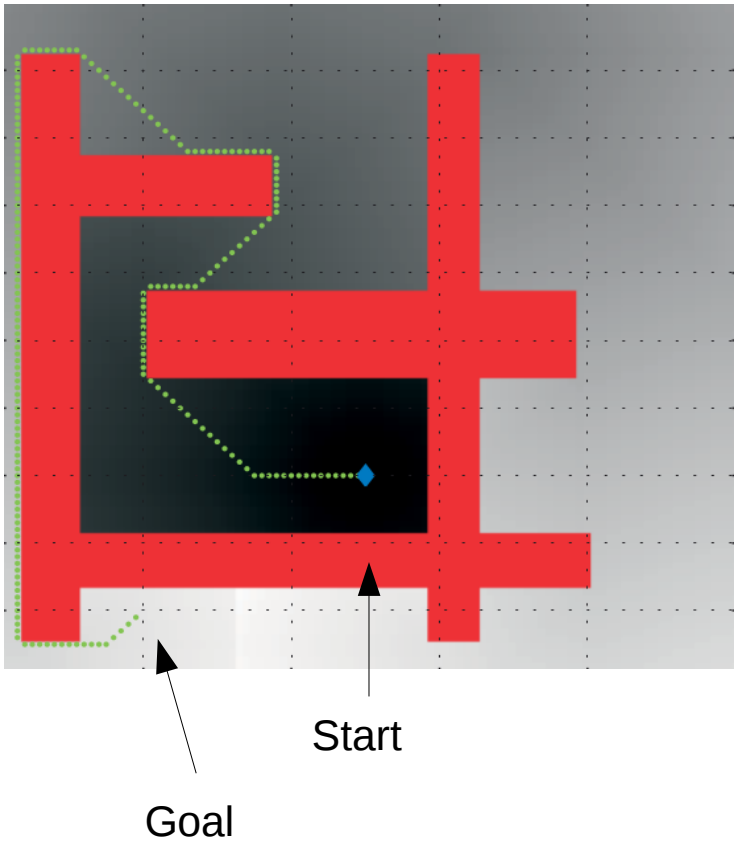
Path Planning: Bugs, Wavefront

Robert Platt
Northeastern University

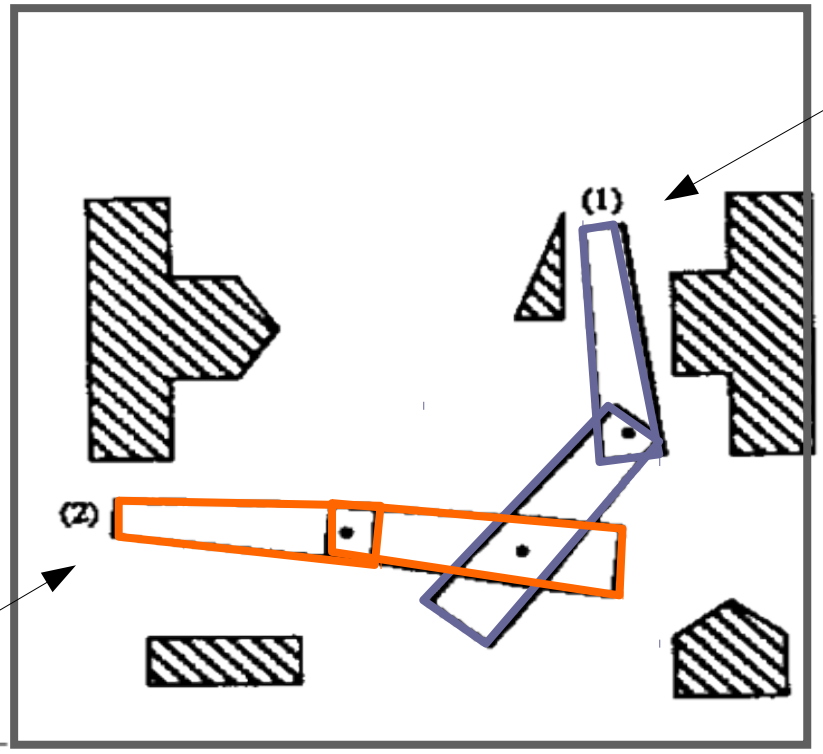


These notes contain materials from Peter Corke's book and from Howie Choset's lecture materials.

Problem we want to solve

Given:
– a point-robot (robot is a point in space)
– a start and goal configuration

Find:
– path from start to goal that does not result in a collision



Starting configuration

Goal configuration

Problem we want to solve

Given:
– a point-robot (robot is a point in space)
– a start and goal configuration

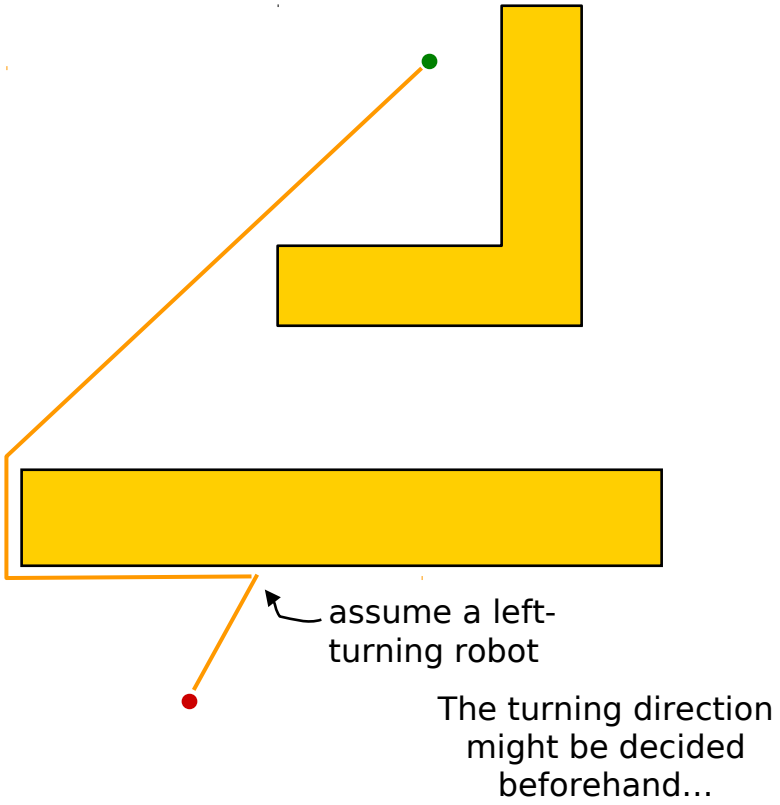
Find:
– path from start to goal that does not result in a collision

Assumptions:
– the position of the robot can always be measured perfectly
– the motion of the robot can always be controlled perfectly
– the robot can move in any directly instantaneously

First attempt: BUGs!

Bug algorithms:
– assume only local knowledge of the environment is available
– simple behaviors: follow a wall; follow straight line toward goal

First attempt: BUG 0

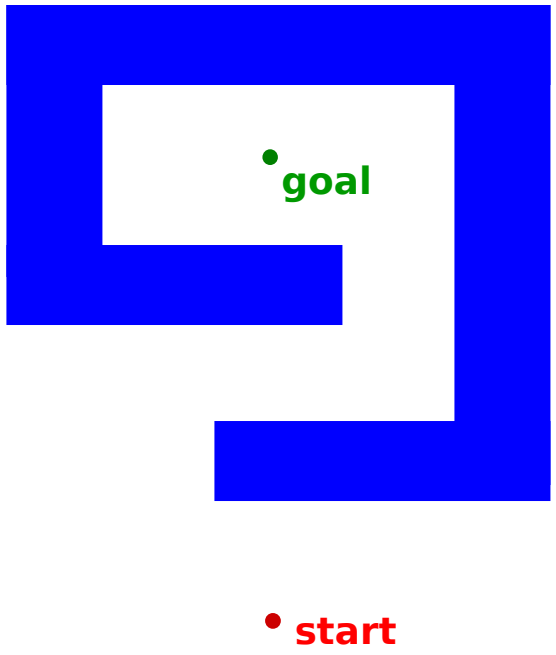


BUG 0:

1. head toward goal
2. if hit a wall, turn left
3. follow wall until a line toward goal will move you away from wall.

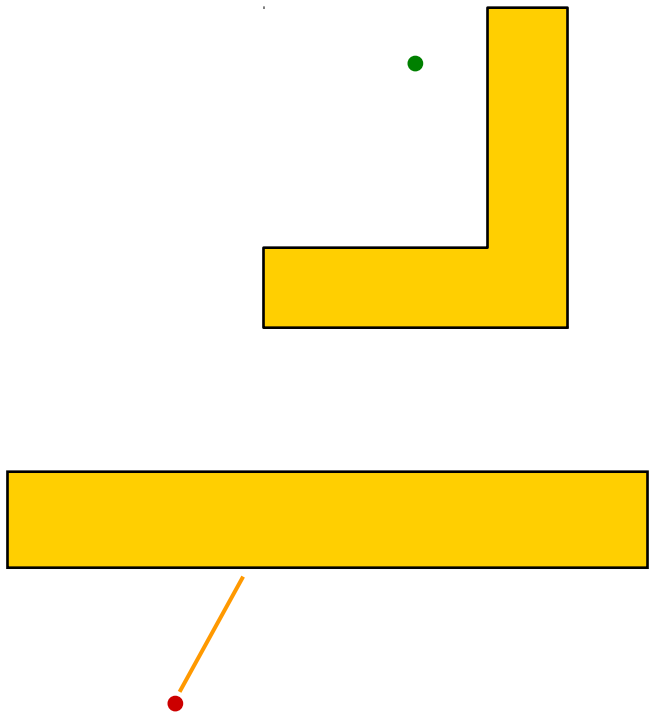
(assume we only have local sensing – we cannot sense position of walls we are not touching)

Question



What does BUG0 do here?

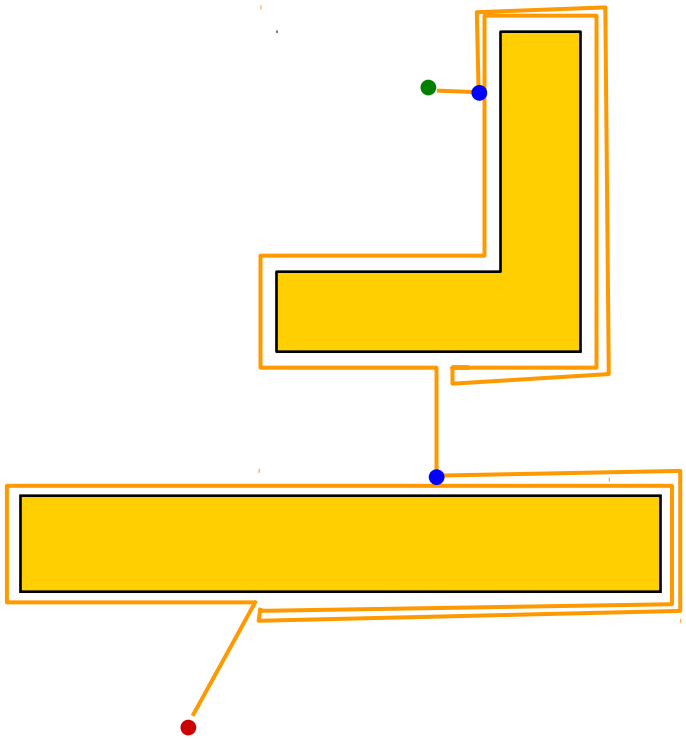
Second attempt: BUG 1



BUG 1:

1. move on straight line toward goal
2. if obstacle encountered, circumnavigate entire obstacle and remember how close bug got to goal
3. return to closest point and continue on a straight line toward goal

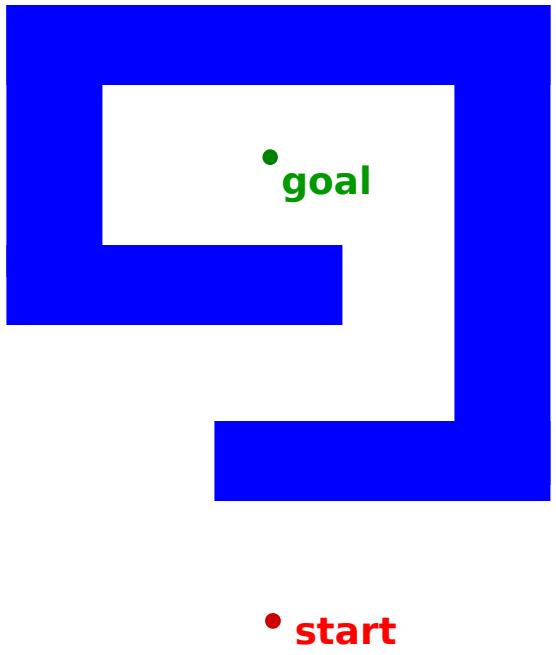
Second attempt: BUG 1



BUG 1:

1. move on straight line toward goal
2. if obstacle encountered, circumnavigate entire obstacle and remember how close bug got to goal
3. return to closest point and continue on a straight line toward goal

Question



What does BUG1 do here?

BUG 1 Performance Analysis

How far does BUG 1 travel before reaching goal?

Best case scenario (lower bound): D

Worst case scenario (upper bound): $D + 1.5 \sum_i P_i$

Where

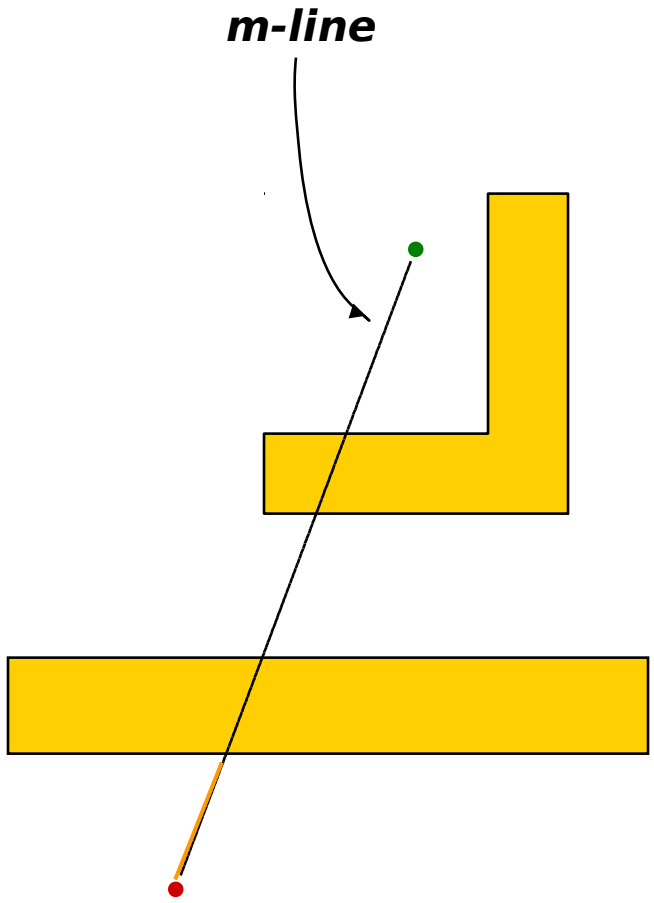
- D denotes distance from start to goal and
- P_i denotes perimeter of ith obstacle

Questions

Is BUG 1 *complete?*

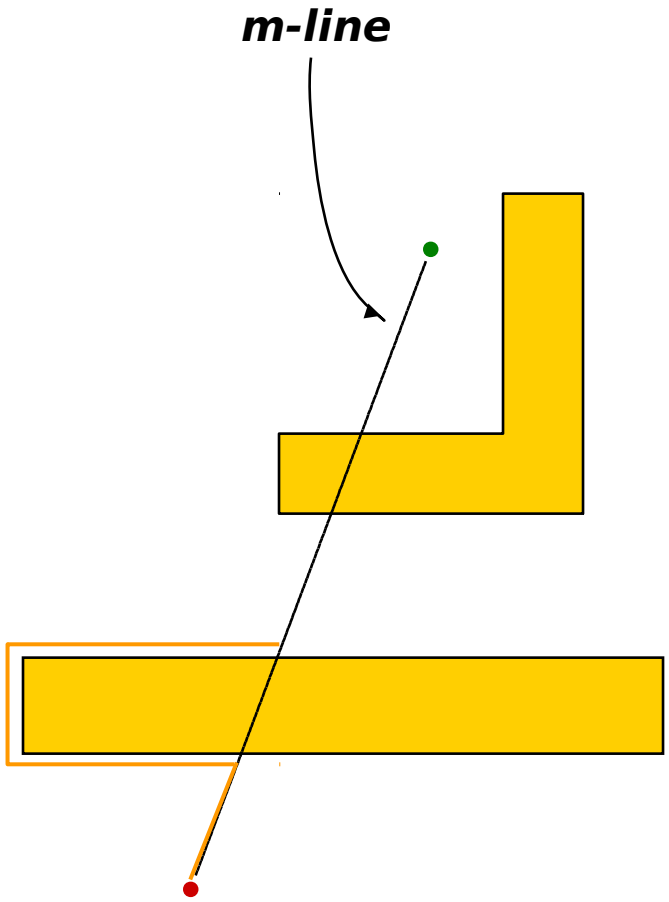
Prove completeness or incompleteness.

Another bug: BUG 2



1. head toward goal on m-line

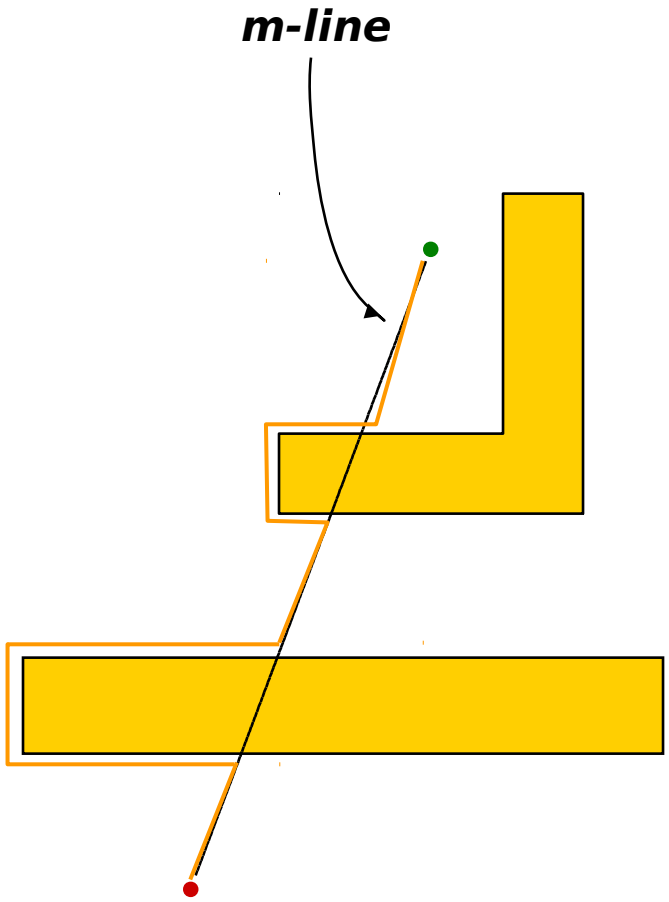
Another bug: BUG 2



1. head toward goal on m-line

2. if you encounter obstacle, follow it until you encounter m-line again at a point closer to goal

Another bug: BUG 2

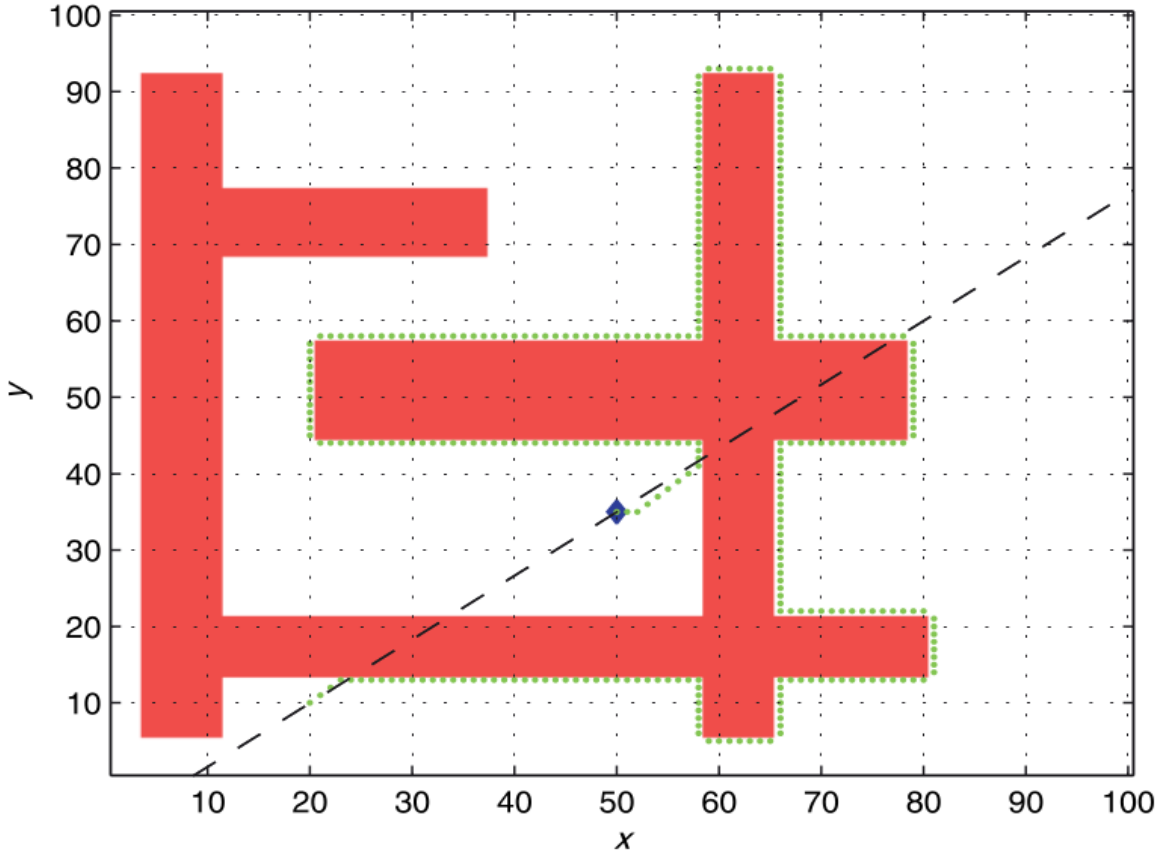


1. head toward goal on m-line

2. if you encounter obstacle, follow it until you encounter m-line again at a point closer to goal

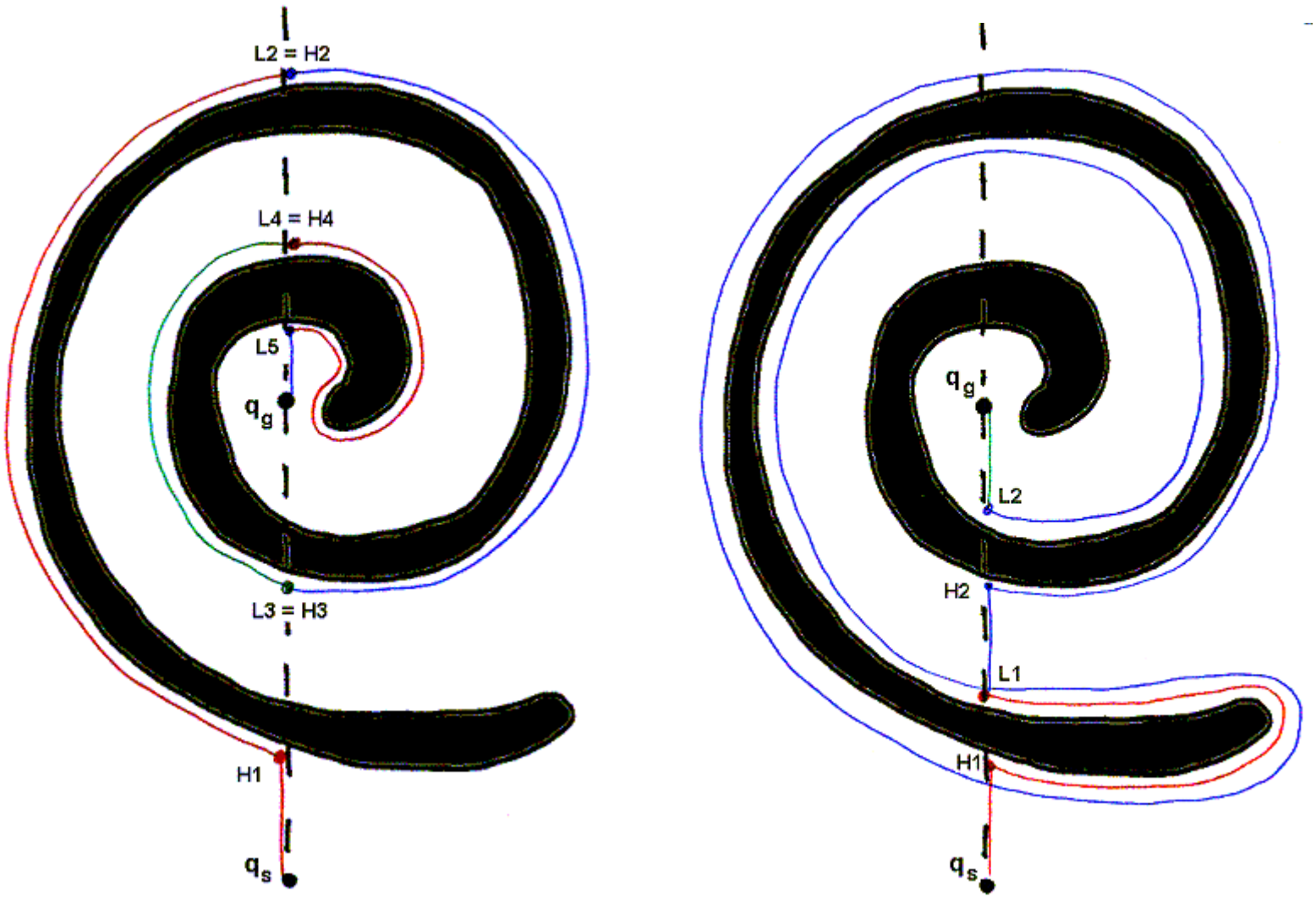
3. leave line and head toward goal again

Another bug: BUG 2

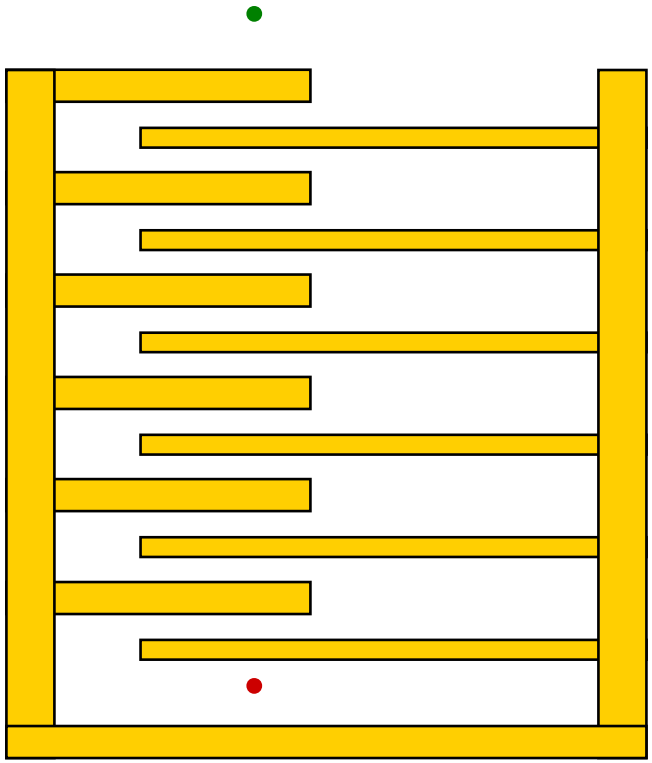


Is BUG 2 complete?
– Why? Why not?

Another bug: BUG 2



Another bug: BUG 2



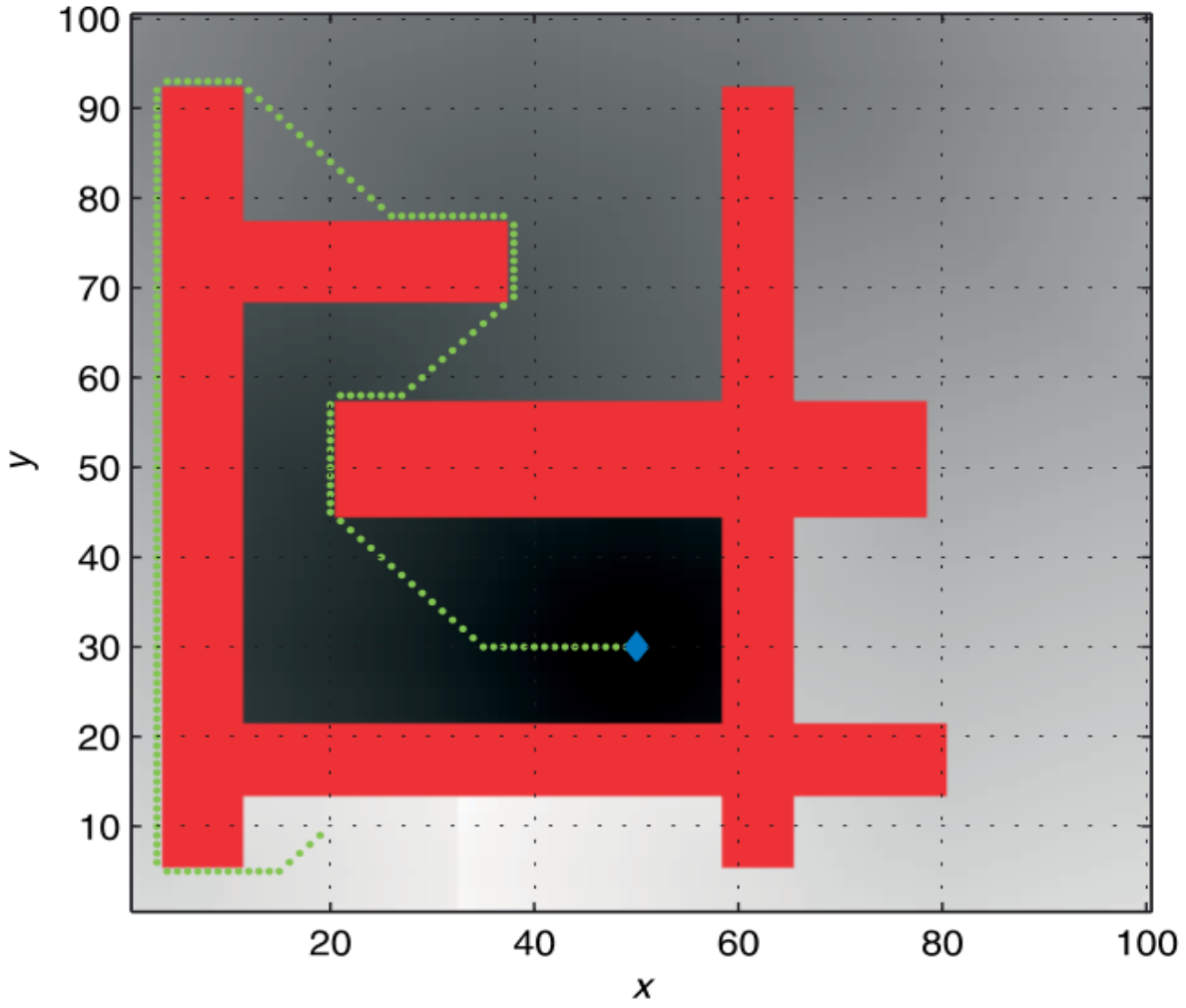
How bad can it get?

Lower bound: D

Upper bound: $D + \sum_i \frac{n_i}{2} P_i$

where n_i is the number of s-line intersections
In the ith obstacle.

Wavefront planner (distance transform)



– intensity of a point denotes its (obstacle-respecting) distance from the goal

Wavefront planner (distance transform)

Algorithm:

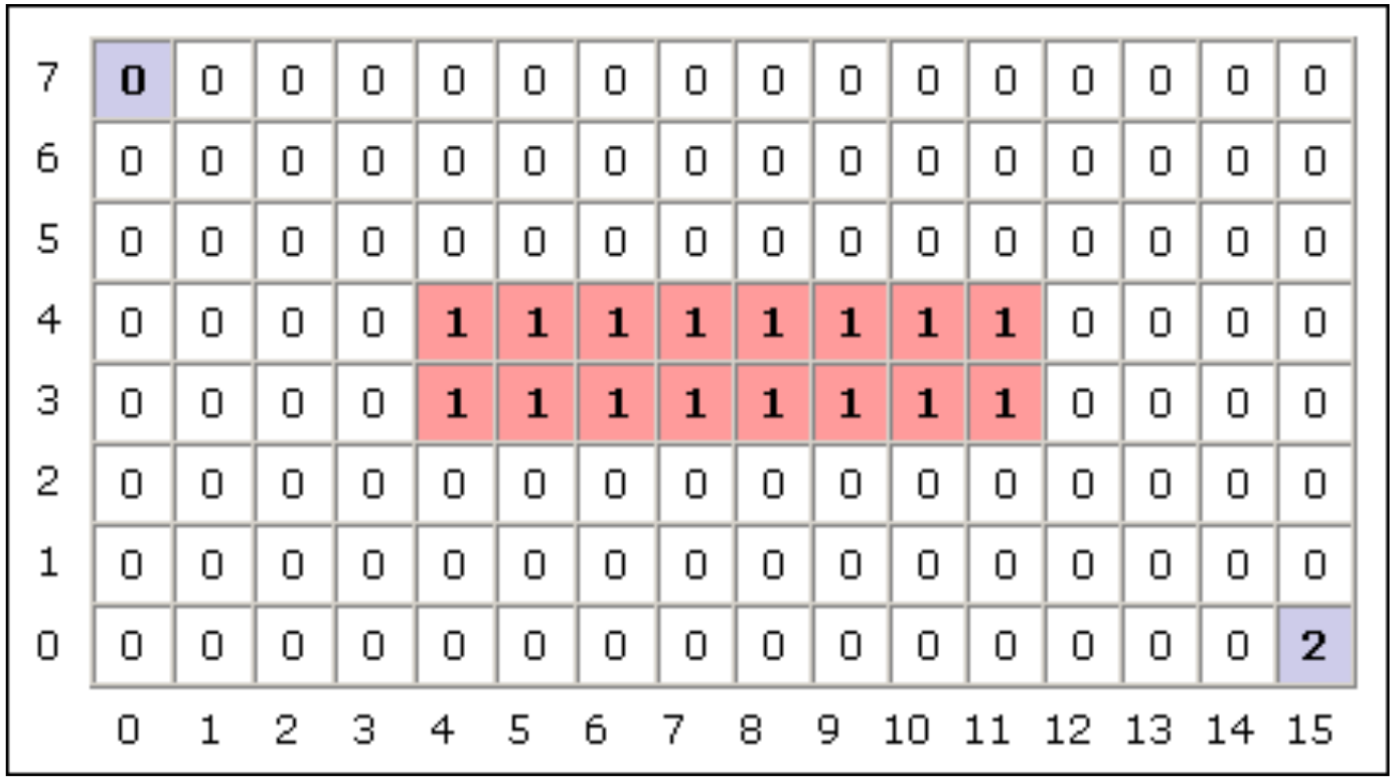
1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L

L: list of nodes in wave front; initially just the goal state
d: distance function over nodes; initially zero everywhere except goal state

Wavefront planner (distance transform)

Algorithm:

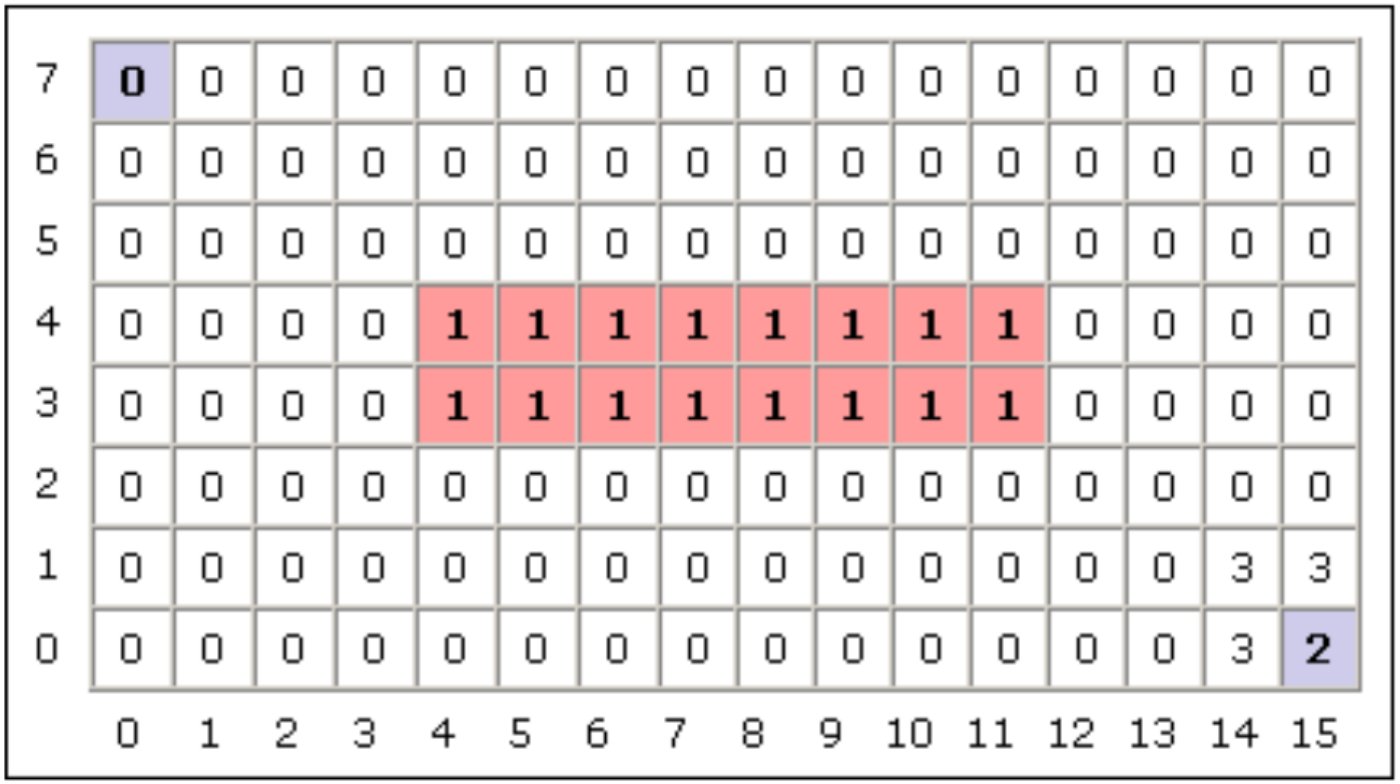
1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L



Wavefront planner (distance transform)

Algorithm:

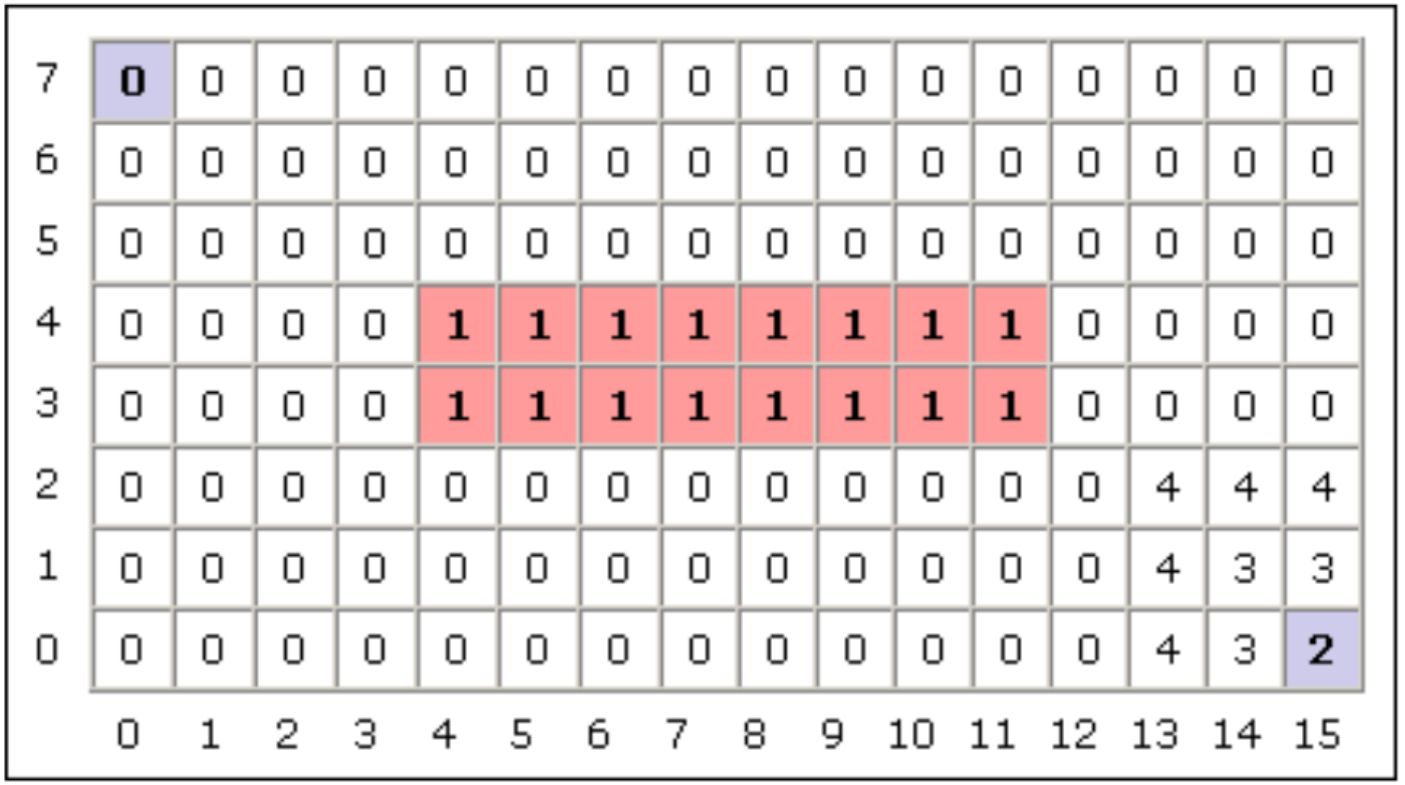
1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L



Wavefront planner (distance transform)

Algorithm:

1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L



Wavefront planner (distance transform)

Algorithm:

1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	**2**

Wavefront planner (distance transform)

Algorithm:

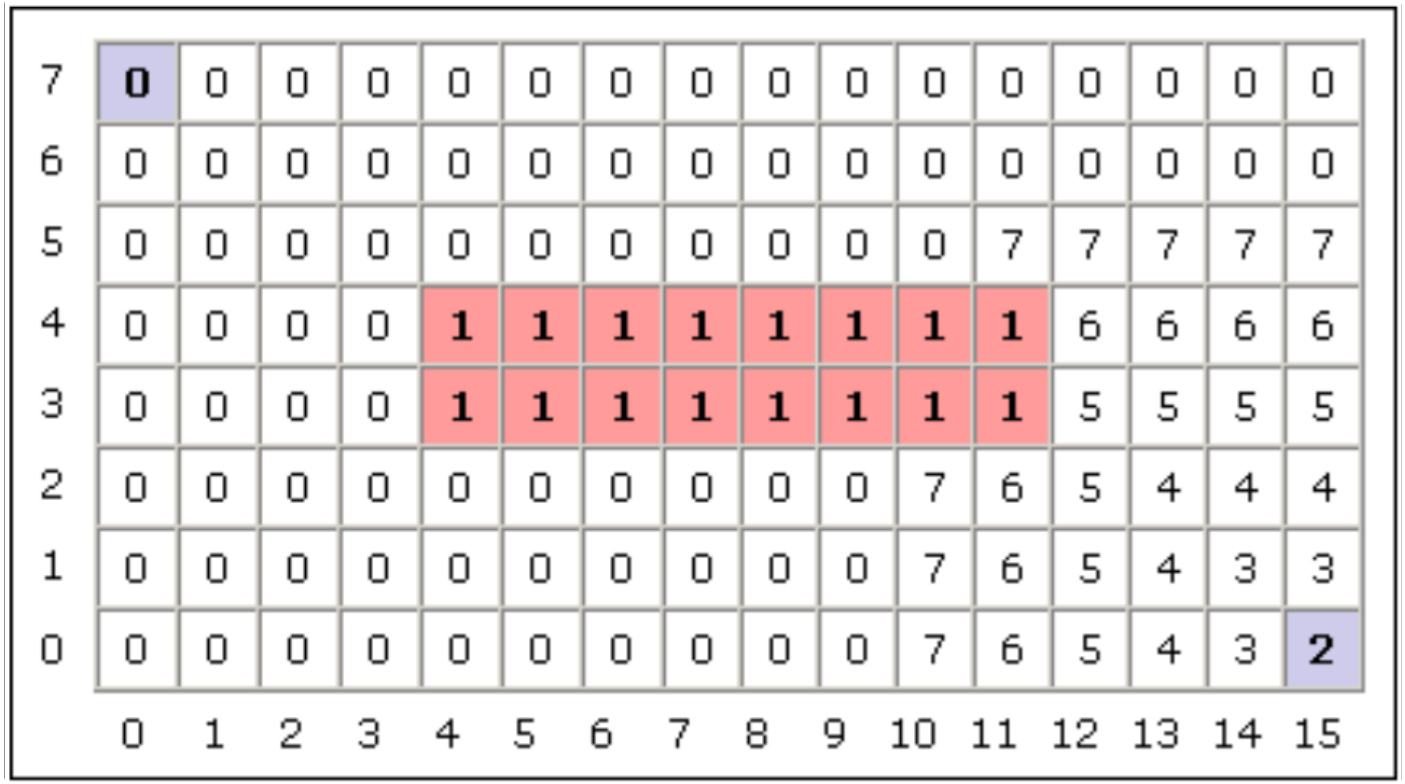
1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	6	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	**2**

Wavefront planner (distance transform)

Algorithm:

1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L



Wavefront planner (distance transform)

Algorithm:

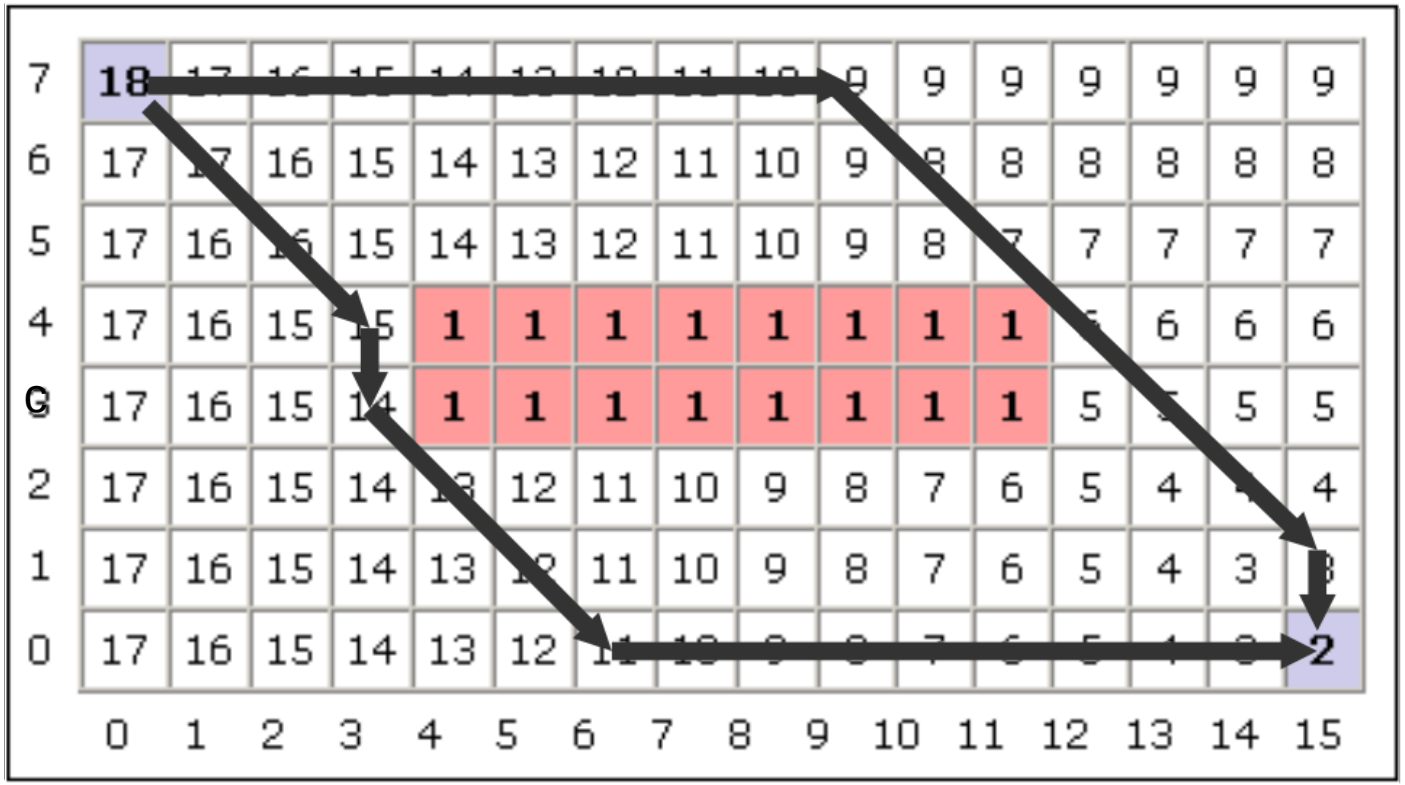
1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	**2**

Wavefront planner (distance transform)

Algorithm:

1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L

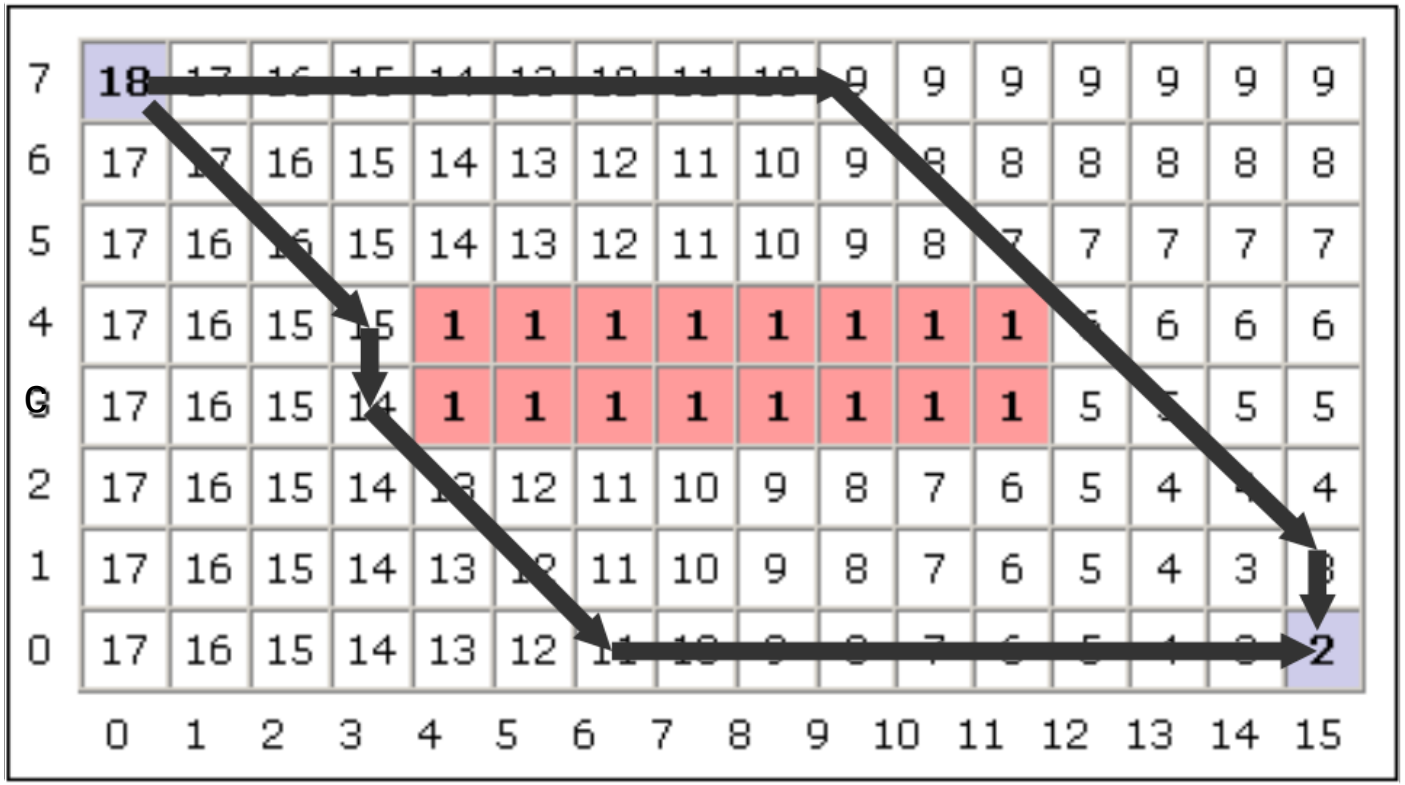


Wavefront planner (distance transform)

Algorithm:

1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L

For node j, how many steps to goal state in terms of d(j)?

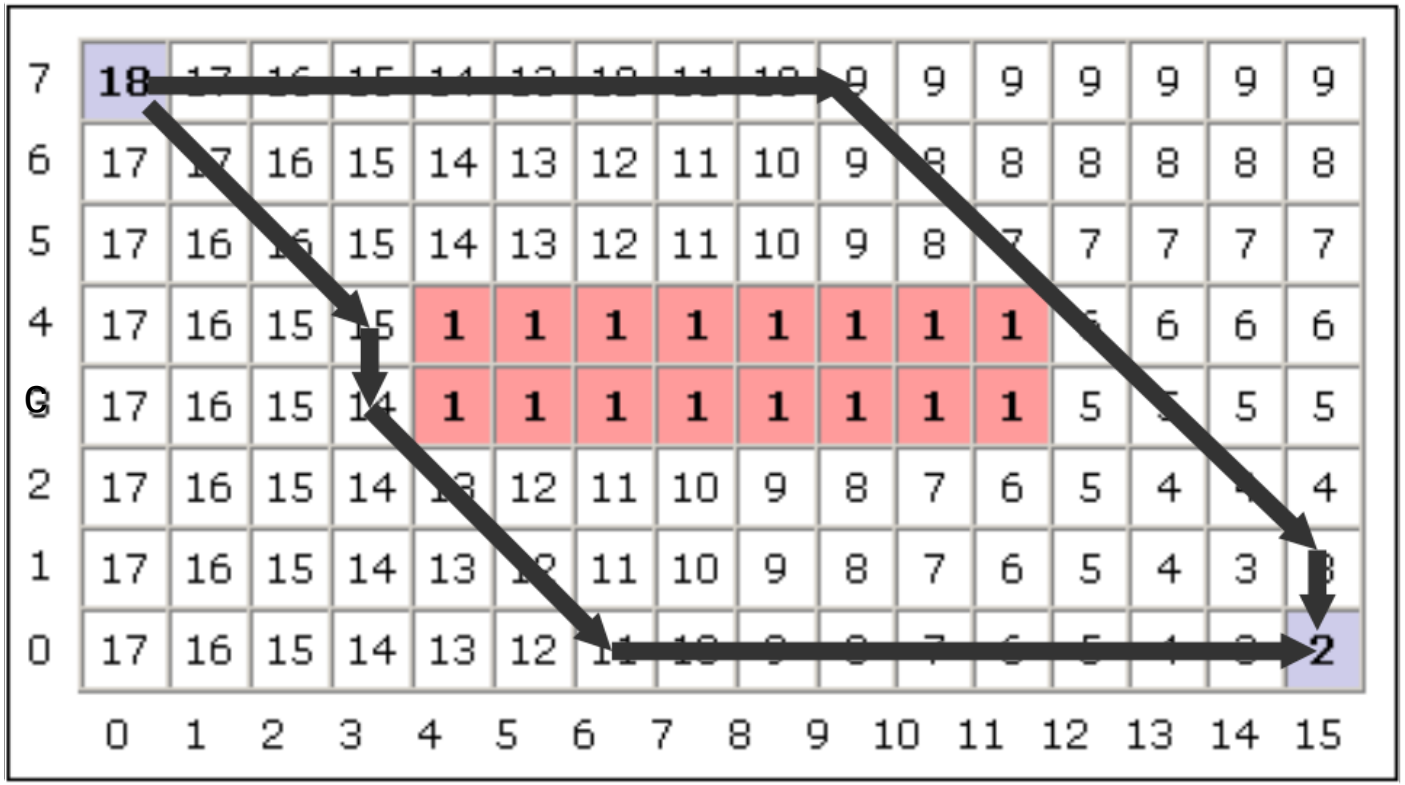


Questions

Complete?

Optimal?

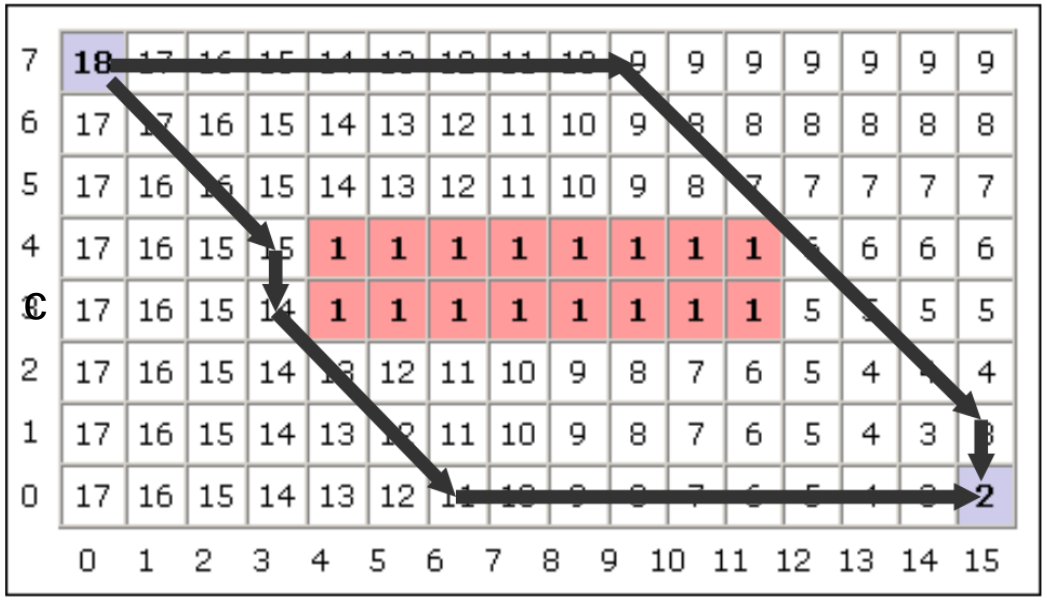
Prove completeness/optimality or incompletness/non-optimality



Wavefront planner (distance transform)

Algorithm:

1. L={goal state}, d(goal state) = 2, d(obstacle states) = 1, d(rest of states) = 0
2. while L != null
3. pop item i from L
4. for all neighbors j of i such that d(j)==0
5. d(j) = d(i)+1
6. push j onto L



L: list of nodes in wave front; initially just the goal state
d: distance function over nodes; initially zero everywhere except goal state