TSFS12 Hand-in 1: Discrete Path Planning in a Structured Road Network

JOHN DOE
E-mail: email@example.com
LiU-ID: johndoe

SAMUEL ÅKESSON
E-mail: email@example.com
LiU-ID: sermuns

2025-11-24

1. Introduction

This hand-in exercise is to implement and explore properties of common search algorithms in the scenario of finding travel routes in a map of Linköping.

2. Results

Exercise 3.1

The state space χ contains all intersections in the given map of Linköping.

$f(x, u)$ moves from x to x', where x and x' always are adjacent intersections in the map.

$U(x)$ contains all possible movements toward intersections adjacent to current one.

f_next returns three vectors where same indices correspond to same neighbor (neighbor, action, cost).

Exercise 3.2

More complex missions require visiting more nodes and therefore are more time-consuming.

A city full of irregularly located intersections could be approximated with a grid of regularly spaced intersections. One can assume that in this map, a longer euclidean distance between the start and goal node is undeniably tied to the number of intersections between these two nodes.

Four classic search algorithms were implemented to find a path between any given start and goal node in the Linköping map:

- Best first
- Dijkstra
- Breadth first
- A*

The results of these algorithms are presented in the figures below. Figure 1 shows the plan length and Figure 2 shows the number of expanded nodes.

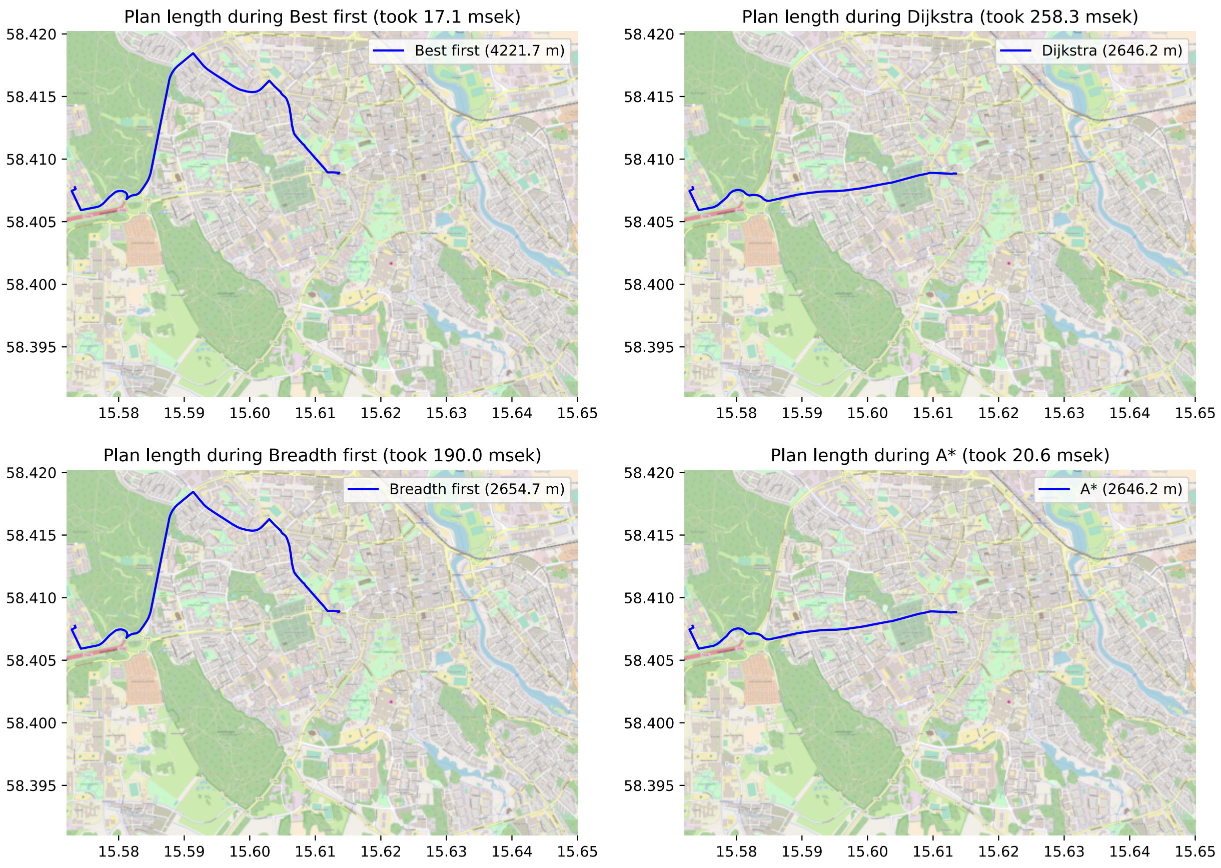
Plan length during Best first (took 17.1 msek)
Best first (4221.7 m)
Plan length during Dijkstra (took 258.3 msek)
Dijkstra (2646.2 m)
Plan length during Breadth first (took 190.0 msek)
Breadth first (2654.7 m)
Plan length during A* (took 20.6 msek)
A* (2646.2 m)
58.420
58.415
58.410
58.405
58.400
58.395
15.58
15.59
15.60
15.61
15.62
15.63
15.64
15.65

Figure 1: Plan length for four algorithms.

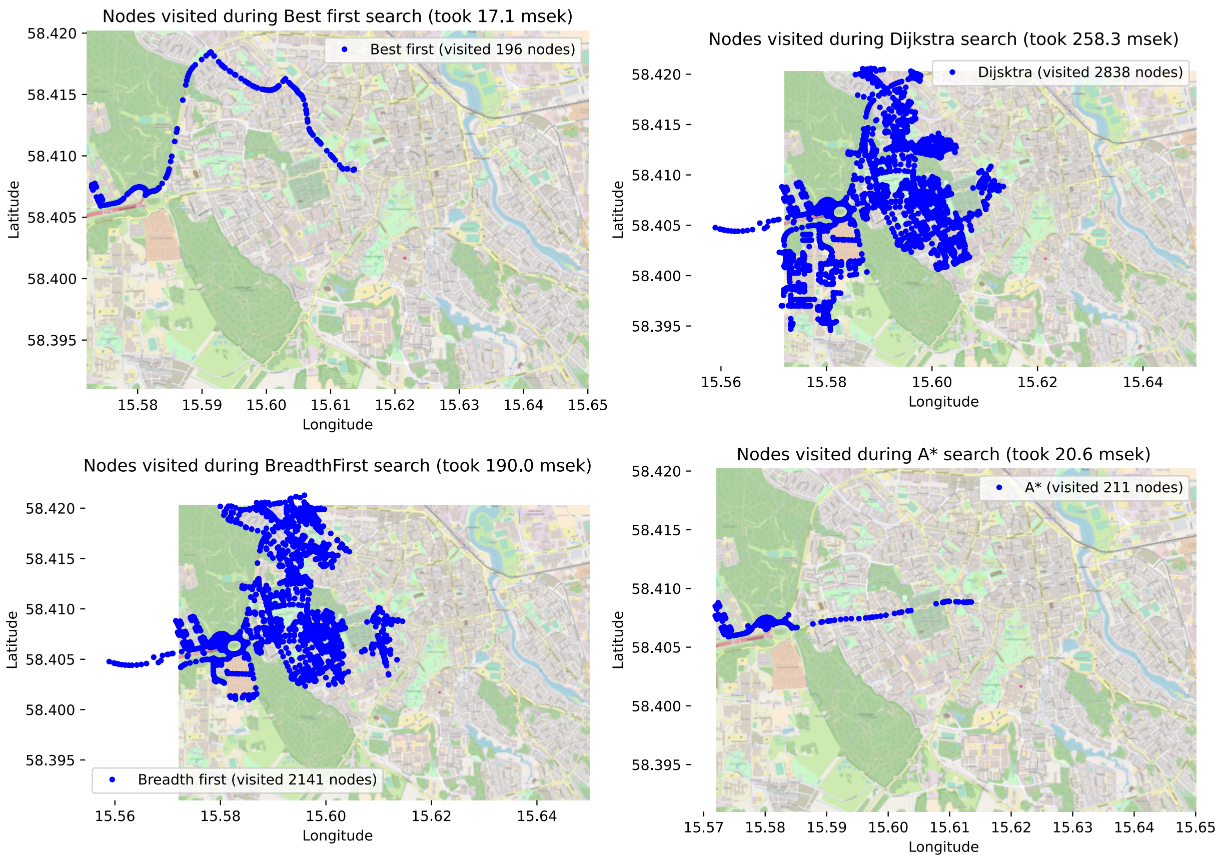
Nodes visited during Best first search (took 17.1 msek)
Best first (visited 196 nodes)
Nodes visited during Dijkstra search (took 258.3 msek)
Dijkstra (visited 2838 nodes)
Nodes visited during BreadthFirst search (took 190.0 msek)
Breadth first (visited 2141 nodes)
Nodes visited during A* search (took 20.6 msek)
A* (visited 211 nodes)
Latitude
Longitude

Figure 2: Number of expanded nodes for four algorithms.

A* is an optimized version of Dijkstra's. Both algorithms produce the same path in the end, but A* does so in a more efficient way, visiting fewer nodes in the process.

A few missions were performed, to compare A* and Dijkstra's against each other. The result is visualized as a scatter plot with Planning time VS. mission length in Figure 3.

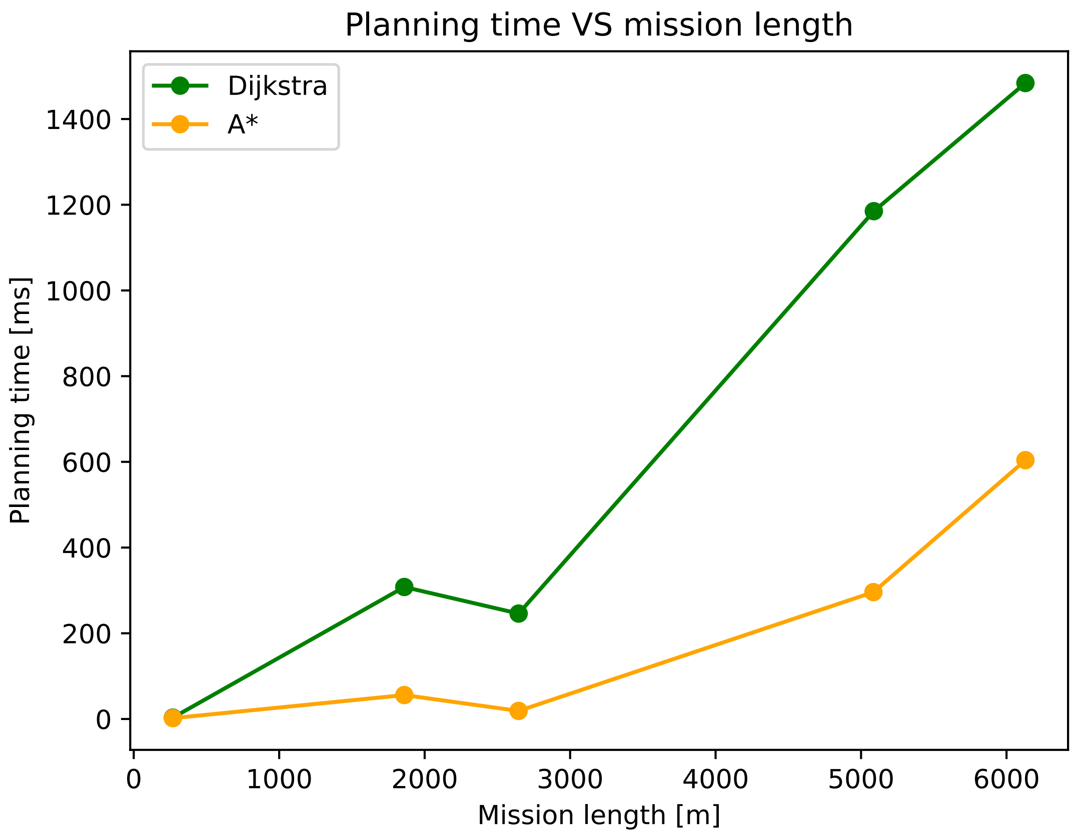

Figure 3: Comparing the performance of Dijkstra vs A*

Exercise 3.3

The best first planner is a very fast planner however would not be a good fit if the path requires backtracking. Whereas breadth is a rather dull planner, it searches in all directions and picks the first path which is not necessarily the most optimal.

Dijkstra always finds the optimal path but is quite slow compared to A* that also always finds the optimal path but is noticeably faster.

The best option would be to always pick the A* planner over Dijkstra. Breadth first would be a good option if for some reason the heuristic function can not be used since breadth first is the only planner that does not require the heuristic function.

Exercise 3.4

If $h(x) > h^*(x)$ (admissibility), the solution is not guaranteed to be optimal.

If $h(x) > d(x, y) + h(y)$ (consistency), the solution will not be found in an efficient manner.

If a heuristic function is admissible, but not consistent, the planner will find the optimal solution, but not in an efficient manner. With a non admissible heuristic a solution will be found but it will not be the optimal solution. However, it may be faster. A both admissible and a consistent heuristic function will find the optimal solution and in an efficient manner.

Exercise 3.5

Euclidean distance would probably be a bad heuristic function for navigating a labyrinth-like environment. The reason for this is that progress toward the goal is not at all guaranteed to be tied to the distance to it, as progressing often involves moving through winding paths that could result in *increasing* euclidean distance to goal node, a "backtracking problem". This is illustrated in Figure 4.

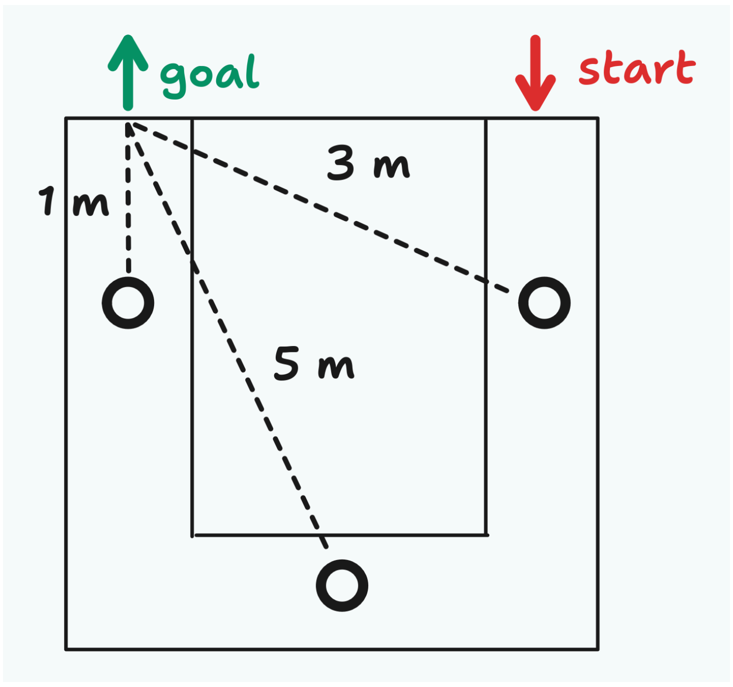

Figure 4: Illustrating the backtracking problem.

Exercise 3.6

A simple way to create the inflated heuristic function $h_c(x)$ is to scale the heuristic function $h(x)$ by a constant c.

$$h_c(x) = c \cdot h(x) \tag{1}$$

By inflating, we risk making the heuristic inadmissible, which makes solutions found using it no longer be guaranteed to be optimal.

40 randomly generated missions were performed, with inflation factor $c = 2^i$ for $i = -4, -3, \ldots, 11, 13$. The results are presented in Figure 5, with each colored line representing a different inflation factor c.

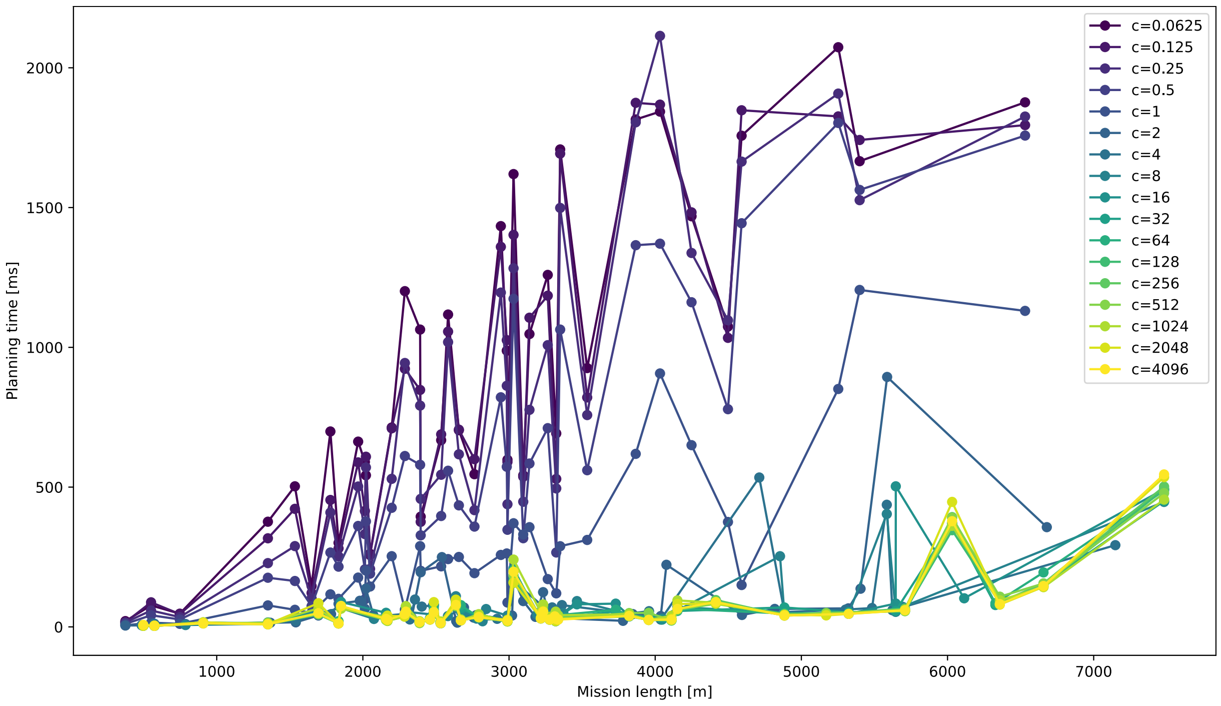

Figure 5: Comparing the effect of different inflation factors on A*.

Two notes on the extremes:

- Deflating the heuristic ($c < 1$), we move further away from the efficient A* algorithm, and eventually fall into Dijkstra's at $c = 0$.
- Inflating towards infinity, the algorithm essentially becomes a best-first algorithm, since the heuristic becomes so much more favored than the cost function. This however, makes the algorithm faster.

3. Discussion

Choosing the right search algorithm for the job boils down to a few aspects. In most cases, you want to pick A*, since it finds the optimal path in an efficient manner. However, if the planning is time-limited, you need to perform an inflated search, possibly to the point where you're better off just choosing best-first search. On the other hand, if there is no way to construct a heuristic function, neither A* nor best-first is on the table. Your only bet is depth- or breadth-first search. Which one to choose depends on the structure of the state-space.