ECE457A - Assignment 2

Written by: Anson Wan

Due date: Jun 5th 2022

Question 1:

Assumptions:

- Repeated states are not expanded if the same state has already been expanded before and is thus already in the closed queue.
- Ties will be broken by the numeric value of the city. The lower numerical value city will have priority.

i) Uniform Cost Search

Nodes will be represented as: $N^{g(N)}$
Where N is the node number.
g(N) is the cost to reach the current node from the start node.

Expanded	Open Queue	Closed Queue
	1^{0}	
1^{0}	$5^{5}, 8^{24}$	
5^{5}	$8^{24}, 6^{40}$	1^{0}
8^{24}	$10^{39}, 6^{40}, 3^{47}$	$1^{0}, 5^{5}$
10^{39}	$6^{40}, 3^{47}, 3^{63}, 9^{65}, 6^{69}$	$1^{0}, 5^{5}, 8^{24}$
6^{40}	$3^{47}, 3^{63}, 9^{65}, 9^{66}, 6^{69}, 2^{78}$	$1^{0}, 5^{5}, 8^{24}, 10^{39}$
3^{47}	$4^{54}, 3^{63}, 9^{65}, 9^{66}, 6^{69}, 2^{78}$	$1^{0}, 5^{5}, 8^{24}, 10^{39}, 6^{40}$
4^{54}	$3^{63}, 9^{65}, 9^{66}, 6^{69}, 9^{72}, 2^{78}$	$1^{0}, 5^{5}, 8^{24}, 10^{39}, 6^{40}, 3^{47}$
9^{65}	$9^{66}, 6^{69}, 9^{72}, 2^{78}, 2^{91}, 7^{100}$	$1^{0}, 5^{5}, 8^{24}, 10^{39}, 6^{40}, 3^{47}, 4^{54}$
2^{78}	$2^{91}, 7^{100}, 7^{110}$	$1^{0}, 5^{5}, 8^{24}, 10^{39}, 6^{40}, 3^{47}, 4^{54}, 9^{65}$
7^{100}	7^{110}	$1^{0}, 5^{5}, 8^{24}, 10^{39}, 6^{40}, 3^{47}, 4^{54}, 9^{65}, 2^{78}$

The found path using UCS is:

$$1^{0} \text{ -> } 8^{24} \text{ -> } 10^{39} \text{ -> } 9^{65} \text{ -> } 7^{100}$$

ii) Greedy Best First Search

Nodes will be represented as: $N^{g(N)}_{f(N)=h(N)}$

Where N is the node number.
Where g(N) is the cost to reach the current node from the start node.
Where f(N) is the evaluation function which in this case is simply the heuristic h(N).

Expanded	Open Queue	Closed Queue
	1^{0}_{78}	
1^{0}_{78}	$8^{24}_{60}, 5^{5}_{75}$	
8^{24}_{60}	$3^{47}_{37}, 10^{39}_{57}, 5^{5}_{75}$	1^{0}_{78}
3^{47}_{37}	$4^{54}_{30}, 10^{39}_{57}, 10^{71}_{57}, 5^{5}_{75}$	$1^{0}_{78}, 8^{24}_{60}$
4^{54}_{30}	$9^{72}_{35}, 10^{39}_{57}, 10^{71}_{57}, 5^{5}_{75}$	$1^{0}_{78}, 8^{24}_{60}, 3^{47}_{37}$
9^{72}_{35}	$7^{107}_{0}, 10^{39}_{57}, 10^{71}_{57}, 5^{5}_{75}$	$1^{0}_{78}, 8^{24}_{60}, 3^{47}_{37}, 4^{54}_{30}$
7^{107}_{0}	$10^{39}_{57}, 10^{71}_{57}, 5^{5}_{75}$	$1^{0}_{78}, 8^{24}_{60}, 3^{47}_{37}, 4^{54}_{30}, 9^{72}_{35}$

The found path using Greedy BFS is:

$$1^{0}_{78} \text{ -> } 8^{24}_{60} \text{ -> } 3^{47}_{37} \text{ -> } 4^{54}_{30} \text{ -> } 9^{72}_{35} \text{ -> } 7^{107}_{0}$$

iii) A* Algorithm

Nodes will be represented as: $N^{g(N)}_{f(N)=g(N)+h(N)}$
Where N is the node number.
Where g(N) is the cost to reach the current node from the start node.
Where f(N) is the evaluation function which in this case is g(N) + the heuristic h(N).

Expanded	Open Queue	Closed Queue
	1^{0}_{78}	
1^{0}_{78}	5^{5}_{80} , 8^{24}_{84}	
5^{5}_{80}	8^{24}_{84} , 6^{40}_{100}	1^{0}_{78}
8^{24}_{84}	3^{47}_{84} , 10^{39}_{96} , 6^{40}_{100}	1^{0}_{78} , 5^{5}_{80}
3^{47}_{84}	4^{54}_{84} , 10^{39}_{96} , 6^{40}_{100} , 10^{71}_{128}	1^{0}_{78} , 5^{5}_{80} , 8^{24}_{84}
4^{54}_{84}	10^{39}_{96} , 6^{40}_{100} , 9^{72}_{107} , 10^{71}_{128}	1^{0}_{78} , 5^{5}_{80} , 8^{24}_{84} , 3^{47}_{84}
10^{39}_{96}	6^{40}_{100} , 9^{65}_{100} , 9^{72}_{107} , 10^{71}_{128} , 6^{69}_{129}	1^{0}_{78} , 5^{5}_{80} , 8^{24}_{84} , 3^{47}_{84} , 4^{54}_{84}
6^{40}_{100}	9^{65}_{100} , 9^{66}_{101} , 9^{72}_{107} , 2^{78}_{110} , 10^{71}_{128} , 6^{69}_{129}	1^{0}_{78} , 5^{5}_{80} , 8^{24}_{84} , 3^{47}_{84} , 4^{54}_{84} , 10^{39}_{96}
9^{65}_{100}	7^{100}_{100} , 9^{66}_{101} , 9^{72}_{107} , 2^{78}_{110} , 2^{91}_{123} , 10^{71}_{128} , 6^{69}_{129}	1^{0}_{78} , 5^{5}_{80} , 8^{24}_{84} , 3^{47}_{84} , 4^{54}_{84} , 10^{39}_{96} , 6^{40}_{100}
7^{100}_{100}	9^{66}_{101} , 9^{72}_{107} , 2^{78}_{110} , 2^{91}_{123} , 10^{71}_{128} , 6^{69}_{129}	1^{0}_{78} , 5^{5}_{80} , 8^{24}_{84} , 3^{47}_{84} , 4^{54}_{84} , 10^{39}_{96} , 6^{40}_{100} , 9^{65}_{100}

The found path using A* Algorithm is:

1^{0}_{78} -> 8^{24}_{84} -> 10^{39}_{96} -> 9^{65}_{100} -> 7^{100}_{100}

Question 2:

BFS:

a) How BFS was implemented:

- A dictionary with keys: position, cost, and path was used to represent a node/state.
- A stack was used to implement the open queue for BFS. A python list data structure was used with .append() and .pop(0) functions being used to simulate a queue which is FIFO.
- A list was used to implement the closed queue.
- Priority of placing node children into the open queue was RIGHT, UP, LEFT, DOWN in that order. So nodes on the right are taken out of the queue first since it is put into the queue first, then upwards nodes, then leftwards nodes, then downward nodes.
- The given maze that is a 2D 25 x 25 array was used to keep track of visited nodes. If a node is visited, that index of the maze would be given an integer value of 2.
- Termination condition is if the open queue is empty or if the exit node is found.

b) Sample output of BFS search from starting point (0,0) to (24,24):

Assignment 2 Question 2 Maze Solver. Written by Anson Wan
Now executing BREADTH-FIRST SEARCH...
Solution found!
The path is:
[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (17, 1), (17, 2), (17, 3), (17, 4), (17, 5), (17, 6), (17, 7), (17, 8), (17, 9), (17, 10), (17, 11), (17, 12), (17, 13), (18, 13), (19, 13), (20, 13), (20, 14), (21, 14), (21, 15), (22, 15), (22, 16), (23, 16), (23, 17), (24, 17), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24)]
The cost is: 49
Number of nodes visited is: 448

c) Test Cases:

S -> E1:
Solution found!
The path is:
[(2, 11), (3, 11), (3, 12), (4, 12), (5, 12), (6, 12), (7, 12), (7, 13), (7, 14), (7, 15), (8, 15), (9, 15), (9, 16), (9, 17), (10, 17), (11, 17), (12, 17), (13, 17), (14, 17), (15, 17), (16, 17), (17, 17), (18, 17), (19, 17), (20, 17), (21, 17), (22, 17), (23, 17), (23, 18), (23, 19)]
The cost is: 30
Number of nodes visited is: 374

S -> E2:
Solution found!
The path is:
[(2, 11), (3, 11), (3, 12), (4, 12), (5, 12), (6, 12), (7, 12), (7, 13), (7, 14), (7, 15), (7, 16), (7, 17), (7, 18), (7, 19), (6, 19), (5, 19), (4, 19), (3, 19), (2, 19), (2, 20), (2, 21)]
The cost is: 21
Number of nodes visited is: 258

(0,0) -> (24,24):
Solution found!
The path is:
[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (17, 1), (17, 2), (17, 3), (17, 4), (17, 5), (17, 6), (17, 7), (17, 8), (17, 9), (17, 10), (17, 11), (17, 12), (17, 13), (18, 13), (19, 13), (20, 13), (20, 14), (21, 14), (21, 15), (22, 15), (22, 16), (23, 16), (23, 17), (24, 17), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24)]
The cost is: 49
Number of nodes visited is: 448

DFS:

a) How DFS was implemented:

- A dictionary with keys: position, cost, and path was used to represent a node/state.
- A stack was used to implement the open queue for DFS. A python list data structure was used with append() and pop() functions being used to simulate a stack which is LIFO.
- A list was used to implement the closed queue.
- Priority of placing node children into the open queue (stack in this case) was DOWN, LEFT, UP, RIGHT in that order. So nodes on the right are taken out of the stack first since it is put into the stack last, then upwards nodes, then leftwards nodes, then downward nodes.
- The given maze that is a 2D 25 x 25 array was used to keep track of visited nodes. If a node is visited, that index of the maze would be given an integer value of 2.
- Termination condition is if the open queue is empty or if the exit node is found.

b) Sample output of DFS search from starting point (0,0) to (24,24):

Assignment 2 Question 2 Maze Solver. Written by Anson Wan
Now executing DEPTH-FIRST SEARCH...
Solution found!
The path is:
[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (18, 0), (19, 0), (20, 0), (21, 0), (22, 0), (23, 0), (24, 0), (24, 1), (24, 2), (24, 3), (24, 4), (24, 5), (24, 6), (24, 7), (24, 8), (24, 9), (24, 10), (24, 11), (23, 11), (23, 12), (22, 12), (22, 11), (22, 10), (23, 10), (23, 9), (22, 9), (22, 8), (23, 8), (23, 7), (22, 7), (21, 7), (20, 7), (20, 8), (20, 9), (20, 10), (20, 11), (20, 12), (20, 13), (20, 14), (21, 14), (21, 15), (22, 15), (22, 16), (23, 16), (23, 17), (24, 17), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24)]
The cost is: 71
Number of nodes visited is: 71

c) Test Cases:

S -> E1:
Solution found!
The path is:

[(2, 11), (3, 11), (3, 12), (4, 12), (5, 12), (6, 12), (7, 12), (8, 12), (9, 12), (9, 13), (8, 13), (7, 13), (7, 14), (7, 15), (8, 15), (9, 15), (9, 16), (9, 17), (10, 17), (11, 17), (12, 17), (13, 17), (14, 17), (15, 17), (16, 17), (17, 17), (18, 17), (19, 17), (20, 17), (21, 17), (22, 17), (23, 17), (24, 17), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24), (23, 24), (22, 24), (21, 24), (20, 24), (19, 24), (19, 23), (20, 23), (21, 23), (22, 23), (23, 23), (23, 22), (22, 22), (21, 22), (21, 21), (22, 21), (23, 21), (23, 20), (22, 20), (21, 20), (21, 19), (22, 19), (23, 19)]
The cost is: 62
Number of nodes visited is: 62

S -> E2:
Solution found!
The path is:
[(2, 11), (3, 11), (3, 12), (4, 12), (5, 12), (6, 12), (7, 12), (8, 12), (9, 12), (9, 13), (8, 13), (7, 13), (7, 14), (7, 15), (8, 15), (9, 15), (9, 16), (9, 17), (10, 17), (11, 17), (12, 17), (13, 17), (14, 17), (15, 17), (16, 17), (17, 17), (18, 17), (19, 17), (20, 17), (21, 17), (22, 17), (23, 17), (23, 16), (22, 16), (21, 16), (20, 16), (19, 16), (18, 16), (17, 16), (16, 16), (15, 16), (14, 16), (13, 16), (12, 16), (11, 16), (11, 15), (12, 15), (13, 15), (14, 15), (15, 15), (16, 15), (17, 15), (18, 15), (19, 15), (20, 15), (21, 15), (21, 14), (20, 14), (19, 14), (18, 14), (17, 14), (17, 13), (18, 13), (19, 13), (20, 13), (20, 12), (19, 12), (19, 11), (20, 11), (20, 10), (19, 10), (19, 9), (20, 9), (20, 8), (19, 8), (19, 7), (20, 7), (21, 7), (22, 7), (23, 7), (24, 7), (24, 6), (24, 5), (23, 5), (22, 5), (22, 4), (23, 4), (24, 4), (24, 3), (23, 3), (22, 3), (21, 3), (21, 4), (20, 4), (20, 3), (20, 2), (21, 2), (22, 2), (23, 2), (24, 2), (24, 1), (23, 1), (22, 1), (21, 1), (20, 1), (20, 0), (19, 0), (18, 0), (17, 0), (17, 1), (17, 2), (17, 3), (17, 4), (17, 5), (17, 6), (17, 7), (17, 8), (17, 9), (16, 9), (15, 9), (14, 9), (14, 10), (14, 11), (14, 12), (14, 13), (14, 14), (13, 14), (13, 13), (12, 13), (11, 13), (11, 12), (12, 12), (13, 12), (13, 11), (12, 11), (11, 11), (10, 11), (9, 11), (8, 11), (7, 11), (6, 11), (5, 11), (5, 10), (6, 10), (7, 10), (8, 10), (9, 10), (10, 10), (11, 10), (12, 10), (13, 10), (13, 9), (12, 9), (11, 9), (10, 9), (9, 9), (8, 9), (7, 9), (6, 9), (5, 9), (4, 9), (4, 10), (3, 10), (2, 10), (1, 10), (1, 11), (1, 12), (2, 12), (2, 13), (3, 13), (4, 13), (5, 13), (6, 13), (6, 14), (6, 15), (6, 16), (7, 16), (7, 17), (7, 18), (7, 19), (7, 20), (7, 21), (7, 22), (6, 22), (6, 21), (6, 20), (6, 19), (5, 19), (4, 19), (3, 19), (2, 19), (2, 20), (2, 21)]
The cost is: 193
Number of nodes visited is: 296

(0,0) -> (24,24):
Solution found!
The path is:
[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (18, 0), (19, 0), (20, 0), (21, 0), (22, 0), (23, 0), (24, 0), (24, 1), (24, 2), (24, 3), (24, 4), (24, 5), (24, 6), (24, 7), (24, 8), (24, 9), (24, 10), (24, 11), (23, 11), (23, 12), (22, 12), (22, 11), (22, 10), (23, 10), (23, 9), (22, 9), (22, 8), (23, 8), (23, 7), (22, 7), (21, 7), (20, 7), (20, 8), (20, 9), (20, 10), (20, 11), (20, 12), (20, 13), (20, 14), (21, 14), (21, 15), (22, 15), (22, 16), (23, 16), (23, 17), (24, 17), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24)]
The cost is: 71
Number of nodes visited is: 71

A* Search:

a) How A* Search was implemented:

- A PriorityEntry object was defined with an evaluation function value and a metadata dictionary as its properties. The metadata dictionary had keys: position, cost, and path. This PriorityEntry object was used to represent a node/state.
- A PriorityQueue was used to implement the open queue for A* Search. This PriorityQueue was imported via Python's standard Queue library.
- A list was used to implement the closed queue.
- Priority of placing node children into the PriorityQueue was RIGHT, UP, LEFT, DOWN in that order. So if there was a tie in the evaluation function, the PriorityQueue would take the one inserted first.
- The heuristic that was used for the A* search was the Manhattan Distance.
- The given maze that is a 2D 25 x 25 array was used to keep track of visited nodes. If a node is visited, that index of the maze would be given an integer value of 2.
- Termination condition is if the open queue is empty or if the exit node is found.

b) Sample output of BFS search from starting point (0,0) to (24,24):

Assignment 2 Question 2 Maze Solver. Written by Anson Wan
Now executing A-STAR SEARCH...
Solution found!
Cost is: 49
Path is: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 6), (5, 6), (5, 7), (5, 8), (6, 8), (7, 8), (7, 9), (8, 9), (9, 9), (10, 9), (11, 9), (11, 10), (12, 10), (12, 11), (13, 11), (13, 12), (14, 12), (14, 13), (14, 14), (14, 15), (15, 15), (16, 15), (17, 15), (18, 15), (19, 15), (19, 16), (20, 16), (21, 16), (22, 16), (22, 17), (22, 18), (23, 18), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24)]
Number of visited nodes is: 219

c) Test Cases:

S -> E1:
Solution found!
Cost is: 30
Path is: [(2, 11), (3, 11), (3, 12), (4, 12), (5, 12), (5, 13), (5, 14), (6, 14), (6, 15), (7, 15), (8, 15), (9, 15), (9, 16), (9, 17), (10, 17), (11, 17), (12, 17), (12, 18), (13, 18), (13, 19), (14, 19), (15, 19), (16, 19), (17, 19), (18, 19), (19, 19), (20, 19), (21, 19), (22, 19), (23, 19)]
Number of visited nodes is: 71

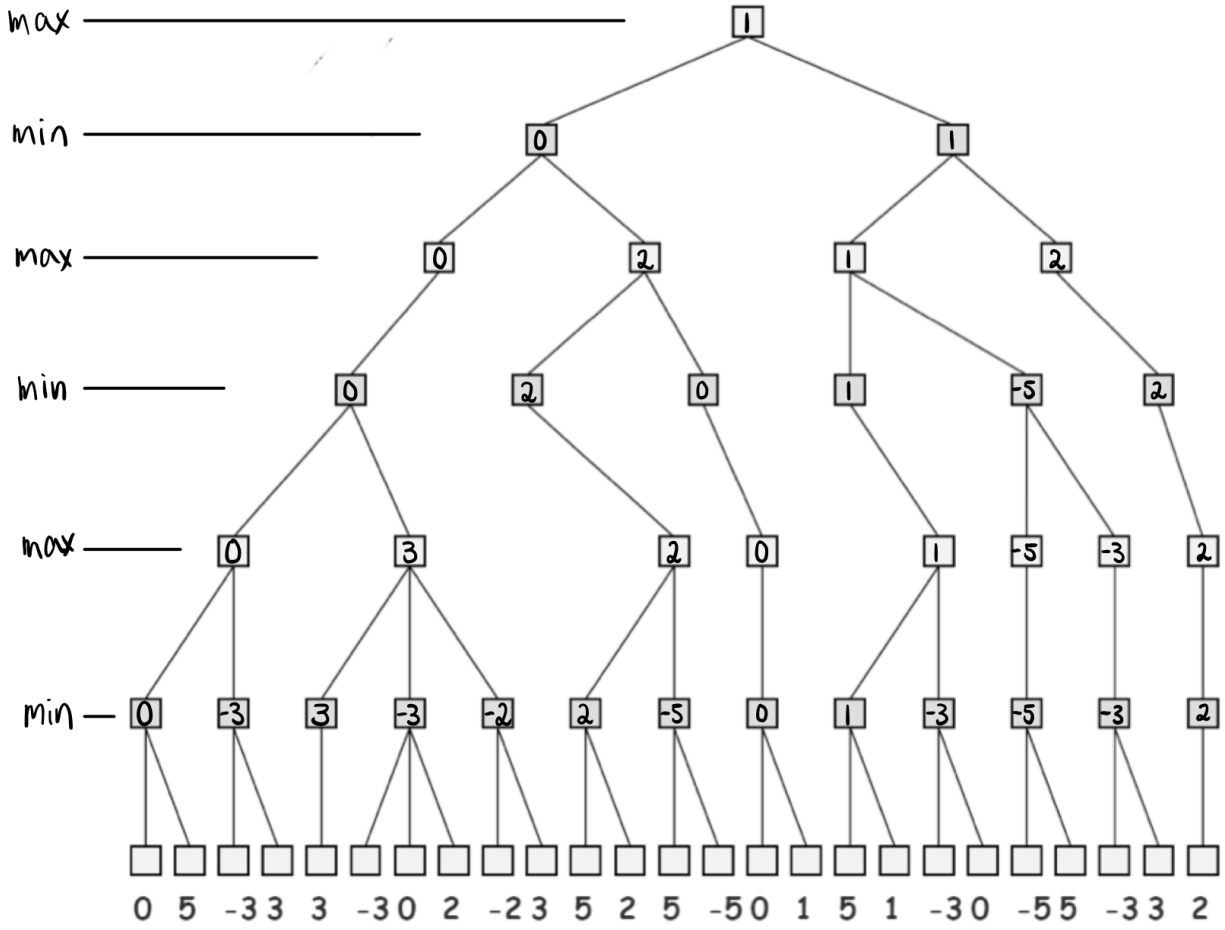
S -> E2:
Solution found!
Cost is: 21
Path is: [(2, 11), (2, 12), (2, 13), (3, 13), (3, 14), (4, 14), (4, 15), (5, 15), (5, 16), (6, 16), (7, 16), (7, 17), (7, 18), (6, 18), (5, 18), (4, 18), (4, 19), (3, 19), (2, 19), (2, 20), (2, 21)]
Number of visited nodes is: 79

(0,0) -> (24,24):

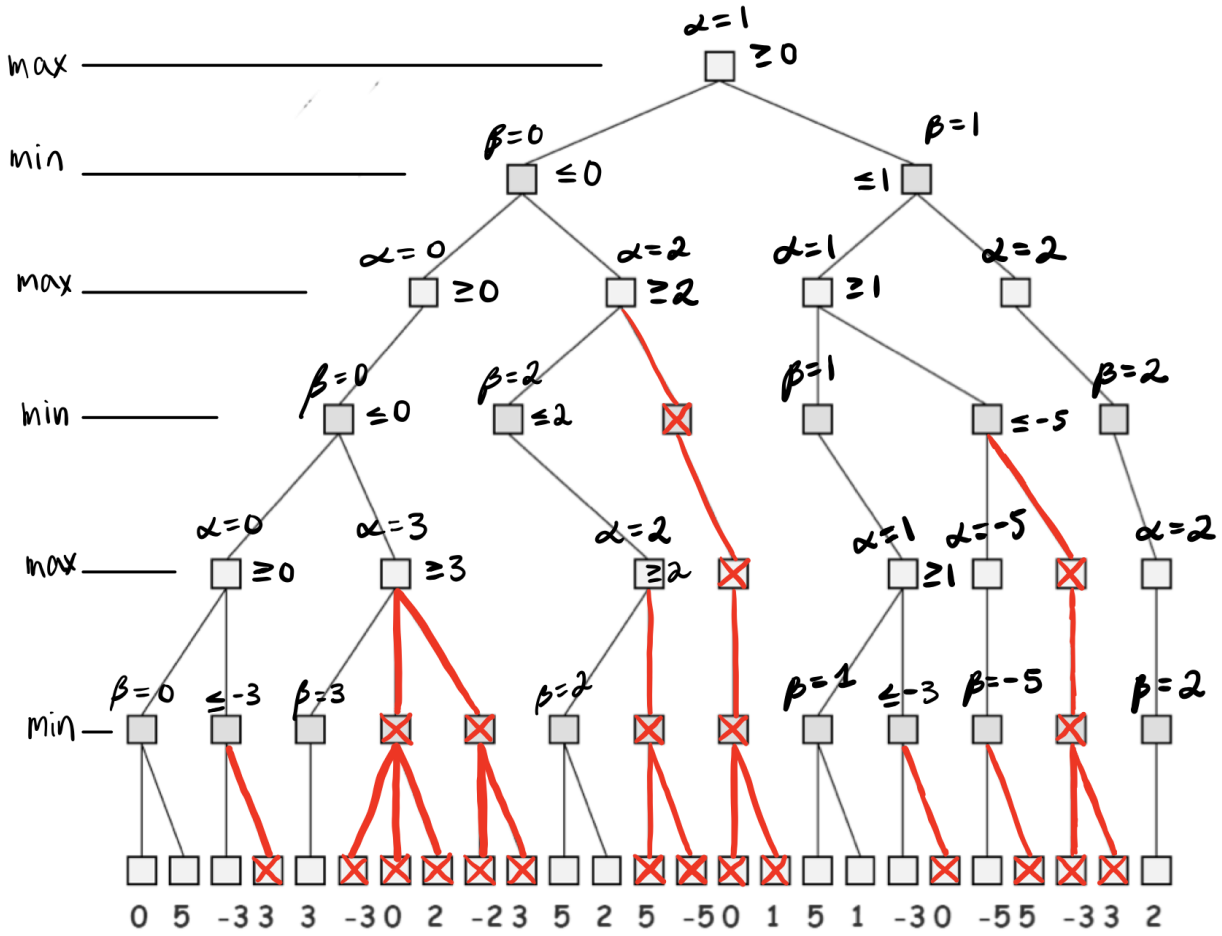
Solution found!
Cost is: 49
Path is: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 6), (5, 6), (5, 7), (5, 8), (6, 8), (7, 8), (7, 9), (8, 9), (9, 9), (10, 9), (11, 9), (11, 10), (12, 10), (12, 11), (13, 11), (13, 12), (14, 12), (14, 13), (14, 14), (14, 15), (15, 15), (16, 15), (17, 15), (18, 15), (19, 15), (19, 16), (20, 16), (21, 16), (22, 16), (22, 17), (22, 18), (23, 18), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (24, 23), (24, 24)]
Number of visited nodes is: 219

Question 3:

a) Minimax Algorithm

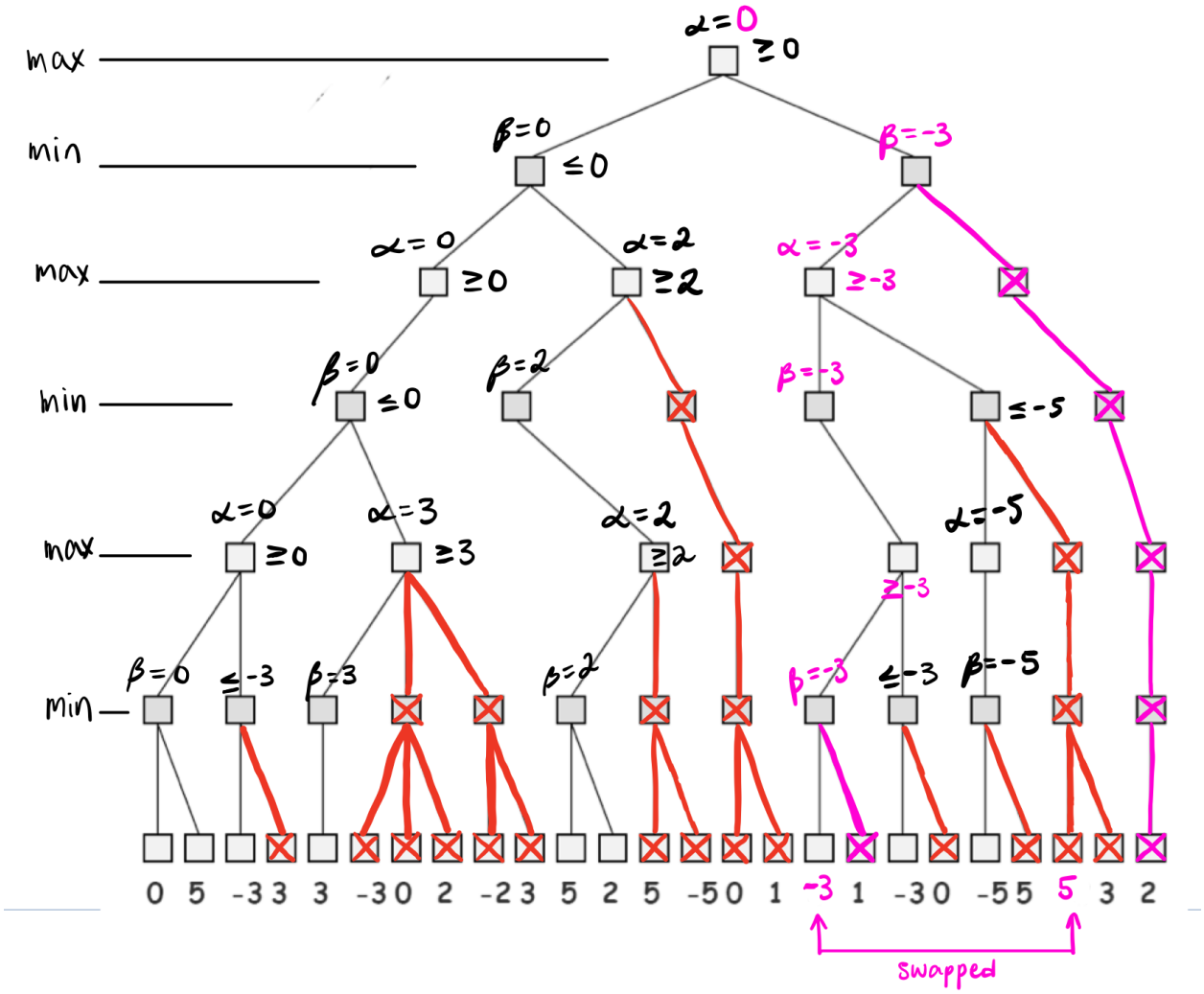


b) Alpha-Beta Pruning



c) How to increase the number of pruned nodes

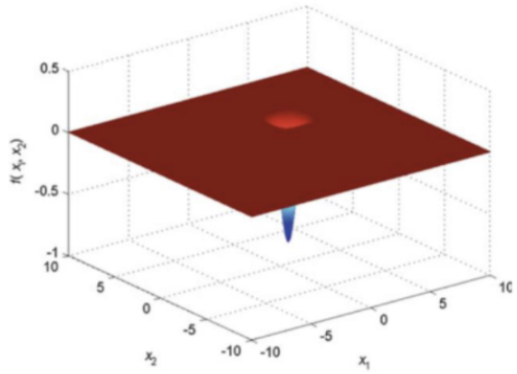
6 additional pruned nodes are created by swapping the 13rd and 9th nodes from the right. The result of alpha-beta pruning after the swap is as follows.



Question 4:

Minimize Eason function of two variables:

$$\min_{x} f(x) = -\cos x_1 \cos x_2 \exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right), \quad x \in [-100, 100]^2$$



a) Problem Formulation

State:

A state is the value of x such that it is a valid input of the Easom Function within its defined range, which in this case is [-100. 100].

This can be represented as:

$$x = (x_1, x_2), \ where \ x \in [-100, 100]^2$$

Initial State:

The initial state can be any value of x that is within the range $x \in [-100, 100]^2$.
In this case, it can be a random point.
This can be represented as:

$$x = (x_1, x_2), \ where \ x \ is \ a \ random \ number \ within \ the \ space \ and \ x \in [-100, 100]^2$$

Goal:

The goal state is the global minimum of the Easom Function within the search space. In this case, we know that the minimum is at $x = (\pi, \pi)$.
Thus, the goal state can be represented as:

$$x^* = (\pi, \pi)$$

Actions:

The action taken will be sampling the neighborhood function to find a candidate solution state. If the candidate solution state is superior to the prior state, it will become the solution state. If it is not superior to the prior state, a random number n will be generated and the probability $e^{-\Delta C/t}$ will be calculated. If $n < e^{-\Delta C/t}$ then we will accept the new state regardless.

Cost:
The cost is the Easom Function value at a state x.

This can be represented as:

$$C = f(x)\ ,\ where\ f\ is\ the\ Easom\ Function.$$

Simulated Annealing Neighborhood Function:
The neighborhood function will be responsible for choosing/sampling candidate solution states near the current state.

This can be done by choosing a random state near a radius of 5 units within the current state.

Thus, the neighborhood function can be represented as:

$$N(x_{current}) = x_{current} + r\ ,\ where\ r\ is\ a\ random\ number\ from\ [-5, 5]\ and\ N \in [-100, 100]^2$$

Simulated Annealing Cost Function:
The cost function will be responsible for signaling whether the new state is superior (closer to the minimum) that the current state.

A cost function that can be used for the simulated annealing algorithm is to subtract the Easom Function values at the candidate x value with value at the current x value since if the value of the difference is less than 0, the candidate state will be taken.
Thus, the cost function can be represented as:

$$C(x_{candidate}, x_{current}) = (f(x_{candidate}) - f(x_{current}))\ ,\ where\ f(x)\ is\ the\ Easom\ Function.$$

b) Observations

i) Using 10 different initial points:
Using an initial temperature of 10, with 1000 iterations per temperature, and a geometric annealing schedule with alpha = 0.95. TThe results using 10 randomly generated initial points were:

(14.036955290028573, 24.783325778032278) -> (3.1342762757665765, 3.1332351724569283)
(-90.15713313761253, 7.759292900007324) -> (3.1618724817977064, 3.1225290671565515)
(-21.571507836761228, -44.86733540329717) -> (3.1708870840353827, 3.1302079484435783)
(-18.51984178120958, 74.88169066077802) -> (3.128219415304672, 3.146256230667836)
(16.17705433167727, -61.24790327233471) -> (3.144884092709308, 3.1746841952061597)
(-65.45245370816835, 38.98233308136011) -> (3.133116396808749, 3.133741750783356)
(43.22830716233321, 80.2020243164009) -> (3.086252565153626, 3.2052957189362923)
(-61.15291270705263, 71.24093959971816) -> (3.11834241573605, 3.136469777486737)
(91.07323966619575, 45.50366291661658) -> (3.173348955044621, 3.1513611783574245)
(87.8162877793421, 79.24764989010379) -> (3.150414168010875, 3.1590489943341127)

The best starting point from these 10 randomly generated points was:
(14.036955290028573, 24.783325778032278)

ii) Using 10 different initial temperatures:
The range of values to generate temperatures from shall be [3, 20]. This range is chosen to vary the balance between exploration and exploitation. This means that larger temperature values will promote exploring the space more while lower temperatures will promote narrowing down a solution within an area in the space. Thus, it will be interesting to see which values work better for our use case.
The starting point will be (14,25) as this was the best starting point found from part(i), the iterations per temperature will be 1000, and the annealing schedule will be geometric with alpha = 0.95.

The results using 10 randomly generated starting temperatures from [500, 3000] were:

Start temp: 17, Start point: (14, 25) -> Result: (3.1544652880050945, 3.1415024002664076)
Start temp: 3, Start point: (14, 25) -> Result: (3.1733284513310966, 3.143138280658592)
Start temp: 4, Start point: (14, 25) -> Result: (3.139618080681572, 3.132479992158351)
Start temp: 10, Start point: (14, 25) -> Result: (3.130285257183245, 3.1311759656325497)
Start temp: 9, Start point: (14, 25) -> Result: (3.1256181579713735, 3.124031646901466)
Start temp: 18, Start point: (14, 25) -> Result: (3.1527693010215154, 3.1278380505282533)
Start temp: 11, Start point: (14, 25) -> Result: (3.1420710834626635, 3.1260312917806163)
Start temp: 18, Start point: (14, 25) -> Result: (3.1352897659363688, 3.1653881789730995)
Start temp: 5, Start point: (14, 25) -> Result: (3.1677124994557984, 3.1125729332567844)
Start temp: 16, Start point: (14, 25) -> Result: (3.165049971621136, 3.0934078812134453)

The best starting temperature from these 10 randomly generated values from [3,20] was: 10

iii) Using 9 different annealing schedules:
The 9 different annealing schedules will be as follows:

1. Linear with $\alpha = 0.001$
2. Linear with $\alpha = 0.01$
3. Linear with $\alpha = 0.1$
4. Geometric with $\alpha = 0.95$
5. Geometric with $\alpha = 0.995$
6. Geometric with $\alpha = 0.999$
7. Slow decrease with $\beta = 0.0001$
8. Slow decrease with $\beta = 0.001$
9. Slow decrease with $\beta = 0.01$

Using the best starting point from part (i) and the best starting temp from part (ii), that being starting point = (14,25) and starting temp = 10, the result of using these 9 different annealing schedules is:

Linear Annealing:

α = 0.001 -> Result: (-58.40930131641905, -40.19464032521961)
α = 0.01 -> Result: (3.4340085272421037, 3.3886371899124184)
α = 0.1 -> Result: (39.042067798332084, -87.56446248757163)

Geometric Annealing:

α = 0.95 -> Result: (3.1429536710393737, 3.1354727638404967)
α = 0.995 -> Result: (3.1443847887081064, 3.1452833440943415)
α = 0.999 -> Result: (3.1273133220583107, 3.1464368665906104)

Slow-decrease Annealing:

β = 0.0001 -> Result: (3.127153762487651, 3.164537505539495)
β = 0.001 -> Result: (3.1713168040089603, 3.1441361188219874)
β = 0.01 -> Result: (3.116005719219605, 3.1410752131730337)

The results of these 9 annealing schedules show that geometric annealing with α = 0.995 resulted in the best solution.

c) Best Solution

Through the experiments done in part(b), the best solution found was the point:

(3.1443847887081064, 3.1452833440943415)

This point was found using the settings:

Starting point: (14,25)
Starting temperature: 10
Temperature annealing schedule: Geometric
α = 0.995
Iterations per temperature: 1000

This configuration of SA performed better than the others due to it having a good balance between exploration and exploitation for this particular user scenario. It is important that initially, the SA algorithm must promote exploration to find a region with good prospects. This is likely to take a large number of iterations due to the nature of the Easom Function. If the temperature is not high enough initially, or the annealing schedule reduces the temperature too quickly, the algorithm may never find the solution region. On the other hand, it is also important that at lower temperatures the algorithm promotes restricting exploration to find the most optimal solution and converge upon it. This is why geometric annealing was the most superior as it had a balance of both.