Optimal and learning control for robotics

Exercise series1

Wenli Shen NET ID: ws1577

● Exercise 1

a) $X_n = [-2,-1,0,1,2]$ $U_n = [-1,0,1]$

$$x_{n+1} = \begin{cases} -x_n + 1 + u_n & \text{if } -2 \le -x_n + 1 + u_n \le 2 \\ 2 & \text{if } -x_n + 1 + u_n > 2 \\ -2 & \text{else} \end{cases}$$

$g_i(X_i, U_i) = 2|X_i| + |U_i|$ (cost calculation function in every stage)

$g_N(X_N) = X_N^2$ （terminal cost）

$$J = \left(\sum_{k=0}^{2} 2|x_k| + |u_k|\right) + x_3^2$$

J_k=$g_k(X_k, U_k)$+ $g_{k+1}(X_{k+1})$

According above formula, J in the table are not corresponding cost in every stage, they are cost to go which means summary of cost from the terminal to present stage.

Results in every stage:

N=3 stage3:

X(value)	J(cost)	U(control)
$X_3 = -2$	$J_3 = 4$	none
$X_3 = -1$	$J_3 = 1$	none
$X_3 = 0$	$J_3 = 0$	none
$X_3 = 1$	$J_3 = 1$	none
$X_3 = 2$	$J_3 = 4$	none

N=2 stage2:

X(value)	J(cost)	U(control)
$X_2 = -2$	$J_2 = 8$	0,1
$X_2 = -1$	$J_2 = 4$	0
$X_2 = 0$	$J_2 = 1$	-1,0
$X_2 = 1$	$J_2 = 2$	-1
$X_2 = 2$	$J_2 = 5$	0

N=1 stage1:

X(value)	J(cost)	U(control)
$X_1 = -2$	$J_1 = 9$	1
$X_1 = -1$	$J_1 = 5$	0
$X_1 = 0$	$J_1 = 2$	-1,0
$X_1 = 1$	$J_1 = 3$	-1
$X_1 = 2$	$J_1 = 6$	0

N=0 stage0:

X(value)	J(cost)	U(control)
$X_0 = -2$	$J_0 = 10$	0
$X_0 = -1$	$J_0 = 6$	-1
$X_0 = 0$	$J_0 = 3$	-1,0
$X_0 = 1$	$J_0 = 4$	0
$X_0 = 2$	$J_0 = 7$	1

b) When $X_0 = 0$,-2,2 Control:
If we know X_0, we can ensure optimal control and corresponding cost from the table in (a) at present stage. And then from $X_{n+1} = -X_n + 1 + U_n$, we can find optimal control and corresponding cost in same way at next stage

(The cost in the table are corresponding cost in every stage with optimal control)

	Stage 0		X1	Stage 1		X2	Stage2		X3	Stage3
Value	Control	Cost		Control	Cost		Control	Cost		Cost
$X_0 = 0$	-1	3	0	-1	2	0	-1	1	0	0
$X_0 = -2$	0	10	2	1	6	0	-1	1	0	0
$X_0 = 2$	1	7	0	-1	2	0	-1	1	0	0

c) $X_n = [-2, -1, 0, 1, 2]$ $\qquad U_n = [-1, 0, 1]$

$$x_{n+1} = \begin{cases} -x_n + \omega_n + u_n & \text{if } -2 \le -x_n + \omega_n + u_n \le 2 \\ 2 & \text{if } -x_n + \omega_n + u_n > 2 \\ -2 & \text{else} \end{cases}$$

$p(w_n = 0) = 0.3$ $\qquad p(w_n = 1) = 0.7$

$(w_n = 1)$: $X_{i1+1} = X_{i1} + U_{i1} + 1$ $\qquad (w_n = 0)$: $X_{i2+1} = X_{i2} + U_{i2} + 0$

$g_i(X_i, U_i) = (2|X_{i1}| + |U_{i1}|) * 0.7 + (2|X_{i2}| + |U_{i2}|) * 0.3$

$g(X_n) = X_n^2$

$$J = \mathbb{E}\left(\left(\sum_{k=0}^{2} 2|x_k| + |u_k|\right) + x_3^2\right)$$

$J_k = g_k(X_k, U_k) + g_{k+1}(X_{k+1})$

According above formula, J in the table are not corresponding cost in every stage, they are cost to go which means summary of cost from the terminal to present stage.

Results in every stage:

N=3 stage3:

X(value)	J(cost)	U(control)
$X_3 = -2$	$J_3 = 4$	none
$X_3 = -1$	$J_3 = 1$	none
$X_3 = 0$	$J_3 = 0$	none
$X_3 = 1$	$J_3 = 1$	none
$X_3 = 2$	$J_3 = 4$	none

N=2 stage2:

X(value)	J(cost)	U(control)
$X_2 = -2$	$J_2 = 8.0$	0
$X_2 = -1$	$J_2 = 3.7$	-1
$X_2 = 0$	$J_2 = 0.7$	0
$X_2 = 1$	$J_2 = 2.3$	0
$X_2 = 2$	$J_2 = 5.3$	1

N=1 stage1:

X(value)	J(cost)	U(control)
$X_1 = -2$	$J_1 = 9.3$	0
$X_1 = -1$	$J_1 = 4.8$	-1
$X_1 = 0$	$J_1 = 1.8$	0
$X_1 = 1$	$J_1 = 3.6$	0
$X_1 = 2$	$J_1 = 6.6$	1

N=0 stage0:

X(value)	J(cost)	U(control)
$X_0 = -2$	$J_0 = 10.6$	0
$X_0 = -1$	$J_0 = 6.1$	-1
$X_0 = 0$	$J_0 = 3.1$	0
$X_0 = 1$	$J_0 = 4.7$	0
$X_0 = 2$	$J_0 = 7.7$	1

d) Compared to these two models. Because uncertainty (w_n) comes out in second situation, we cannot directly ensure the exact value of the next X (X_{n+1}). The value X is floating with w_n, the control would change in a way. We use $J(x)$ = min [($g(x,u)$ + $Jf(x,u,\omega$ = 1) $*$ 0.7 + ($g(x,u)$ + $J(f(x,u,\omega$ = 0)) $*$ 0.3] to find minimum cost J in this stage, and then we can find corresponding control and then ensure the next X(X_{n+1}) . Do this way circularly, the probability model show out. From the date has been calculated, the probability model has bigger range of cost and stable control.

● Exercise 2

a) $X_{n+1} = X_n + U_n$

$X_n = [-4,-3,-2,-1,0,1,2,3,4]$ $\qquad U_n = [-2,-1,0,1,2]$

$$J = \sum_{k=0}^{N-1}((x_k+4)^2 + u_k^2) + g_N(x_N)$$

$$g_N(x_N) = \begin{cases} 0 \text{ if } x_N = 0 \\ \infty \text{ else} \end{cases}$$

$g_i(X_i, U_i) = (X_i + 4)^2 + U_i^2$

Because $-4 < X_{n+1} < 4$, we need to pay attention to the range of control U when programming.

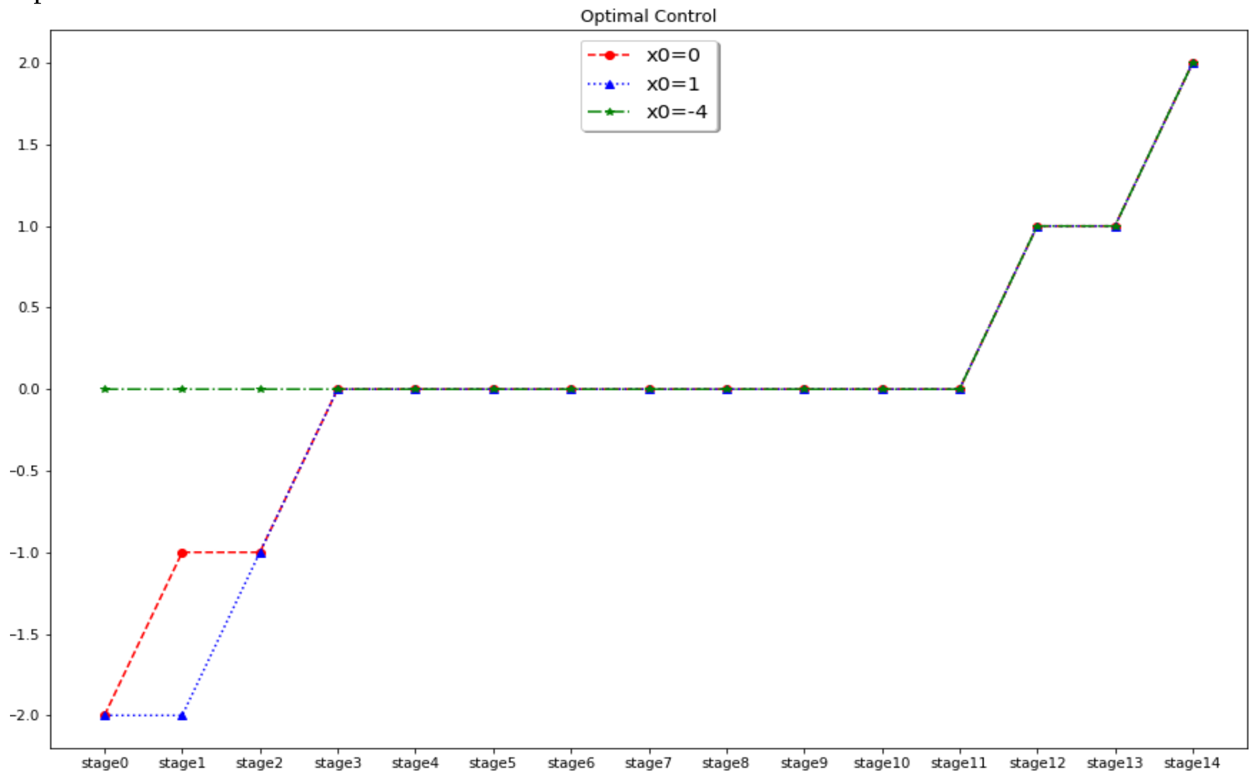
b) According to the programming, when $X_4 = 4$:

In stage 4: (N=4)

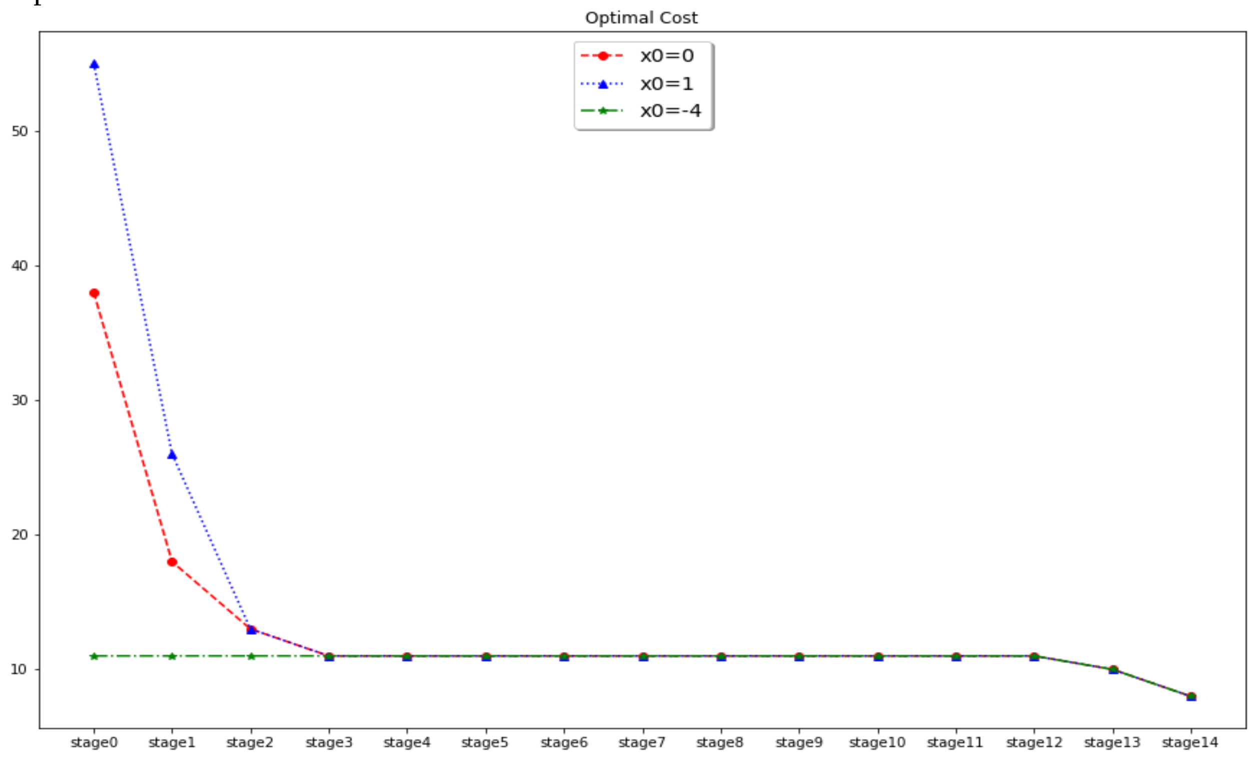
X_4	-4	-3	-2	-1	0	1	2	3	4
Cost to go(J)	11	13	18	26	38	55	78	108	146
Control(U)	0	-1	-1	-2	2	2	2	2	-2

From the table: $J_4 = 146$

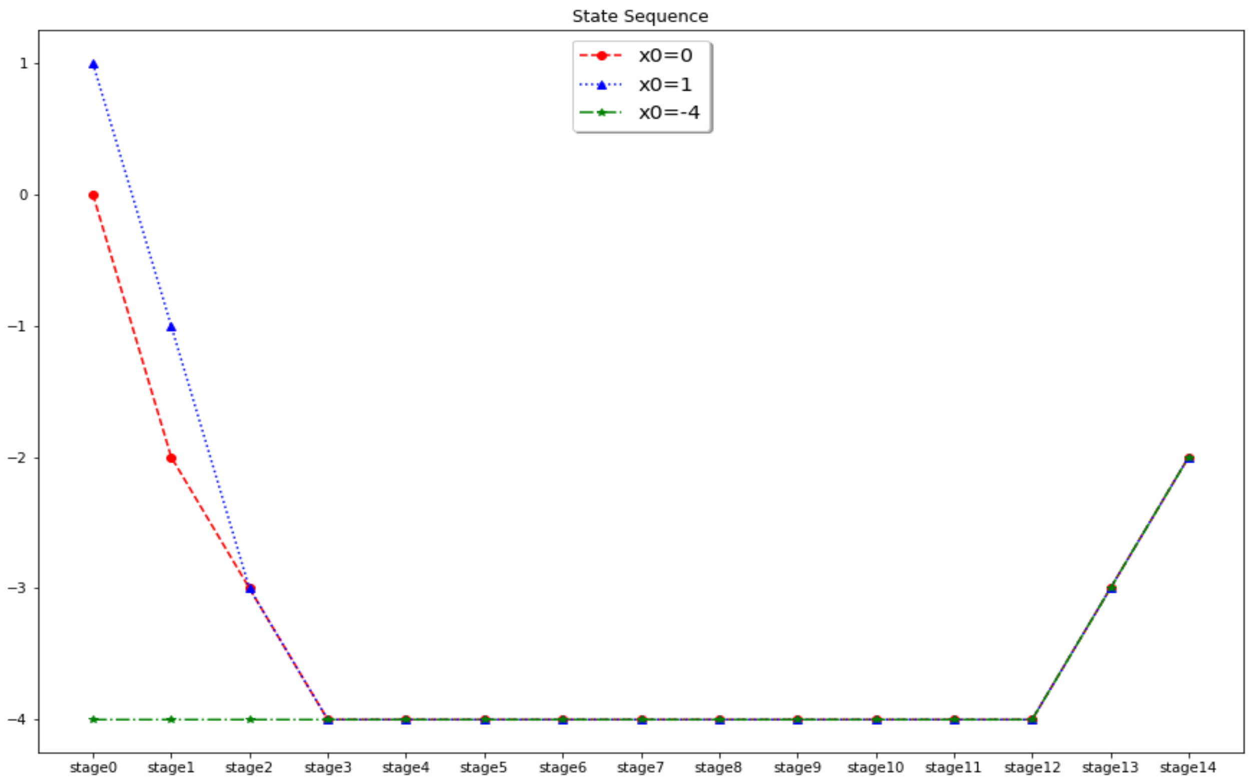
c) Optimal Control:



Optimal Cost:

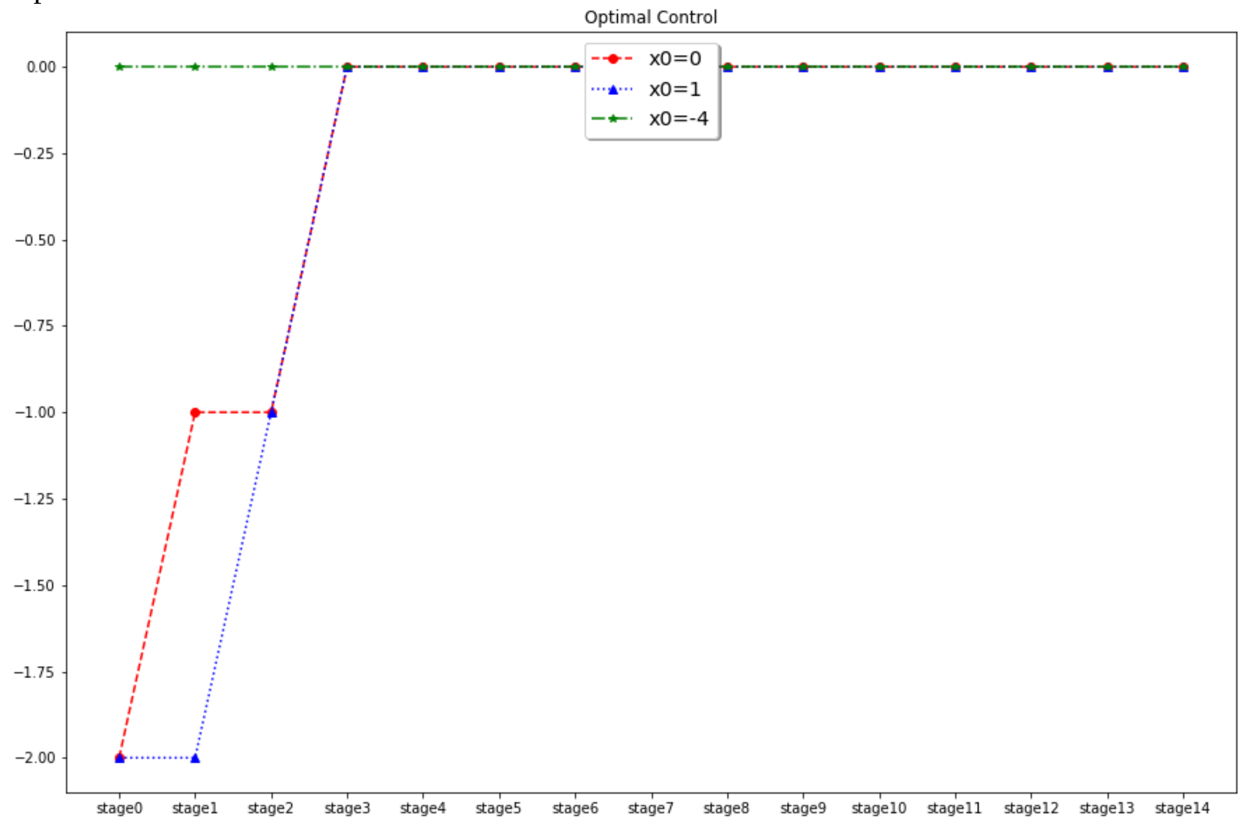


State Sequence:

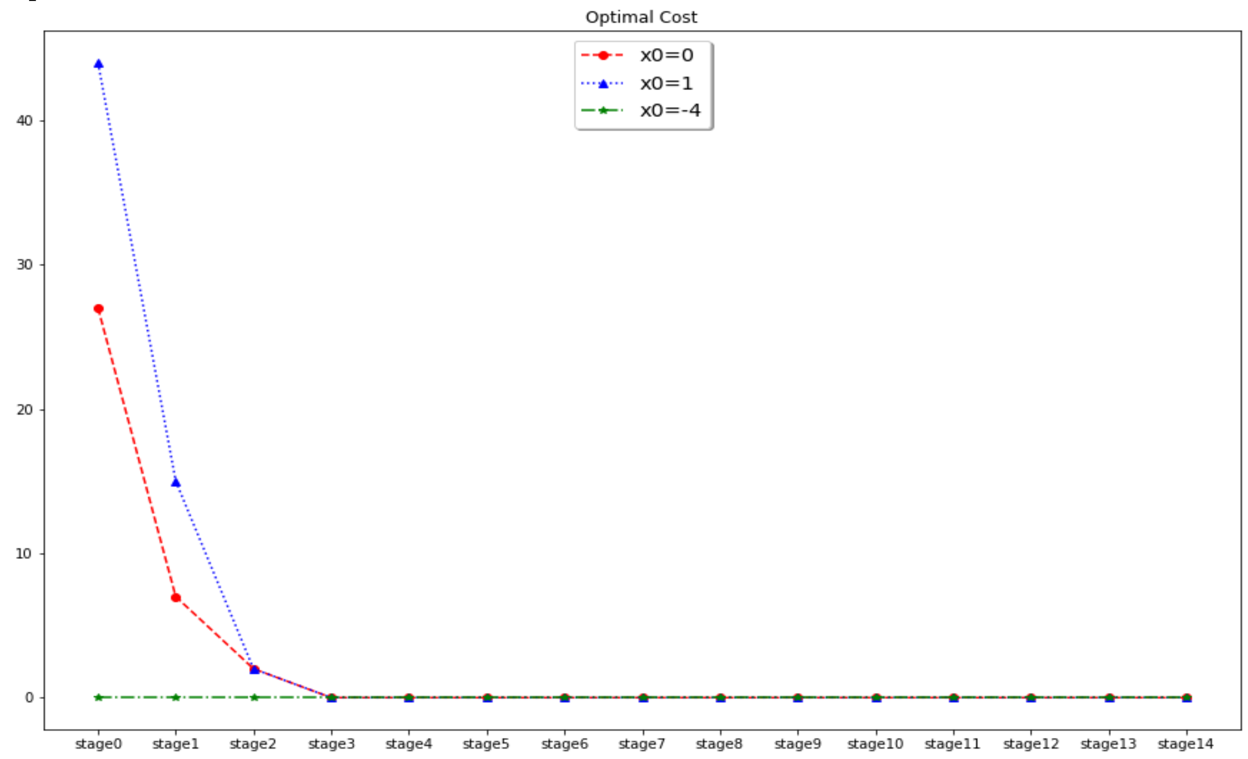


d) Let the terminal cost g_N is 0 for all states. $J_{15} = [0]$ for all X
Compared to the previous function of optimal cost, this situation has less cost than before. When N is getting bigger until final stage, this situation is more stable. And previous situation would keep steady before stage 12, but it would decline again after stage 12.
Compared to the previous function of state sequence, this situation is also steadier. Because this function would not change after stage 12, and previous would rise.
Compared to the previous function of optimal control, it's also steadier and getting to 0 faster.
The reason of second is steadier and with less fluctuation is that it doesn't need to consider final stage must be $J_{15} = 0$.Because every X in stage15 is 0. We don't need much more controls and states to make it approach $J_{15} = 0$.

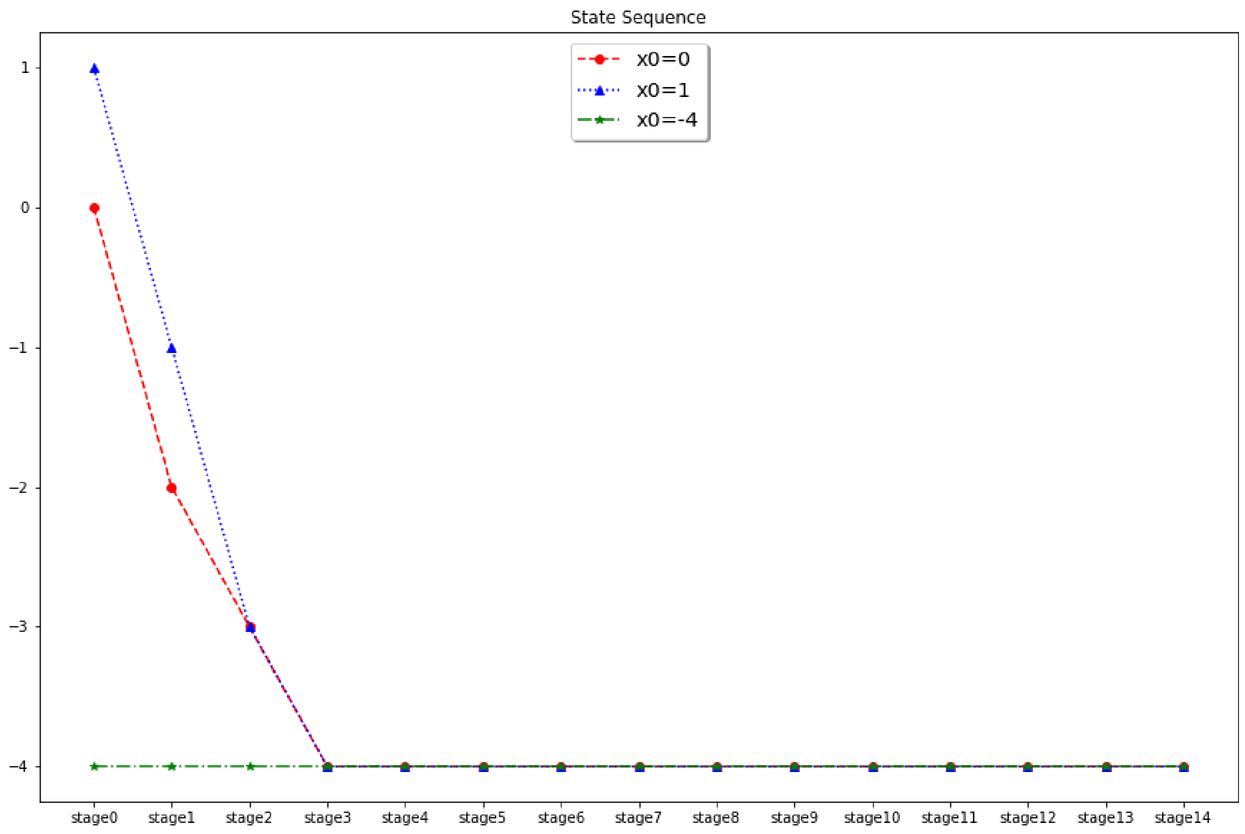
Optimal Control:



Optimal Cost:



State Sequence:



e) $X_{n+1} = X_n + U_n$

$X_n = [-4,-3,-2,-1,0,1,2,3,4]$ $U_n = [-2,-1,0,1,2]$

$$J = \sum_{k=0}^{N-1} (x_k + 4)^2 + g_N(x_N)$$

$$g_N(x_N) = \begin{cases} 0 \text{ if } x_N = 0 \\ \infty \text{ else} \end{cases}$$

$g_i(X_i, U_i) = (X_i + 4)^2$

Because $-4 < X_{n+1} < 4$, we need to pay attention to the range of control U when programming. Let the terminal cost g_N is 0 for all states. $J_{15} = [0]$ for all X.

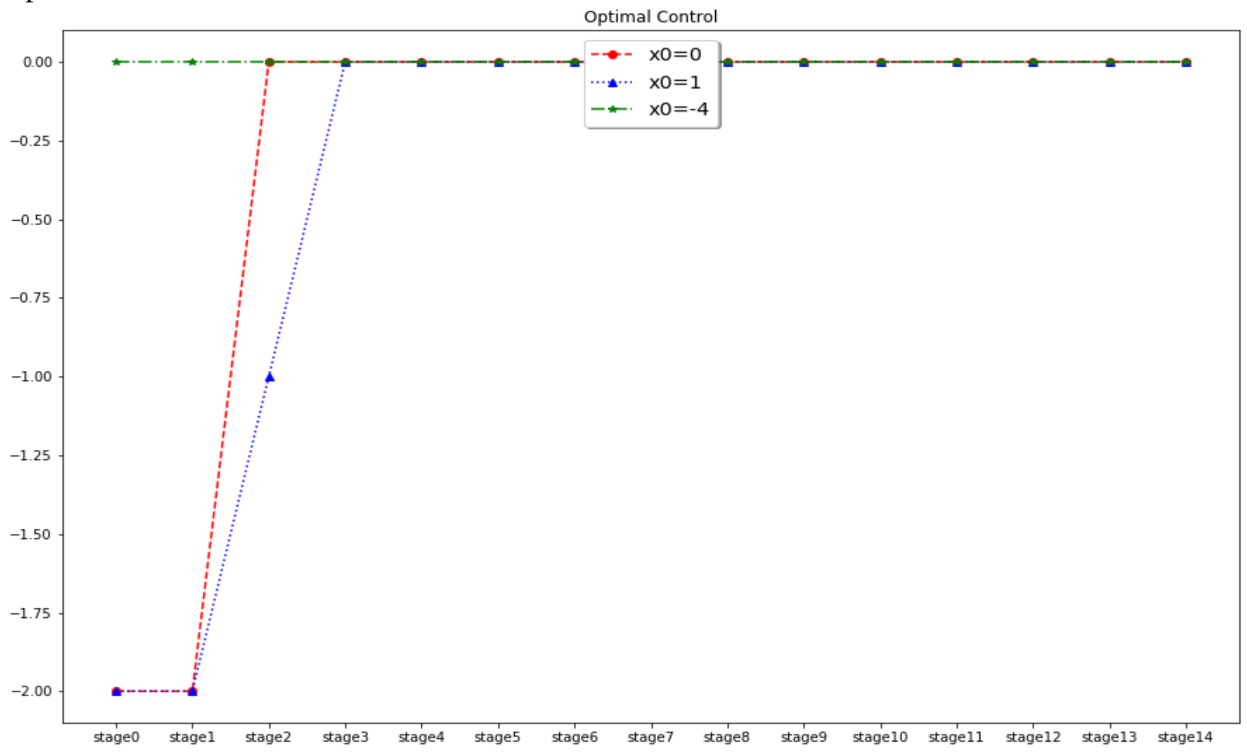
Compared to (c) function of optimal cost, this situation has less cost than before. When N is getting bigger until final stage, this situation is more stable. And previous situation would keep steady before stage 12, but it would decline again after stage 12.

Compared to (c) function of state sequence, this situation is also steadier. Because this function would not change after stage 12, and previous would rise.

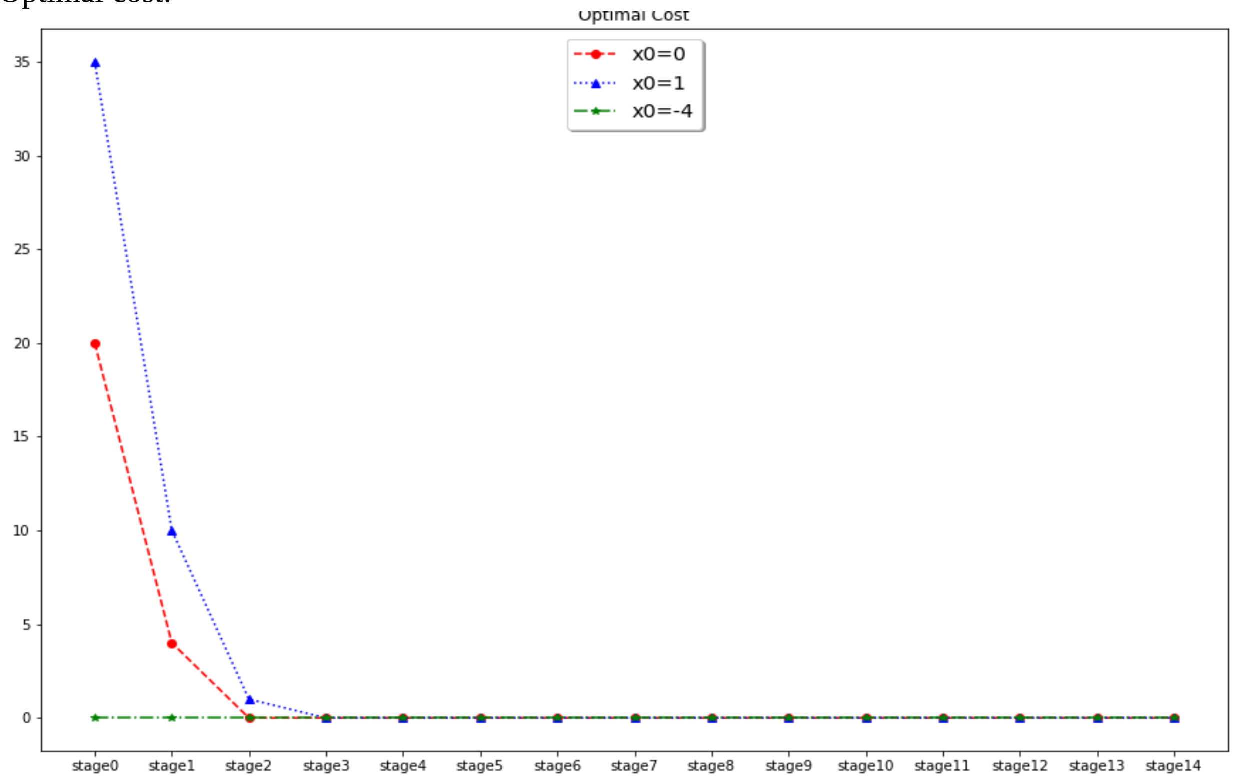
Compared to (c) function of optimal control, it's also steadier and getting to 0 faster.

The reason of second is steadier and with less fluctuation is that it doesn't need to consider final stage must be $J_{15} = 0$.Because every X in stage15 is 0. Apart from this, ${U_i}^2$ is always positive except 0, it would make the cost much larger in every stage. So we don't need much more controls and states to make every stage's cost as small as possible.

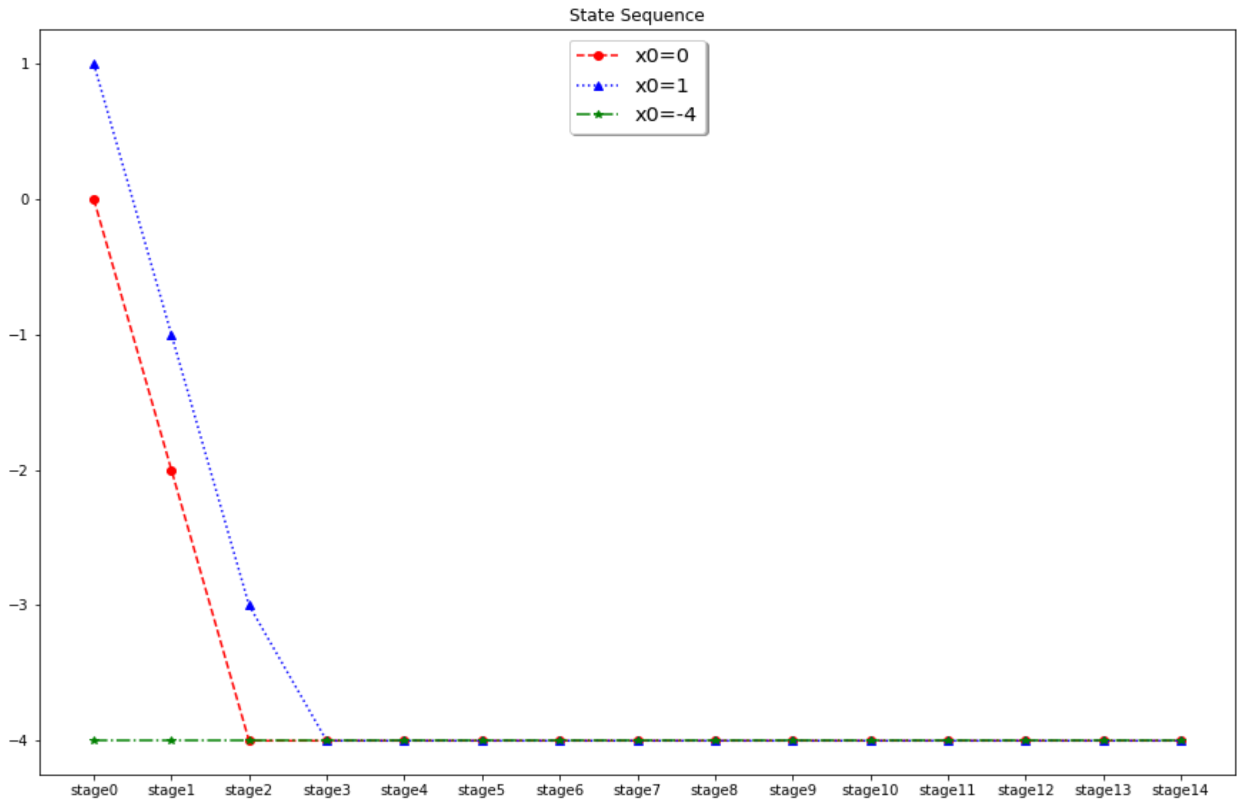
Optimal control:



Optimal cost:



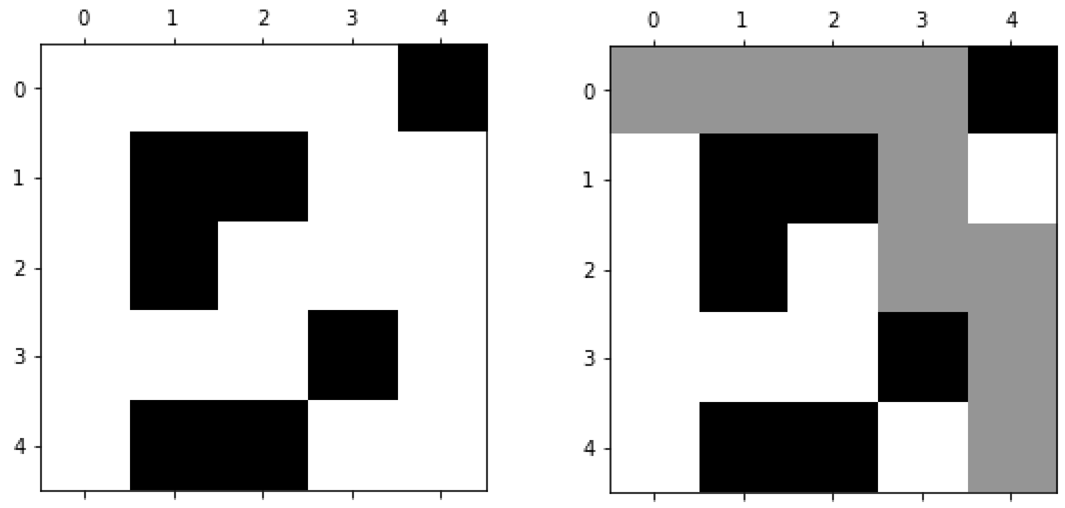
State sequence:



● Exercise 3

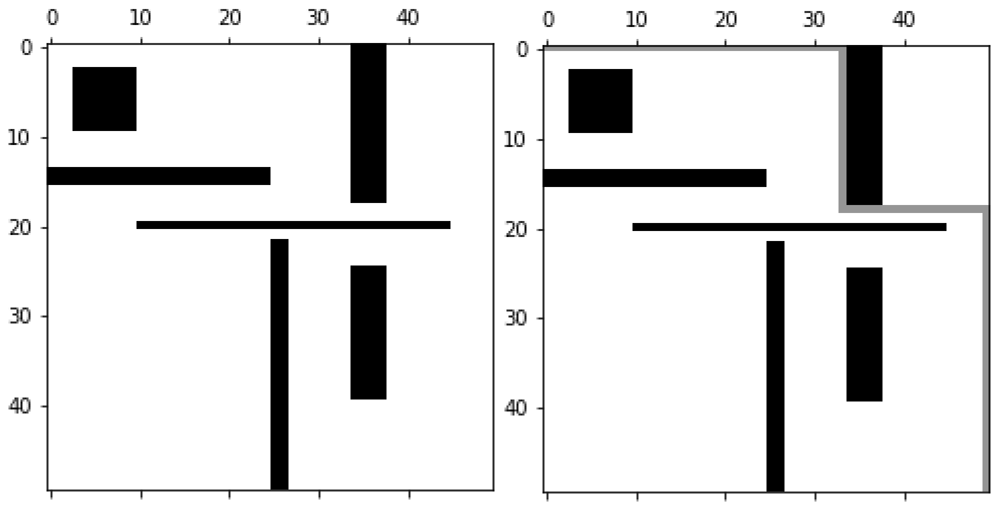
a) For a depth-first algorithm, we put new nodes at the beginning of the list and take the first element in the last as CURRENT.

i. Maze 0:



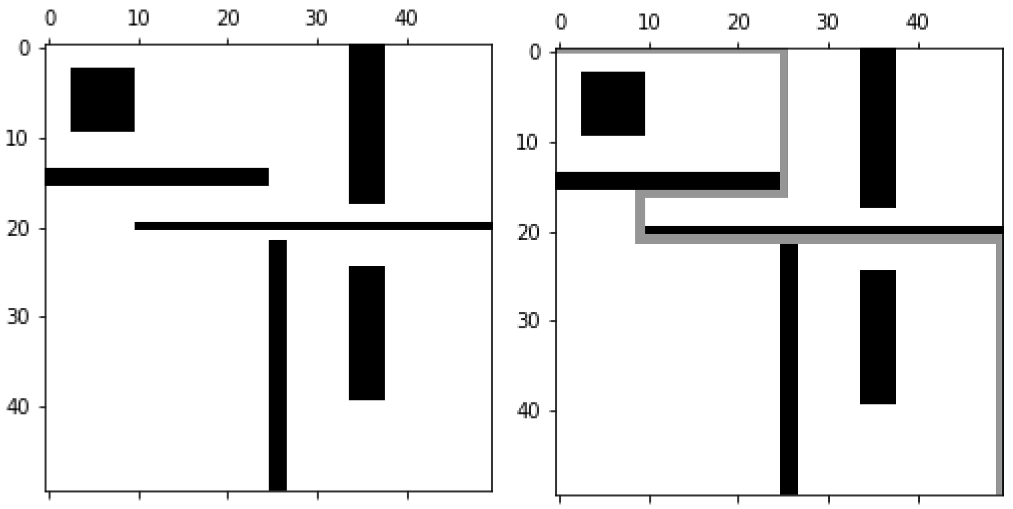
There are 17 nodes tested in CURRENT.
The shortest path is 9.

ii. Maze 1:



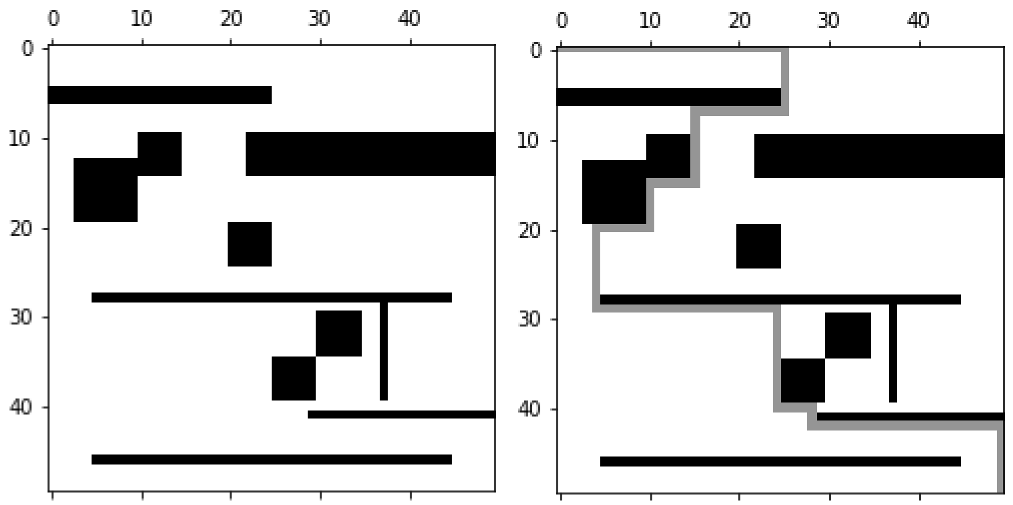
There are 223241 nodes tested in CURRENT.
The shortest path is 99.

iii. Maze 2:



There are 176721 nodes tested in CURRENT.
The shortest path is 131.

iv. Maze 3

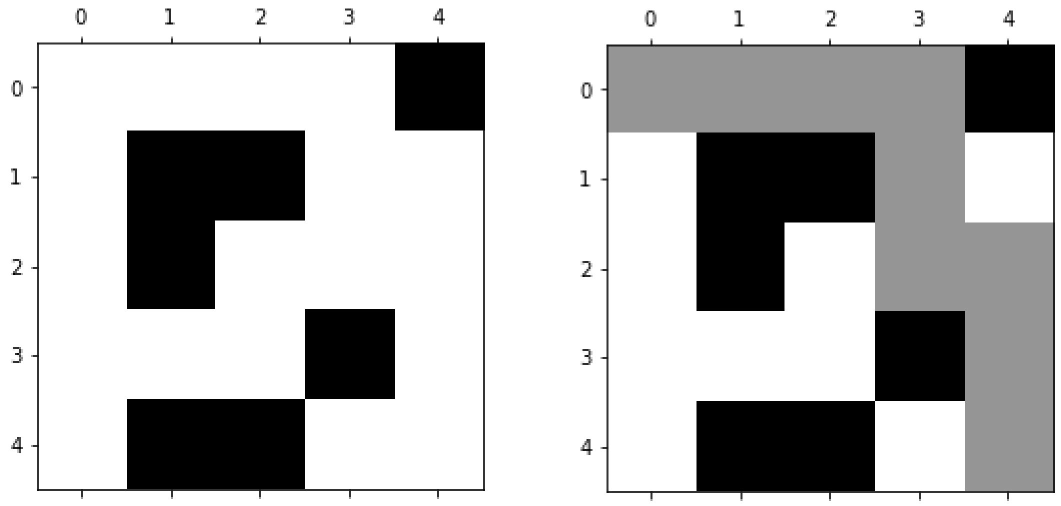


There are 332734 nodes tested in CURRENT.
The shortest path is 141.
To sum up:

Maze	Depth-first	
	Nodes	Path
0	17	9
1	223241	99
2	176721	131
3	332734	141

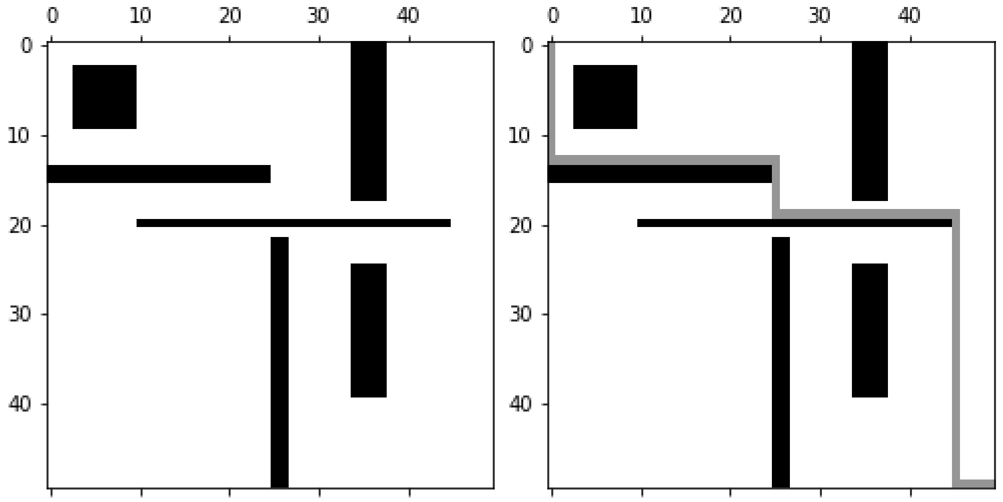
b) For a Bread-first algorithm, we put new nodes at the end of the list and take the first element in the last as CURRENT.

i. Maze 0:



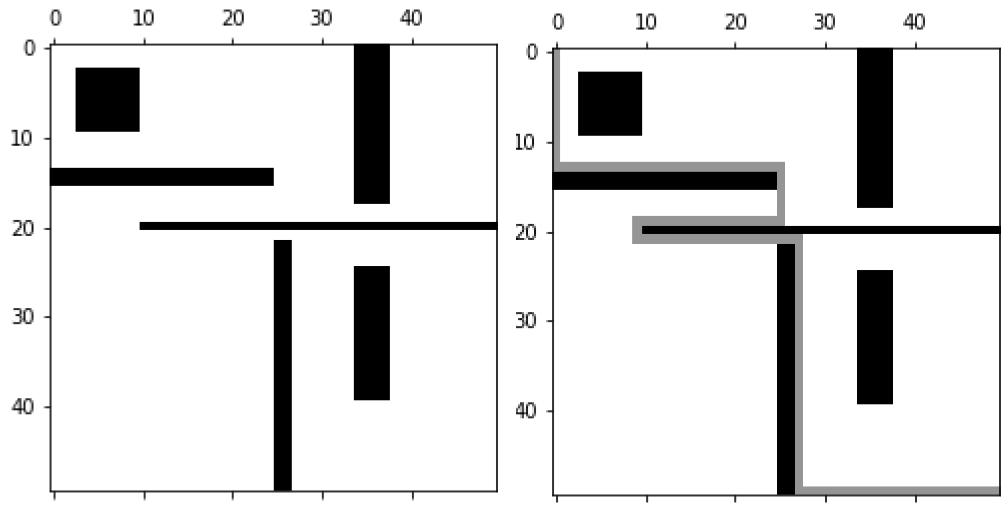
There are 16 nodes tested in CURRENT.
The shortest path is 9.

ii. Maze 1:



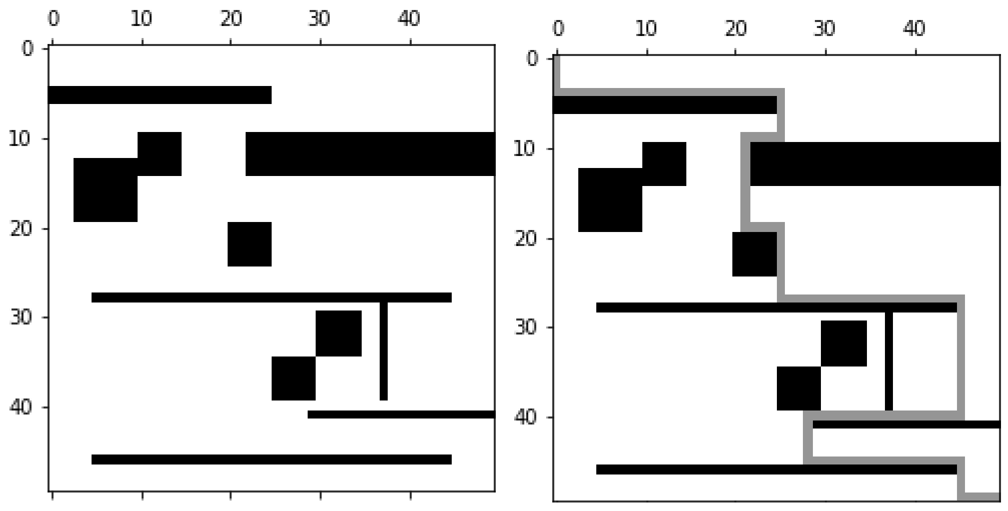
There are 2024 nodes tested in CURRENT.
The shortest path is 99.

iii. Maze 2:



There are 2172 nodes tested in CURRENT.
The shortest path is 131.

iv. Maze 3

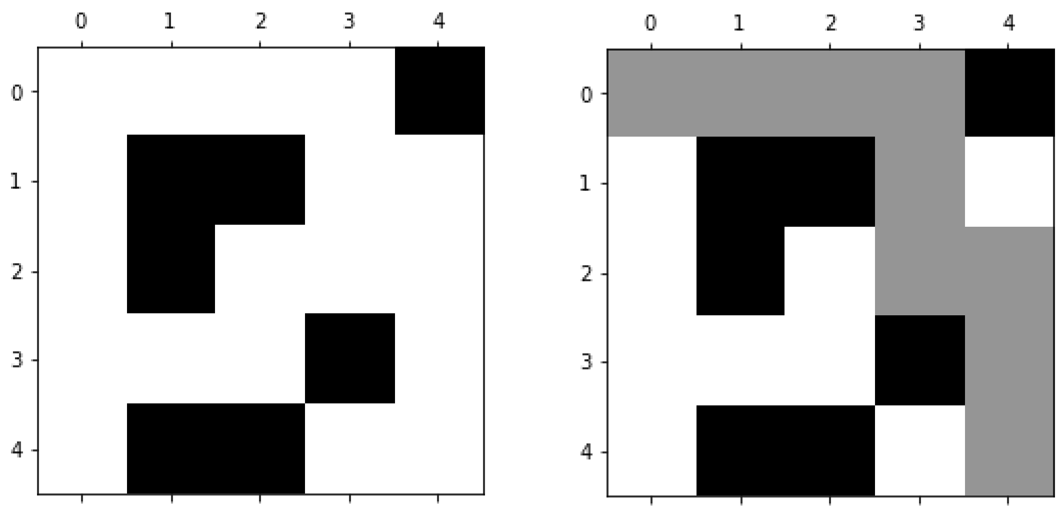


There are 2048 nodes tested in CURRENT.
The shortest path is 141.
To sum up:

Maze	Breadth-first	
	Nodes	Path
0	16	9
1	2024	99
2	2172	131
3	2048	141

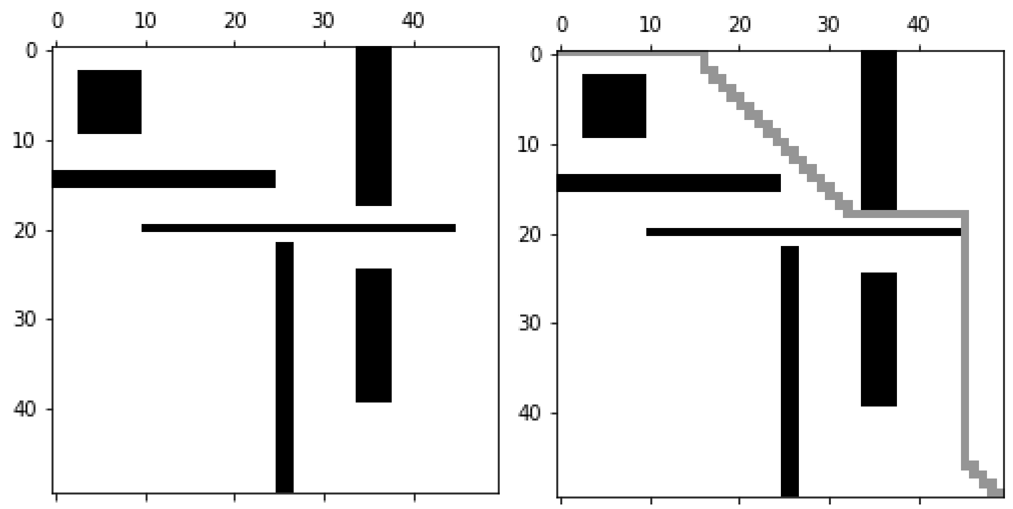
c) For A* algorithm, let heuristic h_{ij}= 0 (where i and j are the i row and j column entry)

i. Maze 0:



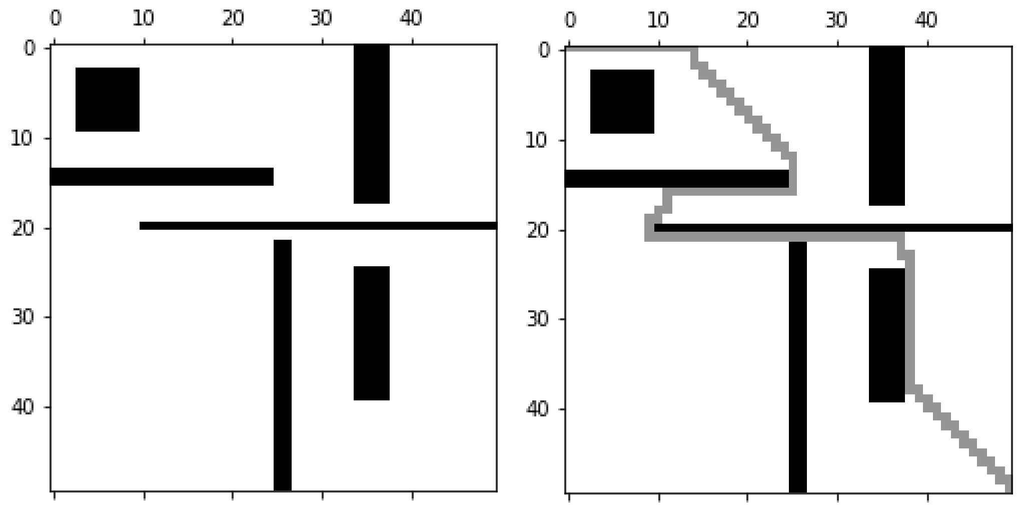
There are 17 nodes tested in CURRENT.
The shortest path is 9.

ii. Maze 1:



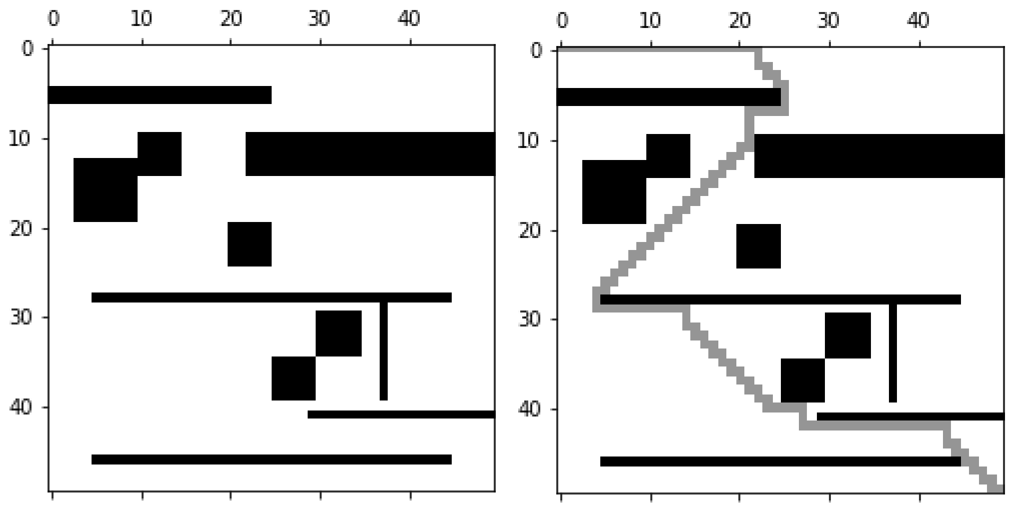
There are 2037 nodes tested in CURRENT.
The shortest path is 99.

iii. Maze 2:



There are 2137 nodes tested in CURRENT.
The shortest path is 131.

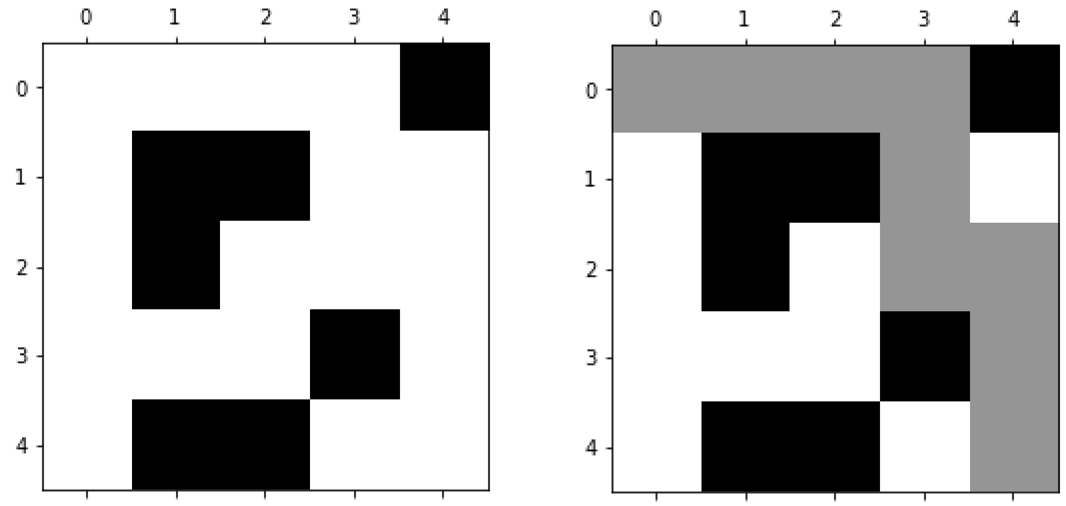
iv. Maze 3



There are 2049 nodes tested in CURRENT.
The shortest path is 141.

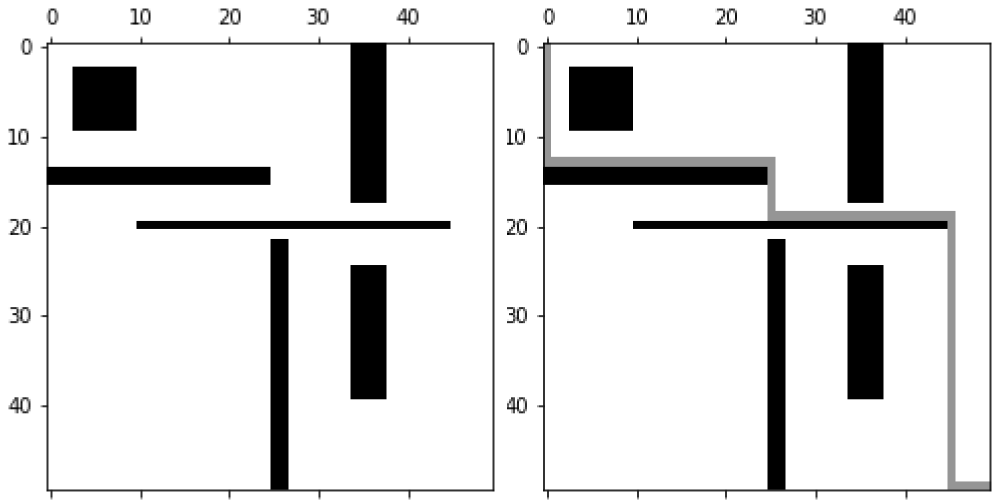
For A* algorithm, let heuristic h_{ij} = $|\mathrm{i} - goal_i|$+$|\mathrm{j} - goal_j|$ where $goal_i$ and $goal_j$ are the coordinates of the goal position.

i. Maze 0:



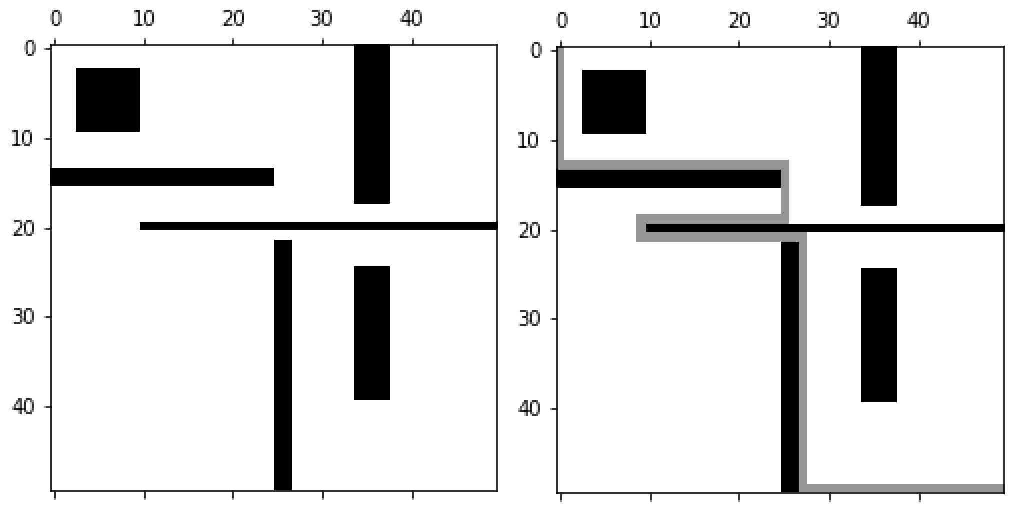
There are 11 nodes tested in CURRENT.
The shortest path is 9.

ii. Maze 1:



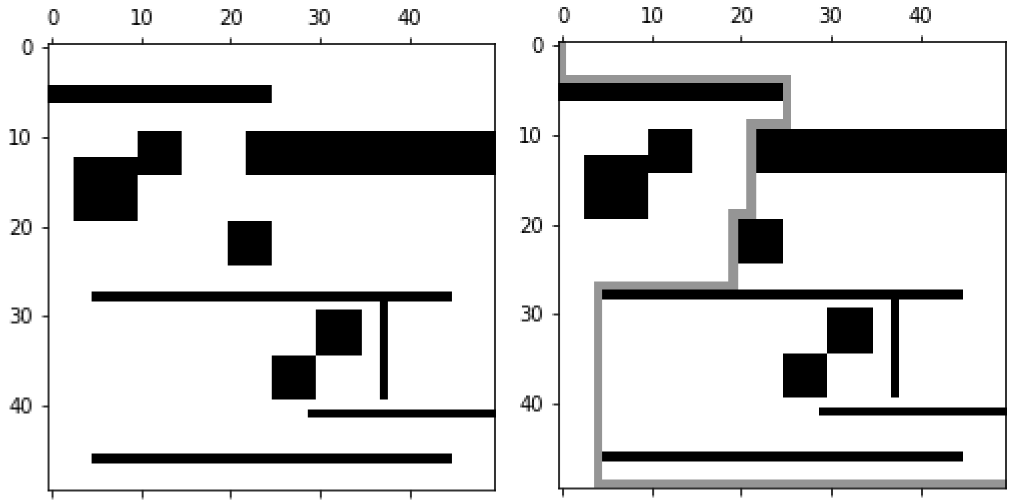
There are 1724 nodes tested in CURRENT.
The shortest path is 99.

iii. **Maze 2:**



There are 2286 nodes tested in CURRENT.
The shortest path is 131.

iv. **Maze 3**



There are 2732 nodes tested in CURRENT.
The shortest path is 141.

Data comparison for 2 situations:

Maze	$h_{ij} = 0$		$h_{ij} = \lvert \mathrm{i} - goal_i \rvert + \lvert \mathrm{j} - goal_j \rvert$	
	Nodes	Path	Nodes	Path
0	17	9	11	9
1	2037	99	1724	99
2	2173	131	2286	131
3	2049	141	2732	141

d) Depth-First:
Pros: If we're looking for global info (such as reachability) then it'll mainly produce a test on as much paths as it can, e.g. DFS is great for cycle detection. Besides DFS consumes less memory and it finds the larger distant element (from source vertex) in less time.
Cons: DFS is recursive, which introduces an overhead, DFS by itself doesn't guarantee the shortest path, only a path. It may get trapped in searching useless path.

Breadth-First:
Pros: BFS is that in unweighted graphs it can be used to construct a shortest path, BFS guarantees visiting all nodes at distance 1 before those at distance 2, etc. If the graph is laid out with neighboring nodes being nearby in memory, then BFS can get some cache locality. BFS can also be better for multiple destination nodes. Used to find the shortest path between vertices and always return the optimal solution. There is nothing like useless path in BFS, since it searches level by level. And it will find the closest goal in less time.
Cons: All of the connected vertices must be stored in memory. So consumes more memory

A* algorithm:
Pros: A* can be morphed into another path-finding algorithm by simply playing with the heuristics it uses and how it evaluates each node. This can be done to simulate BFS and DFS. A* will always find a solution if it exists. And A* expands on a node only if it seems promising. Its only focus is to reach the goal node as quickly as possible from the current node, not to try and reach every other node
Cons: If you have many target nodes and you don't know which one is closest to the main one, A* is not very optimal. This is because it needs to be run several times (once per target node) in order to get to all of them.