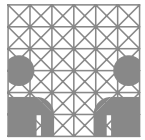


Variants of the Dijkstra and A* algorithms used in path planning

Emy Arts

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics

Technical Aspects of Multimodal Systems

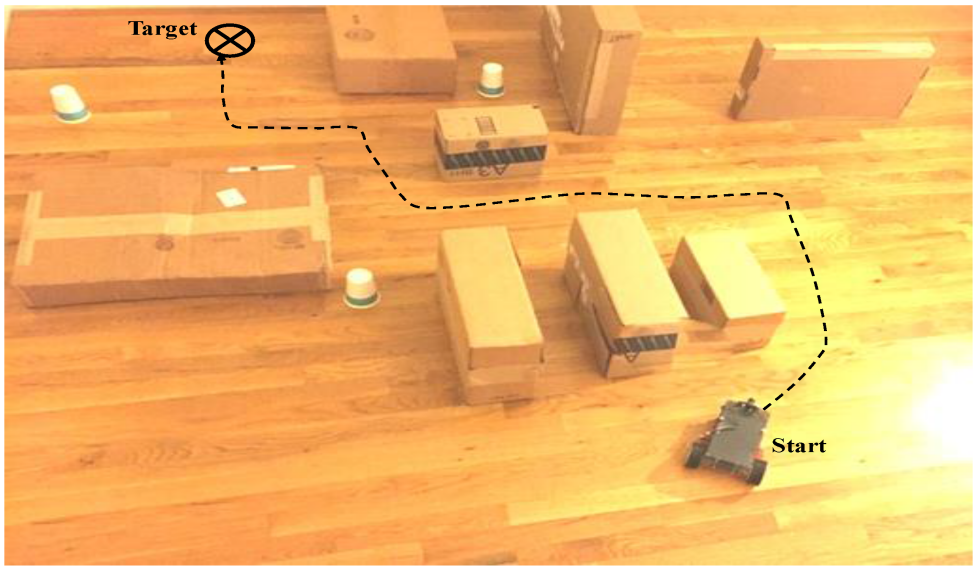
03/12/2018

Outline

Path planning

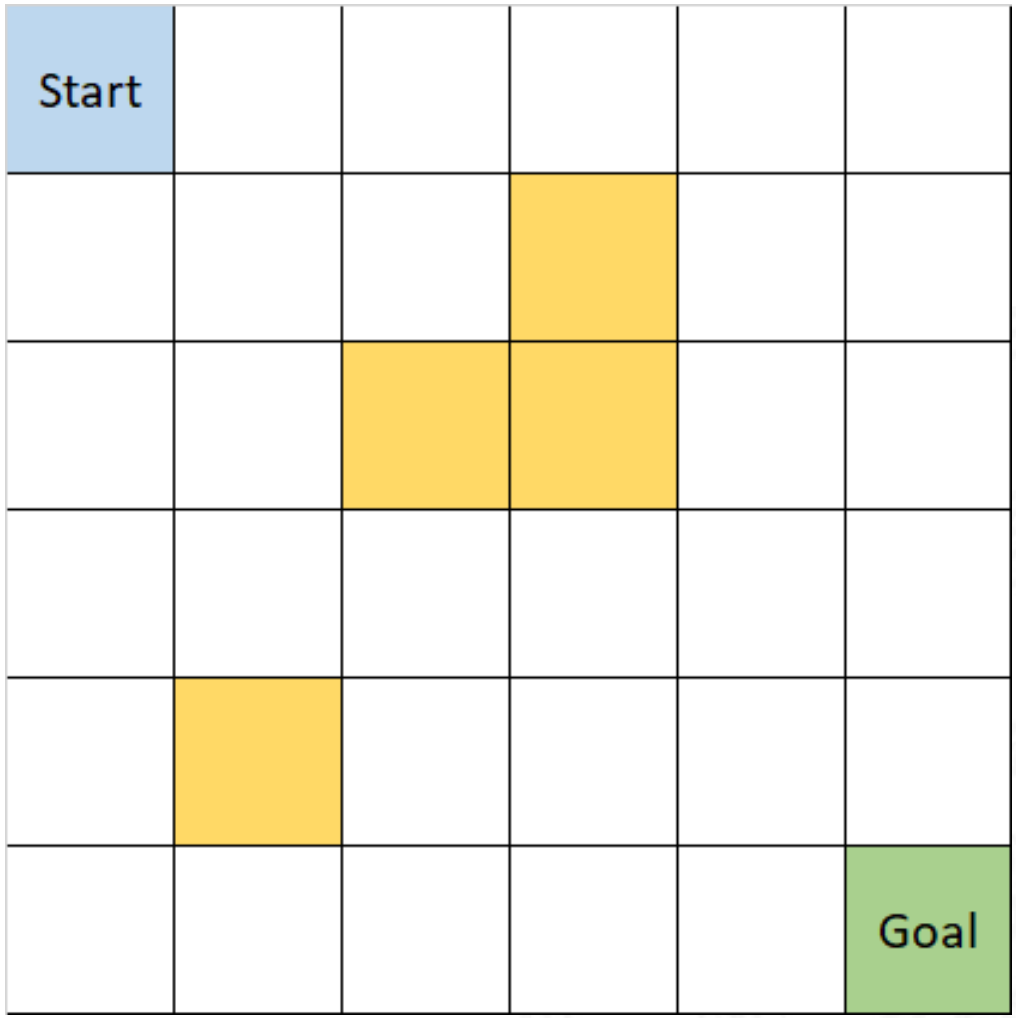
Definition

"A path is a geometric representation of a plan to move from a start to a target pose. The task of planning is to find a collision-free path among a collection of static and dynamic obstacles." [3]

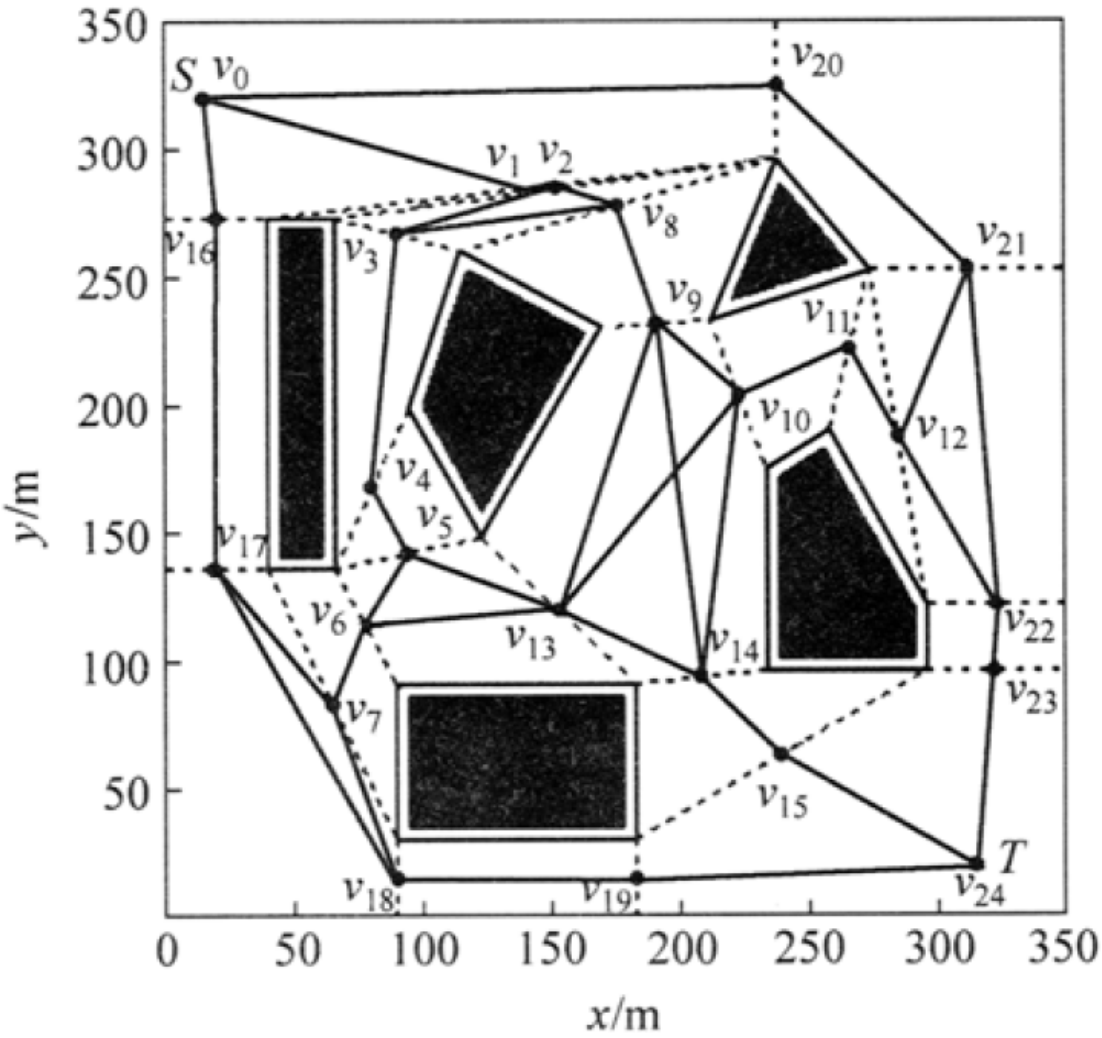


An optimal and energy efficient multi-sensor collision-free path planningalgorithm for a mobile robot in dynamic environments [1]

Space as a graph - Grid-Map

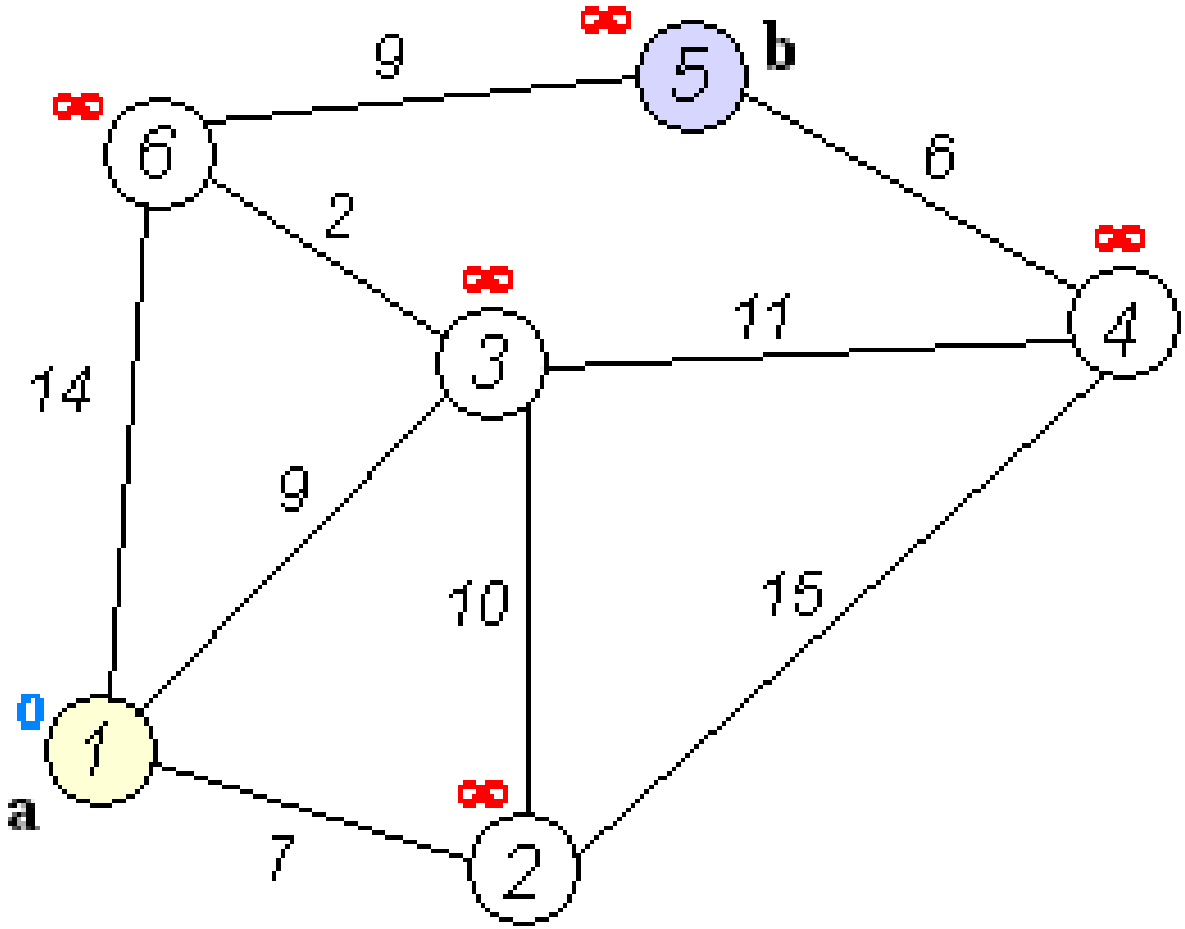


Space as a graph - MAKLINK



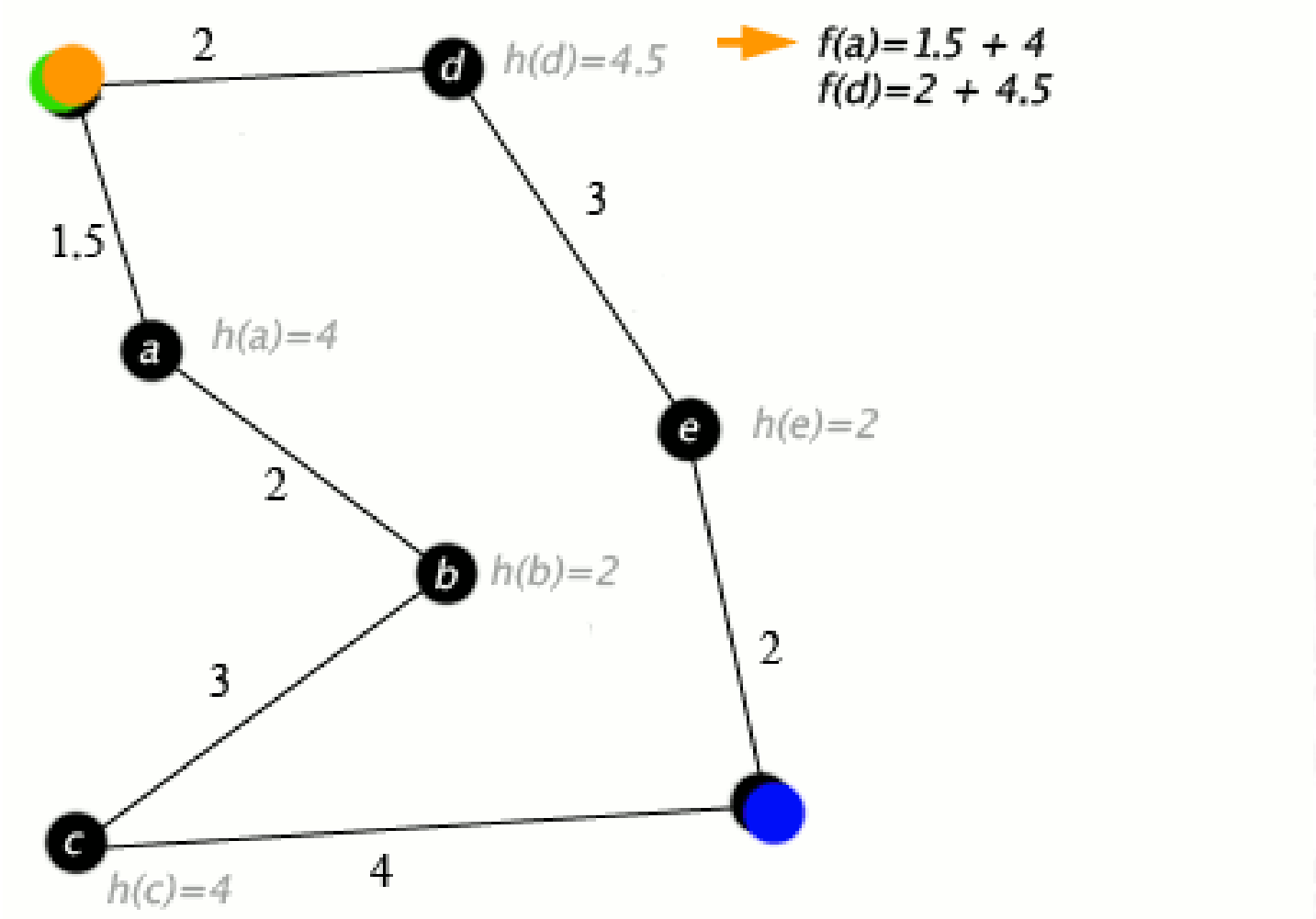
Global optimal path planning for mobile robot based on improved Dijkstra algorithm and ant system algorithm [4]

The Dijkstra algorithm



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

The A* algorithm



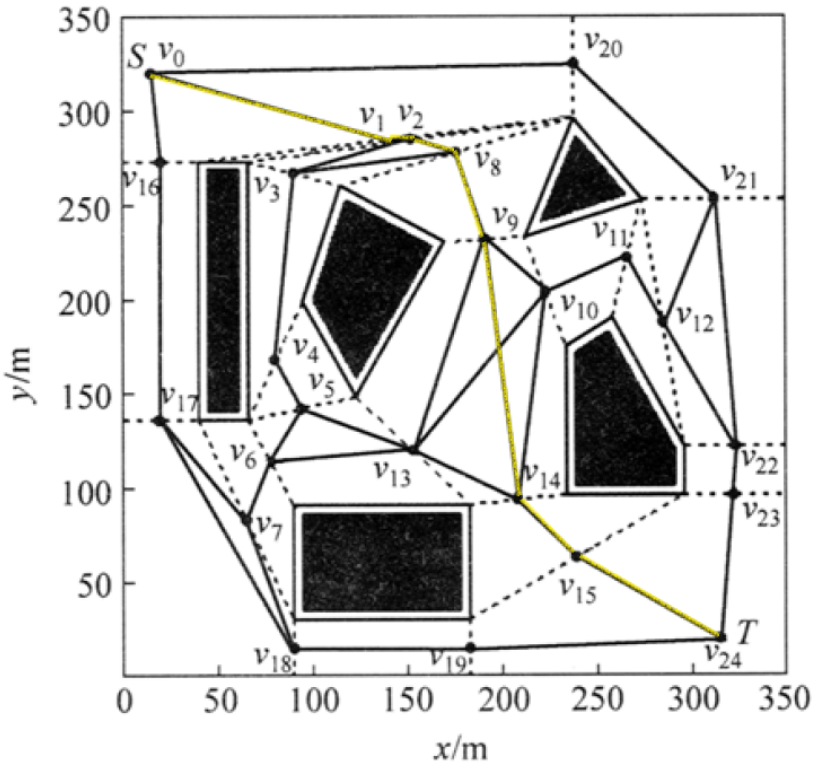
https://en.wikipedia.org/wiki/A*_search_algorithm

What can be improved?

Dijkstra combined with the Ant System Algorithm I

Possible points of improvement:

- The passage through some nodes could be unnecessary
- Path is sub-optimal since edges don't define every possible movement.

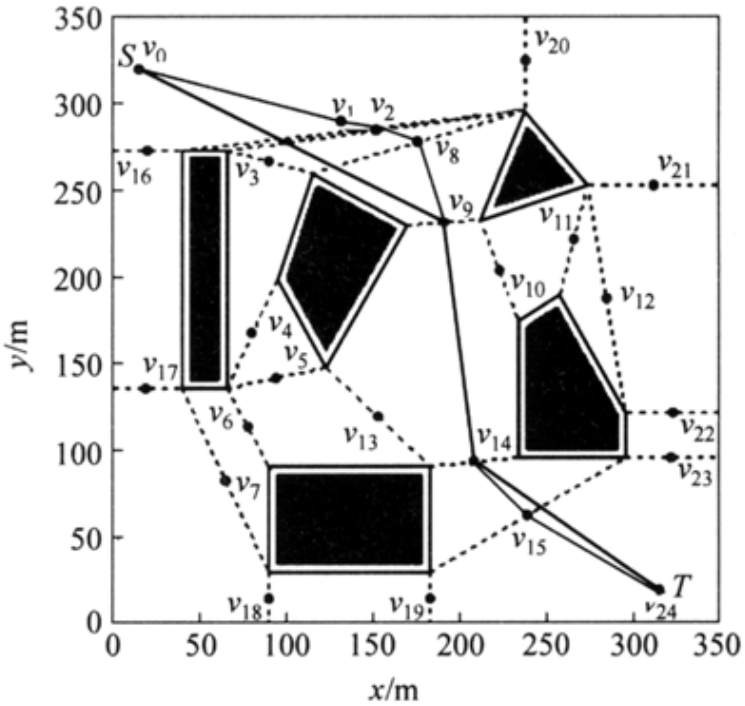


Global optimal path planning for mobile robot based on improved Dijkstra algorithm and ant system algorithm [4]

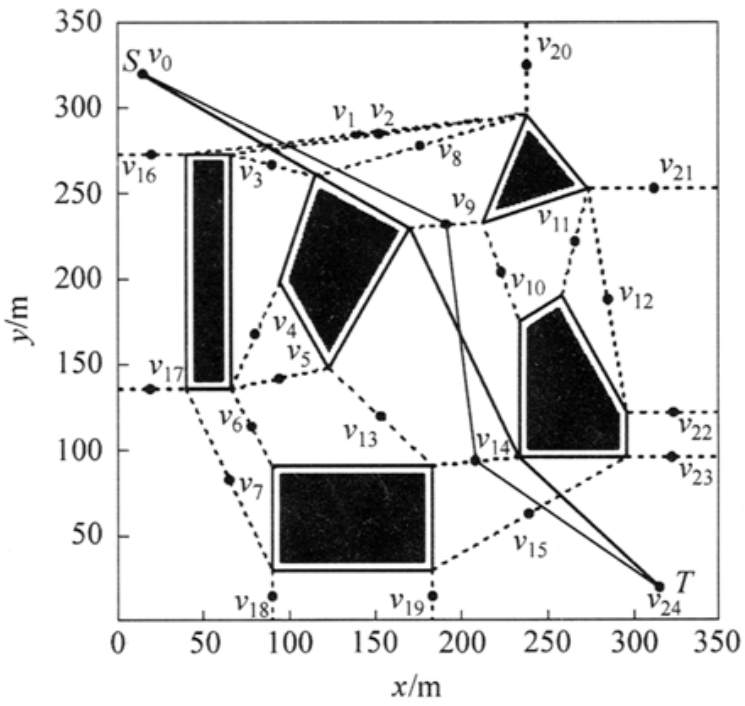
Improved Dijkstra combined with the Ant System Algorithm II

Solution approach

Improve Dijkstra



Apply ant Algorithm



Global optimal path planning for mobile robot based on improved Dijkstra algorithm and ant system algorithm [4]

The standard Dijkstra algorithm


```
   input  : Grid, Start, Goal
   output : dist
 1 for each vertex v in Grid do
 2 |   dist(v)= infinity // Mark distance from
   |   start to each node v as not yet
   |   computed;
 3 |   visited(v)= false // Mark all nodes as
   |   unvisited;
 4 |   previous(v)=undefined // Previous node in
   |   optimal path;
 5 end
 6 dist(Start)=0;
 7 insert Start into Q // Initially, only the
   start node is in the Queue;
 8 while (Q is not empty) and (not visited(Goal)) do
 9 |   u = not visited vertex in Q with smallest distance;
10 |   remove u from Q;
11 |   visited(u)=true // mark this node as
   |   visited;
12 |   for each neighbor v of u do
13 |   |   g = dist(u) + distEdge(u, v) // distance
   |   |   from Start;
14 |   |   if g ≤ dist(v) then
15 |   |   |   dist(v)=g;
16 |   |   |   previous(v)=u ;
17 |   |   |   if !visited(v) then
18 |   |   |   |   insert v into Q // Add unvisited
   |   |   |   |   node into the Queue to be
   |   |   |   |   processed;
19 |   |   |   end
20 |   |   end
21 |   end
22 end
```

Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments [2]

Relaxed Dijkstra Algorithm


```
   input  : Grid, Start, Goal
   output : dist
 1 for each vertex v in Grid do
 2 |   dist(v)= infinity // Mark distance from
   |   start to each node v as not yet
   |   computed;
 3 end
 4 dist(Start)=0;
 5 insert Start into Q // Initially, only the
   start node is in the Queue;
 6 while (Q is not empty) and (dist(Goal)==infinity) do
 7 |   u = head of Q;
 8 |   remove u from Q;
 9 |   for each neighbor v of u do
10 |   |   if dist(v)==infinity then
11 |   |   |   insert v at the tail of Q // Add
   |   |   |   unvisited node into the Queue
   |   |   |   to be processed;
12 |   |   end
13 |   |   dist(v)=dist(u)+distEdge(u, v);
14 |   end
15 end
```

Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments [2]

The standard A* algorithm


```
   input  : Grid, Start, Goal
   // Initialisation:
1  closedSet = empty set // Set of already evaluated
   nodes;
2  openSet = Start // Set of nodes to be
   evaluated;
3  came_from = the empty map // map of
   navigated nodes;
4  tBreak = 1+1/(length(Grid)+width(Grid));
   // coefficient for breaking ties;
5  g_score[Start] = 0; // Cost from Start along
   best known path;
   // Estimated total cost from Start to
   Goal:
6  f_score[Start] = heuristic_cost(Start, Goal);
.
.
.
```

```
7  while openSet is not empty do
8      current = the node in openSet having the lowest
       f_score;
9      if current = Goal then
10         return reconstruct_path(came_from, Goal);
11     end
12     remove current from openSet;
13     add current to closedSet;
14     for each free neighbor v of current do
15         if v in closedSet then
16             continue;
17         end
18         tentative_g_score = g_score[current] +
           dist_edge(current, v);
19         if v not in openSet or tentative_g_score <
           g_score[v] then
20             came_from[v] = current;
21             g_score[v] = tentative_g_score;
22             f_score[v] = g_score[v] + tBreak *
               heuristic_cost(v, Goal);
23             if neighbor not in openSet then
24                 add neighbor to openSet;
25             end
26         end
27     end
28 end
29 return failure;
```

Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments [2]

Relaxed A* Algorithm


```
   input   : Grid, Start, Goal
 1 tBreak = 1+1/(length(Grid)+width(Grid));
   // Initialisation:
 2 openSet = Start // Set of nodes to be
   evaluated;
 3 for each vertex v in Grid do
 4 |  g_score(v)= infinity;
 5 end
 6 g_score[Start] = 0;
   // Estimated total cost from Start to
   Goal:
 7 f_score[Start] = heuristic_cost(Start, Goal);
 8 while openSet is not empty and g_score[Goal]==
   infinity do
 9 |  current = the node in openSet having the lowest
   |  f_score;
10 |  remove current from openSet;
11 |  for each free neighbor v of current do
12 |  |  if g_score(v) == infinity then
13 |  |  |  g_score[v] = g_score[current] +
   |  |  |  dist_edge(current, v);
14 |  |  |  f_score[v] = g_score[v] + tBreak *
   |  |  |  heuristic_cost(v, Goal);
15 |  |  |  add neighbor to openSet;
16 |  |  end
17 |  end
18 end
19 if g_score(goal) != infinity then
20 |  return reconstruct_path(g_score) // path will
   |  be reconstructed based on g_score
   |  values;
21 else
22 |  return failure;
23 end
```

Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments [2]

Results

Improved Dijkstra with Ant System Algorithm	Hybrid genetic algorithm
439 m path	562 m path

[4]

Relaxed Dijkstra / A*	Regular Dijkstra / A*
Linear time complexity bounded by the number of cells of the grid.	Quadratic time complexity bounded by the number of cells of the grid (for A* it depends on the heuristic)

[2]

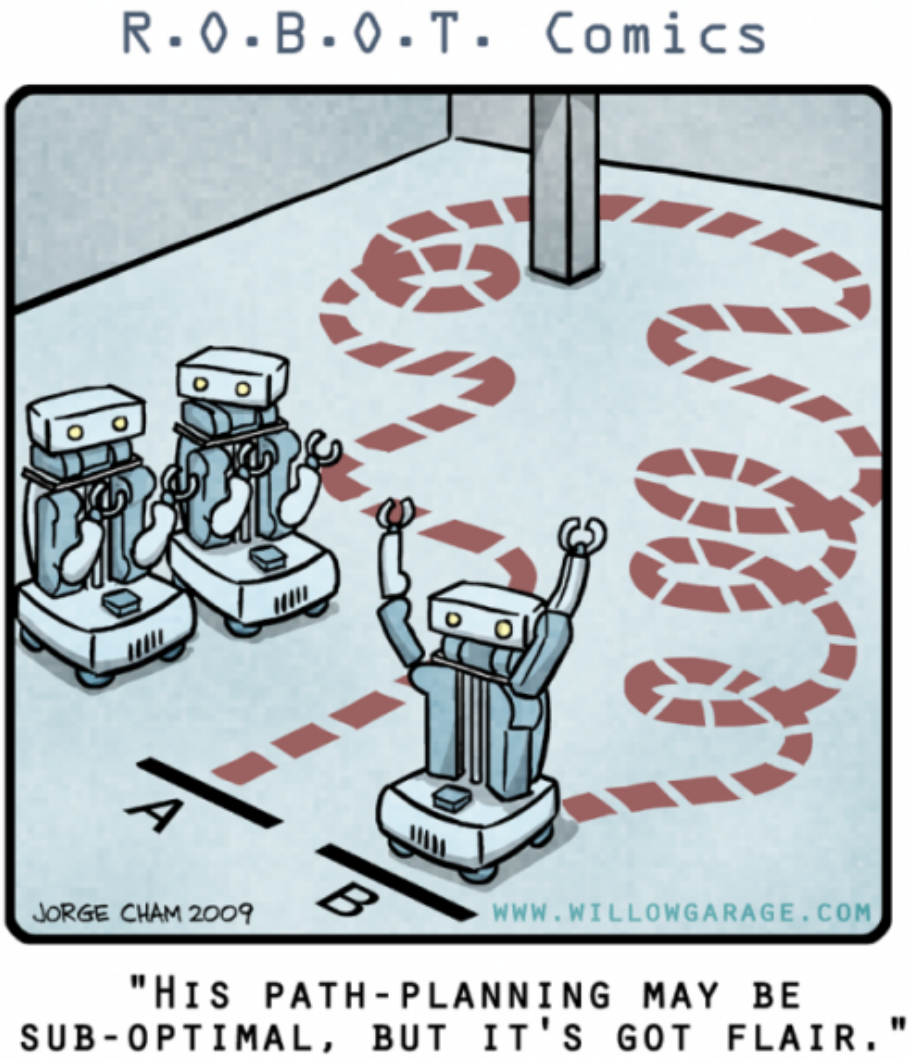
RA*	navfn (ROS)
(6.170884 $\pm$ 0.504922) ms execution 19.41165 m path	(14.95496 $\pm$ 1.174413) ms execution 18.83176 m path

[2]

Conclusion

Pros	Cons
▶ Optimal path	▶ No dynamic environments
▶ Minimal computation	▶ Finite world
▶ Guaranteed solution (if exists)	▶ No irregular shaped obstacles

Thank you for your attention.



References

[1] Abrar Alajlan, Khaled Elleithy, Marwah Almasri, and Tarek Sobh.
An optimal and energy efficient multi-sensor collision-free path planning algorithm for a mobile robot in dynamic environments.
Robotics, 6(2), 2017.

[2] Adel Ammar, Hachemi Bennaceur, Imen Châari, Anis Koubâa, and Maram Alajlan.
Relaxed dijkstra and a* with linear complexity for robot path planning problems in large-scale grid environments.
Soft Computing, 20(10):4149–4171, Oct 2016.

[3] Bruno Siciliano and Oussama Khatib.
Springer Handbook of Robotics.
Springer-Verlag, Berlin, Heidelberg, 2016.

[4] Guan-zheng Tan, Huan He, and Sloman Aaron.
Global optimal path planning for mobile robot based on improved dijkstra algorithm and ant system algorithm.
Journal of Central South University of Technology, 13(1):80–86, Feb 2006.