CIE 458

Artificial Intelligence

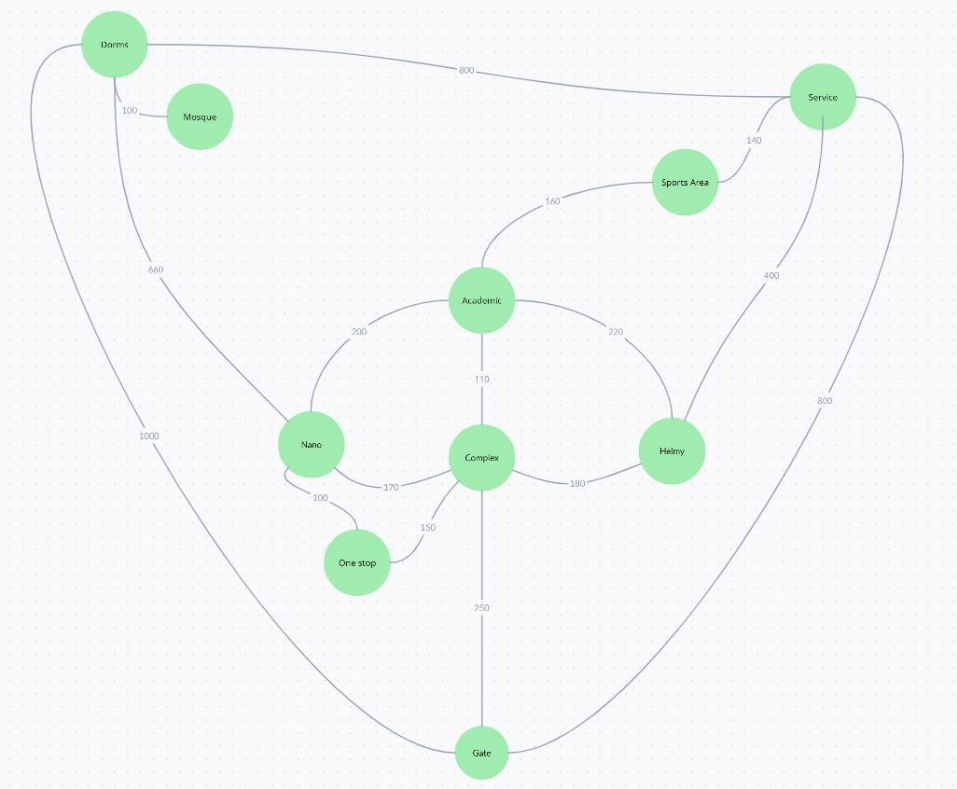
Project 1 Final Report

Abdullah Kamal Muhammad 201801271
Mohamed Nasr 201801675
Amr Abdelaziz 201801041

Introduction

Programming is generally proposed to solve problems in an appropriate way. Artificial intelligence is about solving this problem with generic searching algorithms each having its consequences of actions. Each algorithm is trying to reach a goal state starting from the initial state. In this project, we aim to model the problem of finding the optimal way to get from one place on Zewail City campus to another place as a search problem.

Design



Zewail City Campus

Work Distribution

Abdullah

- Complete ZC map
- BFS
- Greedy search
- ZC_problem Class: findpath, calculate_cost, and solution
- find_path_steps

Amr

- Designing the code from scratch
- DFS
- ZC_problem Class: Actions, results, get_child, find, and found_room
- Hill climbing
- Simulated annealing

Nasr

- Complete ZC map
- IDS
- A_star
- ZC_problem Class: generate_h, heuristic, and goal_test
- Documentation

Comparison

1)BFS

Breadth-first search is an uniformed algorithm that searches for the goal without any previous knowledge of its location. The algorithm starts from the initial node as its root and starts to expand its children nodes until it reaches the goal. It uses levels to name the children, because it expands all children from the same level before moving to the next level. It tests each node to be the goal when it is expanded before moving to the next level. it processes all nodes above the shallowest solution whose depth is **‘s’**. It takes time **O(bs).**

- If a solution exists, it is a complete solution.
- It is optimal only when costs are all 1.

Test case: we set our location **Dorms** and the destination **Gate A:**

```
[52]
      init_state = 'Dorms'
      goal_state = "Gate A"
      test = ZC_problem(init_state, goal_state)
```

```
[53] bfs_graph(test, verbose=True)
```

and it returns the following output

```
the current state is: Dorms
the current state is: DO_NA
the current state is: DO_SE
the current state is: Nano
the current state is: road2
the current state is: service
the current state is: NA_ON
the current state is: Academic
the current state is: road3
the current state is: OneStopShop
the current state is: Complex
the current state is: Helmy
the current state is: road9
the current state is: Gate A
```

```
path_actions(stations_indecies)
```

```
['left',
 'down',
 'right',
 'up',
 'left',
 'down',
 'right',
 'up',
 'right',
 'up',
 'left',
 'down',
 'right',
 'up',
 'right',
 'up',
 'left',
 'down',
 'right',
 'up',
 'right',
 'up',
 'left',
 'down',
 'left',
 'up']
```

Comment: So we find that it expands all node children in the same level and does not move to the next level unless it did not find the goal.

2) DFS

Depth first search is **complete** but **not optimal.**In our case it reaches the destination but it doesn't return the optimal path between the two points.
Test case: we set our location **Dorms** and the destination **Gate A,** and it returns the following output

```
init_state = 'Dorms'
goal_state = "Gate A"

first_try = ZC_problem(init_state,goal_state)
%prun DFS(first_try)
```

```
now we are at Dorms
now we are at DO_SE
now we are at service
now we are at road3
now we are at Helmy
now we are at Academic
now we are at Nano
now we are at NA_ON
now we are at OneStopShop
now we are at GA_CO
Now we reached  Gate A
```

Runtime : 0.008 seconds

You can refer to the ZC map to trace the path. The solution navigates through unnecessary nodes. Thus, giving a longer than needed path **(Not optimal)**

3) IDS

Iterative Deepening-Depth-First Search is a **complete** search. It is also an **optimal search only when step cost = 1** which is the situation in our case. Depth-limited-search couldn't find a solution if the solution is deeper than limit. Thus, IDS solves this problem by iterating on depth limit each time and calling DLS.

- Another test where we start from **room 207** inside the **Dorms** and our destination is also a room **security office** at **Gate A**
- **We note that it's optimal in our case and it is only 6 moves. It is the shortest path among BFS, DFS, and IDS.**

```
[183] init_state = 'room 207'
      goal_state = 'security office'

      first_try = ZC_problem(init_state,goal_state)
      %prun ids(first_try)
```

```
now we are at Dorms
now we are at DO_NA
now we are at Nano
now we are at NA_ON
now we are at OneStopShop
now we are at road9
Now we reached  Gate A
```

```
461 function calls in 0.001 seconds
```

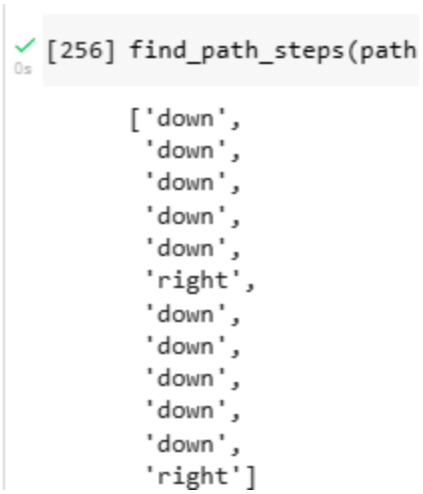
4) Greedy best-first

- Greedy algorithm is an informed algorithm that uses previous knowledge to guide the algorithm to the goal state.
- It always selects the path which appears beat at that moment
- It is a combination between depth-first search and breadth-first search algorithms
- It uses the heuristic function and search
- BFS helps it to choose the most promising node
- BFS can finds the closest node to the goal and the heuristic function finds the minimum estimated cost
- It is not complete.

```
3129 function calls in 0.014 seconds
```

```
init_state = 'room 207'
goal_state = 'security office'

first_try = ZC_problem(init_state,goal_state)
%prun greedy_best_first(first_try)
```

```
I am at        Dorms
hu      0 0
Next is   DO_NA
Next is   DO_SE
I am at        DO_NA
hu      1 0
Next is   Nano
Next is   road2
I am at        Nano
hu      2 0
Next is   NA_ON
Next is   Academic
I am at        NA_ON
hu      3 0
Next is   OneStopShop
Next is   Complex
I am at        OneStopShop
hu      4 0
Next is   road9
Next is   GA_CO
I am at        road9
hu      5 0
Next is   Gate A
I am at        Gate A
```



5) A* search

- It is an informed search that depends on both heuristic and total path cost.
- A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.
- It's optimal and complete algorithm.

```
1108 function calls in 0.002 seconds
```

```
[258] init_state = 'room 207'
      goal_state = 'security office'

      first_try = ZC_problem(init_state,goal_state)
      %prun A_star(first_try)
```

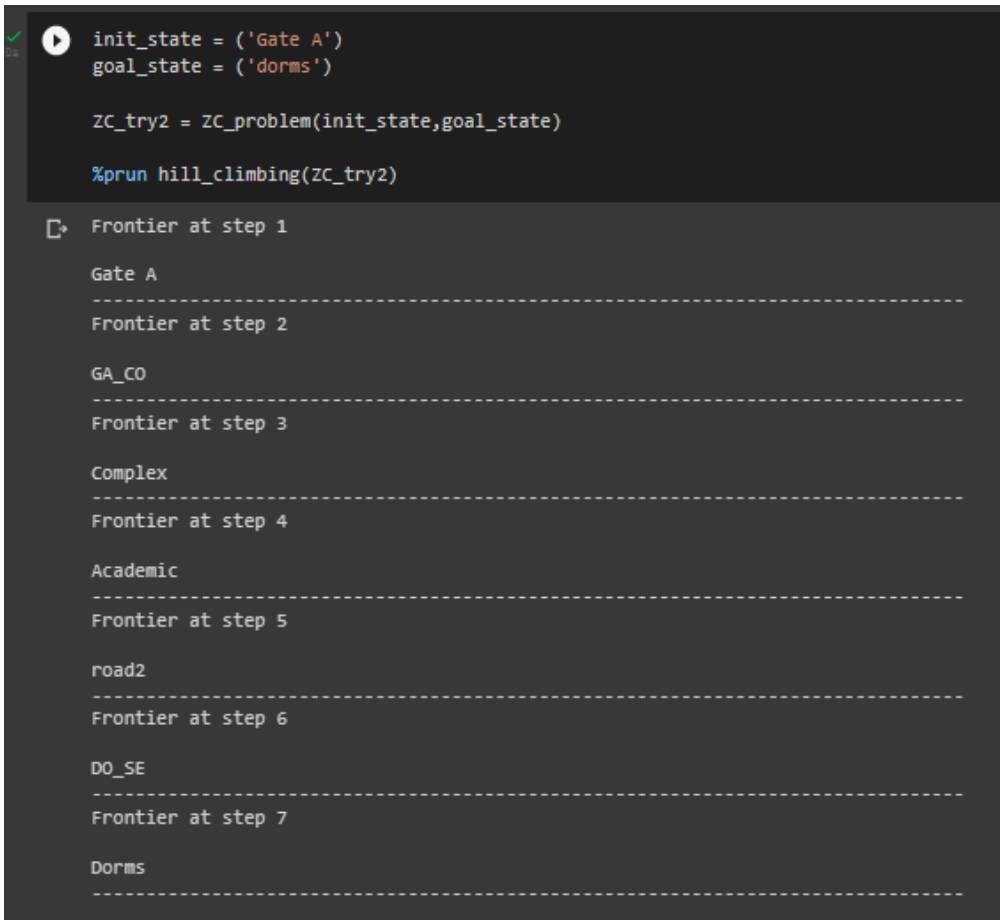
```
I am at        Dorms
Next is    DO_NA
Next is    DO_SE
I am at        DO_NA
Next is    Nano
Next is    road2
I am at        Nano
Next is    NA_ON
Next is    Academic
I am at        NA_ON
Next is    OneStopShop
Next is    Complex
I am at        OneStopShop
Next is    road9
Next is    GA_CO
I am at        road9
Next is    Gate A
I am at        Gate A
```

```
[231] find_path_steps(path2)
```

```
['down',
 'down',
 'down',
 'down',
 'down',
 'right',
 'down',
 'down',
 'down',
 'down',
 'down',
 'right']
```

Hill Climbing

We implemented the Hill Climbing algorithm to run on the ZC_probelm.
As a test, we set our location (initial state) as “Gate A” and our destination (goal state) is “Dorms”. These points are the furthest two points on the map.
Here the test results showing the path taken to reach the destination:



```
init_state = ('Gate A')
goal_state = ('dorms')

ZC_try2 = ZC_problem(init_state,goal_state)

%prun hill_climbing(ZC_try2)
```

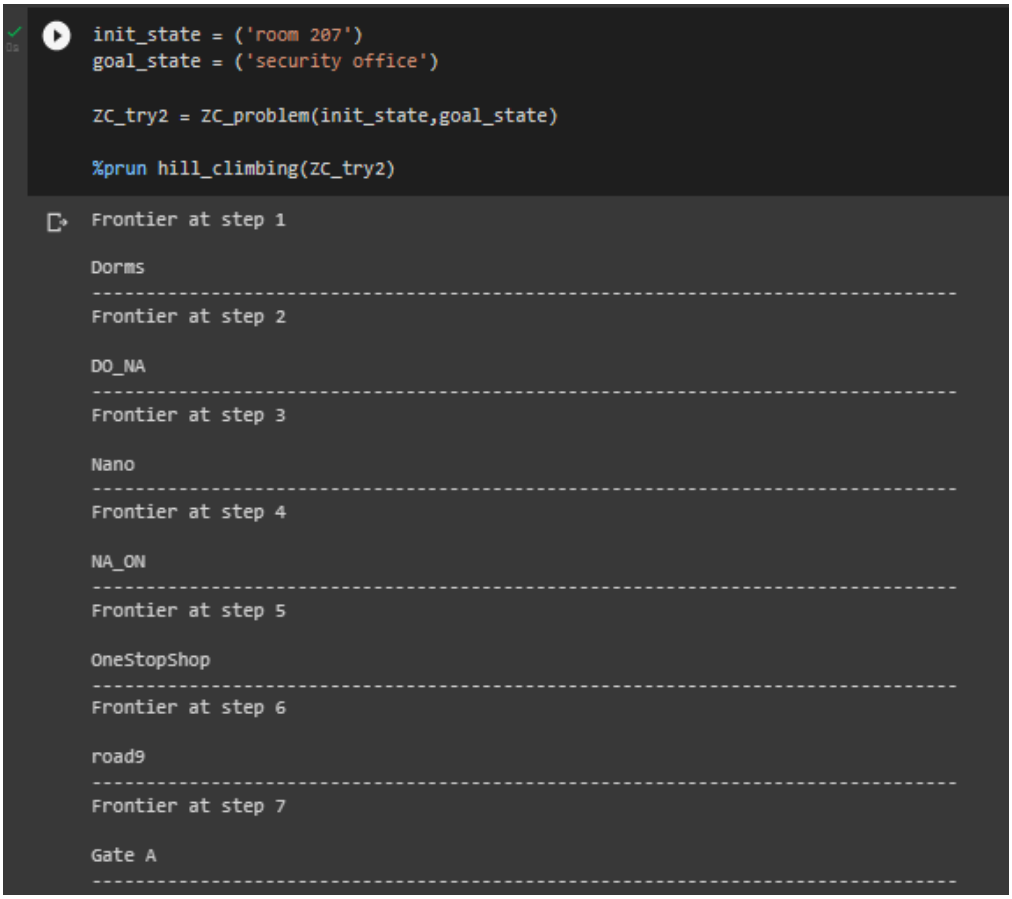
```
Frontier at step 1

Gate A
---------------------------------------------------------------------------
Frontier at step 2

GA_CO
---------------------------------------------------------------------------
Frontier at step 3

Complex
---------------------------------------------------------------------------
Frontier at step 4

Academic
---------------------------------------------------------------------------
Frontier at step 5

road2
---------------------------------------------------------------------------
Frontier at step 6

DO_SE
---------------------------------------------------------------------------
Frontier at step 7

Dorms
---------------------------------------------------------------------------
```

```
823 function calls in 0.004 seconds
```

As we can see, the algorithm successfully reached the destination. It didn’t get stuck because our heuristic (Euclidean distance) values are different.

- Another test where we start from **room 207** inside the **Dorms** and our destination

- is also a room **security office** at **Gate A**



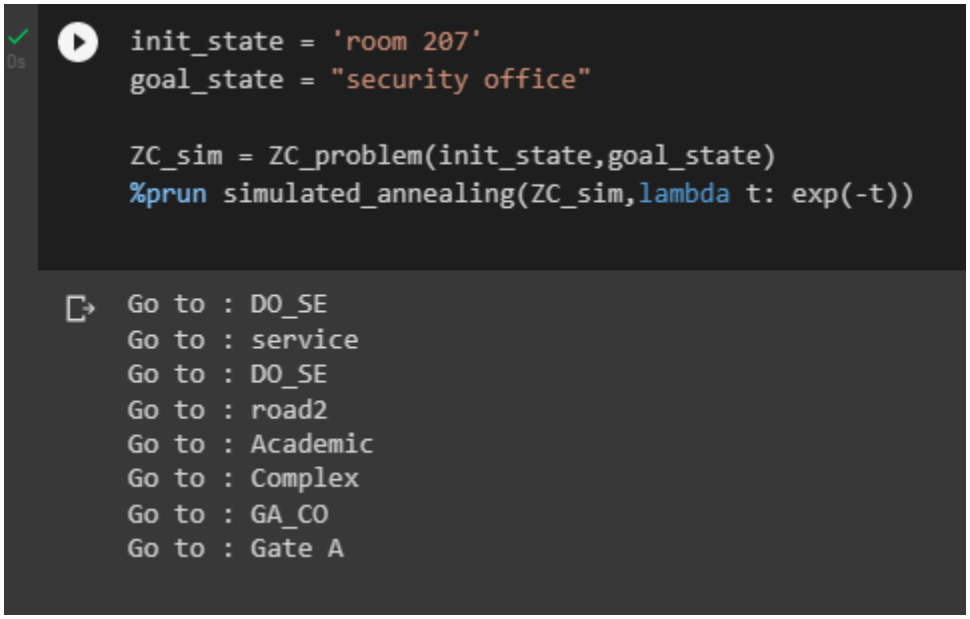
```
init_state = ('room 207')
goal_state = ('security office')

ZC_try2 = ZC_problem(init_state,goal_state)

%prun hill_climbing(ZC_try2)
```

```
Frontier at step 1

Dorms
--------------------------------------------------------------------------------
Frontier at step 2

DO_NA
--------------------------------------------------------------------------------
Frontier at step 3

Nano
--------------------------------------------------------------------------------
Frontier at step 4

NA_ON
--------------------------------------------------------------------------------
Frontier at step 5

OneStopShop
--------------------------------------------------------------------------------
Frontier at step 6

road9
--------------------------------------------------------------------------------
Frontier at step 7

Gate A
--------------------------------------------------------------------------------
```

Hill Climb is generally **incomplete**, it can get stuck easily, but in our case that didn't happen.

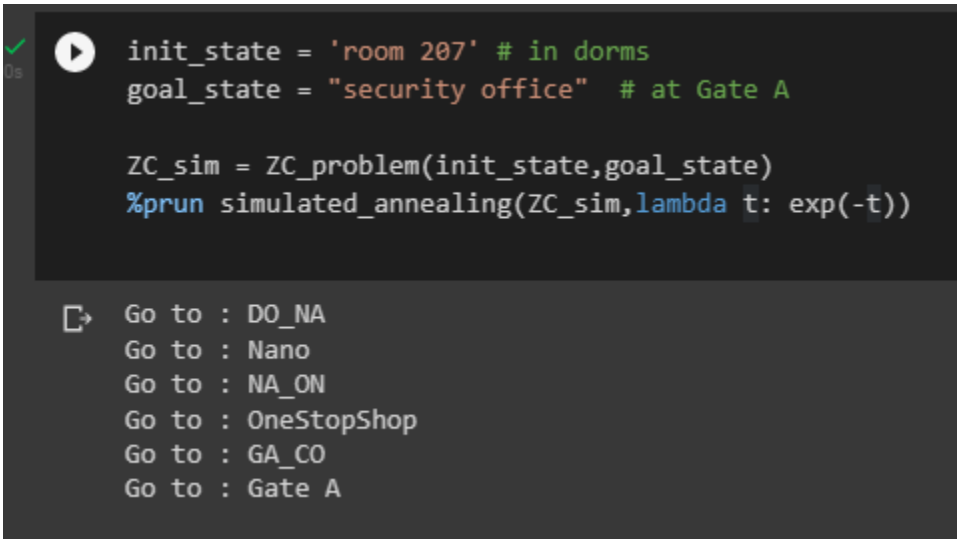
Simulated Annealing

We also implemented the Simulated Annealing algorithm. Here's the results of the same test case we used previously



Runtime : 0.004 seconds

As we can see, the algorithm reached the destination but the path wasn't **optimal**, we moved to **Service** building and moved back to the same road **DO_SE.** This is because of the **random factor**. Also, running the same test case with the same destination and location would yield different results for the same reason.



Same Test Case

Therefore, Simulated annealing isn't optimal but it would usually produce a good solution.