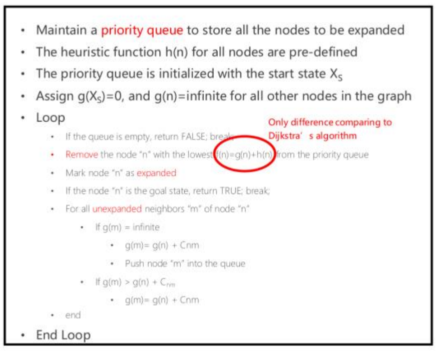
A* search algorithm



- Maintain a priority queue to store all the nodes to be expanded
- The heuristic function h(n) for all nodes are pre-defined
- The priority queue is initialized with the start state X_S
- Assign $g(X_S)=0$, and g(n)=infinite for all other nodes in the graph
- Loop
  - If the queue is empty, return FALSE; break;
  - Remove the node "n" with the lowest $f(n)=g(n)+h(n)$ from the priority queue
  - Mark node "n" as expanded
  - If the node "n" is the goal state, return TRUE; break;
  - For all unexpanded neighbors "m" of node "n"
    - If g(m) = infinite
      - g(m)= g(n) + Cnm
      - Push node "m" into the queue
    - If $g(m) > g(n) + C_{nm}$
      - g(m)= g(n) + Cnm
  - end
- End Loop

Simulation Result

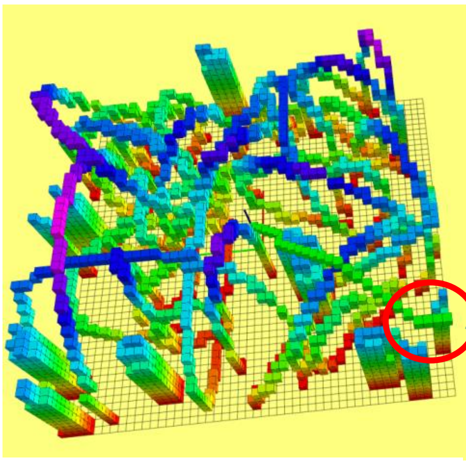


Fig. 1: A* search

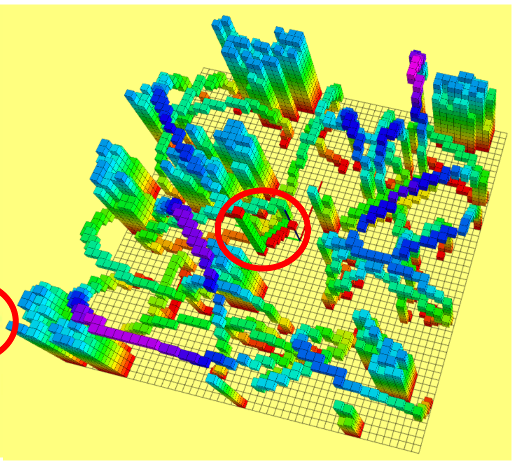


Fig. 2: A* search and JPS

Different Heuristics comparison

5 random maps for each heuristic, target is the same, which locates at the right-down corner with z equal to 0. In the following Table: h1 is Manhattan; h2 is Euclidean; h3 is Diagonal Heuristic; TB is Tie Breaker. h1 might overestimate the path cost, it is not admissible but greedier. It shows less visited nodes. h2 and h3 are admissible and h3 is closer to the real path cost, which shows better performance compared to h2.

Table 1 Heuristics Comparison

	Map 1	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Average**
Run time /ms (h1)	0.136	0.363	0.233	0.221	0.534	0.297
Visited node sizes (h1)	29	27	28	26	40	30

Run time /ms (h2)	1.499	0.392	3.686	5.897	4.658	3.226
Visited node sizes (h2)	998	237	771	505	2218	946
Run time /ms (h3)	0.864	0.600	0.341	0.497	0.642	0.589
Visited node sizes (h3)	70	93	33	130	30	71
Run time /ms (h3+TB)	0.376	0.108	0.101	0.287	0.083	0.191
Visited node sizes (h3+TB)	27	26	40	27	27	29

Tie Breaker

See the table above, because there are many obstacles in the map, tie breaker (slightly increasing h) can significantly reduce the visited nodes perhaps without breaking the optimality.

Problems and Solutions

1. Bug of catkin_make

See link: http://www.shenlanxueyuan.com/course/191/thread/420

2. Segmentation fault

A pointer is pointed to a non-existing memory. I checked whether the index of an array is out of range.

```
if( (n_x < 0) || (n_x > (GLX_SIZE - 1)) || (n_y < 0) || (n_y > (GLY_SIZE - 1) ) || (n_z < 0) || (n_z > (GLZ_SIZE - 1)))
    continue; // to avoid index problem
```

Comparison: A* and JPS

Normally, JPS works better in complicated maps, with a small amount of open space. But A* works better in a map with much open space or in a scenario where the perception of a robot is limited.

Remember **assumption** behind JPS:accessing the contents of many points on a grid in a few iterations of A* is more efficient than maintaining a priority queue over many iterations of A*. Here, it's O(1) versus O(n), though with a better queue implementation it'd be O(1) versus O(n log n).

Reference

1) Lecture 2 SEARCH-BASED PATH FINDING

2) https://zerowidth.com/2013/a-visual-explanation-of-jump-point-search.html