Assignment 2

Question 1

For all parts of Question 1, repeated states are ignored as per the assignment instructions, therefore optimality is no longer guaranteed.

(i) Uniform Cost Search

Form: $n^{g(n)}$, where n is the city number, and g(n) is the actual travel distance from city 1 to city n.

Expanded Node	**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} 9^{65} }	{ 1^{0} 5^{5} 8^{24} }
6^{40}	{ 3^{47} 9^{65} 2^{78} }	{ 1^{0} 5^{5} 8^{24} 10^{39} }
3^{47}	{ 4^{54} 9^{65} 2^{78} }	{ 1^{0} 5^{5} 8^{24} 10^{39} 6^{40} }
4^{54}	{ 9^{65} 2^{78} }	{ 1^{0} 5^{5} 8^{24} 10^{39} 6^{40} 3^{47} }
9^{65}	{ 2^{78} 7^{100} }	{ 1^{0} 5^{5} 8^{24} 10^{39} 6^{40} 3^{47} 4^{54} }
2^{78}	{ 7^{100} }	{ 1^{0} 5^{5} 8^{24} 10^{39} 6^{40} 3^{47} 4^{54} 9^{65} }
7^{100}	{ }	{ 1^{0} 5^{5} 8^{24} 10^{39} 6^{40} 3^{47} 4^{54} 9^{65} 2^{78} }

Path: 1 → 8 → 10 → 9 → 7

Cost: 100

Assignment 2

(ii) Greedy Best First Search

Form: $n^{h(n)}$, where n is the city number, and h(n) is an estimate of distance from city n to city 7.

Expanded Node	**Open Queue**	**Closed Queue**
	{ 1^{78} }	
1^{78}	{ 8^{60} 5^{75} }	
8^{60}	{ 3^{37} 10^{57} 5^{75} }	{ 1^{78} }
3^{37}	{ 4^{30} 10^{57} 5^{75} }	{ 1^{78} 8^{60} }
4^{30}	{ 9^{35} 10^{57} 5^{75} }	{ 1^{78} 8^{60} 3^{37} }
9^{35}	{ 7^{0} 2^{32} 10^{57} 6^{60} 5^{75} }	{ 1^{78} 8^{60} 3^{37} 4^{30} }
7^{0}	{ 2^{32} 10^{57} 6^{60} 5^{75} }	{ 1^{78} 8^{60} 3^{37} 4^{30} 9^{35} }

Path: 1 → 8 → 3 → 4 → 9 → 7

Cost: 107

Paolo Torres

Assignment 2

(iii) A* Search

Form: $n^{g(n)+h(n)}$, where n is the city number, g(n) is the actual travel distance from city 1 to city n, and h(n) is an estimate of distance from city n to city 7.

Expanded Node	**Open Queue**	**Closed Queue**
	{ 1^{78} }	
1^{78}	{ 5^{80} 8^{84} }	
5^{80}	{ 8^{84} 6^{100} }	{ 1^{78} }
8^{84}	{ 3^{84} 10^{96} 6^{100} }	{ 1^{78} 5^{80} }
3^{84}	{ 4^{84} 10^{96} 6^{100} }	{ 1^{78} 5^{80} 8^{84} }
4^{84}	{ 10^{96} 6^{100} 9^{107} }	{ 1^{78} 5^{80} 8^{84} 3^{84} }
10^{96}	{ 6^{100} 9^{100} 9^{107} }	{ 1^{78} 5^{80} 8^{84} 3^{84} 4^{84} }
6^{100}	{ 9^{100} 9^{101} 9^{107} 2^{110} }	{ 1^{78} 5^{80} 8^{84} 3^{84} 4^{84} 10^{96} }
9^{100}	{ 7^{100} 9^{101} 9^{107} 2^{110} 2^{123} }	{ 1^{78} 5^{80} 8^{84} 3^{84} 4^{84} 10^{96} 6^{100} }
7^{100}	{ 2^{110} 2^{123} }	{ 1^{78} 5^{80} 8^{84} 3^{84} 4^{84} 10^{96} 6^{100} 9^{100} }

Path: 1 → 8 → 10 → 9 → 7

Cost: 100

Paolo Torres

Assignment 2

Question 2

To simplify setup of the maze in my program, let A represent E1 and B represent E2. This is to allow the maze to be structured as a 2D vector of characters as opposed to strings, saving on memory requirements and simplifying usage. So, in the code A and B are used, but in this assignment document E1 and E2 are used. Additionally, I assumed (0, 0) to be the top left of the maze, and (24, 24) be the bottom right. This simplifies coding since this is naturally how 2D vectors are indexed. So, (24, 0) and (0, 24) is used in the code to represent bottom left and top right, but (0, 0) and (24, 24) is used in the document.

(i) Breadth First Search

Breadth first search was implemented using a queue data structure, a 2D vector to track visited nodes, and a 2D vector containing adjacent directions of search. While the queue is not empty, pop the front of the queue and check if it is equal to the destination. If so return, but if not, iterate through the adjacent directions. For each adjacent direction, if that node is valid, meaning within bounds, not yet visited, and is not a wall, then push it to the queue and count up the number of nodes explored. Additionally, have a map that tracks the parent nodes with all its children nodes, with the keys being the parent nodes, and the values being its list of children nodes. Once the destination is found, trace its complete path using the map, count up its cost, and return these along with the nodes explored.

Starting at S and ending at E1
Path: (13, 2), (13, 3), (12, 3), (12, 4), (12, 5), (12, 6), (12, 7), (11, 7), (10, 7), (9, 7), (9, 8), (9, 9), (8, 9), (7, 9), (7, 10), (7, 11), (7, 12), (7, 13), (7, 14), (7, 15), (7, 16), (7, 17), (7, 18), (7, 19), (7, 20), (7, 21), (7, 22), (7, 23), (6, 23), (5, 23) Cost: 30 Nodes: 380

Starting at S and ending at E2
Path: (13, 2), (13, 3), (12, 3), (12, 4), (12, 5), (12, 6), (12, 7), (11, 7), (10, 7), (9, 7), (8, 7), (7, 7), (6, 7), (6, 6), (6, 5), (6, 4), (6, 3), (6, 2), (5, 2), (4, 2), (3, 2) Cost: 21

Nodes: 278

Starting at (0, 0) and ending at (24, 24)
Path: (24, 0), (24, 1), (24, 2), (24, 3), (24, 4), (24, 5), (24, 6), (24, 7), (24, 8), (24, 9), (24, 10), (24, 11), (24, 12), (24, 13), (24, 14), (24, 15), (24, 16), (24, 17), (23, 17), (22, 17), (21, 17), (20, 17), (19, 17), (18, 17), (17, 17), (16, 17), (15, 17), (14, 17), (13, 17), (12, 17), (11, 17), (11, 18), (11, 19), (11, 20), (10, 20), (10, 21), (9, 21), (9, 22), (8, 22), (8, 23), (7, 23), (7, 24), (6, 24), (5, 24), (4, 24), (3, 24), (2, 24), (1, 24), (0, 24) Cost: 49 Nodes: 447

(ii) Depth First Search

Depth first search was implemented using the call stack, a 2D vector to track visited nodes, and a 2D vector to store the path. A recursive approach was taken where the call stack was used to search in a last in, first out manner. If the next node to be searched has not yet been visited, set it to visited, push this coordinate to the path, and update the cost and node counters. The base case is if the destination has been reached, then return true and output the complete path, cost, and number of nodes explored. The recursive case is to search all the way left, then up, then right, then down, while ensuring the moves are valid, meaning within bounds and not a wall. If it passes all these recursive cases within the call stack, then set visited to false, pop from the path, and decrease the cost and nodes explored.

Starting at S and ending at E1
Path: (13, 2), (13, 1), (13, 0), (12, 0), (11, 0), (10, 0), (9, 0), (8, 0), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8), (8, 9), (7, 9), (7, 10), (7, 11), (6, 11), (5, 11), (4, 11), (3, 11), (3, 10), (3, 9), (2, 9), (2, 10), (1, 10), (0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (0, 15), (0, 16), (1, 16), (1, 15), (1, 14), (1, 13), (1, 12), (1, 11), (2, 11), (2, 12), (2, 13), (3, 13), (3, 12), (4, 12), (4, 13), (5, 13), (5, 12), (6, 12), (6, 13), (7, 13), (7, 12), (8, 12), (8, 11), (9, 11), (9, 12), (9, 13), (8, 13), (8, 14), (7, 14), (7, 15), (7, 16), (7, 17), (7, 18), (7, 19), (7, 20), (7, 21), (6, 21), (5, 21), (4, 21), (3, 21), (2, 21), (1, 21), (1, 20), (1, 19), (0, 19), (0, 20), (0, 21), (0, 22), (0, 23), (0, 24), (1, 24),

(1, 23), (1, 22), (2, 22), (2, 23), (2, 24), (3, 24), (3, 23), (3, 22), (4, 22), (4, 23), (4, 24), (5, 24), (5, 23)
Cost: 98
Nodes: 98

Starting at S and ending at E2
Path: (13, 2), (13, 1), (13, 0), (12, 0), (11, 0), (10, 0), (9, 0), (8, 0), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (7, 7), (6, 7), (6, 6), (6, 5), (6, 4), (6, 3), (6, 2), (6, 1), (6, 0), (5, 0), (4, 0), (3, 0), (3, 1), (3, 2)
Cost: 29
Nodes: 29

Starting at (0, 0) and ending at (24, 24)
Path: (24, 0), (23, 0), (22, 0), (21, 0), (21, 1), (21, 2), (21, 3), (20, 3), (19, 3), (18, 3), (18, 4), (18, 5), (17, 5), (16, 5), (15, 5), (15, 4), (15, 3), (15, 2), (15, 1), (15, 0), (14, 0), (13, 0), (12, 0), (11, 0), (10, 0), (9, 0), (8, 0), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8), (8, 9), (7, 9), (7, 10), (7, 11), (6, 11), (5, 11), (4, 11), (3, 11), (3, 10), (3, 9), (2, 9), (2, 10), (1, 10), (0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (0, 15), (0, 16), (1, 16), (1, 15), (1, 14), (1, 13), (1, 12), (1, 11), (2, 11), (2, 12), (2, 13), (3, 13), (3, 12), (4, 12), (4, 13), (5, 13), (5, 12), (6, 12), (6, 13), (7, 13), (7, 12), (8, 12), (8, 11), (9, 11), (9, 12), (9, 13), (8, 13), (8, 14), (7, 14), (7, 15), (7, 16), (7, 17), (7, 18), (7, 19), (7, 20), (7, 21), (6, 21), (5, 21), (4, 21), (3, 21), (2, 21), (1, 21), (1, 20), (1, 19), (0, 19), (0, 20), (0, 21), (0, 22), (0, 23), (0, 24)
Cost: 103
Nodes: 103

(iii) A* Search

A* search was implemented using a priority queue, where the data type was a pair, with the first element being an integer to store the F cost, and the second element being a pair to store the corresponding (x, y) coordinates. The F cost was calculated as the summation of the G cost and H cost, where G cost is the actual cost from the start node to the current node, while the H cost is the estimated cost from the current node to the goal node. In particular, the heuristic function (H cost) was determined using the Manhattan distance, namely the sum of the absolute values of the differences between the current node and goal node. The priority queue was modified to work as a min heap, meaning the data is sorted by lowest F cost, with the top element having the lowest one. When data is popped from the min heap, the next lowest F cost value is moved to the top, and when data is added to the heap, it automatically inserts it into the correct position, again based on the F cost.

Starting at S and ending at E1
Path: (13, 2), (12, 2), (11, 2), (10, 2), (9, 2), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8), (8, 9), (7, 9), (7, 10), (7, 11), (6, 11), (5, 11), (5, 12), (5, 13), (5, 14), (5, 15), (5, 16), (5, 17), (5, 18), (5, 19), (5, 20), (5, 21), (5, 22), (5, 23) Cost: 30 Nodes: 76

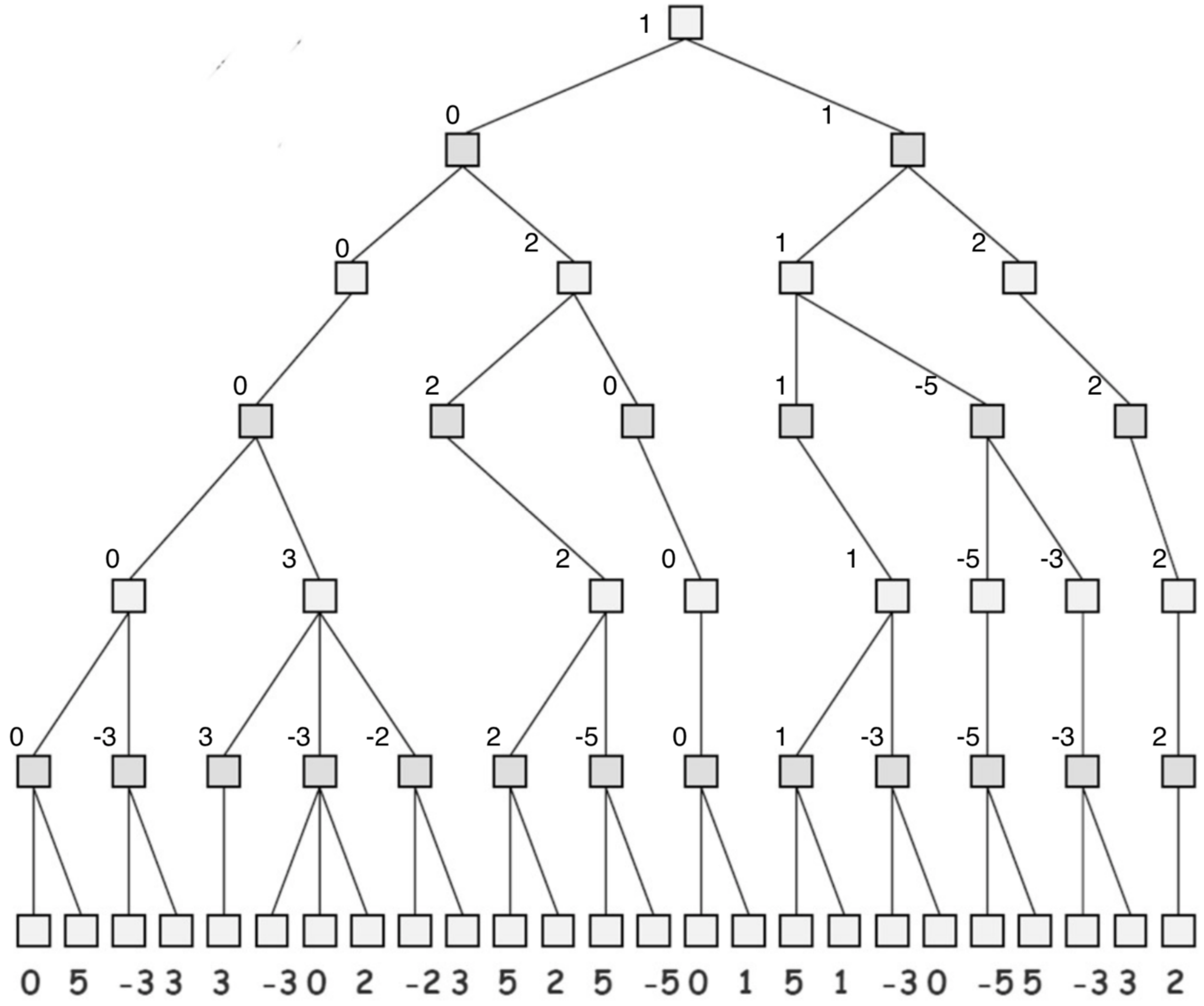
Starting at S and ending at E2
Path: (13, 2), (12, 2), (11, 2), (10, 2), (9, 2), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (7, 7), (6, 7), (6, 6), (6, 5), (6, 4), (6, 3), (6, 2), (5, 2), (4, 2), (3, 2) Cost: 21 Nodes: 81

Starting at (0, 0) and ending at (24, 24)
Path: (24, 0), (23, 0), (22, 0), (21, 0), (21, 1), (21, 2), (21, 3), (20, 3), (19, 3), (18, 3), (18, 4), (18, 5), (17, 5), (16, 5), (15, 5), (14, 5), (13, 5), (12, 5), (11, 5), (10, 5), (9, 5), (8, 5), (8, 6), (8, 7), (8, 8), (8, 9), (7, 9), (7, 10), (7, 11), (6, 11), (5, 11), (4, 11), (3, 11), (2, 11), (1, 11), (0, 11), (0, 12), (0, 13), (0, 14), (0, 15), (0, 16), (1, 16), (1, 17), (1, 18), (1, 19), (0, 19), (0, 20), (0, 21), (0, 22), (0, 23), (0, 24) Cost: 51 Nodes: 139

Assignment 2

Question 3

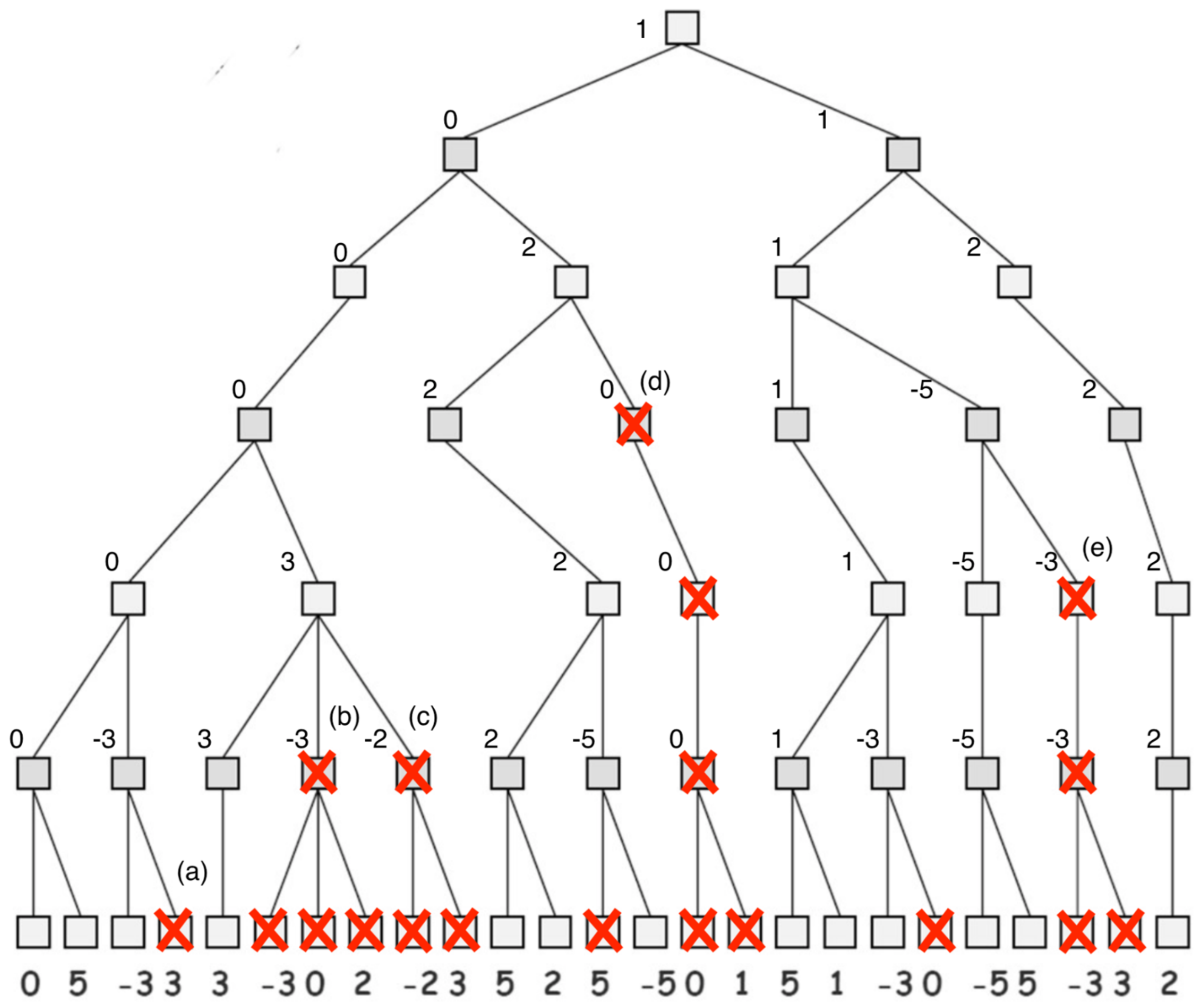
(i) Backed-Up Values by Minimax Algorithm



Since the root node corresponds to the maximizing player, then all light-coloured nodes represent max levels, whereas all dark-coloured nodes represent min levels. The strategy is to back up the highest value nodes for maximum levels and lowest value nodes for minimum levels.

Assignment 2

(ii) Pruning by Alpha-Beta Pruning Algorithm



(a) $\alpha = 0, \beta = -3$

Since $\alpha \geq \beta$ and on max level, can prune 3 since α will take 0 no matter the value

(b) $\beta = 0, \alpha = 3$

Since $\beta \leq \alpha$ and on min level, can prune -3 and its children since β will always be 0

(c) $\beta = 0, \alpha = 3$

Since $\beta \leq \alpha$ and on min level, can prune -2 and its children since β will always be 0

(d) $\beta = 0, \alpha = 2$

Since $\beta \leq \alpha$ and on min level, can prune 0 and its children since β will always be 0

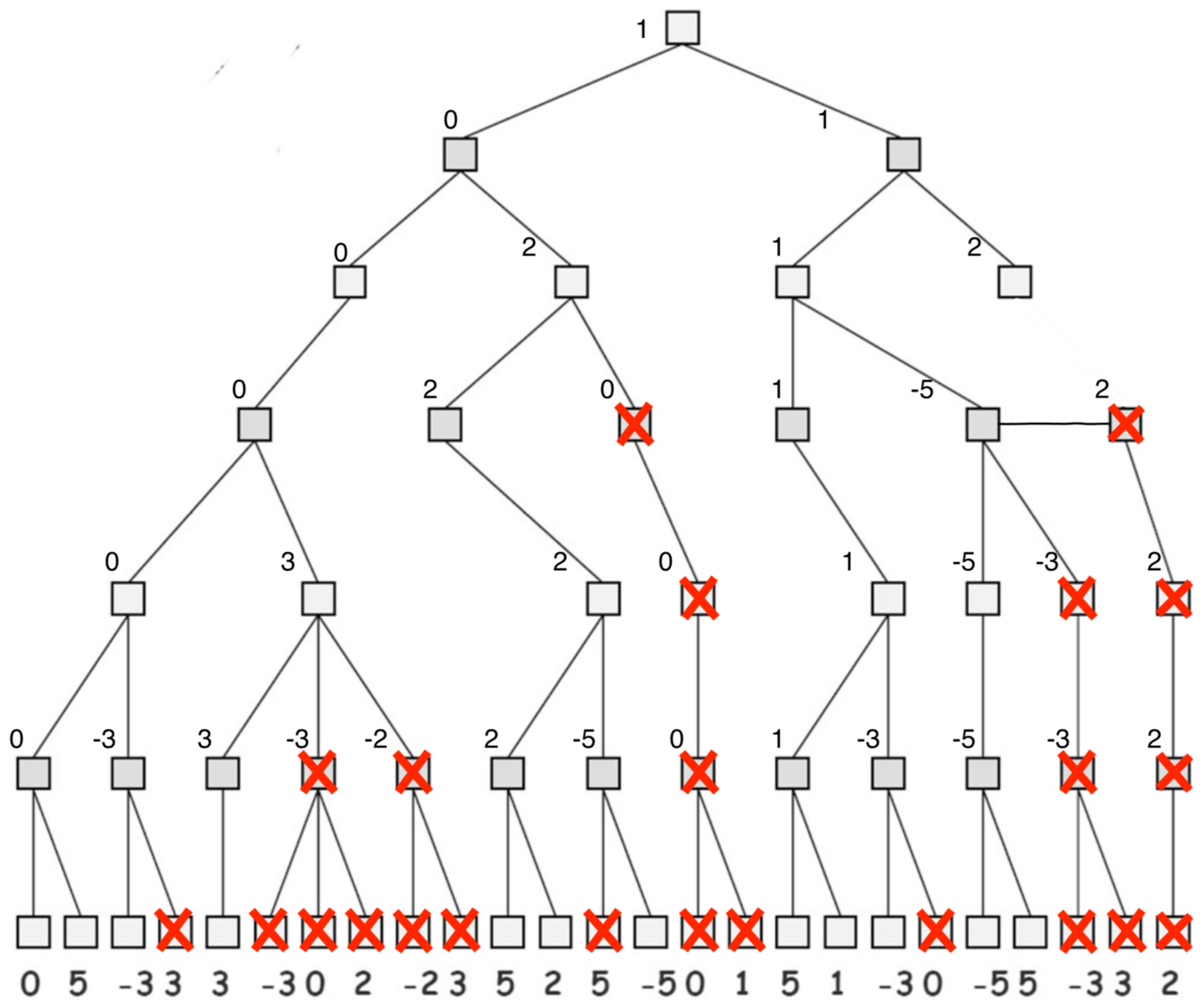
(e) $\alpha = 1, \beta = -5$

Since $\alpha \geq \beta$ and on max level, can prune -3 and its children since α will always be 1

Paolo Torres

Assignment 2

(iii) Tree Restructuring to Increase Pruned Branches



The tree has been restructured by connecting the branch with the 2's, far on the right, at the second min level, to the -5 node. This increases the number of pruned branches from 19 to 23. In this case, $\alpha = 1$ at the second max level and $\beta = -5$ at the second min level, so this newly added 2 node, as well as all its associated children, can be pruned. This is because it does not matter what its value is, as α will always be taking the 1.

Paolo Torres

Assignment 2

Question 4

(a) Problem Formulation, Neighbourhood, and Cost

Problem Formulation:

- States: The current values of x_1 and x_2 in the range $[-100, 100]$.
- Initial State: The values of x_1 and x_2 take some random initial value in the range $[-100, 100]$, known as the current solution $s = s_0$.
- Goal State: The values of x_1 and x_2 converge to some values near π, where the Easom function has a global minimum of -1 at $x = (\pi, \pi)^{\boldsymbol{T}}$.
- Actions:
    - Use simulated annealing (described below) to reach the goal state
    - Select a new solution s_i randomly from the neighbourhood $N(s)$
    - Accept solutions that are better, where $\Delta c < 0$, meaning it is improving: $s = s_i$
    - If $\Delta c \geq 0$, generate a random number x in the range $(0, 1)$, and if $x < e^{-\Delta c/t}$, then accept the new solution: $s = s_i$
    - Look for new solutions in neighbourhood for a max no. of iterations for the temp
    - Decrease temperature with α to decrease probability of accepting worse solutions
- Goal Test: The final solution (x_1, x_2), when substituted into the Easom function, minimizes the output, meaning (x_1, x_2) is as close as possible to (π, π), and the output is as close as possible to the global optimum of -1.
- Cost: Each new solution found has a cost of 1, where it increases the number of iterations for the temperature by 1 and gets 1 unit closer to a decrease of the temperature.

Neighbourhood Function: The following neighbourhood function, $N(s)$, is used: Once the maximum number of iterations are reached for a certain temperature, before the next temperature is evaluated, the search space is scaled linearly by α towards the solution, to guide the neighbourhood in the right direction and help the algorithm converge to final values.

Cost Function: The cost function, c and Δc, used is a subtraction between the current Easom output with the best Easom output, and if the result is negative (improving), or satisfies the $x < e^{-\Delta c/t}$ if positive, then this new solution becomes the best solution.

Assignment 2

(b) Simulated Annealing Results

The performance of the simulated annealing was evaluated based on random selections of 10 different initial points, combined with 10 different initial temperatures, and with 9 different annealing schedules. The initial points chosen were in the range $[-100, 100]$, while the initial temperatures within $[1000, 10000]$, and the alpha values between $[0.1, 0.9]$. This temperature range was chosen to be high enough to allow a move to possibly any state in the search space, but also not too high such that the simulated annealing would behave as a random search for a number of iterations. The alpha range was chosen to reasonably reduce the temperature to reach a solution, but at the same time, iterate enough times such that better solutions are found. The 9 different annealing schedules consist of 3 linear, 3 exponential, and 3 slow, with results below:

Initial x_1: -78.9476, Initial x_2: -72.286, Initial Temp: 9481.16, Alpha: 0.174792, Schedule: 0 Final x_1: 3.16513, Final x_2: 3.13615, $f(x_1, x_2)$: -0.999125
Initial x_1: -67.7296, Initial x_2: 69.1835, Initial Temp: 4005.21, Alpha: 0.151423, Schedule: 1 Final x_1: 3.14919, Final x_2: 3.13425, $f(x_1, x_2)$: -0.999833
Initial x_1: -62.0198, Initial x_2: 33.0069, Initial Temp: 3099.1, Alpha: 0.866287, Schedule: 2 Final x_1: 3.59248, Final x_2: 3.20922, $f(x_1, x_2)$: -0.729456
Initial x_1: 28.6863, Initial x_2: -69.8211, Initial Temp: 1737.84, Alpha: 0.794804, Schedule: 3 Final x_1: 3.55682, Final x_2: 2.70171, $f(x_1, x_2)$: -0.574218
Initial x_1: -34.5205, Initial x_2: 14.6698, Initial Temp: 3464.11, Alpha: 0.572197, Schedule: 4 Final x_1: 2.93512, Final x_2: 2.76133, $f(x_1, x_2)$: -0.753661
Initial x_1: -87.9205, Initial x_2: -79.9336, Initial Temp: 3510, Alpha: 0.327247, Schedule: 5 Final x_1: 3.38523, Final x_2: 3.43785, $f(x_1, x_2)$: -0.801202
Initial x_1: 47.2179, Initial x_2: -9.19138, Initial Temp: 1923.25, Alpha: 0.189572, Schedule: 6 Final x_1: 3.31081, Final x_2: 3.18791, $f(x_1, x_2)$: -0.954815
Initial x_1: 29.526, Initial x_2: 43.4346, Initial Temp: 5710.65, Alpha: 0.801723, Schedule: 7

Final x_1: 3.04701, Final x_2: 3.10042, $f(x_1, x_2)$: -0.984157
Initial x_1: -84.3189, Initial x2: 52.8392, Initial Temp: 8610.32, Alpha: 0.782635, Schedule: 8 Final x_1: 2.62725, Final x_2: 3.54392, $f(x_1, x_2)$: -0.522993
Initial x_1: 38.3917, Initial x2: 50.1344, Initial Temp: 5888.72, Alpha: 0.423727, Schedule: 9 Final x_1: 3.39002, Final x_2: 3.38999, $f(x_1, x_2)$: -0.830467

The schedule numbers correspond to the different annealing schedules, where 0-3 are increasingly scaled linear schedules, 4-6 exponential, and 7-9 slow.

(c) Simulated Annealing Evaluation

The table shown above depicts great results for the use of the simulated annealing algorithm to minimize the Easom function. It appears each trial successfully converged to the vicinity of the solution, with the range of answers being $[-0.522993, -0.999833]$. The best solution found was virtually a tie between the first 2 solutions, with their x values being $(3.16513, 3.13615)$ and $(3.14919, 3.13425)$, corresponding to Easom values of -0.999125 and -0.999833, respectively. Analyzing their parameters, it appears these best solutions have high initial temperature values and low alpha values. This contributes to its optimality, as higher temperatures give the algorithm the ability to readily escape local optimums, while low alpha values allow the temperature to more slowly and steadily decrease when finding better solutions. All of the solutions had a maximum number of iterations of 100, to appropriately search the neighbourhood before moving on. Additionally, the annealing schedules of the best solutions were linear, which allows a steady decrease over time. The results show that when the decrementing of the temperature is increased, the Easom value reached is lower. This is seen in the schedule 2 and schedule 3 results, which have higher linear schedule multipliers, resulting in lower final values of -0.729456 and -0.574218. The exponential and slow annealing schedules were also able to reach to reasonable solutions, but not as good as the best ones. However, the slow schedule trials had relatively high alpha values, and decreasing them results in more optimal values. The 2 lowest values, -0.522993 and -0.574218, have alpha values of 0.782635 and 0.794804, which convey a slight trend that increasing alpha values result in lower Easom values. Lastly, the initial x values do not appear to affect the algorithm, as they all converged to a solution despite being randomly chosen between the range of $[-100, 100]$.