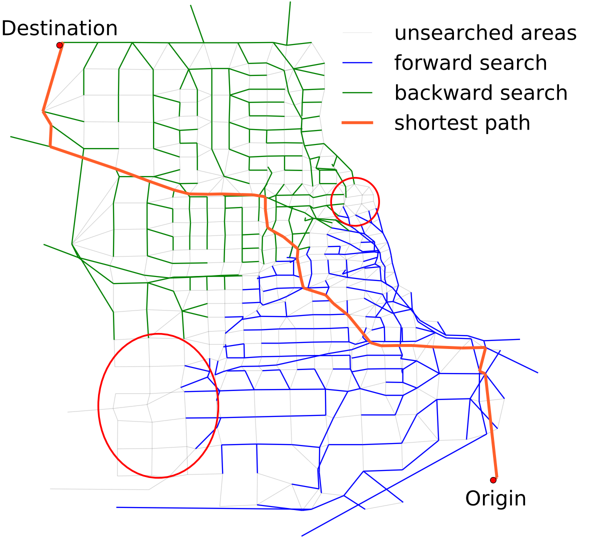
Bidirectional Dijkstra's algorithm



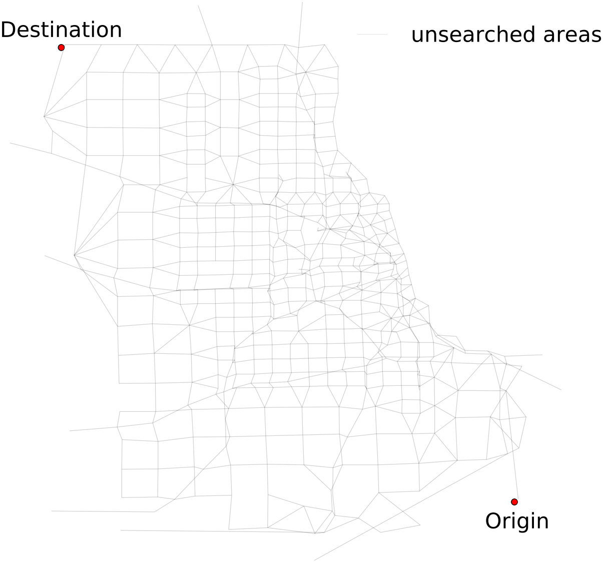
Bidirectional Dijkstra's algorithm



Bidirectional Dijkstra's algorithm

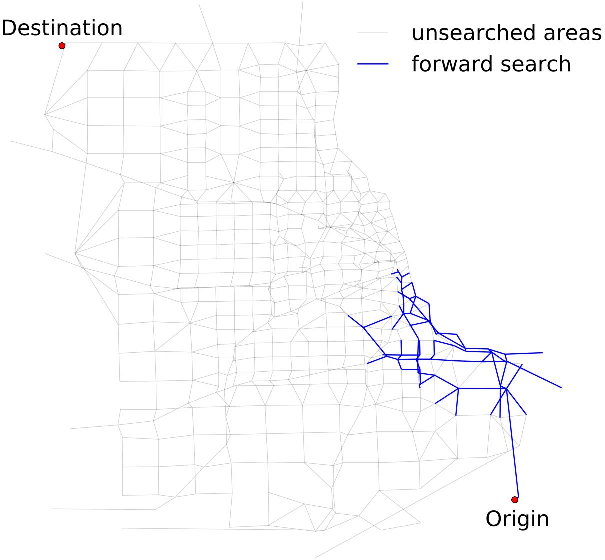


A* search (goal-directed search)



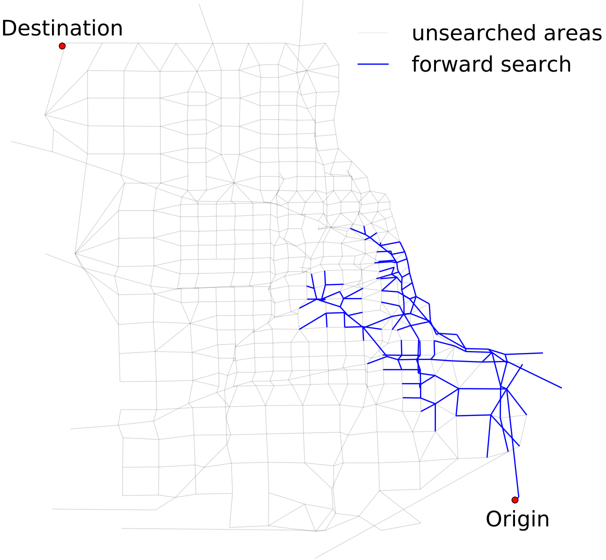
- $\min_{u \in \mathcal{Q}} [d_u + h_u]$
- d_u: shortest path from origin to u
- h_u: shortest path estimate from u to destination
- $\mathcal{Q}$: set of labelled nodes

A* search (goal-directed search)



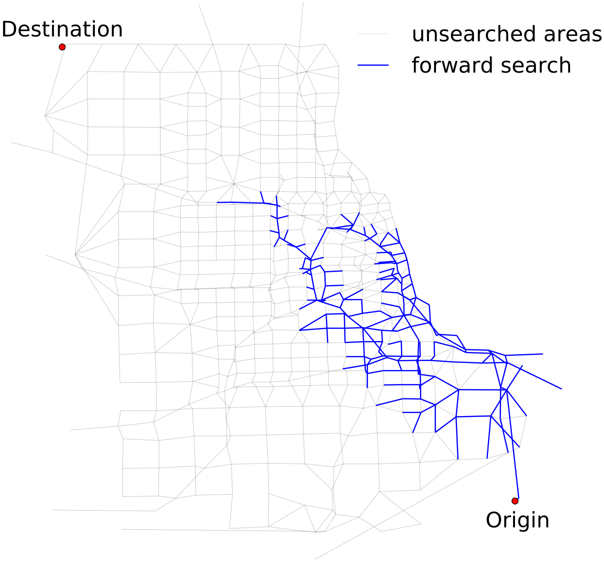
- $\min_{u \in \mathcal{Q}} [d_u + h_u]$
- d_u: shortest path from origin to u
- h_u: shortest path estimate from u to destination
- $\mathcal{Q}$: set of labelled nodes

A* search (goal-directed search)



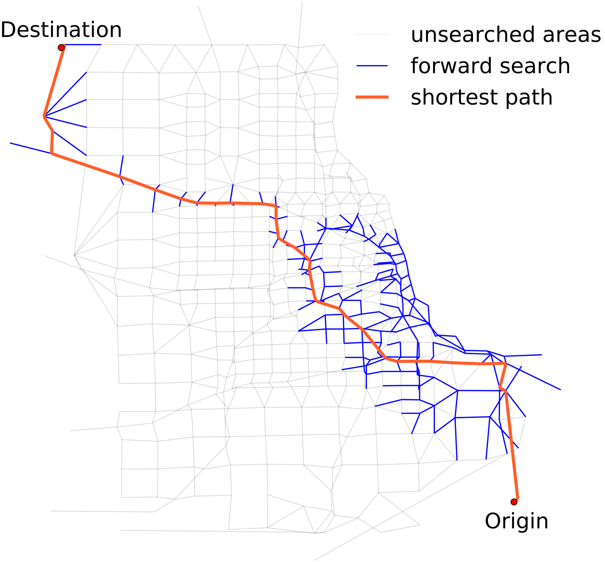
- $\min_{u \in \mathcal{Q}} [d_u + h_u]$
- d_u: shortest path from origin to u
- h_u: shortest path estimate from u to destination
- $\mathcal{Q}$: set of labelled nodes

A* search (goal-directed search)



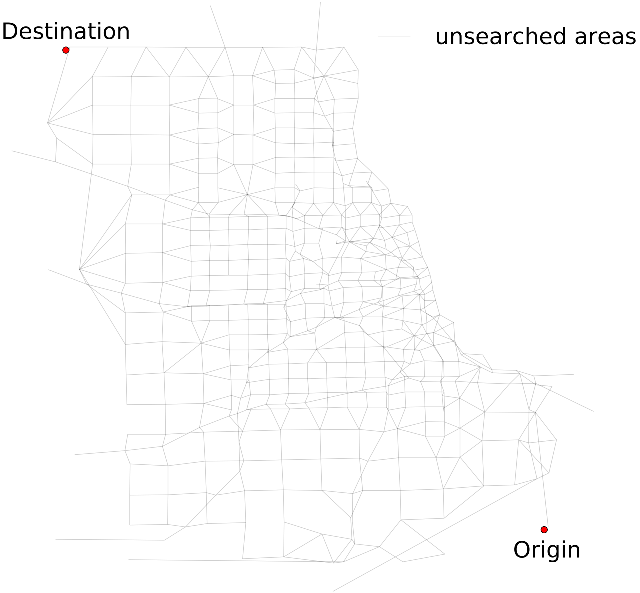
- $\min\limits_{u \in \mathcal{Q}} [d_u + h_u]$
- d_u: shortest path from origin to u
- h_u: shortest path estimate from u to destination
- $\mathcal{Q}$: set of labelled nodes

A* search (goal-directed search)

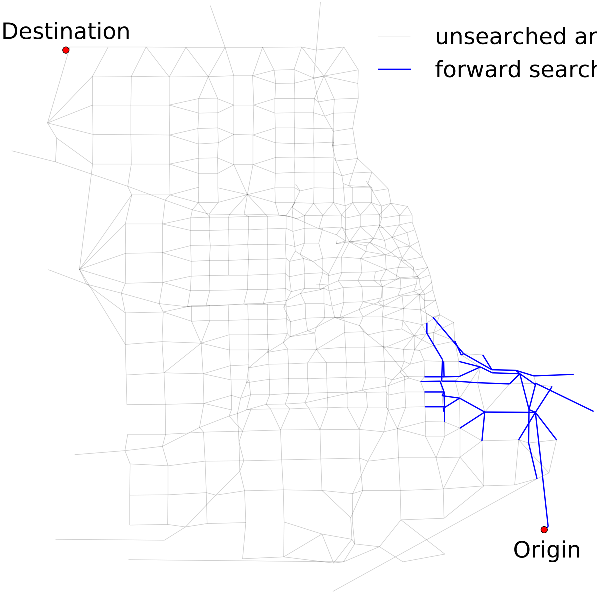


- $\min\limits_{u \in \mathcal{Q}} [d_u + h_u]$
- d_u: shortest path from origin to u
- h_u: shortest path estimate from u to destination
- $\mathcal{Q}$: set of labelled nodes

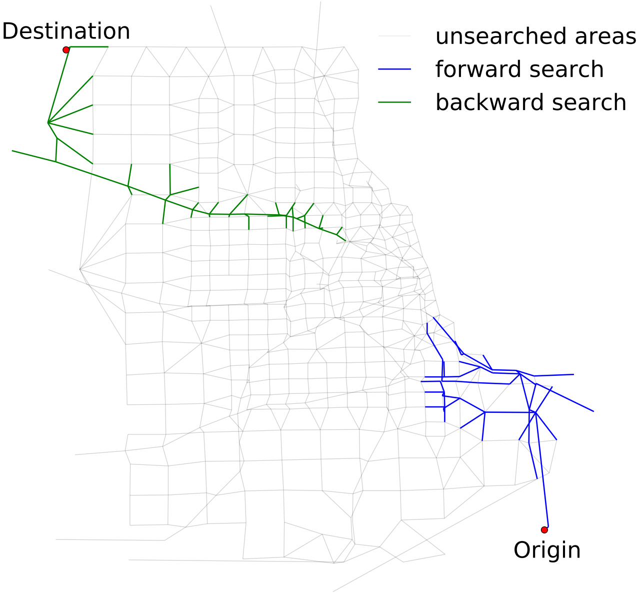
Bidirectional A* search



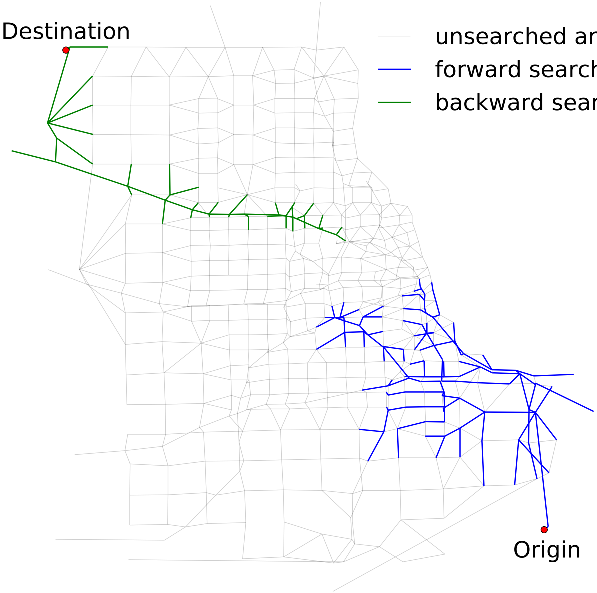
Bidirectional A* search



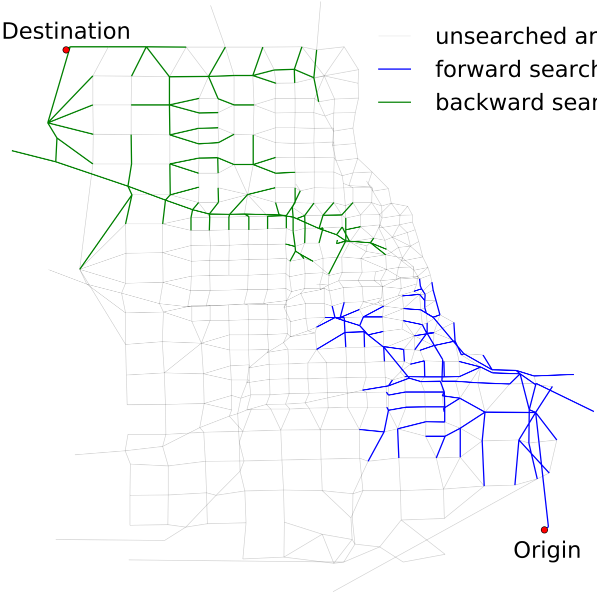
Bidirectional A* search



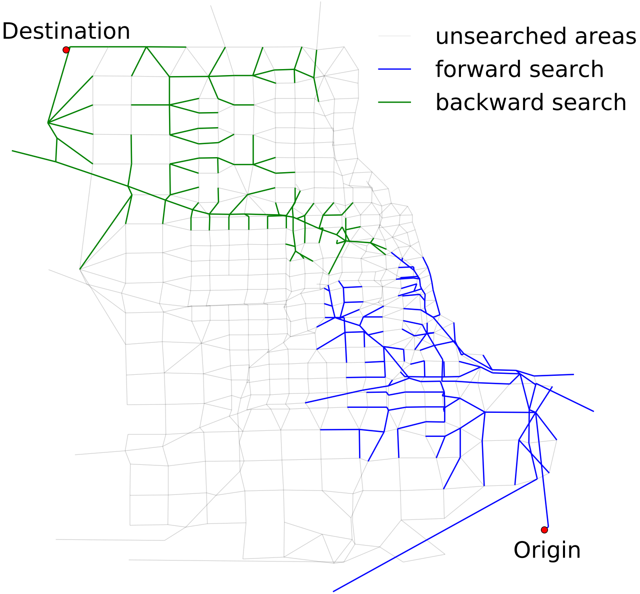
Bidirectional A* search



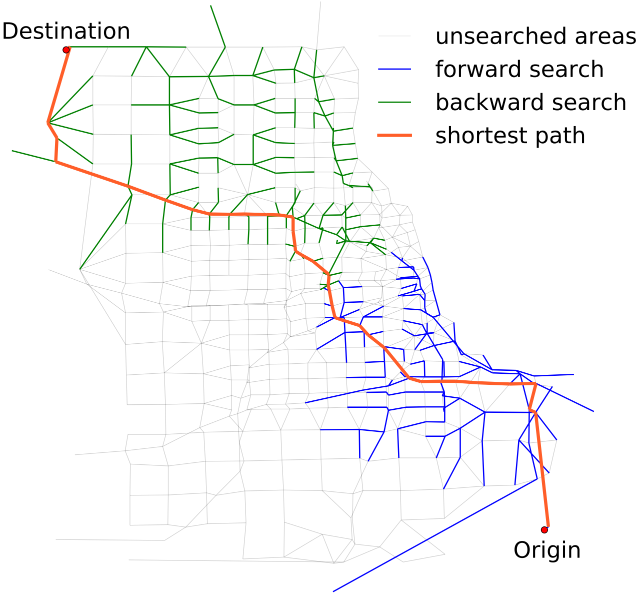
Bidirectional A* search



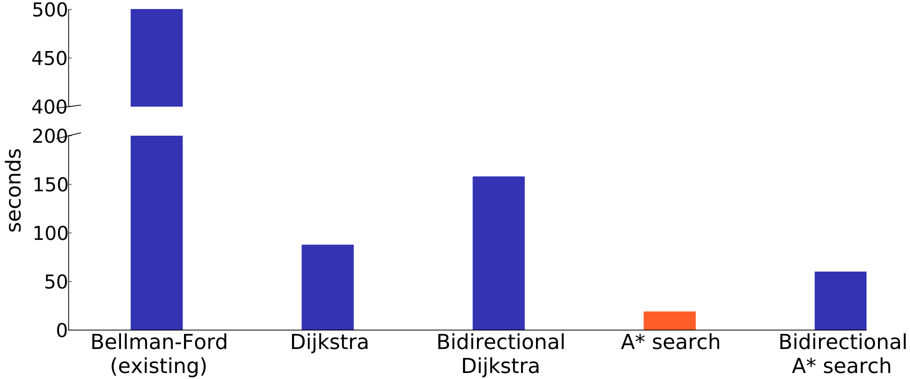
Bidirectional A* search



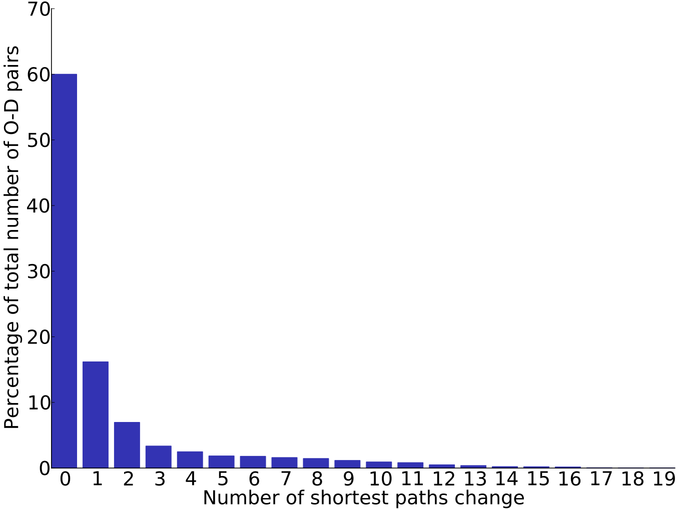
Bidirectional A* search



Shortest path algorithm results

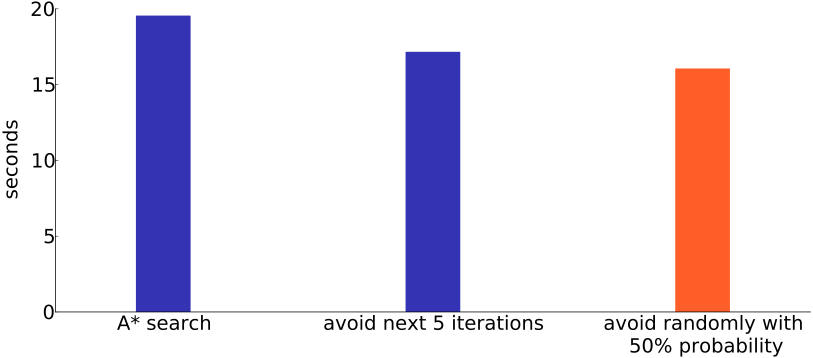


Shortest paths change between iterations for Path Equilibration



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



Conclusion and future work

Conclusion

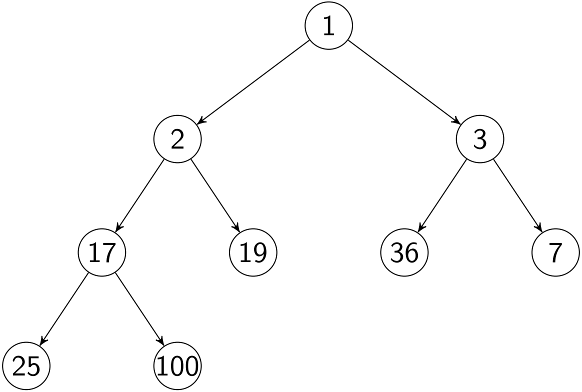
- Best performance: A* search algorithm using min-heap tree with random avoiding strategy
- 30 times faster than the existing implemented Bellman-Ford algorithm
- Bidirectional algorithms are worse compared to the unidirectional ones

Future work

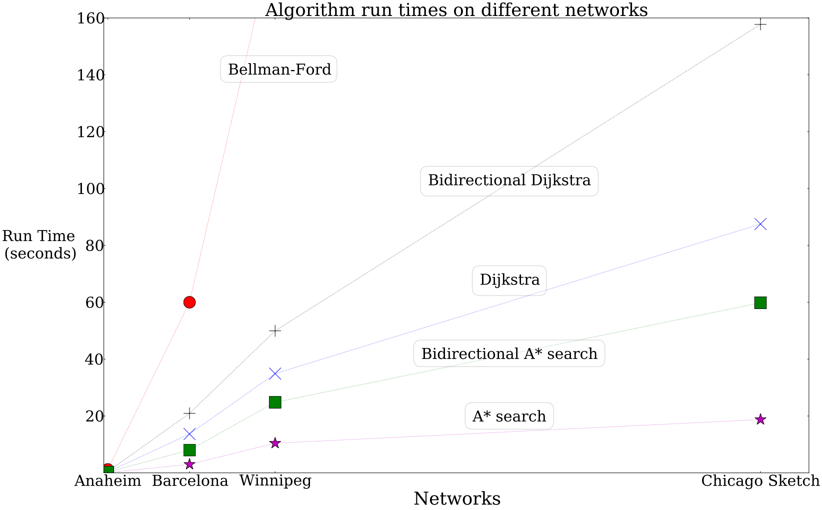
- Pre-processing: A* search with landmarks
- Multi-thread on GPU
- Test the avoiding strategies on other algorithms that solve the traffic assignment problem

Appendix

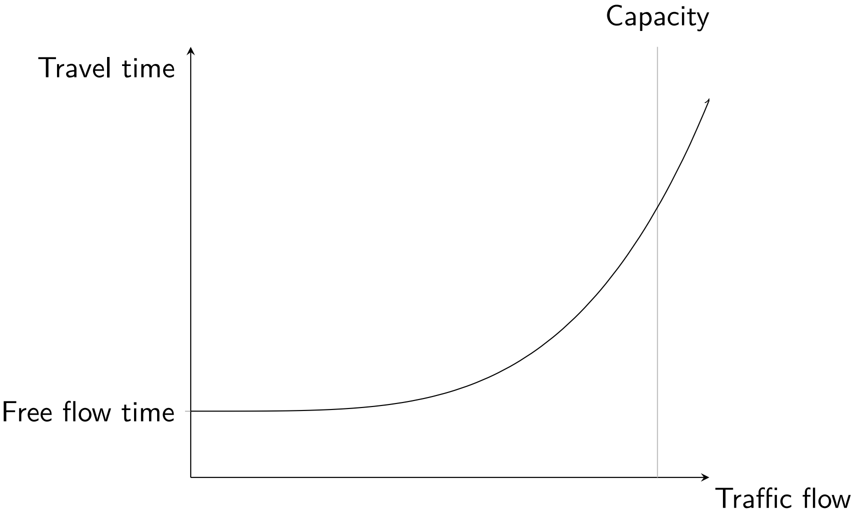
Binary min-heap tree



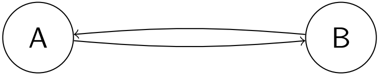
Shortest path algorithm results on different networks



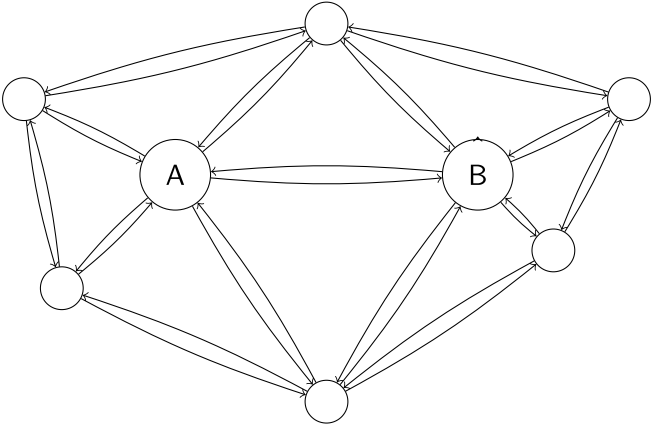
Non-linear travel time function



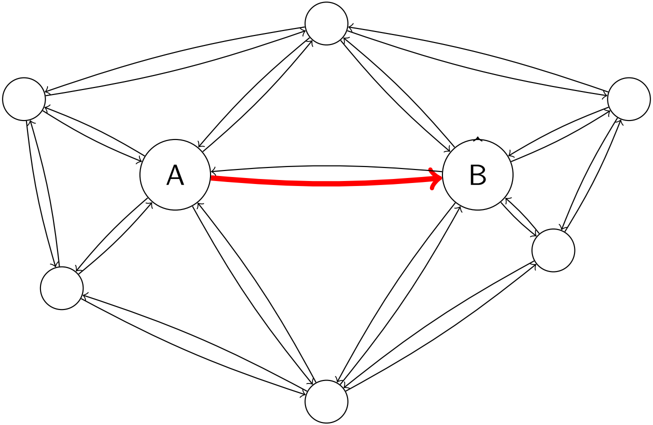
Traffic assignment illustration



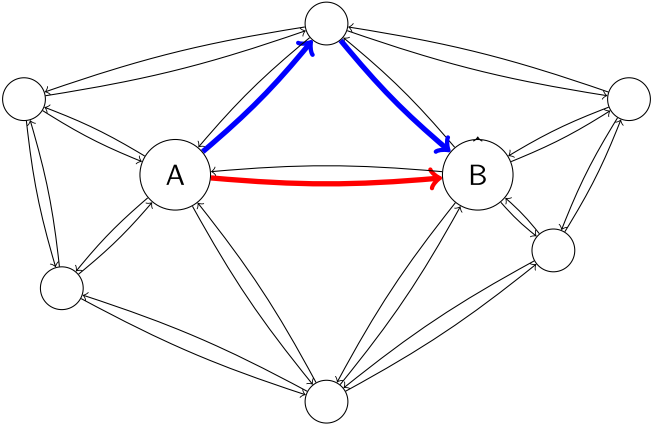
Traffic assignment illustration



Traffic assignment illustration



Traffic assignment illustration



Traffic assignment illustration