Introduction to Robotics

Felipe P. Vista IV

Grading

➢ Attendance 5%

Name (Original Name)	User Email	Join Time	Leave Time	Duration (Minutes)
		4/12/2021 9:12	4/12/2021 10:14	62
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:13	4/12/2021 9:13	1
		4/12/2021 9:13	4/12/2021 9:14	2
		4/12/2021 9:14	4/12/2021 9:14	1
		4/12/2021 9:14	4/12/2021 9:14	1
		4/12/2021 9:14	4/12/2021 10:14	60

Bad ZOOM User Name (Absent)

- ➢ Iphone → Not your name
- ➢ SiAko 202100001 → Wrong order
- ➢ SiAko → Name only
- ➢ 202100001 → ID Num only

ZOOM User Name (Present)

- ➢ University ID Num_Name
- ➢ 202100001 SiAko → GOOD (Present)

Name (Original Name)	User Email	Total Duration (Minutes)
		62
		63
		62
		62
		63
		62
		63

Student Responsibilities

- Download/Install ZOOM app for online lecture
  - *Zoom profile must be your **OASIS ID+name** similar to OASIS*
  - *Ex.: **202061234 YourName***
  - *If you are asked, but no reply, then you'll be out of zoom & mark **absent***
- Regularly login, check OLD IEILMS for updates, notifications
  - *https://ieilmsold.jbnu.ac.kr*
  - *Presentations & lecture videos will be uploaded after class*
- Regularly check Kakao Group Chat for class
  - *Everybody must have a Kakao talk account*
  - *Search & add account "**botjok**", introduce yourself and name of class ("Robotics"), then you will be added to the group chat*

Intro To Robotics

MAPPING-BASED NAVIGATION

Intro

- After having a map
  - *Can be supplied by user or*
  - *Can be created by robot*
  - *We discuss path-planning (a higher-level algorithm)*
- **Ex.: robot in hospital**
  - *Transport medications/supplies*
  - *From storage areas to nurse/doctors*
- What is best way for Point A → Point B?
  - *Maybe multiple ways of moving through corridors*
  - *Also maybe short paths robot not allowed to take*
    - i.e. corridors near operating rooms

Intro

- Three algorithms
  - *Planning shortest path from start* **S** *to goal* **G**
  - *Assuming map of area indicate position of obstacles in the area*

1) Ed~~s~~gar W. Dijkstra
   - *A pioneer of CS, proposed algorithms for shortest path problem*
   - *One for a grid map and one for a continuous map*
2) A* algorithm
   - *Improvement on Dijkstra's algorithm*
3) Combined algorithm
   - *High-level path planning w/ a low-level obstacle avoidance algorithm*

Mapping-Based Navigation

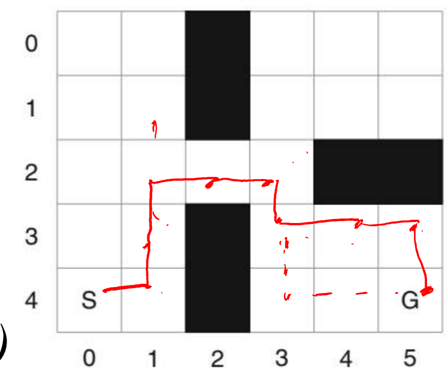
- Dijkstra's Algorithm for a Grid Map
- Dijkstra's Algorithm for a Continuous Map
- Path-planning with the A* Algorithm
- Path Following and Obstacle Avoidance

Dijkstra's Algorithm for a Grid Map

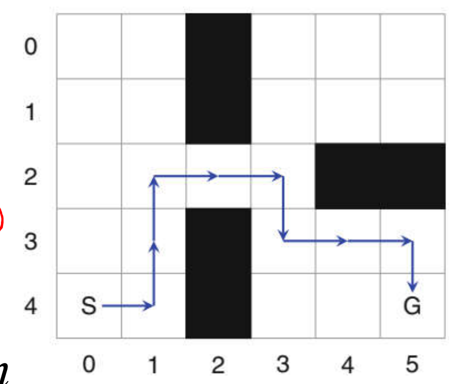
- Algorithm for discrete graph of nodes & edges
- Described for a grid map of cells (a)
  - *Starting pt* → **S**, *Goal* → **G**, *Obstacle* → **Black**
  - *Robot sense & move to neighbor of cell* **c** *it occupies*
  - *Simplicity, neighbors are 4-cells (Horizontally/Vertically)*
  - *Shortest path for* **S** → **G** *is given in (b)*

$$(4,0) \rightarrow (4,1) \rightarrow (3,1) \rightarrow (2,1) \rightarrow (2,2) \rightarrow$$
$$(2,3) \rightarrow (3,3) \rightarrow (3,4) \rightarrow (3,5) \rightarrow (4,5).$$

- Two versions of algorithm will be presented
  1) *Cost of moving from one cell to a neighbor is* ***constant***
  2) *Each cell can have* ***different cost****, therefore shortest path geometrically is not necessarily shortest path with costs*



(a) Grid map



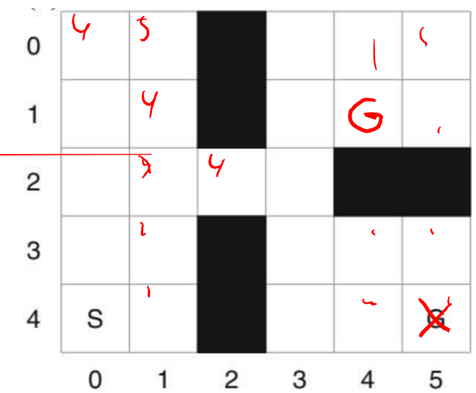
(b) Shortest path found

For a Grid Map w/ Constant Cost

```
integer n ← 0                   // Distance from start
cell array grid ← all unmarked  // List of frontier cells
cell list path ← empty          // Shortest path
cell current                    // Current cell in path
cell c                          // Index over cells
cell S ← …                      // Source cell
cell G ← …                      // Goal cell
```

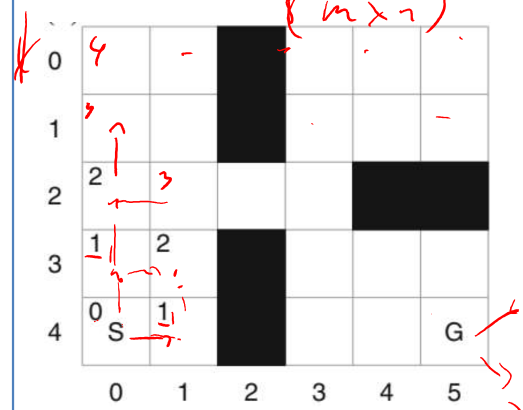
```
mark S with n
while G is unmarked
   n ← n + 1
   for each unmarked cell c in grid
         next to a marked cell
      mark c with n
current ← G
append current to path
while S not in path
   append lowest marked neighbor c
      of current to path
   current ← c
```

pseudocode



(a) Grid map

(m×n)



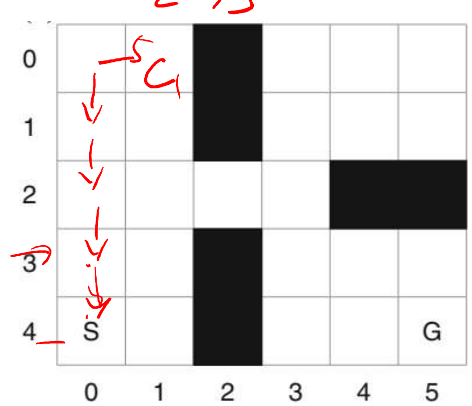
(b) First two iterations

(r_m, c_n)

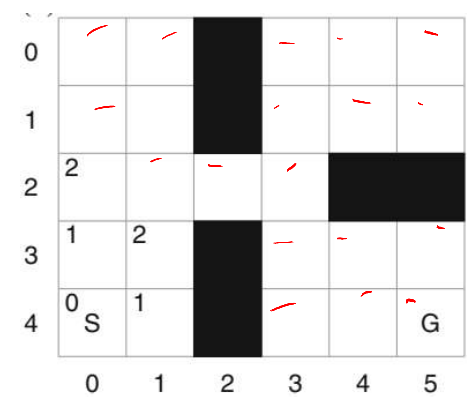
A10.a Dijkstra's algorithm on a grid map

For a Grid Map w/ Constant Cost

- Algorithm incrementally marks each cell **c**
  - *With number of steps needed to reach ~~s~~ c from* **S**
  - *Step count number in upper-left hand corner of cell*
- Initially, mark "0" → cell **S**
  - *Since no steps needed from* **S** → **S**
- Now, mark "1" → every neighbor of **S**
  - *Since one step away from* **S**
- Then, mark "2" → every neighbor of cell w/ "1"
  - *Since two steps away from* **S**
- Figure (b)
  - *Grid map after two iterations of the algorithm*



(a) Grid map



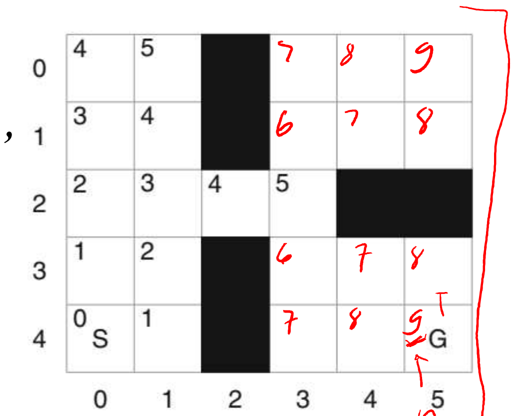
(b) First two iterations

For a Grid Map w/ Constant Cost

- Algorithm continues iteratively
  - *If cell marked "**n**" → mark unmarked neighbors "**n+1**"*
  - *When **G** is finally marked,*
    *We now know shortest distance for **S** → **G** is n*
- Now easy to find shortest path

G → S

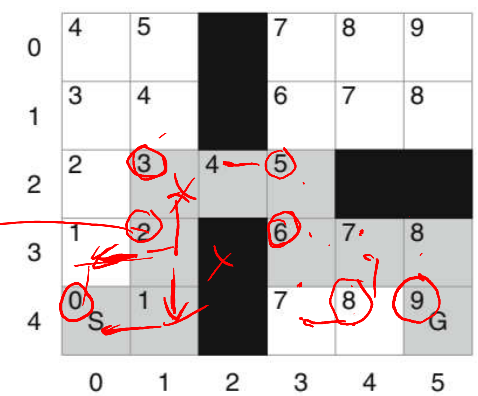
  - *By working backwards from **G** (shortest path (b) → gray)*
  - *Start from **G**(4,5) → prev cell either (4, 4) or (3, 5)*
    *both 8-steps from **S** → shows there's more than 1 path*
  - *Arbitrarily choose (3, 5)*
  - *From each chosen cell marked **n** → choose cell*
    *marked (**n -1**)*
  - *Until cell **S** marked **0** is selected*



(a) After 5 iterations

2 = n

(n-1 = 1)



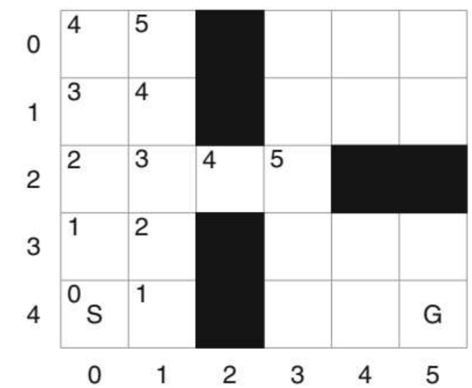
(b) Final map w/ shortest path

For a Grid Map w/ Constant Cost

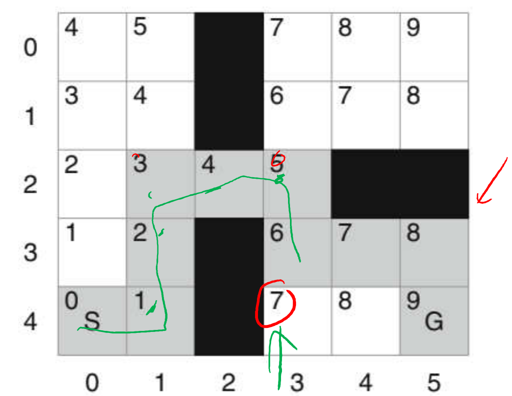
- Now easy to find shortest path
  - *Working backwards from **G** (shortest path (b) → gray)*
  - *Start from **G**(4,5) → prev cell either (4, 4) or (3, 5) both 8-steps from **S** → shows there's more than 1 path*
  - *Arbitrarily choose (3, 5)*
  - *From each chosen cell marked **n** → choose cell marked (**n -1**)*
  - *Until cell **S** marked **0** is selected*

$$(4,5)\to(3,5)\to(3,4)\to(3,3)\to(2,3)\to$$
$$(2,2)\to(2,1)\to(3,1)\to(4,1)\to(4,0).$$

  - *By reversing list → shortest path for (**S** →**G**) obtained*
  - *We check its same from what found intuitively previously*



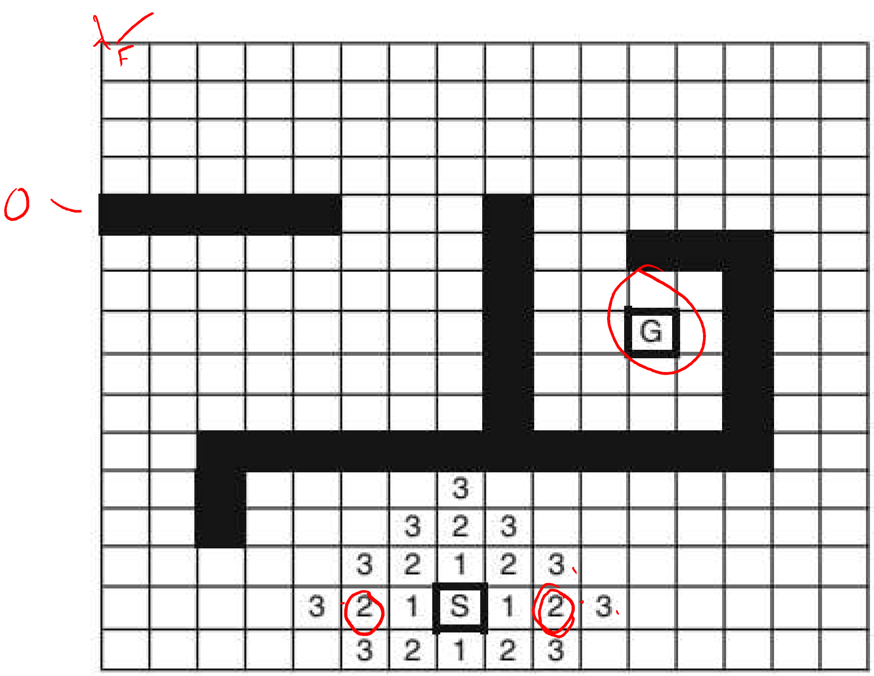
(a) After 5 iterations



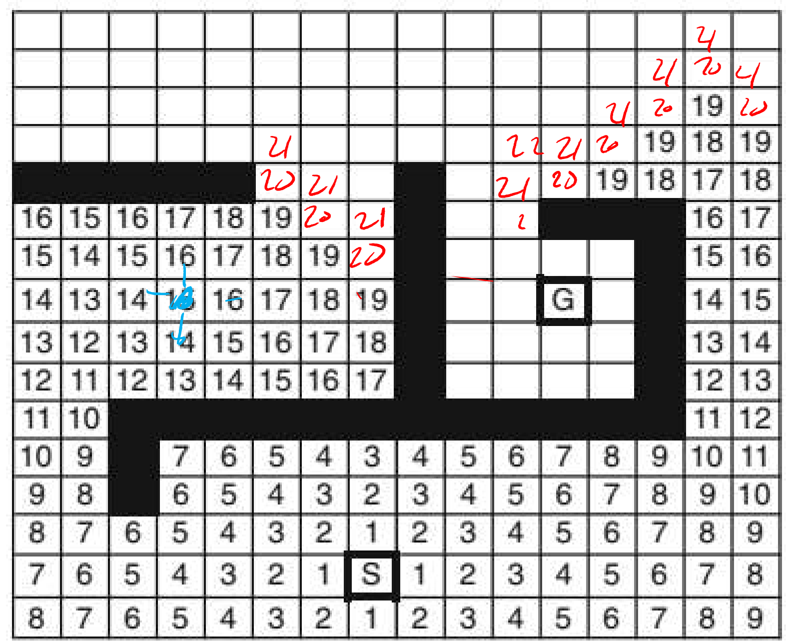
(b) Final map w/ shortest path

For a Grid Map w/ Constant Cost

- **Ex. : A more complicated example**
  - *16 x 16 grid; Goal G enclosed in an obstacle and hard to reach*



(a) After 3 iterations

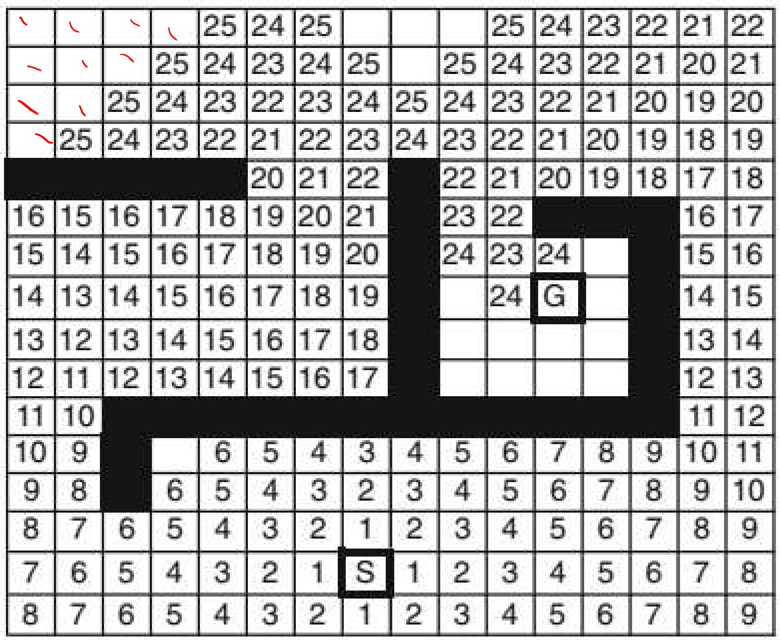


(b) After 19 iterations

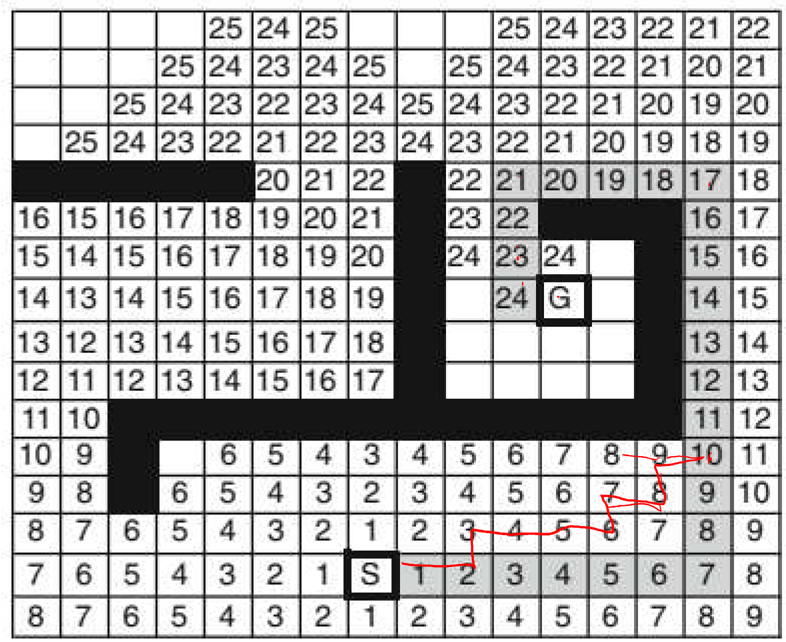
Mapping-Based Navigation

For a Grid Map w/ Constant Cost

- **Ex. : A more complicated example**



(c) After 25 iterations (G is found)

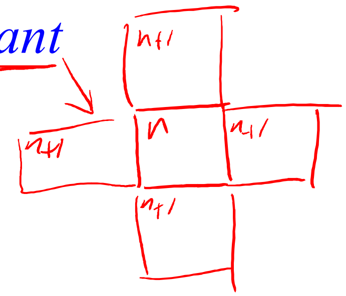


(d) Shortest path shown

G found after 25 steps only but algorithm is not efficient for this map,
since algorithm explored 256 – 25 = 231 cells!

For a Grid Map w/ Variable Cost

- Dijkstra's previous algorithm
  - *Assume cost of taking a step from one cell to next is constant*
  - *Line 3 : Adds 1 to cost for each neighbor*



- Dijkstra's algorithm modified
  - *take into account variable cost of each step*
- **Ex. : Area in environment covered w/ sand**
  - *more difficult to for robot move*
- In the modified algorithm
  - *Instead of adding "1" to the cost for each neighboring cell*
  - *Add "k" to each neighboring cell to reflect additional cost*

For a Grid Map w/ Variable Cost

- **Ex .: Area in environment covered w/ sand**
  - *Cells w/ diagonal lines → sandy → cost of moving through is 4 instead of 1*

2	1	2	3	4	5	6	7	8	9	10
1	S	1	2	3	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9	10
3	2	3	4	5	9	10	11	12	13	11
4	3	4	5	9	12	14	15	16	13	12
5	4	5		13	14	15	16	15	14	13
6	5	6		14	15	16		16	15	14
7	6	7		15	16			G	16	15
8	7	8		14	15	16				16
9	8	9		13	14	15	16			
10	9	10	11	12	13	14	15	16		

(a) Variable cost per cell, cost = 4 / 3

2	1	2	3	4	5	6	7	8	9	10
1	S	1	2	3	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9	10
3	2	3	4	5	7	8	9	11	12	11
4	3	4	5	7	9	10	11	13	13	12
5	4	5		9	10	11	12	13		13
6	5	6		10	11	12	13			
7	6	7		11	12	13	G			
8	7	8		12	13					
9	8	9		13						
10	9	10	11	12	13					

(b) Variable cost per cell, cost = 2

n → n-1
n → (n - n_d)

C

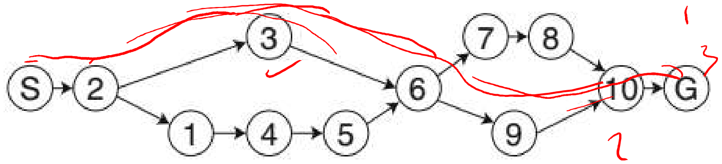
  - *(a) Shortest path(gray): 17 steps & costs 17 since it go around the sand(=4)*
  - *(b) Shortest path(gray): 12 steps & costs 14 since only 2 steps through sand*

Mapping-Based Navigation

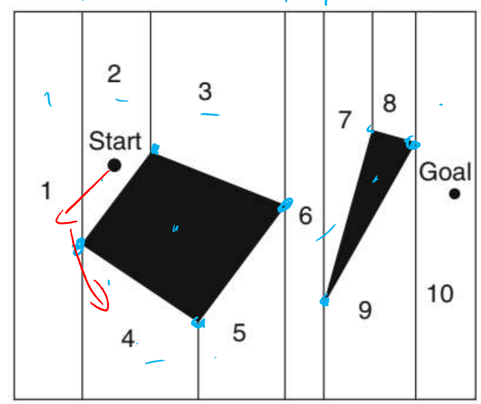
- Dijkstra’s Algorithm for a Grid Map
- Dijkstra’s Algorithm for a Continuous Map
- Path-planning with the A* Algorithm
- Path Following and Obstacle Avoidance

Dijkstra’s Algorithm for Continuous Map

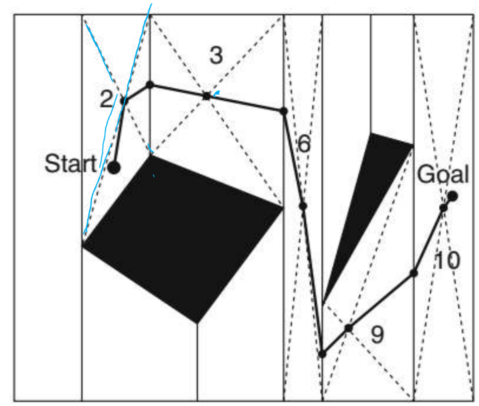
- Continuous map
  - *Area is ordinary two-dimensional geometric plane*
- An approach for Dijkstra’s algorithm
  - *Transform (map → discrete map)*
  - *Drawing vertical lines from upper & lower edges of env to each corner of the obstacle → Dividing area to finite number of segments → Each represented as node in a graph*
  - (a) *7 vertical lines divide map to 10 segments* (c)
  - (b) *Adjacency relation of segments define graph edges*



(c) Graph constructed from segmented continuous map



(a) Segmenting by vertical lines



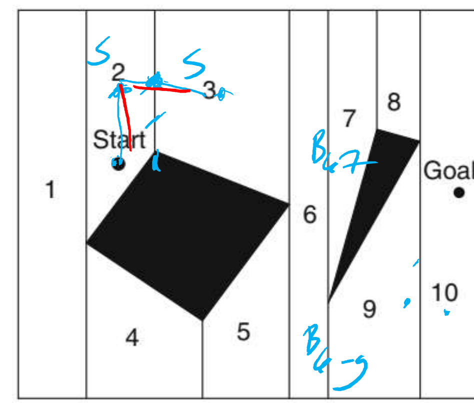
(b) Path through the segments

Dijkstra's Algorithm for Continuous Map

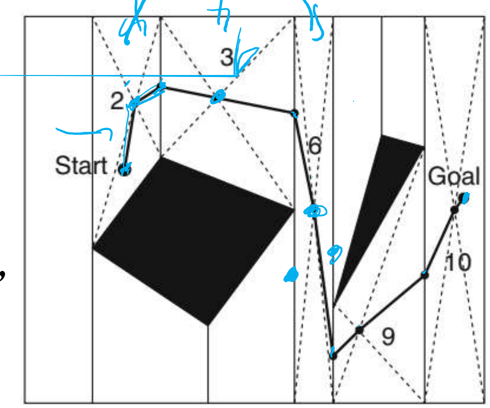
- There is directed edge from segments A to B
  - *If A and B share common border*
  - *i.e. There are edges from node 2 to nodes 1 & 3*
    - Since they share an edge with segment 2
- Shortest path between vertices 2 & 10?
  - *Vertex 2 represent segment w/ starting point*
  - *Vertex 10 segment with goal*
  - *Using Dijkstra's algorithm*

$$\boxed{S \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 10 \rightarrow G}$$

  - *It is shortest in terms of number of edges of the graph, but not shortest path in the environment*
  - *Because constant cost assigned to each edge, even if segments have various sizes*



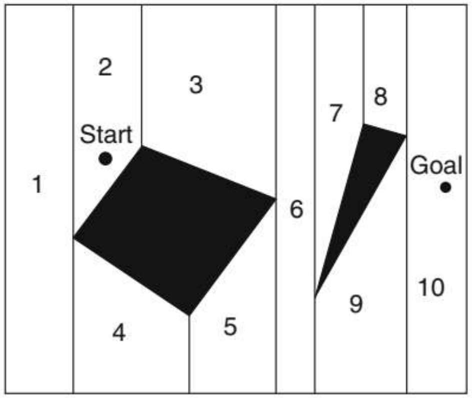
(a) Segmenting by vertical lines



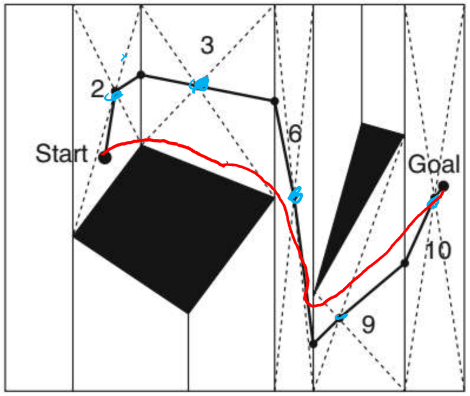
(b) Path through the segments

Dijkstra's Algorithm for Continuous Map

- Each vertex represent large segment of env
  - *Must know how moving from one vertex to another translates to moving from one segment to another*
- Figure (b) show one possibility
  - *Each segment associated w/ its geometric center*
  - *Indicated by intersection of dotted diagonal lines*
  - *Path is through segment centers except geometric locations of start and goal*
  - *Although this is a reasonable method without further knowledge of environment*
  - *Does not optimal path*
    - Which should stay close to the borders of the obstacles



(a) Segmenting by vertical lines

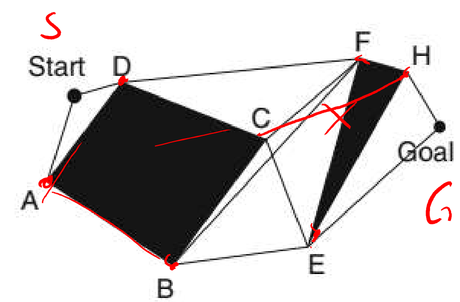


(b) Path through the segments

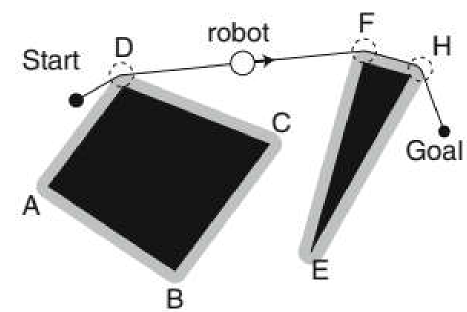
Dijkstra’s Algorithm for Continuous Map

- Another approach use **visibility graph** (a)(b)
  - *Each vertex of graph represent corner of obstacle and there are vertices for start & goal positions*
  - *There is an edge from vertex v_1 to v_2 if its corresponding corners are visible*
- **Ex.: There is edge C → E**

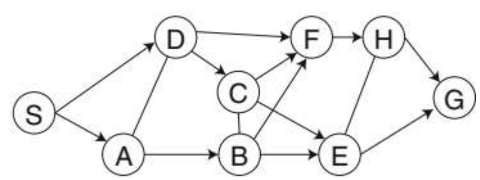
  not blocked – direct path
  - *Because corner E of right obstacle is visible from corner C of left obstacle*
  - *(c) graph formed by the edges and nodes*
    - Represent all candidates for shortest path



(a) W/ lines from corner to corner



(b) Path through the corners



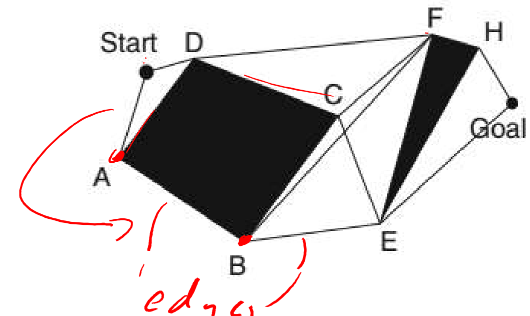
(c) Graph constructed from segmented continuous map

Dijkstra's Algorithm for Continuous Map

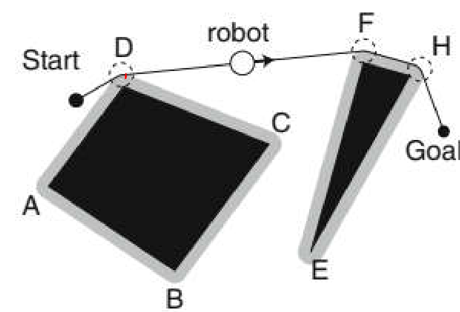
- Paths in graph represent paths in environment
  - *Robot can simply move from corner to corner*
- These paths are shortest paths
  - *Since no path like A→B,*
    is shorter than straight line from A to B
- Dijkstra's algorithm gives shortest path as:

$$\boxed{S \to D \to F \to H \to G}$$

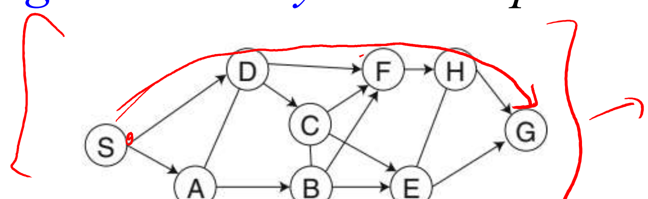
  - *In this case shortest path in terms of number of edges is also the geometrically shortest path.*



(a) W/ lines from corner to corner



(b) Path through the corners



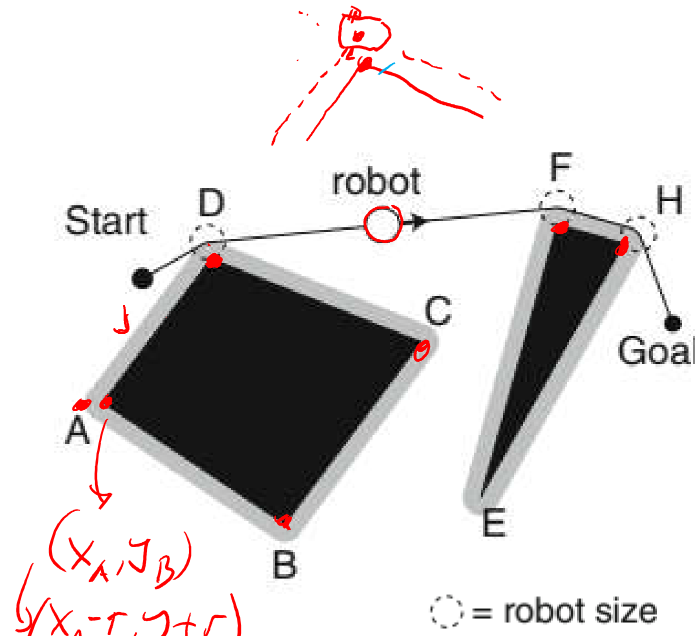
(c) Graph constructed from segmented continuous map

Dijkstra's Algorithm for Continuous Map

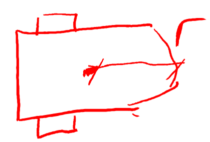
- Dijkstra's algorithm gives shortest path as:

$$\boxed{S \to D \to F \to H \to G}$$

  - *In this case shortest path in terms of number is also the geometrically shortest*
- Real robot cannot follow this path
  - *It has finite size so its center cannot follow border of an obstacle*
  - *It must maintain minimum distance from each of the obstacle*
  - *Can be implemented by expanding size of obstacles with size of the robot (b)*
  - *Resulting path is still optimal*
  - *Path can now be traversed by robot*



(b) Path through the corners



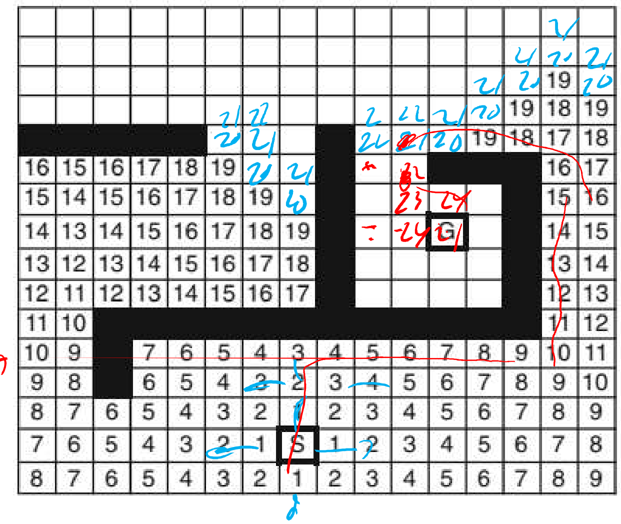
Mapping-Based Navigation

- Dijkstra's Algorithm for a Grid Map — constant variable
- Dijkstra's Algorithm for a Continuous Map — geometric visual
- **Path-planning with the A* Algorithm**
- Path Following and Obstacle Avoidance

Path Planning with A* Algorithm

- Dijkstra's algorithm
  - *Search for goal in all directions*
  - *Can be efficient in complex environment*
  - *But not for simple path, ex: straight line to goal*
- From figure (a)
  - Cell w/ distance 19 from S :
    Near upper right corner of center obstacle
  - *Two more steps to left → there's cell marked 21→ there is path to G not blocked by an obstacle*
  - *Definitely no reason to explore but Dijkstra's will continue to do so*
- More efficient algorithm
  - *If it knows that it was close to the goal cell*

$g(x,y)$



(a) After 19 iterations

Path Planning with A* Algorithm

- A* ("A star") Algorithm
  - *Similar to Dijkstra's,*
    But often more efficient since extra information used to guide the search
  - *Consider not only num of steps from start cell,*
    *But also **heuristic function** that give idea of preferred direction to search*
- Cost function g(x, y) previously used
  - *Give actual number of steps from start cell*

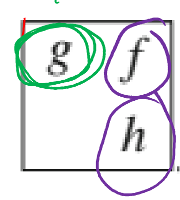
current → start
c → s

- *Dijkstra's*
  - *Expanded search to start w/ cells marked highest values of g(x, y)*
- *A* algorithm*
  - *Cost function includes the heuristic function:*

$$f(x,y) = g(x,y) + h(x,y)$$

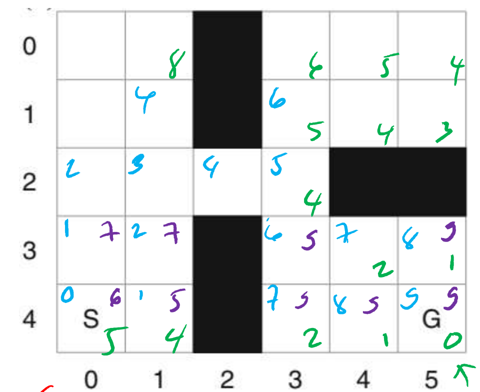
Path Planning with A* Algorithm

- Number of steps (G → cell(x, y))
  - ***w/o taking obstacles into account***
  - *used as heuristic function to demonstrate A**
  - *Can be precomputed & remain available throughout*
- Keep track of ***f***, ***g***, ***h*** values in each cell
  - *By displaying them in different corners*
    - Cost func, ***g*** → *upper left*
    - Cost func, ***f*** → *upper right*
    - Heuristic func, ***h*** → *lower right*

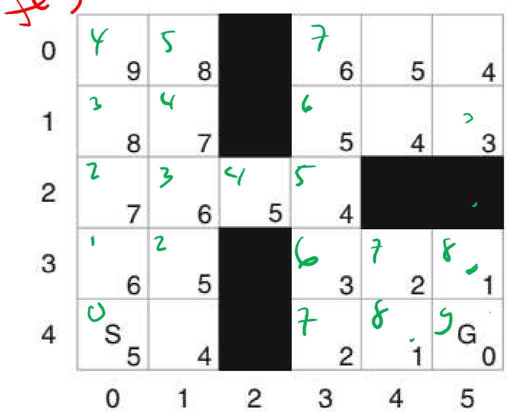


> ****g*** = num of steps from S → Cell(x, y)
> ****f*** = g(x, y) + *h*(x, y)
> ****h*** = num of steps from G → Cell(x, y)

  - *Heuristic function of grid map (a) shown in (b)*



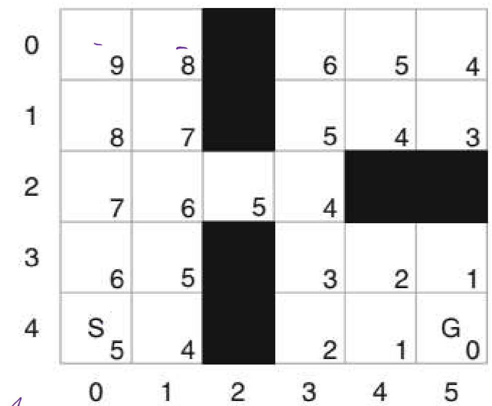
(a) Grid map



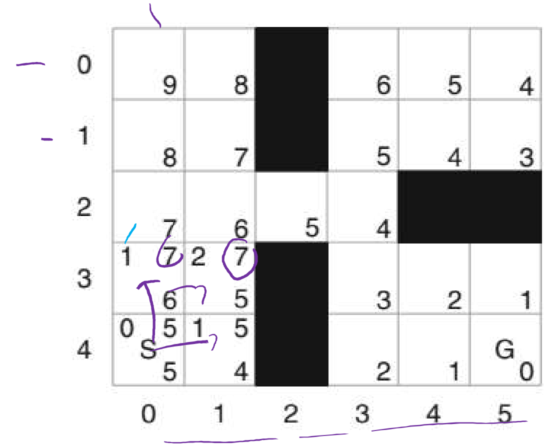
(b) Heuristic function

Path Planning with A* Algorithm

- After first two iterations of A*(b)
  - *Cells (3,1) & (3,0) get same cost function **f** = "7"*
  - *Cell(3,1) closer to S (by number of steps counted, 1)*
    Cell(3,0) closer to G (by heuristics, 5)
- Open cells
  - *Cells not yet expanded*
  - *Must maintain data structure of it*
  - *Use notation (**r**, **c**, **v**)*
    - ***r** & **c*** → row & column of cell
    - ***v*** → ***f*** value of cell, $f = g + h$
  - *Each time open cell expanded → remove from list → new cells added*



(a) Heuristic function



(b) First 2 iterations of A*

Path Planning with A* Algorithm

- Ordered list
  - *Cells w/ lowest values appear first*
  - *Make it easy to decide w/c cell to expand first*
- First three lists (***r***, ***c***, ***v***) corresponding to (a):

$$(4,0,5)$$

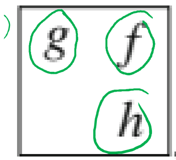
$$(4,1,5),(3,0,7)$$

$$(3,0,7),(3,1,7)$$

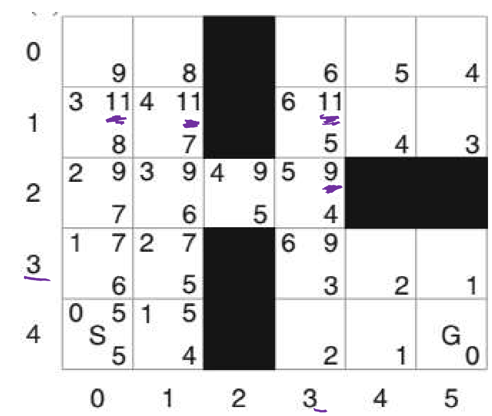
- List of open cells (***r***, ***c***, ***v***) after 6 steps (b):
  - *By checking values of g (upper left)*

$$(3,3,9),(1,0,11),(1,1,11),(1,3,11)$$

  - *Recall "Open cells are cells not yet expanded"*



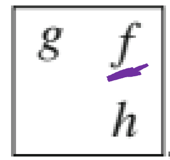
(a) First 2 iterations of A*



(b) A* after 6 steps

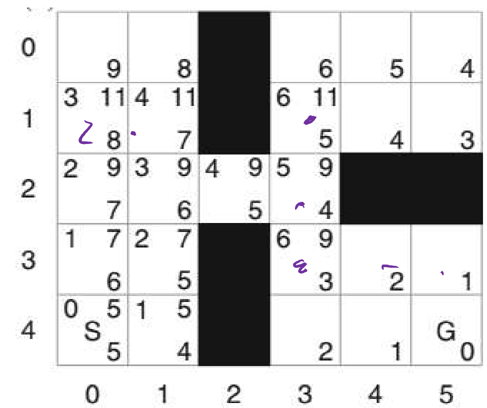
More Complex Example of A*

- A* expands cell (3, 3, 9) since lowest ***f***
  - *Other cells in list has a value of "11"*

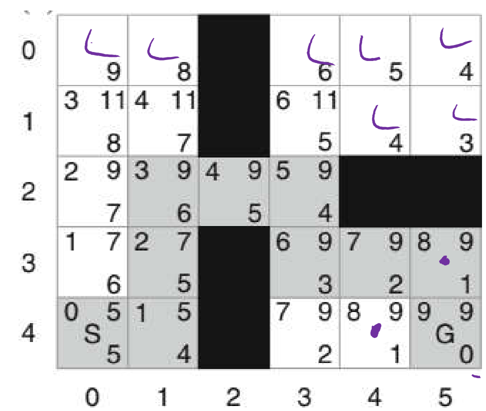


- Continuing A* (b)
  - *Goal cell is reached with **f** value of "9"*
  - *Shortest path (in gray) is displayed*
- Last list before reaching goal is :

  (3,5,9),(4,4,9),(1,10,11),(1,1,11),(1,3,11)
  - *Doesn't matter which nodes with "9" is chosen, Both reaches goal cell (4, 5, 9)*
- All upper-right cells in grid not explored
  - *Cell (1, 3) has f = "11", it will never be smallest value*



(a) A* after 6 steps



(b) A* reach goal cell & find shortest path

****Dijkstra's*** *explored all **24** non-obstacle cells but **A*** only **17**.*

More Complex Example of A*

- Recall previous grid map with sand on some cells
  - **g** *func will give higher values for cost of moving to these cells*
  - *(a)* **g** *func as computed by Dijkstra's*
  - *(b) heuristic func* ***h*** *in absence of of obstacles and sand*

2	1	2	3	4	5	6	7	8	9	10
1	S	1	2	3	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9	10
3	2	3	4	5	7	8	9	11	12	11
4	3	4	5	7	9	10	11	13	13	12
5	4	5		9	10	11	12	13		13
6	5	6		10	11	12	13			
7	6	7		11	12	13	G			
8	7	8		12	13					
9	8	9		13						
10	9	10	11	12	13					

g(x,y)

(a) Number of steps to goal

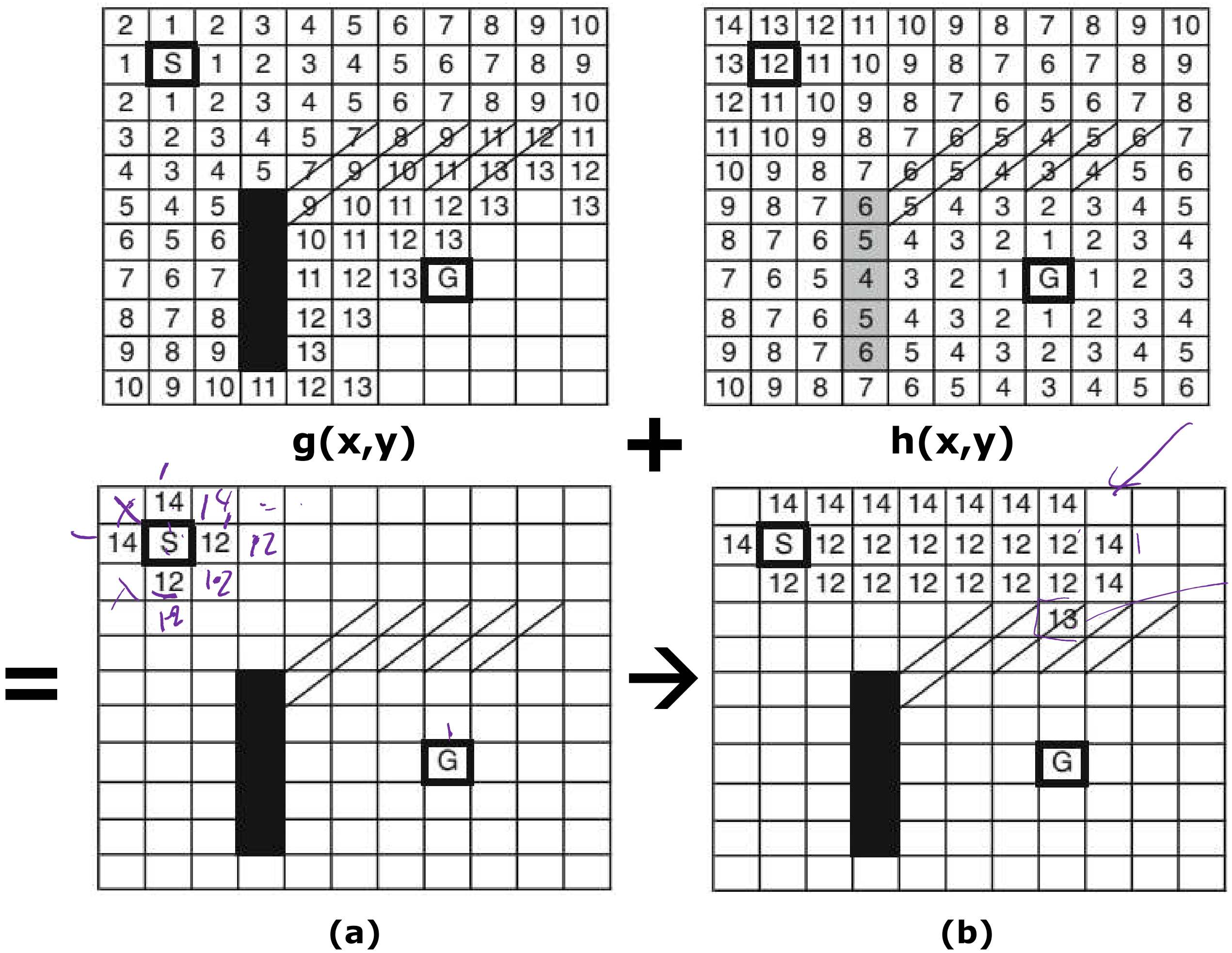
+

14	13	12	11	10	9	8	7	8	9	10
13	12	11	10	9	8	7	6	7	8	9
12	11	10	9	8	7	6	5	6	7	8
11	10	9	8	7	6	5	4	5	6	7
10	9	8	7	6	5	4	3	4	5	6
9	8	7	6	5	4	3	2	3	4	5
8	7	6	5	4	3	2	1	2	3	4
7	6	5	4	3	2	1	G	1	2	3
8	7	6	5	4	3	2	1	2	3	4
9	8	7	6	5	4	3	2	3	4	5
10	9	8	7	6	5	4	3	4	5	6

h(x,y)

(b) Heuristic function

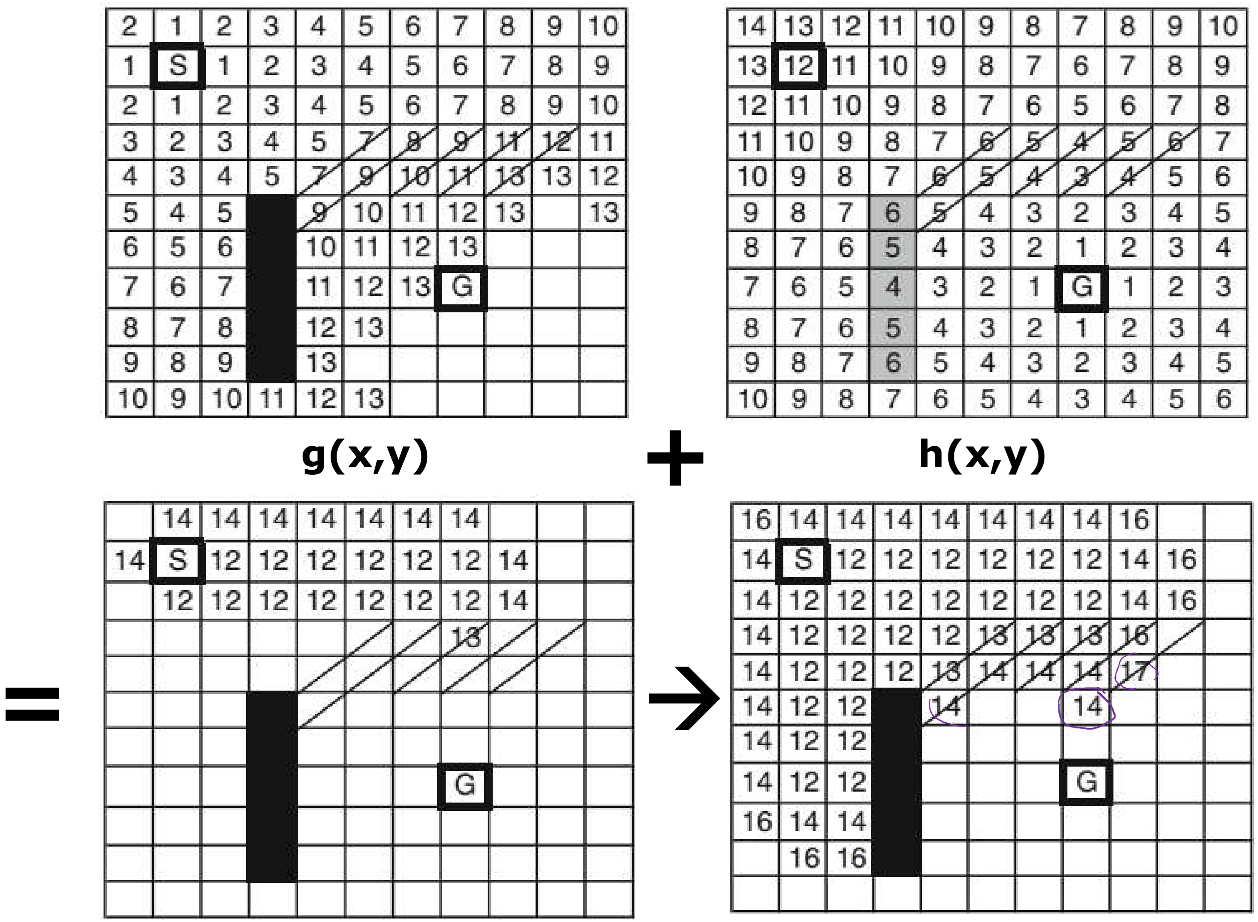
More Complex Example of A*



- **Figure (a)**
  - Need search to top-left
  - Since *f* values of cells above & left of S higher than to right & below

$f = g + h$

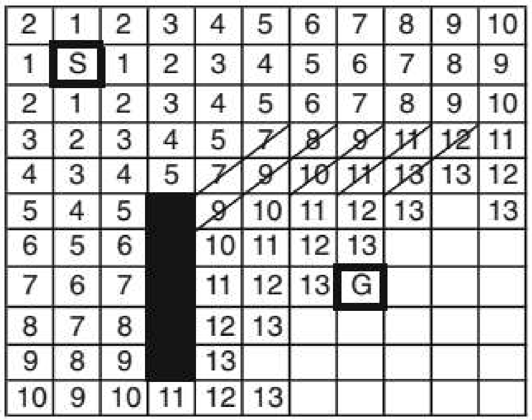
More Complex Example of A*



(b) (c)

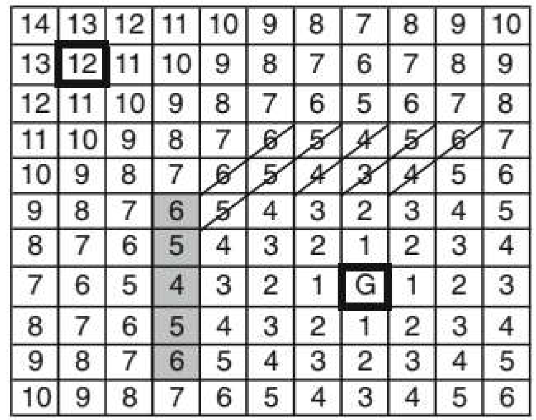
- **Figure (b)**
  - First sand cell has value 13, so expand to cells with lower cost of 12 to the left

More Complex Example of A*



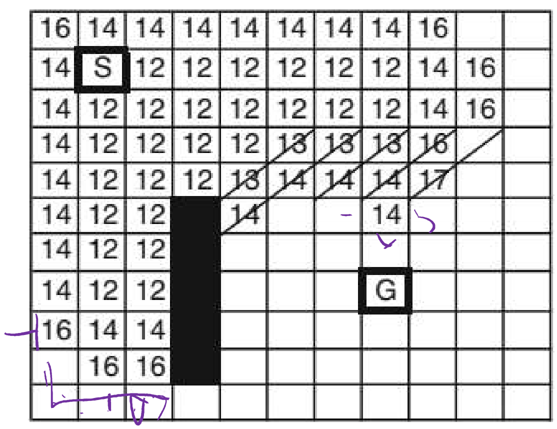
g(x,y)

+

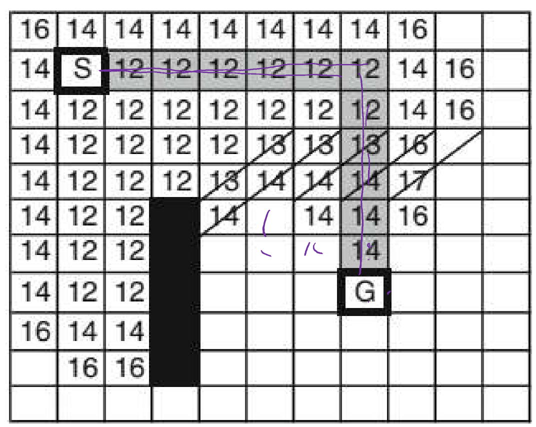


h(x,y)

=



(c)

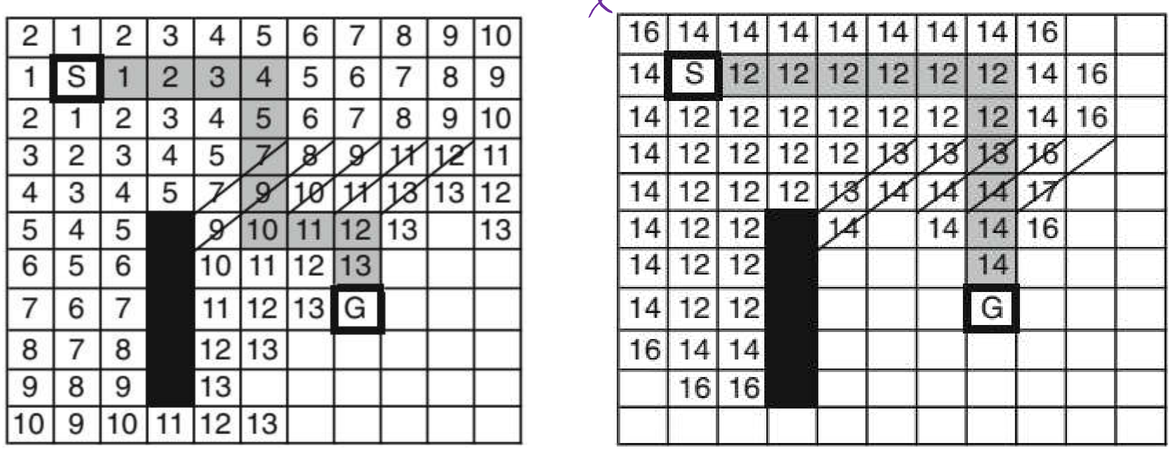


(d)

- **Figure (c)**
  - Search does not continue to lower left of map
  - Cost of 16 higher than cost of 14 once search leaves the sand
- **Figure (d)**
  - Goal cell **G** is now found very quickly
- **Similar to Dijkstra's**
  - Shortest path found by tracing back through cells with lower g values until start cell is reached.

More Complex Example of A*

- Comparing (a) Dijkstra's & (b) A*
  - *A* only had to visit 71% of the cells visited by Dijkstra's* g(x,y)
  - *Although A* performed additional work to compute heuristic function* g(x,y) h(x,y)
  - *Reduced number of cells visited made A* more efficient*
  - *Also, heuristic function depend only on area searched & not on obstacles*
  - *Even if obstacle set changed, no need to recompute heuristic function*



(a) Dijkstra's **(b) A***

Mapping-Based Navigation

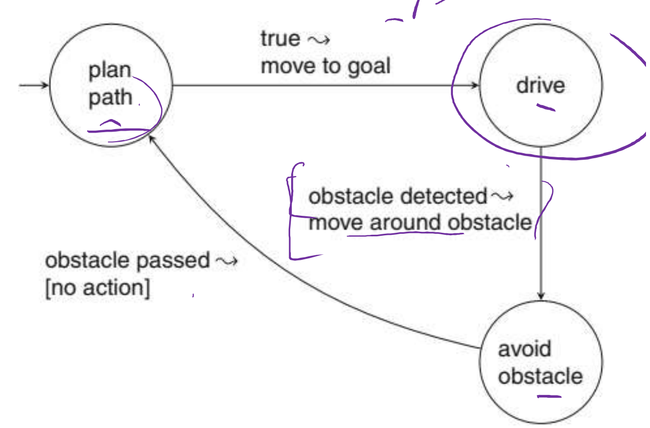
- Dijkstra's Algorithm for a Grid Map
- Dijkstra's Algorithm for a Continuous Map
- Path-planning with the A* Algorithm
- Path Following and Obstacle Avoidance

Path Following & Obstacle Avoidance

- High-level path planning & low-level obstacle avoidance
  - *How to integrate?*
- Simplest approach is prioritize low-level algorithm (a)
  - *Of course, more important avoid hitting people or drive around pothole*
  - *Rather than shortest route to airport*
- Robot normally in **drive** state
  - *Transition to* **avoid obstacle** *if obstacle detected*
  - *After obstacle has been passed → transition to* **plan path** *state to recompute path*

SLAM

OA



(a) Integrating path planning & obstacle avoidance

Path Following & Obstacle Avoidance

- Strategy for integrating the two algo depends on environment
- **Ex. : Road repair might take weeks**
  - *Makes sense to add this obstacle to map*
  - *Path planning algo will take this into account*
  - *Resulting path likely better than one changed at last minute by obstacle avoidance algo*
- **Ex. : On the other extreme**
  - *If lots of moving obstacles like pedestrians crossing a street*
  - *Obstacle avoidance algo can be simply to stop moving →*
    wait until obstacles move away →
    original path plan can be resumed without detours

Summary

- **Path planning**
  - *High-level behavior: finding shortest path from S → G in the environment*
  - *Based upon a map showing obstacles*
  - *Can be based on both grid & continuous map*
    - Creating graph of obstacles from the map
    - Account for constant or variable costs for each cell
  - *Dijsktra's algorithm* $g(x,y)$ = min steps S → cell .. obs(✓)
    - Expand shortest path to any cell encountered so far
  - *A* algorithm* $f(x) = g(x,y) + h(x,y)$
    - Reduce number of cells visited   min steps: G → cell obs(X)
    - Using heuristic function that indicates direction to the goal cell
- **Low-level avoidance must be integrated with high-level planning**

Thank you.