Week 7 – Path Planning

ELEC0144 Machine Learning for Robotics

Dr. Chow Yin Lai

Email: uceecyl@ucl.ac.uk

Schedule

Week	Lecture	Workshop	Assignment Deadlines
1	Introduction; Image Processing	Image Processing	
2	Camera and Robot Calibration	Camera and Robot Calibration	
3	Introduction to Neural Networks	Camera and Robot Calibration	Friday: Camera and Robot Calibration
4	MLP and Backpropagation	MLP and Backpropagation	
5	CNN and Image Classification	MLP and Backpropagation	
6	Object Detection	MLP and Backpropagation	Friday: MLP and Backpropagation
7	Path Planning	Path Planning	
8	Kalman Filter SLAM	Path Planning	
9	Extended Kalman Filter SLAM	Path Planning	
10	Particle Filter SLAM	Path Planning	Friday: Path Planning

Content

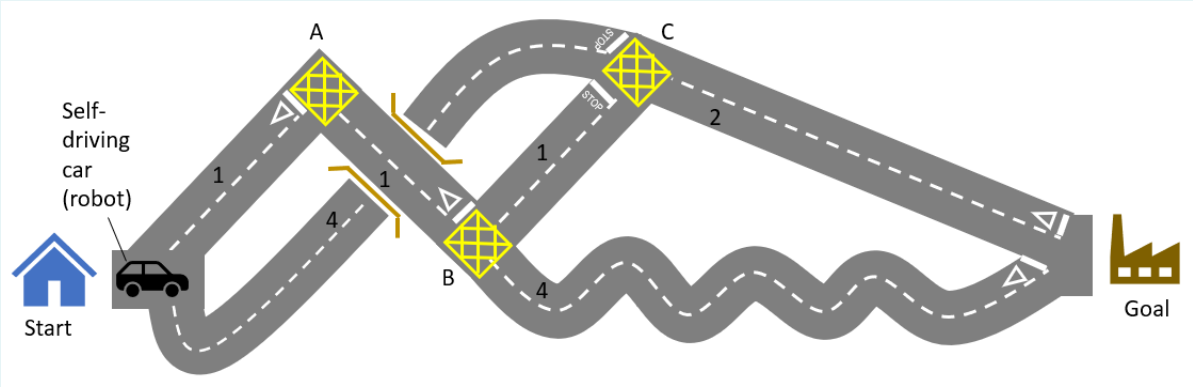
- Dijkstra’s Algorithm: Nodes and Edges
- Dijkstra’s Algorithm: Empty Space (with/out obstacles)
- Greedy Algorithm
- A* Algorithm
- Potential Field

Content

- Dijkstra’s Algorithm: Nodes and Edges
- Dijkstra’s Algorithm: Empty Space (with/out obstacles)
- Greedy Algorithm
- A* Algorithm
- Potential Field

Introduction (1)

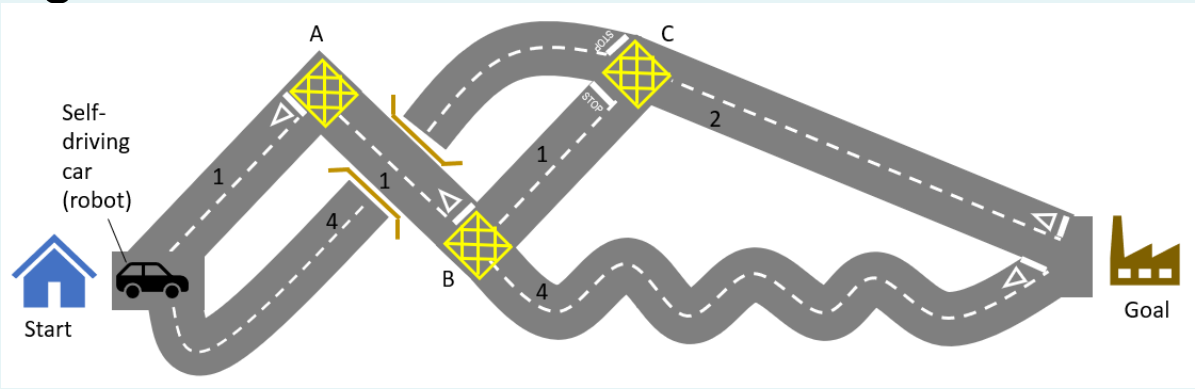
- Assume that you have the following map:



  - The numbers on the road represent the "lengths" of the segments, or "cost" in going through that segment.

Introduction (2)

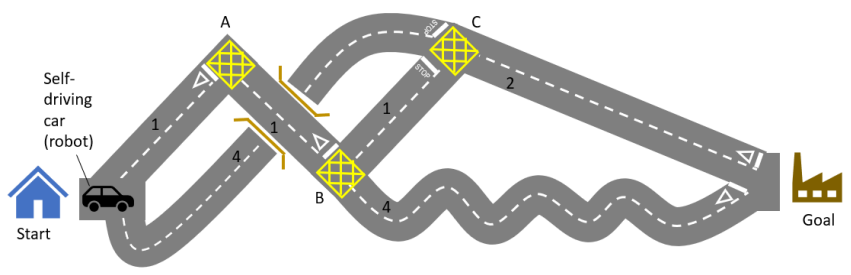
- Question: What is the shortest path or lowest cost from start to goal?

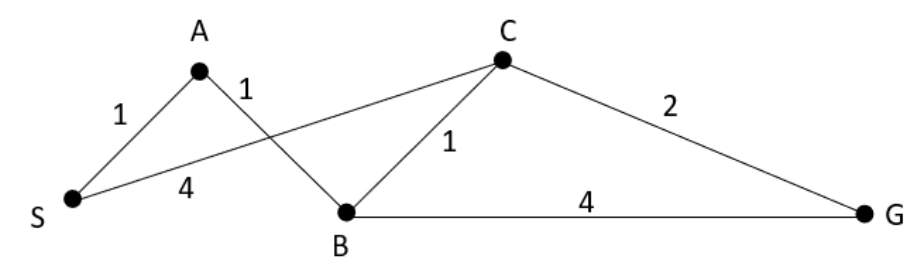


  - In this simple example, we can (quickly) see that the shortest path would be going along the segments with lengths [1,1,1,2].
  - Other options e.g. [4,2], [4,1,4], [1,1,4] are all longer.

Introduction (3)

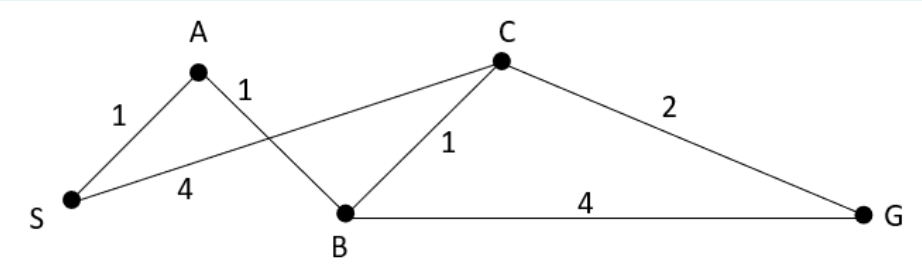
- We would now like to write an algorithm which can solve the problem for us automatically, even for more complicated maps!
- Firstly, we transform the “physical” map into “nodes and edges”:
  - Segments → Edges; Intersections → Nodes; Lengths → Weights.





Dijkstra Algorithm – Simple Example (1)

- Using this simple example, we will explain an algorithm called “Dijkstra algorithm”.



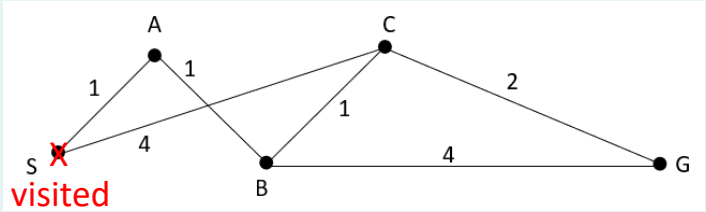
  - Firstly, we define the “front” as the set of nodes to be considered at the current moment.
  - At the beginning, the only node to be considered is the “start” node.
  - The “start” node also has 0 cost associated with it.
  - The “start” node has no predecessor.

Dijkstra Algorithm – Simple Example (2)

- Therefore: we write front = {[0,S,none]}, where we keep track of the cost associated with that node, and predecessor of that node.
- The Dijkstra algorithm now proceeds as follows:
- While “front” is not empty:
  - Choose the node n in “front” which has the lowest cost;
  - Move n to another set called “visited”;
  - Delete n from the “front” set;
  - Determine direct unvisited neighbors of n,
    - Add them into the “front” set if not visited before,
    - Or adjust cost if visited before and the new cost is lower.
- End While.

Dijkstra Algorithm – Simple Example (3)

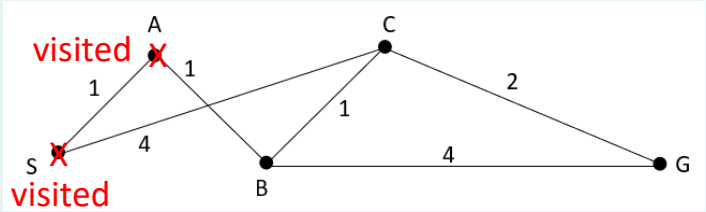
- Let's see how it goes.
- We had: front = {[0,S,none]}.
- While "front" is not empty: → TRUE



  - Choose the node n in "front" which has the lowest cost → [0,S,none];
  - Move n (S) to another set called "visited"; → visited = {[0,S,none]}
  - Delete n (S) from the "front" set; → front = ~~{[0,S,none]}~~ = ∅
  - Determine direct <u>unvisited</u> neighbors of n (S), → $[0_{(S)}+1,A,S]$, $[0_{(S)}+4,C,S]$
    - Add them into the "front" set if not visited before, → front= $\begin{Bmatrix}[1,A,S],\\ [4,C,S]\end{Bmatrix}$
    - Or adjust cost if visited before and the new cost is lower. → not needed.

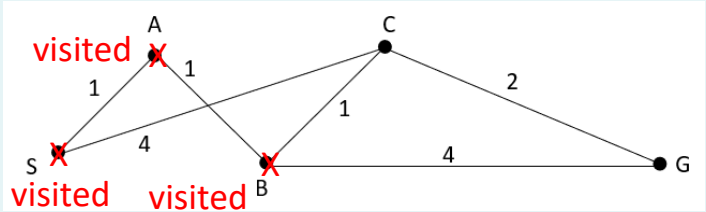
Dijkstra Algorithm – Simple Example (4)

- Now we have: front= $\begin{Bmatrix}[1,A,S],\\ [4,C,S]\end{Bmatrix}$
- While "front" is not empty: → TRUE
  - Choose the node n in "front" which has the lowest cost → [1,A,S];
  - Move n (A) to another set called "visited"; → visited= $\begin{Bmatrix}[0,S,none],\\ [1,A,S]\end{Bmatrix}$
  - Delete n (A) from the "front" set; → front = $\begin{Bmatrix}\sout{[1,A,S],}\\ [4,C,S]\end{Bmatrix}$ = {[4,C,S]}
  - Determine direct unvisited neighbors of n (A), → $[1_{(A)}+1,B,A]$
  - Add them into the "front" set if not visited before, → front= $\begin{Bmatrix}[2,B,A],\\ [4,C,S]\end{Bmatrix}$
    - Or adjust cost if visited before and the new cost is lower. → not needed.



Dijkstra Algorithm – Simple Example (5)

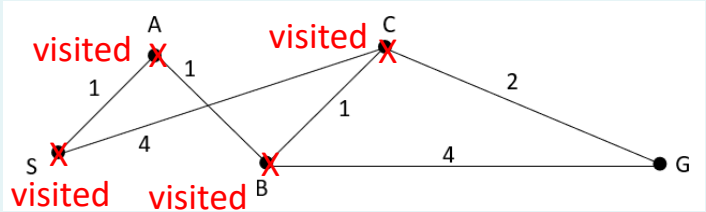
- Now we have: front= $\begin{Bmatrix}[2,B,A],\\ [4,C,S]\end{Bmatrix}$
- While "front" is not empty: → TRUE



  - Choose the node n in "front" which has the lowest cost → [2,B,A];
  - Move n (B) to another set called "visited"; → visited= $\begin{Bmatrix}[0,S,none],\\ [1,A,S],\ [2,B,A]\end{Bmatrix}$
  - Delete n (B) from the "front" set; → front = $\begin{Bmatrix}\sout{[2,B,A],}\\ [4,C,S]\end{Bmatrix}$ = {[4,C,S]}
  - Determine direct unvisited neighbors of n (B), → $[2_{(B)}+1,C,B]$, $[2_{(B)}+4,G,B]$
  - Add them into the "front" set if not visited before, or adjust cost if visited before and the new cost is lower → front= $\begin{Bmatrix}[6,G,B],\\ \sout{[4,C,S]} \rightarrow [3,C,B]\end{Bmatrix}$

Dijkstra Algorithm – Simple Example (6)

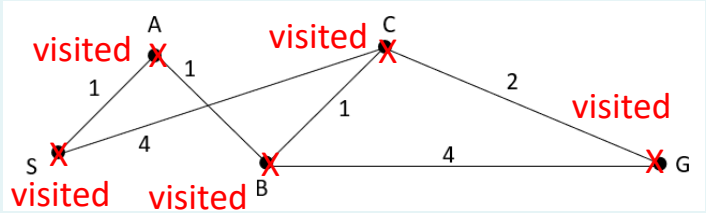
- Now we have: front=$\begin{Bmatrix}[6,G,B],\\ [3,C,B]\end{Bmatrix}$
- While "front" is not empty: → TRUE



  - Choose the node n in "front" which has the lowest cost → [3,C,B];
  - Move n (C) to another set called "visited"; → visited=$\begin{Bmatrix}[0,S,none], [1,A,S]\\ [2,B,A], [3,C,B]\end{Bmatrix}$
  - Delete n (C) from the "front" set; → front =$\begin{Bmatrix}[6,G,B],\\ \sout{[3,C,B]}\end{Bmatrix}$= {[6,G,B]}
  - Determine direct <u>unvisited</u> neighbors of n (C), →$[3_{(C)}+2,G,C]$
  - Add them into the "front" set if not visited before, or adjust cost if visited before and the new cost is lower → front={$\sout{[6,G,B]}$ → [5,G,C]}

Dijkstra Algorithm – Simple Example (7)

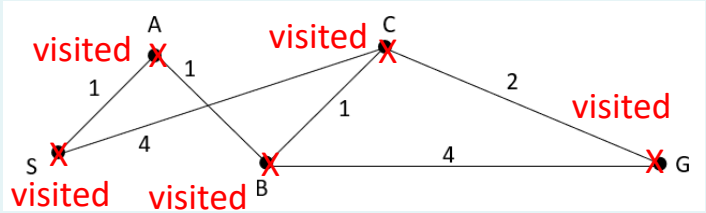
- Now we have: front={[5,G,C]}.



- While “front” is not empty: → TRUE
  - Choose the node n in “front” which has the lowest cost → [5,G,C];
  - Move n (G) to another set called “visited”; → visited={[0,S,none], [1,A,S], [2,B,A], [3,C,B], [5,G,C]}
  - Delete n (G) from the “front” set; → front = {~~[5,G,C]~~} = ∅
  - Determine direct unvisited neighbors of n (G), →None
  - Add them into the “front” set if not visited before, or adjust cost if visited before and the new cost is lower → front = ∅

Dijkstra Algorithm – Simple Example (8)

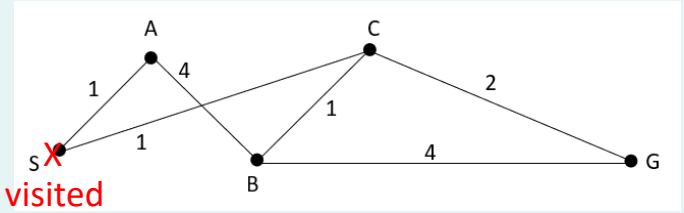
- Now we have: front = ∅
- While “front” is not empty: → FALSE
- End of While loop.



- Now, we can look at the “visited” set to determine the shortest path: visited= $\{[0,S,none], [1,A,S], [2,B,A], [3,C,B], [5,G,C]\}$
  - G’s predecessor is C, C’s predecessor is B, B’s predecessor is A, A’s predecessor is S, and S’s predecessor is none.
  - The path is therefore S-A-B-C-G, and the final cost is 5.

Dijkstra Algorithm – Example B (1)

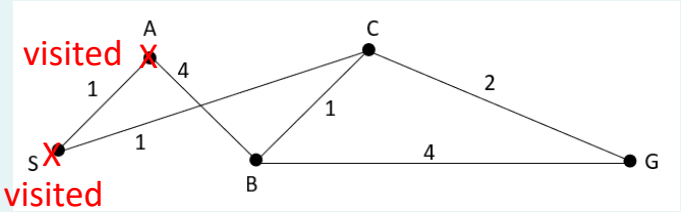
- Let's do another example.
- Similar graph but weights different.
- Start with: front = {[0,S,none]}.



- While "front" is not empty: → TRUE
  - Choose the node n in "front" which has the lowest cost → [0,S,none];
  - Move n (S) to another set called "visited"; → visited = {[0,S,none]}
  - Delete n (S) from the "front" set; → front = ~~{[0,S,none]}~~ = ∅
  - Determine direct <u>unvisited</u> neighbors of n (S), → $[0_{(S)}+1,A,S]$, $[0_{(S)}+1,C,S]$
    - Add them into the "front" set if not visited before, → front= $\begin{Bmatrix}[1,A,S],\\ [1,C,S]\end{Bmatrix}$
    - Or adjust cost if visited before and the new cost is lower. → not needed.

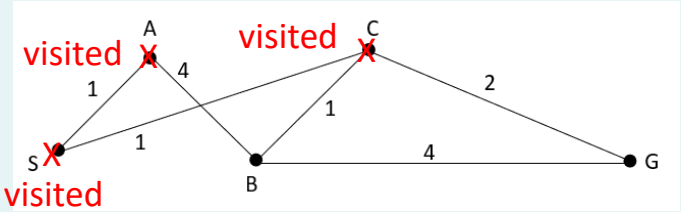
Dijkstra Algorithm – Example B (2)

- Now we have: front= $\begin{Bmatrix}[1,A,S],\\ [1,C,S]\end{Bmatrix}$.
- While "front" is not empty: → TRUE
  - Choose the node n in "front" which has the lowest cost → Both fronts have the same cost. Choose either one. [1,A,S];
  - Move n (A) to another set called "visited"; → visited = {[0,S,none],[1,A,S]}
  - Delete n (A) from the "front" set; → front= $\begin{Bmatrix}\sout{[1,A,S]},\\ [1,C,S]\end{Bmatrix}$ = [1,C,S]
  - Determine direct <u>unvisited</u> neighbors of n (A), → $[1_{(A)}+4,B,A]$
    - Add them into the "front" set if not visited before, → front= $\begin{Bmatrix}[5,B,A],\\ [1,C,S]\end{Bmatrix}$
    - Or adjust cost if visited before and the new cost is lower. → not needed.



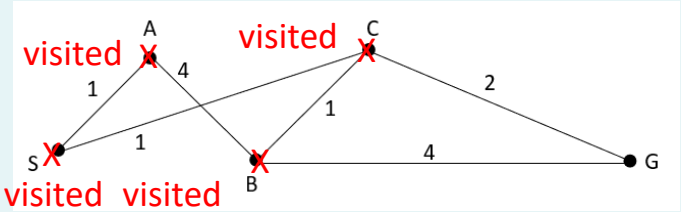
Dijkstra Algorithm – Example B (3)

- Now we have: front= $\begin{Bmatrix}[5,B,A],\\ [1,C,S]\end{Bmatrix}$.
- While "front" is not empty: → TRUE
  - Choose the node n in "front" which has the lowest cost → [1,C,S];
  - Move n (C) to another set called "visited"; → visited = $\begin{Bmatrix}[0,S,none],\\ [1,A,S],[1,C,S]\end{Bmatrix}$
  - Delete n (C) from the "front" set; → front= $\begin{Bmatrix}[5,B,A],\\ \sout{[1,C,S]}\end{Bmatrix}$= [5,B,A]
  - Determine direct unvisited neighbors of n (C), → $[1_{(C)}+1,B,C]$, $[1_{(C)}+2,G,C]$
    - Add them into the "front" set if not visited before, or adjust cost if visited before and the new cost is lower. → front= $\begin{Bmatrix}\sout{[5,B,A]} \rightarrow [2,B,C],\\ [3,G,C]\end{Bmatrix}$.



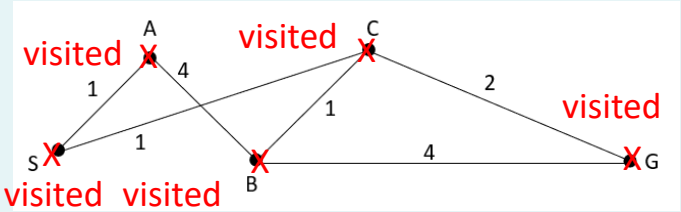
Dijkstra Algorithm – Example B (4)

- Now we have:front= $\begin{Bmatrix}[2,B,C],\\ [3,G,C]\end{Bmatrix}$.
- While "front" is not empty: → TRUE
  - Choose the node n in "front" which has the lowest cost → [2,B,C];
  - Move n (B) to another set called "visited"; → visited = $\begin{Bmatrix}[0,S,none],[1,A,S],\\ [1,C,S],[2,B,C]\end{Bmatrix}$
  - Delete n (B) from the "front" set; → front= $\begin{Bmatrix}\sout{[2,B,C]},\\ [3,G,C]\end{Bmatrix}$= [3,G,C]
  - Determine direct unvisited neighbors of n (B), → $[3_{(B)}+4,G,B]$
    - Add them into the "front" set if not visited before, or adjust cost if visited before and the new cost is lower. → front remains = {[3,G,C]}.



Dijkstra Algorithm – Example B (5)

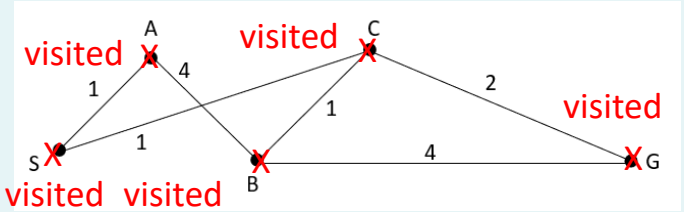
- Now we have:front={[3,G,C]}.



- While “front” is not empty: → TRUE
  - Choose the node n in “front” which has the lowest cost → [3,G,C];
  - Move n (G) to another set called “visited”; → visited = {[0,S,none],[1,A,S], [1,C,S], [2,B,C], [3,G,C]}
  - Delete n (G) from the “front” set; → front = {~~[3,G,C]~~} = ∅
  - Determine direct unvisited neighbors of n (G), → None
    - Add them into the “front” set if not visited before, or adjust cost if visited before and the new cost is lower. → front remains = ∅

Dijkstra Algorithm – Example B (6)

- Finally, front = ∅ and we won't go into the while loop.



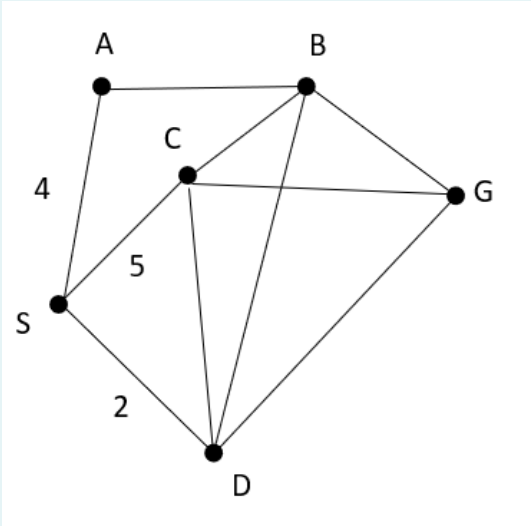
- Now, we can look at the "visited" set to determine the shortest path: visited = {[0,S,none],[1,A,S], [1,C,S], [2,B,C], [3,G,C]}
  - G's predecessor is C, C's predecessor is S, S's predecessor is none.
  - The path is therefore S-C-G, with the final cost of 3.

Dijkstra Algorithm – Example B (7)

- From this second example, we learnt a few things:
  - Sometimes, the front set has more than 1 nodes with the same minimum cost.
    - It is ok to choose one and proceed with the calculation, because in the next while iteration, the other node(s) will be dealt with, since they are most likely to be still the node(s) with minimum cost in the next iteration.
  - It is important to keep track of the predecessors.
    - Even though the visited set has 5 nodes, we see that G's predecessor is C, C's predecessor is S, and S's predecessor is none.
    - So nodes A and B will not be passed through at all.

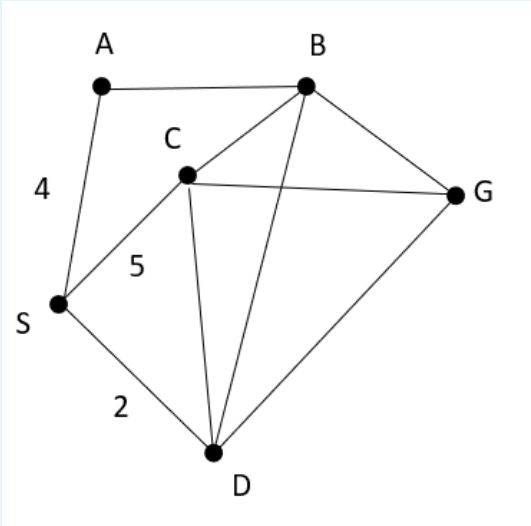
Discussion about Visited Set (1)

- In the examples, we also saw that once a node is moved into the “visited” set, we will not look at it anymore.
- But why? Is there any chance that the cost of the nodes in the visited set can be lower?
  - Let’s look at the example on the right.
  - After S is moved into the visited set, the front set (direct neighbors of S) consists of A, C and D.
  - D would have the lowest cost, and thus is moved into the visited set.



Discussion about Visited Set (2)

- Can D have a lower cost if it comes through any other path, for e.g. S-A-B-D, S-C-D, S-C-B-D etc?
  - No, because as soon as you go from S to A, or S to C, the cost is already higher than the cost from S to D.
  - Therefore, the cost of D in the visited set (2 in this example) will definitely be the lowest possible for D.
  - Visited set can also be thought of as optimal set!
  - Of course, this argument only holds if we allow only positive edge cost.

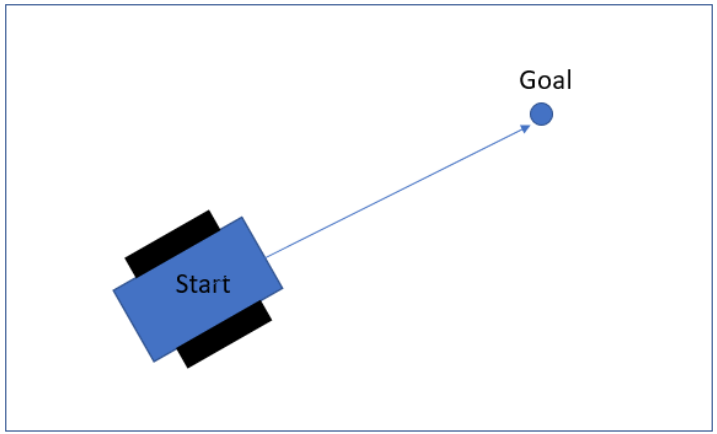


Content

- Dijkstra’s Algorithm: Nodes and Edges
- Dijkstra’s Algorithm: Empty Space (with/out obstacles)
- Greedy Algorithm
- A* Algorithm
- Potential Field

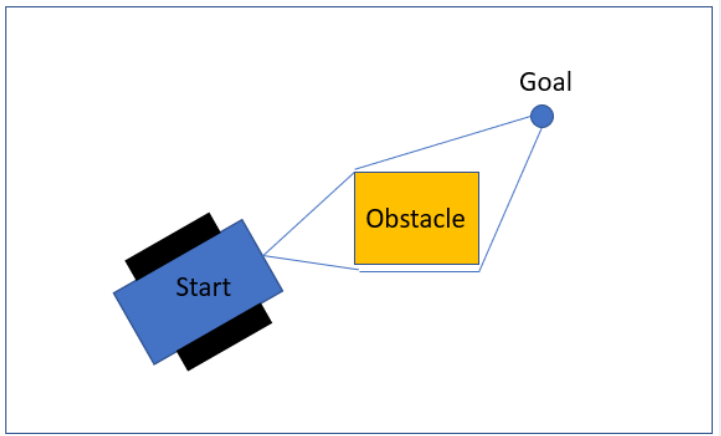
Navigation in Empty Space (1)

- So far, we have looked at the case of possible paths along specific nodes and edges.
- What if we want to plan a path for a robot moving in a flat empty space without obstacle?
  - The solution is straightforward → straight line.



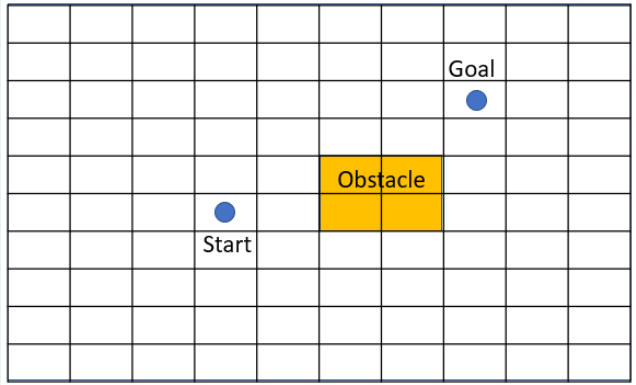
Navigation in Empty Space (2)

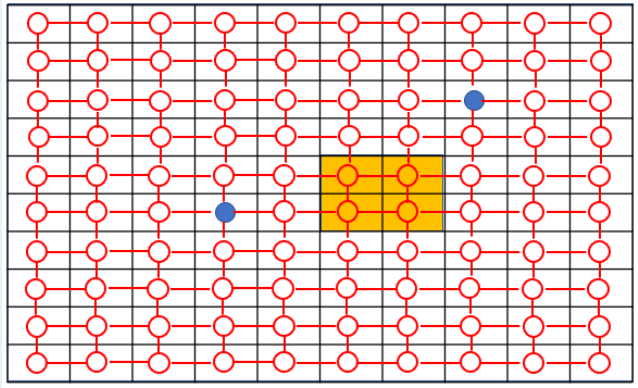
- However, if there are some obstacles, we will need to find an optimal path in a continuous space of all possible paths!
  - Complicated!



Navigation in Empty Space (3)

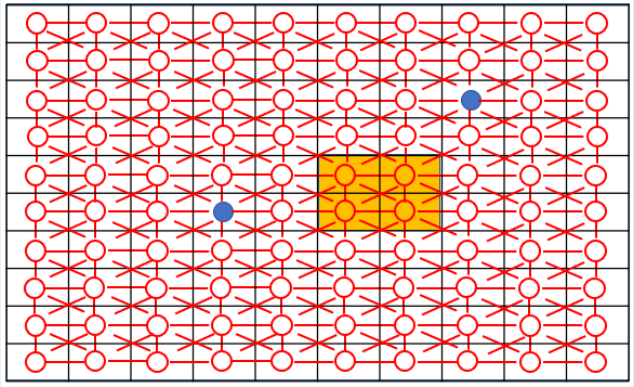
- Solution: Divide the empty space into a raster of cells.
  - We then constraint the navigation to be only in the center of cells.
  - This then implicitly defines a graph with edges and nodes.
  - We give the vertical and horizontal movement a cost of 1.
  - But if a node coincides with the obstacle, we set the cost to ∞.





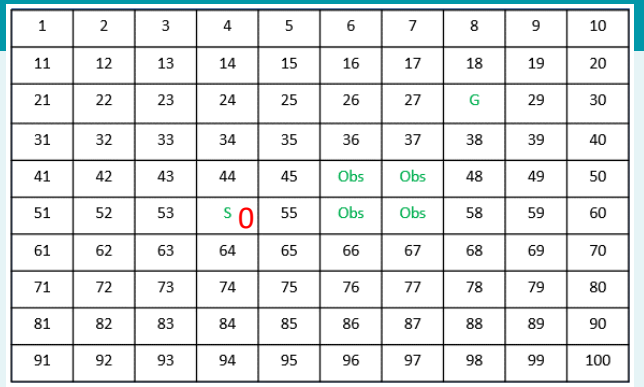
Navigation in Empty Space (4)

- We can further improve the graph by allowing diagonal movements, with cost of $\sqrt{2}$.
- Now let's apply Dijkstra's algorithm on this example.
- Firstly, we assign numbers to the cells.



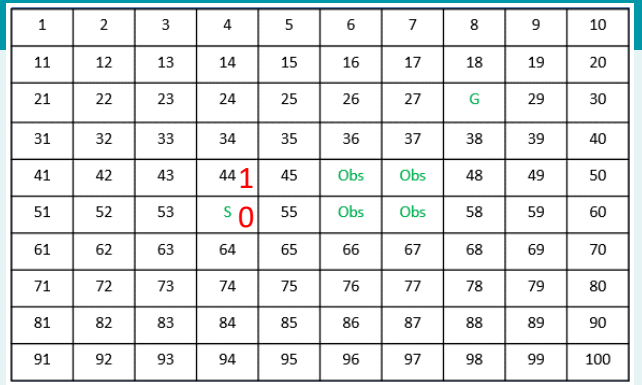
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	G	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	Obs	Obs	48	49	50
51	52	53	S	55	Obs	Obs	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Navigation in Empty Space (5)



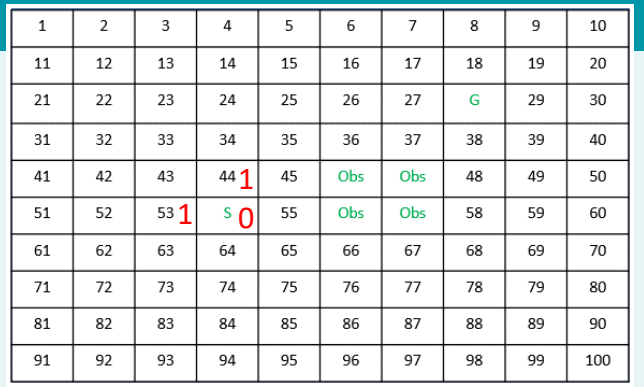
New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[0, S, none]$	S	$[0, S, none]$	$\emptyset$	$[1,44,0], [1,53,0],$ $[1,55,0], [1,64,0],$ $[\sqrt{2}, 43,0], [\sqrt{2}, 45,0],$ $[\sqrt{2}, 63,0], [\sqrt{2}, 65,0]$

Navigation in Empty Space (6)



New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[1,44,0], [1,53,0], [1,55,0], [1,64,0], [\sqrt{2},43,0], [\sqrt{2},45,0], [\sqrt{2},63,0], [\sqrt{2},65,0]$	44	$[0,S,none], [1,44,0]$	$[1,53,0], [1,55,0], [1,64,0], [\sqrt{2},43,0], [\sqrt{2},45,0], [\sqrt{2},63,0], [\sqrt{2},65,0]$	$[1,53,0]$, $[1,55,0]$, $[1,64,0]$, $[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[1+1,34,44]$, ~~$[1+1,43,44]$~~, ~~$[1+1,45,44]$~~, $[1+\sqrt{2},33,44]$, $[1+\sqrt{2},35,44]$, ~~$[1+\sqrt{2},53,44]$~~, ~~$[1+\sqrt{2},55,44]$~~

Navigation in Empty Space (7)



New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[1,53,0]$, $[1,55,0]$, $[1,64,0]$, $[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[2,34,44]$, $[1+\sqrt{2},33,44]$, $[1+\sqrt{2},35,44]$	53	$[0,S,none]$, $[1,44,0]$, $[1,53,0]$	$[1,55,0]$, $[1,64,0]$, $[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[2,34,44]$, $[1+\sqrt{2},33,44]$, $[1+\sqrt{2},35,44]$,	$[1,55,0]$, $[1,64,0]$, $[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[2,34,44]$, $[1+\sqrt{2},33,44]$, $[1+\sqrt{2},35,44]$, $[1+1,52,53]$, ~~$[1+1,43,53]$~~, ~~$[1+1,63,53]$~~, $[1+\sqrt{2},42,53]$, $[1+\sqrt{2},62,53]$, ~~$[1+\sqrt{2},64,53]$~~

Navigation in Empty Space (8)

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	G	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44 1	45	Obs	Obs	48	49	50
51	52	53 1	S 0	55 1	Obs	Obs	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

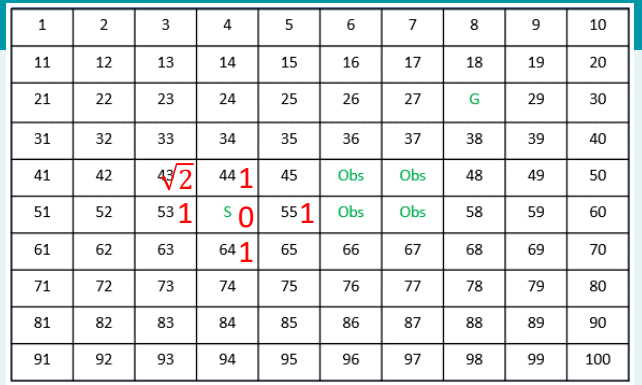
New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[1,55,0], [1,64,0],$ $[\sqrt{2},43,0], [\sqrt{2},45,0],$ $[\sqrt{2},63,0], [\sqrt{2},65,0],$ $[2,34,44],$ $[1+\sqrt{2},33,44],$ $[1+\sqrt{2},35,44],$ $[2,52,53],$ $[1+\sqrt{2},42,53],$ $[1+\sqrt{2},62,53]$	55	$[0,S,none], [1,44,0],$ $[1,53,0], [1,55,0]$	$[1,64,0],$ $[\sqrt{2},43,0], [\sqrt{2},45,0],$ $[\sqrt{2},63,0], [\sqrt{2},65,0],$ $[2,34,44],$ $[1+\sqrt{2},33,44],$ $[1+\sqrt{2},35,44],$ $[2,52,53],$ $[1+\sqrt{2},42,53],$ $[1+\sqrt{2},62,53]$	$[1,64,0],$ $[\sqrt{2},43,0], [\sqrt{2},45,0],$ $[\sqrt{2},63,0], [\sqrt{2},65,0],$ $[2,34,44], [1+\sqrt{2},33,44],$ $[1+\sqrt{2},35,44], [2,52,53],$ $[1+\sqrt{2},42,53], [1+\sqrt{2},62,53],$ ~~$[1+1,45,55]$~~, ~~$[1+1,65,55]$~~, ~~$[1+\sqrt{2},64,55]$~~, $[1+\sqrt{2},66,55]$

Navigation in Empty Space (9)

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	G	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44 1	45	Obs	Obs	48	49	50
51	52	53 1	S 0	55 1	Obs	Obs	58	59	60
61	62	63	64 1	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[1,64,0]$, $[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[2,34,44]$, $[1+\sqrt{2},33,44]$ $[1+\sqrt{2},35,44]$, $[2,52,53]$, $[1+\sqrt{2},42,53]$, $[1+\sqrt{2},62,53]$ $[1+\sqrt{2},66,55]$	64	$[0,S,none]$, $[1,44,0]$, $[1,53,0]$, $[1,55,0]$, $[1,64,0]$	$[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[2,34,44]$, $[1+\sqrt{2},33,44]$ $[1+\sqrt{2},35,44]$, $[2,52,53]$, $[1+\sqrt{2},42,53]$, $[1+\sqrt{2},62,53]$ $[1+\sqrt{2},66,55]$	$[\sqrt{2},43,0]$, $[\sqrt{2},45,0]$, $[\sqrt{2},63,0]$, $[\sqrt{2},65,0]$, $[2,34,44]$, $[1+\sqrt{2},33,44]$ $[1+\sqrt{2},35,44]$, $[2,52,53]$, $[1+\sqrt{2},42,53]$, $[1+\sqrt{2},62,53]$ $[1+\sqrt{2},66,55]$, ~~$[1+1,63,64]$~~, $[1+1,74,64]$, ~~$[1+1,65,64]$~~, $[1+\sqrt{2},73,64]$, $[1+\sqrt{2},75,64]$

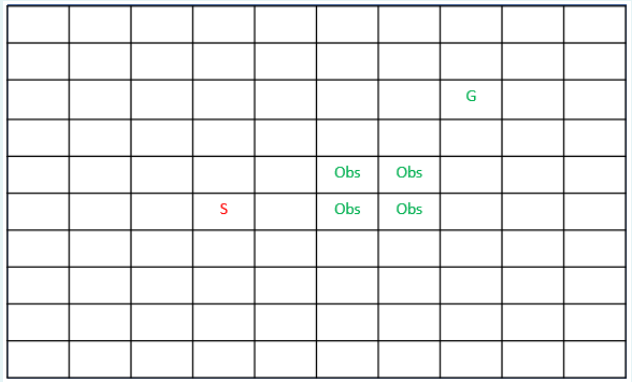
Navigation in Empty Space (10)

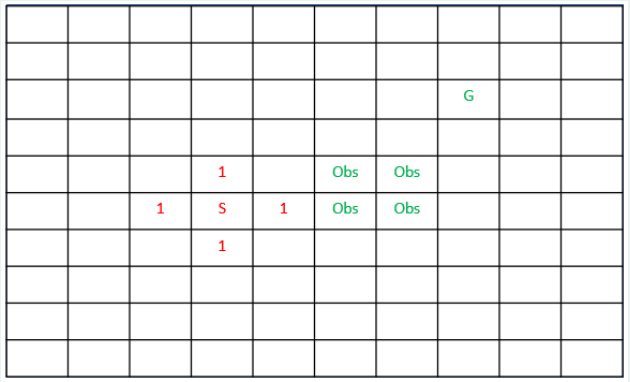


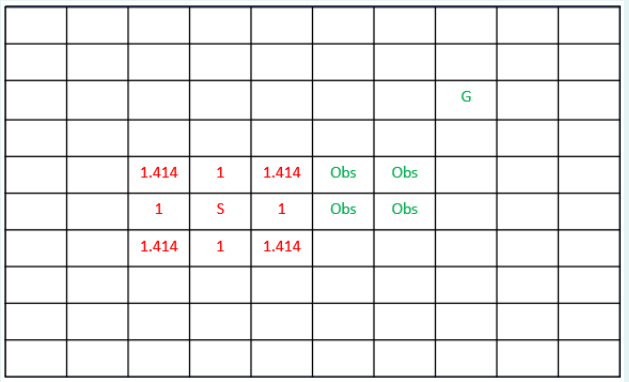
New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[\sqrt{2}, 43,0], [\sqrt{2}, 45,0],$ $[\sqrt{2}, 63,0], [\sqrt{2}, 65,0],$ $[2,34,44],$ $[1+\sqrt{2}, 33,44],$ $[1+\sqrt{2}, 35,44],$ $[2,52,53], [2,74,64],$ $[1+\sqrt{2}, 42,53],$ $[1+\sqrt{2}, 62,53],$ $[1+\sqrt{2}, 66,55],$ $[1+\sqrt{2}, 73,64],$ $[1+\sqrt{2}, 75,64]$	43	$[0, S, none], [1,44,0],$ $[1,53,0], [1,55,0],$ $[1,64,0], [\sqrt{2}, 43,0]$	$[\sqrt{2}, 45,0],$ $[\sqrt{2}, 63,0], [\sqrt{2}, 65,0],$ $[2,34,44],$ $[1+\sqrt{2}, 33,44],$ $[1+\sqrt{2}, 35,44],$ $[2,52,53], [2,74,64],$ $[1+\sqrt{2}, 42,53],$ $[1+\sqrt{2}, 62,53],$ $[1+\sqrt{2}, 66,55],$ $[1+\sqrt{2}, 73,64],$ $[1+\sqrt{2}, 75,64]$	$[\sqrt{2}, 45,0], [\sqrt{2}, 63,0], [\sqrt{2}, 65,0],$ $[2,34,44], [1+\sqrt{2}, 33,44],$ $[1+\sqrt{2}, 35,44], [2,52,53],$ $[2,74,64], [1+\sqrt{2}, 42,53],$ $[1+\sqrt{2}, 62,53], [1+\sqrt{2}, 66,55],$ $[1+\sqrt{2}, 73,64], [1+\sqrt{2}, 75,64],$ $[\sqrt{2}+1,42,43], [\sqrt{2}+1,33,43],$ $[\sqrt{2}+\sqrt{2}, 32,43],$ ~~$[\sqrt{2}+\sqrt{2}, 34,43]$~~, ~~$[\sqrt{2}+\sqrt{2}, 52,43]$~~

Navigation in Empty Space (11)

- The same process continues until G is moved into the visited (optimal) set.
- Evolution of the “visited” set:







Navigation in Empty Space (12)

							G		
			2						
		1.414	1	1.414	Obs	Obs			
	2	1	S	1	Obs	Obs			
		1.414	1	1.414					
			2						

							G		
		2.414	2	2.414					
	2.414	1.414	1	1.414	Obs	Obs			
	2	1	S	1	Obs	Obs			
	2.414	1.414	1	1.414	2.414				
		2.414	2	2.414					

							G		
	2.828	2.414	2	2.414	2.828				
	2.414	1.414	1	1.414	Obs	Obs			
	2	1	S	1	Obs	Obs			
	2.414	1.414	1	1.414	2.414				
	2.828	2.414	2	2.414	2.828				

			3				G		
	2.828	2.414	2	2.414	2.828				
	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs			
	2.414	1.414	1	1.414	2.414				
	2.828	2.414	2	2.414	2.828				
			3						

		3.414	3	3.414			G		
	2.828	2.414	2	2.414	2.828				
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs			
3.414	2.414	1.414	1	1.414	2.414	3.414			
	2.828	2.414	2	2.414	2.828				
		3.414	3	3.414					

	3.828	3.414	3	3.414	3.828		G		
3.828	2.828	2.414	2	2.414	2.828	3.828			
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs			
3.414	2.414	1.414	1	1.414	2.414	3.414			
3.828	2.828	2.414	2	2.414	2.828	3.828			
	3.828	3.414	3	3.414	3.828				

Navigation in Empty Space (13)

			4						
	3.828	3.414	3	3.414	3.828		G		
3.828	2.828	2.414	2	2.414	2.828	3.828			
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs			
3.414	2.414	1.414	1	1.414	2.414	3.414			
3.828	2.828	2.414	2	2.414	2.828	3.828			
	3.828	3.414	3	3.414	3.828				
			4						

			4						
4.242	3.828	3.414	3	3.414	3.828	4.242	G		
3.828	2.828	2.414	2	2.414	2.828	3.828			
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs			
3.414	2.414	1.414	1	1.414	2.414	3.414			
3.828	2.828	2.414	2	2.414	2.828	3.828			
4.242	3.828	3.414	3	3.414	3.828	4.242			
			4						

		4.414	4	4.414					
4.242	3.828	3.414	3	3.414	3.828	4.242	G		
3.828	2.828	2.414	2	2.414	2.828	3.828			
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs			
3.414	2.414	1.414	1	1.414	2.414	3.414	4.414		
3.828	2.828	2.414	2	2.414	2.828	3.828			
4.242	3.828	3.414	3	3.414	3.828	4.242			
		4.414	4	4.414					

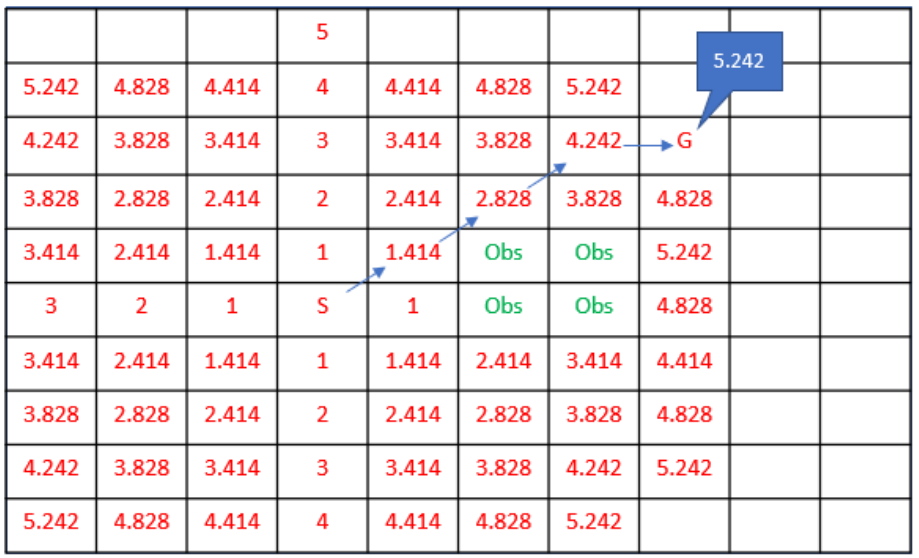
	4.828	4.414	4	4.414	4.828				
4.242	3.828	3.414	3	3.414	3.828	4.242	G		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs	4.828		
3.414	2.414	1.414	1	1.414	2.414	3.414	4.414		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
4.242	3.828	3.414	3	3.414	3.828	4.242			
	4.828	4.414	4	4.414	4.828				

			5						
	4.828	4.414	4	4.414	4.828				
4.242	3.828	3.414	3	3.414	3.828	4.242	G		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
3.414	2.414	1.414	1	1.414	Obs	Obs			
3	2	1	S	1	Obs	Obs	4.828		
3.414	2.414	1.414	1	1.414	2.414	3.414	4.414		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
4.242	3.828	3.414	3	3.414	3.828	4.242			
	4.828	4.414	4	4.414	4.828				

			5						
5.242	4.828	4.414	4	4.414	4.828	5.242			
4.242	3.828	3.414	3	3.414	3.828	4.242	G		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
3.414	2.414	1.414	1	1.414	Obs	Obs	5.242		
3	2	1	S	1	Obs	Obs	4.828		
3.414	2.414	1.414	1	1.414	2.414	3.414	4.414		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
4.242	3.828	3.414	3	3.414	3.828	4.242	5.242		
5.242	4.828	4.414	4	4.414	4.828	5.242			

Navigation in Empty Space (14)

- So the shortest path is (3 x diagonal) + (1 x right). Makes sense!
  - And the cost/length is 5.242.

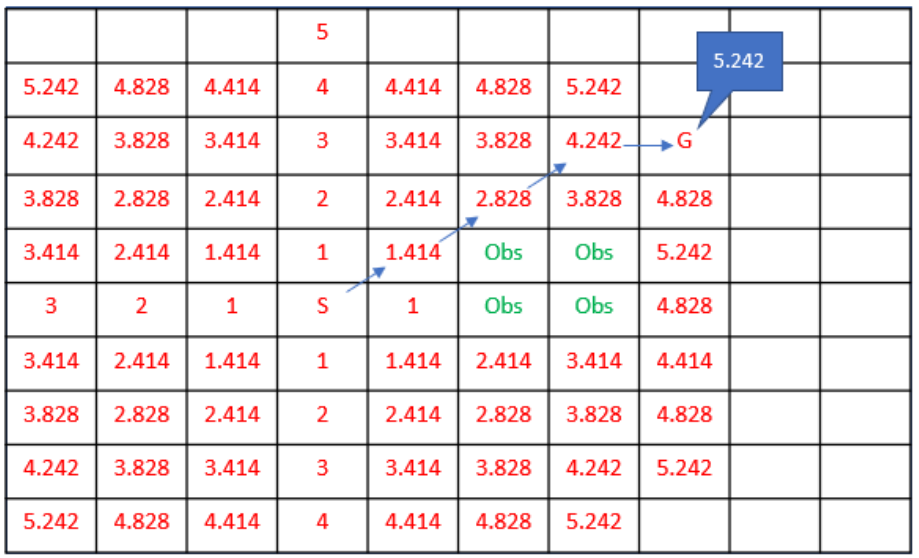


Content

- Dijkstra’s Algorithm: Nodes and Edges
- Dijkstra’s Algorithm: Empty Space (with/out obstacles)
- Greedy Algorithm
- A* Algorithm
- Potential Field

Drawback of Dijkstra's Algorithm

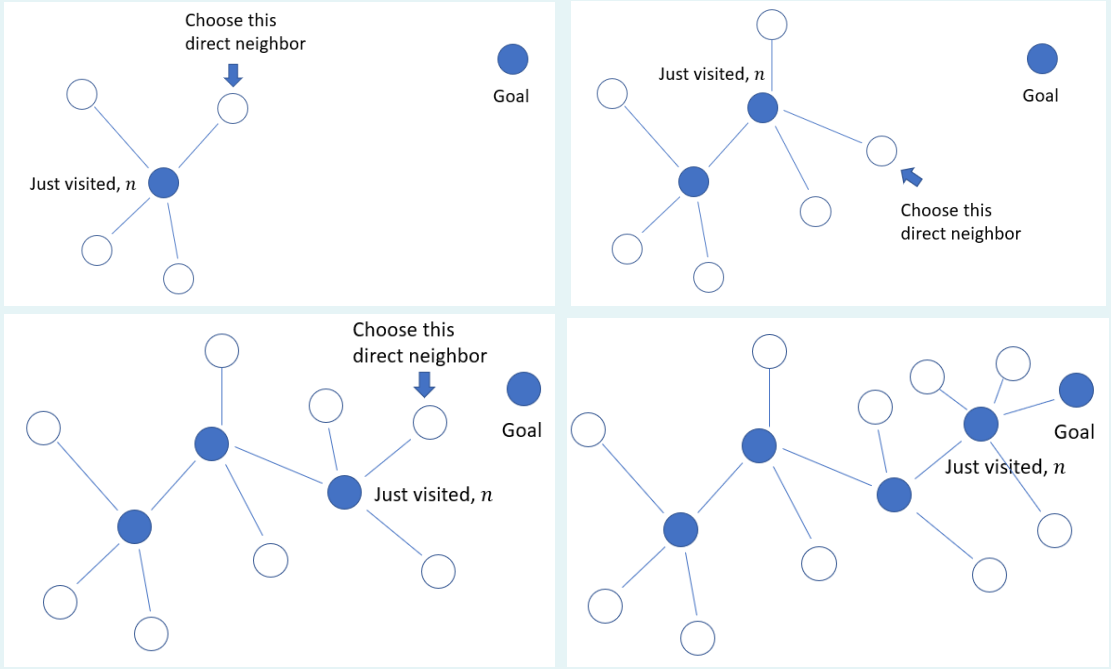
- In the previous example, we saw that the search space expands in all directions.
  - Nodes away from the goal are also being visited, and their neighbors evaluated.
  - This is not efficient.
  - Question: How do we improve the search, to be "only" in the direction of the goal?



			5						
5.242	4.828	4.414	4	4.414	4.828	5.242			
4.242	3.828	3.414	3	3.414	3.828	4.242	G		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
3.414	2.414	1.414	1	1.414	Obs	Obs	5.242		
3	2	1	S	1	Obs	Obs	4.828		
3.414	2.414	1.414	1	1.414	2.414	3.414	4.414		
3.828	2.828	2.414	2	2.414	2.828	3.828	4.828		
4.242	3.828	3.414	3	3.414	3.828	4.242	5.242		
5.242	4.828	4.414	4	4.414	4.828	5.242			

Greedy Algorithm (1)

- The idea is: When looking at direct neighbors of the visited node, look for the one closest physically to the goal.



Greedy Algorithm (2)

- When using the Greedy algorithm, we assume that we do not know how the underlying graph (edges and nodes) looks like.
  - All we care about is the physical / Euclidean distance between the node and the goal.
  - For instance, in the Dijkstra algorithm, we recorded the nodes as: [cost g = path length, node n, predecessor].
  - Here, we record the nodes as: [cost h = Euclidean distance to goal, node n, predecessor].

Greedy Algorithm (3)

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	G	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	Obs	Obs	48	49	50
51	52	53	S 5	55	Obs	Obs	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

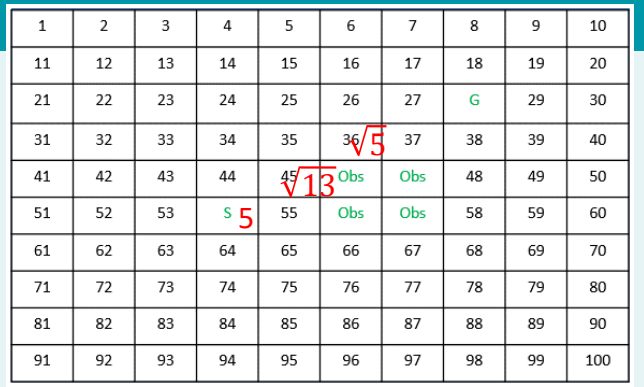
New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Find the one closest to goal.
$\left[\sqrt{4^2+3^2}, S, none\right]$	S	$[5, S, none]$	$\emptyset$	Choose between: Node 53, distance $\sqrt{5^2+3^2}$; Node 43, distance $\sqrt{5^2+2}$; Node 44, distance $\sqrt{4^2+2^2}$; Node 45, distance $\sqrt{3^2+2^2}$; Node 55, distance $\sqrt{3^2+3^2}$; Node 65, distance $\sqrt{3^2+4}$; Node 64, distance $\sqrt{4^2+4}$; Node 63, distance $\sqrt{5^2+4}$; Chosen: 45: $\left[\sqrt{13}, 45, S\right]$

Note: We don't actually need these 3 middle columns, but they are kept here to be similar to the Dijkstra algorithm

Greedy Algorithm (4)

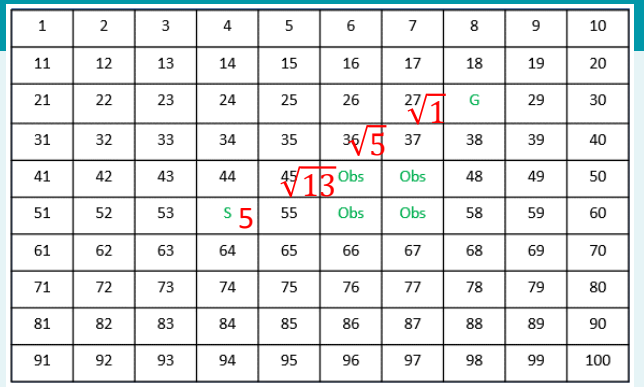
New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Find the one closest to goal.
$[\sqrt{13}, 45, S]$	45	$[5, S, none]$, $[\sqrt{13}, 45, S]$	$\emptyset$	Choose between: Node 44, distance $\sqrt{4^2 + 2^2}$; Node 34, distance $\sqrt{4^2 + 1}$; Node 35, distance $\sqrt{3^2 + 1}$; Node 36, distance $\sqrt{2^2 + 1^2}$; Node 55, distance $\sqrt{3^2 + 3^2}$; Chosen: 36: $[\sqrt{5}, 36, 45]$

Greedy Algorithm (5)



New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Find the one closest to goal.
$[\sqrt{5}, 36,45]$	36	$[5, S, none]$, $[\sqrt{13}, 45, S]$, $[\sqrt{5}, 36,45]$	$\emptyset$	Choose between: Node 35, distance $\sqrt{3^2+1^2}$; Node 25, distance $\sqrt{3^2+0}$; Node 26, distance $\sqrt{2^2+0}$; Node 27, distance $\sqrt{1^2+0^2}$; Node 37, distance $\sqrt{1^2+1^2}$; Chosen: 27: $[1,27,36]$

Greedy Algorithm (6)



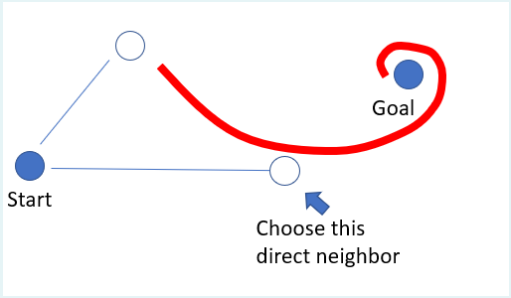
New Front	Node n in "front" which has the lowest cost	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Find the one closest to goal.
$[1,27,36]$	27	$[5, S, none]$, $[\sqrt{13}, 45, S]$, $[\sqrt{5}, 36,45]$, $[1,27,36]$	$\emptyset$	Choose between: Node 26, distance $\sqrt{2^2+0^2}$; Node 16, distance $\sqrt{2^2+1^2}$; Node 17, distance $\sqrt{1^2+1^2}$; Node 18, distance $\sqrt{0^2+1^2}$; Node G, distance $\sqrt{0^2+0^2}$; Node 38, distance $\sqrt{0^2+1^2}$; Node 37, distance $\sqrt{1^2+1^2}$; Chosen: G: $[0, G, 27]$

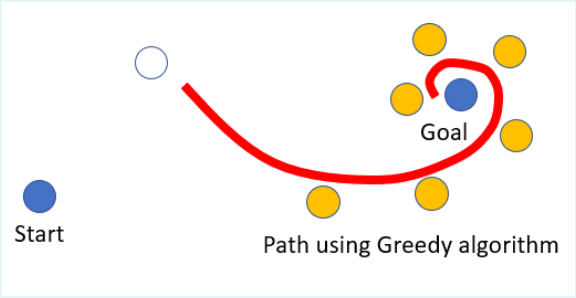
Greedy Algorithm (8)

- We obtain the same path as Dijkstra's algorithm!
- Why do we still want to learn about Dijkstra's algorithm?
- Reason:
  - The same path obtained by Greedy algorithm here is merely a coincidence due to the simple example.
  - It can be shown than Dijkstra' algorithm is optimal, while Greedy algorithm doesn't guarantee optimality!

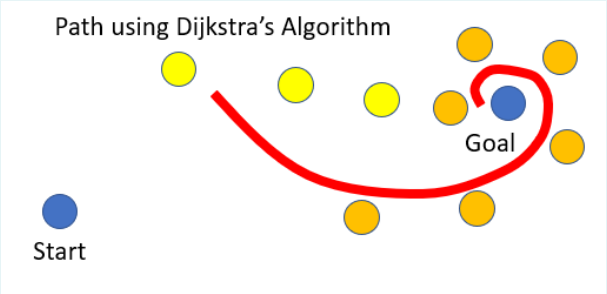
Greedy Algorithm (9)

- Example when Greedy algorithm is not optimal:





- If we had used Dijkstra's algorithm, the path would be shorter!



Content

- Dijkstra’s Algorithm: Nodes and Edges
- Dijkstra’s Algorithm: Empty Space (with/out obstacles)
- Greedy Algorithm
- A* Algorithm
- Potential Field

Dijkstra vs Greedy Algorithms

- A comparison between Dijkstra and Greedy Algorithms:

	Dijkstra	Greedy
Cost	g, path length from start to n	h, estimated distance from n to the goal. ("Estimated" because graph ignored)
Advantage	Optimal solution	Expands fewer nodes
Disadvantage	Expands many nodes	Optimality not guaranteed

A* Algorithm (1)

- How do we get the best of both algorithms?
- We define a new cost by combining both Dijkstra's and Greedy's cost!

$$f = \underbrace{g}_{\text{Dijkstra}} + \underbrace{h}_{\text{Greedy}}$$

  - This is called the A* Algorithm.
  - It has been shown that optimality is guaranteed.
  - Usually expands less nodes than Dijktra.

A* Algorithm (2)

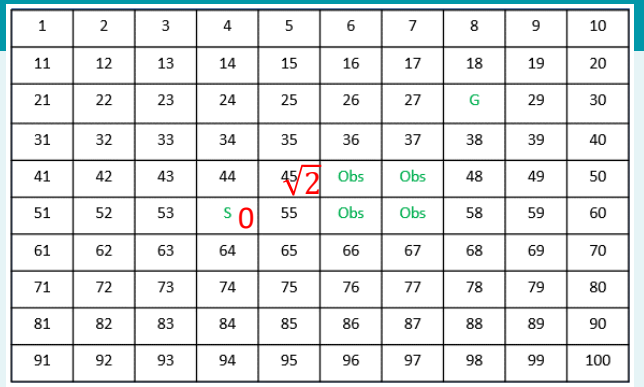
- Implementation:
  - Previously, our front was [g, n, predecessor] in Dijkstra's case, or [h, n, predecessor] in Greedy's case.
  - With A*, we should not just replace g or h by f, because it is still useful to keep track of g, i.e. the length of path travelled.
  - Instead, we add f to the front, i.e. [f, g, n, predecessor].

A* Algorithm (3)

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	G	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	Obs	Obs	48	49	50
51	52	53	S 0	55	Obs	Obs	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

New Front	Node n in "front" which has the lowest cost f	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$\left[0 + \sqrt{4^2+3^2}, 0, S, none\right]$	S	$[5,0,S,none]$	$\emptyset$	$\left[1+\sqrt{4^2+2^2}, 1,44,0\right]$, $\left[1+\sqrt{5^2+3^2}, 1,53,0\right]$, $\left[1+\sqrt{3^2+3^2}, 1,55,0\right]$, $\left[1+\sqrt{4^2+4^2}, 1,64,0\right]$, $\left[\sqrt{2}+\sqrt{5^2+2^2}, \sqrt{2},43,0\right]$, $\left[\sqrt{2}+\sqrt{3^2+2^2}, \sqrt{2},45,0\right]$, $\left[\sqrt{2}+\sqrt{5^2+4^2}, \sqrt{2},63,0\right]$, $\left[\sqrt{2}+\sqrt{3^2+4^2}, \sqrt{2},65,0\right]$

A* Algorithm (4)



New Front	Node n in "front" which has the lowest cost f	Visited	Front set after n deleted	Determine direct <u>unvisited</u> neighbors of n. Update "front"
$[5.472,1,44,0]$, $[6.831,1,53,0]$, $[5.243,1,55,0]$, $[6.657,1,64,0]$, $[6.799,\sqrt{2},43,0]$, $[5.020,\sqrt{2},45,0]$, $[7.817,\sqrt{2},63,0]$, $[6.414,\sqrt{2},65,0]$	45	$[5.020,\sqrt{2},45,0]$	$[5.472,1,44,0]$, $[6.831,1,53,0]$, $[5.243,1,55,0]$, $[6.657,1,64,0]$, $[6.799,\sqrt{2},43,0]$, $[7.817,\sqrt{2},63,0]$, $[6.414,\sqrt{2},65,0]$	Same idea as shown previously.

A* Algorithm (5)

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	G	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45 $\sqrt{2}$	Obs	Obs	48	49	50
51	52	53	S 0	55	Obs	Obs	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

New Front

$[5.472,1,44,0]$,
$[6.831,1,53,0]$,
$[5.243,1,55,0]$,
$[6.657,1,64,0]$,
$[6.799,\sqrt{2},43,0]$,
$[5.020,\sqrt{2},45,0]$,
$[7.817,\sqrt{2},63,0]$,
$[6.414,\sqrt{2},65,0]$

f g

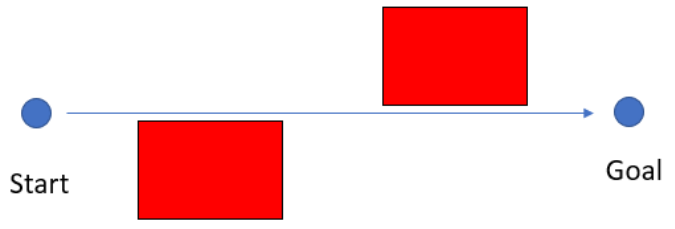
- Note: When running Dijkstra's algorithm, we had 4 nodes with the same cost ($g = 1$).
- So we needed to explore all of them, sooner or later.
- With A*, the cost f are less likely to be repeated. Thus we expand less nodes.
- Much faster!

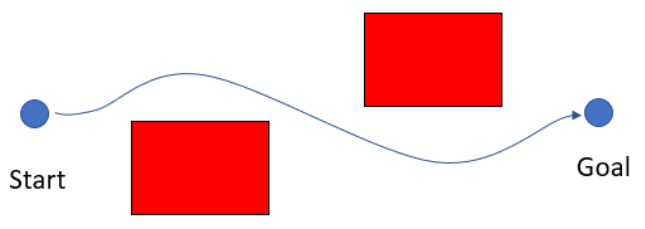
Content

- Dijkstra’s Algorithm: Nodes and Edges
- Dijkstra’s Algorithm: Empty Space (with/out obstacles)
- Greedy Algorithm
- A* Algorithm
- Potential Field

Potential Field (1)

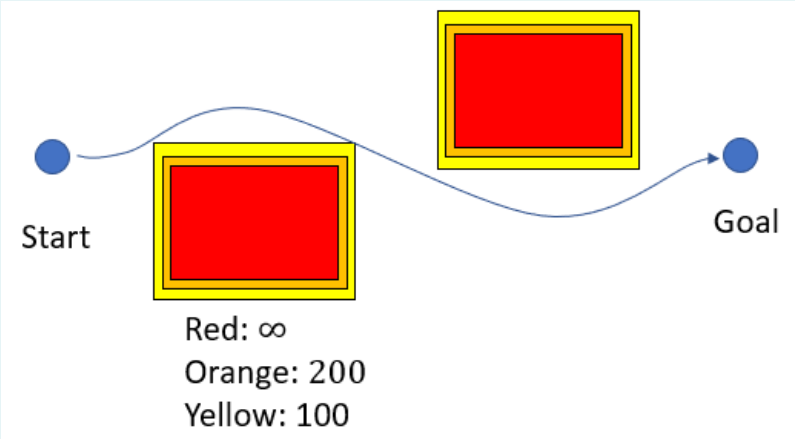
- In the final section of today's lecture, we will look at a concept called potential field.
  - What is it use for?
  - Using what we learnt so far, if given 2 obstacles on "both sides of the road", the robot will move straight to the goal, squeezing in between the 2 obstacles (left image) → Not realistic.
  - A more realistic path is shown on the right image.





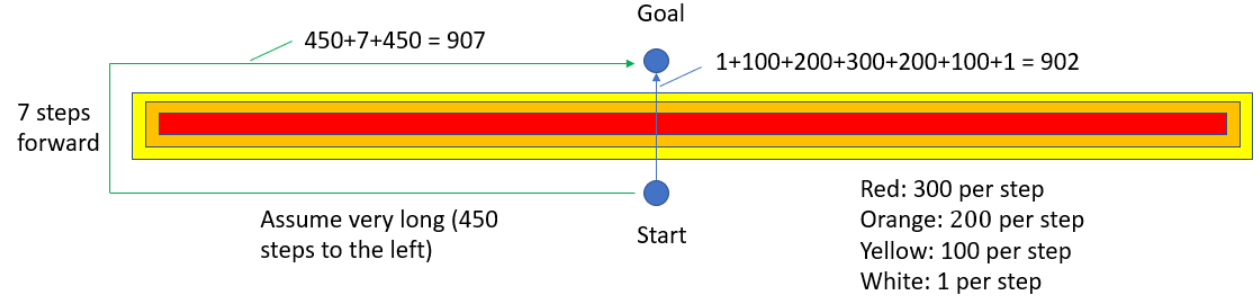
Potential Field (2)

- This can be achieved by adding a potential field around the obstacle.
  - Rather than 0 for free space and ∞ for obstacle, we now give a continuous cost around the obstacle.



Potential Field (3)

- In fact, the cost for the real obstacle need not be ∞. We can reduce it to a finite number.
  - This would allow the robot to "climb over" the obstacle, if the path going around the obstacle is more costly.



- You are free to design the cost function / potential field to achieve a desired outcome!

Thank you for your attention!

Any questions?