



American University of Armenia

Zaven and Sonia Akian College of Science and Engineering

PROJECT TITLE

CS246 / CS346 Project Report

Levon Gevorgyan, Silva Vardanyan, Anzhelika Simonyan

Instructor: *Vahram Tadevosyan*

Fall 2025

Abstract

This project illustrates a real-life navigation problem in a university that many students face, mostly at the beginning of their first year at the university: being lost on campus. The problem becomes a big headache when you also have limited time to reach the professor's office throughout the semester before the exam period, to ask your remaining questions during the office hour, and be fully prepared for the exam!

The *Office Hour Race* is a real-life project simulated in the *American University of Armenia (AUA)* environment, solved with *Artificial Intelligence* algorithms for three students, in our terms: agents, who want to find the professor's room as quickly as possible from the *Main Building* to *PAB*.

Key words: *Artificial Intelligence, Multi-agent systems, Partial observability, Stochastic environment, Decision-making under uncertainty, Search algorithms, Simulation*

Contents

<i>Abstract</i>	i	
1	Introduction	2
2	Literature Review	5
3	Method	7
3.1	<i>Baseline</i>	7
3.2	<i>Medium Extensions</i>	8
3.2.1	<i>Hill Climbing</i>	8
3.2.2	<i>Stochastic Hill Climbing</i>	8
3.2.3	<i>Simulated Annealing</i>	8
3.3	<i>Full Extensions</i>	9
3.3.1	<i>Hill Climbing</i>	9
3.3.2	<i>Stochastic Hill Climbing</i>	9
3.3.3	<i>Simulated Annealing</i>	9
4	Experiments	11
5	Conclusion	15

Chapter 1

Introduction

The environment of the project consists of 3 main components of the AUA campus: *the Main Building, the Bridge, and the PAB*. As in real life, the bridge connects the 4th floor of the Main Building with the 1st floor of PAB, as shown in Figure 1.0.1. The rows of Main and PAB grids are the floors.

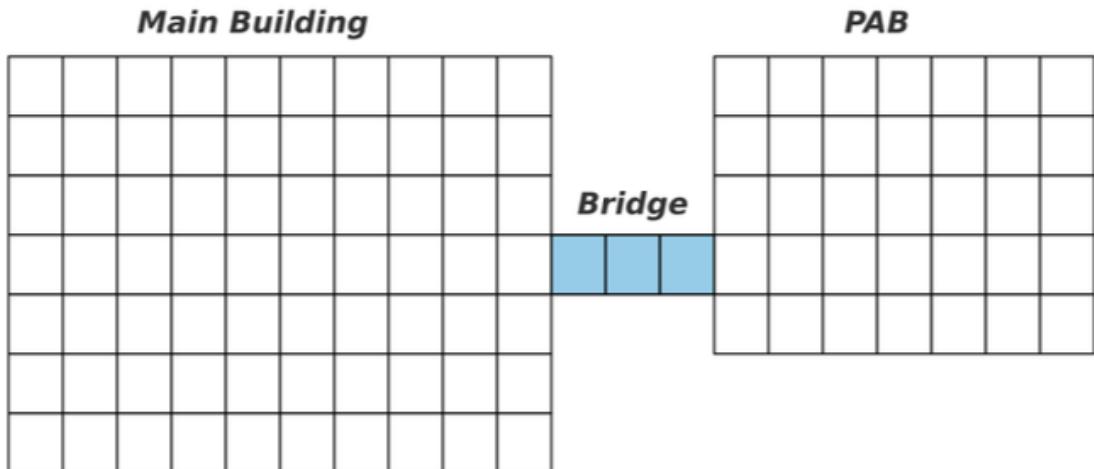


Figure 1.0.1: Episode length mean

The Bridge is fixed with a height of 1 and a width of 3. The heights of Main and PAB are fixed: 7 and 5, respectively. The widths of Main and PAB can be adjusted in each run.

The game also has obstacles: *walls* and *angry professors' rooms* in each building. The sizes are again adjusted according to the width of the grid.

$$\text{MaxCount}_{\text{walls}} = \text{GridWidth} - \text{Count}(\text{angry})$$

$$\text{MaxCount}_{\text{angry}} = \text{GridWidth} - \text{Count}(\text{walls})$$

Important note: during the creation of each random environment walls, and angry rooms are not allowed to be in front of the bridge entrances. The *Professor OH* room—agents' goal state—is always in PAB. There is also a *Program Chair* room in PAB. The points for the different rooms will be discussed later, after presenting our agents.

There are 3 agents, starting at the same random cell of the grid in each run. The agents are trying to get to Professor OH's room, which is their goal. Every agent's algorithm is different from the others. Every agent can move (UP, DOWN, LEFT, or RIGHT). There can be many agents in the same room at the same time. All three agents have access to the adjacent cells of their current position and do not know the entire grid. Moves outside the grid of that run time are not allowed. A full state of an agent includes: *agent position* (x, y, and which grid: Main / Bridge / Pab), *current time*, and *score values*.

In the beginning, all agents have 1000 points and 300 seconds. Points decrease according to the agents' actions, and time decreases automatically after the race starts.

Agents lose points in the following cases:

- Entering an *angry professor's room* (-150 points).
- Entering the *Program Chair's room* — this truncates the episode (all points = 0) and the game terminates for that agent.
- Performing any action (-1 point).

The entire game terminates for an agent when one of the following conditions is met:

- The agent reaches *Professor OH's room*.
- The agent enters the *Program Chair's room*.
- The *agent's points reach 0*.
- The *agent runs out of time* (time = 0).

At the end of the game, we evaluate each agent's final score and the remaining time, making conclusions about:

- the *most optimal* agent (the algorithm with the maximum score),
- the *most efficient* agent (the algorithm with the maximum remaining time).

Overall, our environment is:

- *partially observable*: each agent perceives only the adjacent cells of its current location,
- *deterministic*: each action results in a well-defined next state,
- *sequential*: the next state depends on previous actions, lost points, and elapsed time,
- *discrete*: both the action space and state space are discrete,
- *known*: all agents know about the existence of all dynamics given to the environment before starting the game,
- *static*: the environment itself doesn't change; the agents cannot reform it or make it better/worse for any other agents.
- *competitive multi-agent*.

Chapter 2

Literature Review

There are several known algorithms that could be used to solve grid-based problems; however, there are constraints in our problem that allow us to eliminate several algorithms from the beginning. A group of *uninformed search algorithms*, such as *Breadth-First Search (BFS)*, *Depth-First Search (DFS)*, *UCS*, or *Iterative Deepening Search*, are unsuitable for our problem because our agents do not have knowledge of the entire map, and the successor states are unknown. So, we are unable to construct a fully known tree of states, which is a required parameter of these types of algorithms.

The next group of algorithms is *informed search algorithms*—*A**, *Greedy*—which also requires a full map to construct the trees. Besides that, here we also need to estimate distances of the agent's current positions to the goal, which we can't compute, as we initially don't know the location of the goal.

Choosing to formulate the problem as a *Constraint Satisfaction Problem (CSP)* would not be feasible, as the set of variables would take an unpredictable size (considering that each cell in the grid becomes a variable, and the player can choose any grid width between 0–100, resulting in 7×100 variables for the Main Building alone). This makes the formulation computationally hard, and we still cannot fully describe the environment using constraints. Several constraints would simply state things like “do not enter a wall cell” or “do not enter an angry professor’s room,” which capture only invalid states but do not provide a complete formulation or a path to a solution.

Using *Propositional* or *First-Order Logic* would also be computationally expensive. Our world consists of two buildings, penalties, and a timing component, which would dramatically increase the required logical inference. Even representing that an agent can occupy exactly one cell at a single time step requires encoding $(7 \times N + 5 \times n + 3)$ sentences, where $7 \times N$ corresponds to all Main Building cells, $5 \times n$ corresponds to PAB cells, and 3 corresponds to the Bridge cells.

Based on the structure of the environment, *local search algorithms*—*Hill Climbing*, *Stochastic Hill Climbing*, *Simulated Annealing*—are the best choice to be used for our game. It makes decisions step-by-step using the current local information. Our research showed that many people use local search for our kind of maze-like environments, where the goal is to get from A to B without having knowledge of the environment.

The article “*Climbing the Mazes: A cognitive model of spatial planning*” [2] is very similar to our project baseline. Their interactive environment *PathWorld* generates a 22x22 maze-like grid. There are randomly drawn obstacles, like our walls and angry professors. The start and end positions are also in the grid. They conducted 5 mazes for every 12 experimental conditions, with other parameters, and kept the records for each maze for the analysis. After their experiment, they support the hypothesis that *incremental hill-climbing minimizes the number of steps used to achieve the target in the maze-like environments*.

The *Travelling Tournament Problem* [1] is different at first glance. It has a scheduling task, such as finding the best schedule for teams, rather than solving a navigation problem. However, it has the same underlying conditions to find the optimal solution for the problem. In both problems, the agent has to explore many possible states and improve them step by step using local information in a huge state space. The TTP article shows how simulated annealing and hill climbing can be useful for searching in a huge space while avoiding local optima, which is very relevant for our problem, when *agents would need to avoid poor states, making local improvements*.

After the thorough research, we have decided to use local search algorithms as the best solution for our agents to move in a partially observable environment while gradually improving the states step-by-step.

Chapter 3

Method

As stated in Chapter 2, the main algorithms we worked with belong to the family of local search algorithms. These are *hill climbing*, *stochastic hill climbing*, and *simulated annealing*. The decision to use these three was motivated by the following reasoning: hill climbing is the simplest and most limited algorithm, giving us an opportunity to improve and extend it; stochastic hill climbing introduces randomness into the search; and simulated annealing allows us to explore the effect of controlled exploration versus exploitation. In particular, simulated annealing helps us understand whether the exploration–exploitation trade-off plays a meaningful role in a problem such as ours.

To conduct a proper analysis and evaluate which algorithm performs best, we decided to perform three batches of experiments, each with different variations of our algorithms, to observe and compare their behavior.

3.1 *Baseline*

We first ran all algorithms in their original, unmodified form. The motivation behind this was simple: if an algorithm performs well in its baseline form, then there is no need for additional modifications. However, in practice, the decision to test multiple variations was made even before running the experiments, as our goal was to see how far each algorithm could be improved. Even hill climbing, considered the “baseline” of the local search family, was expected to benefit from extensions. The results of these baseline experiments are presented in the next chapter.

3.2 Medium Extensions

After experimenting with the unmodified versions of the local search algorithms, we introduced several extensions to each of them.

3.2.1 Hill Climbing

- *random restarts* - We decided to add random restarts extension to hill climbing, not because it actually is from known-to-us family of local search methods, but to later be able to integrate this to stochastic hill climbing and compare in which random restarts made the biggest change.

3.2.2 Stochastic Hill Climbing

- *sideways moves* - The motivation for adding sideways moves is straightforward: to enable escape from plateaux where no strictly better neighboring state exists. We applied this extension specifically to stochastic hill climbing so that we could later integrate it into ordinary hill climbing and observe the differences in performance between the two variants.

3.2.3 Simulated Annealing

- *stagnation counter* - The stagnation counter depicts how many consecutive steps produced no improvement in the search. If the algorithm keeps accepting moves that lead to no progress, it will mean that the search is no longer exploring useful areas of the state space. So by stagnation counter, we can detect plateau-like conditions and force the algorithm to either jump to a new random state or decrease the temperature faster.
- *oscillation guard* - As is known, simulated annealing may occasionally bounce back and forth between a small set of states, and this wastes time and pushes the agent away from the solution. So when the oscillation is detected, the guard forces a deviation from the loop, pushing the algorithm to explore new directions instead of revisiting the same few states. Later on, this will also be used for hill-climbing.

3.3 Full Extensions

And finally, after all previous trainings with different extensions, we have concluded all final additions, and our training process —simulation and learning the algorithm’s patterns—got a better vision with this.

3.3.1 Hill Climbing

- *oscillation guard*
- *sideways moves*
- *two-step look-ahead* - Instead of looking only at its neighbor, by this custom extension, we make it possible for hill climbing to also look at its neighbors’ neighbors—so the second child (grandchild)—by this do no in advance if successors are good ones.
- *local escape moves* - This is somewhat similar to simulated annealing logic, as here we give our algorithm possibilities with a small deviation, sometimes going into a worse state in order to escape local maxima. Unlike random restarts, which bring the agent into a completely new position, local escape moves remain in the nearby region of the current state, giving the algorithm a chance to explore promising alternative paths.

3.3.2 Stochastic Hill Climbing

- *random restarts*
- *two-step look-ahead*
- *local escape moves*
- *oscillation guard*

3.3.3 Simulated Annealing

- *stagnation counter*
- *two-step look-ahead*
- *cooling schedule* - besides the other two implemented in the medium extensions, we refined the cooling schedule to control how quickly the temperature

decreases during the search. By adjusting the rate of cooling, we can influence how long the algorithm continues to explore suboptimal states before becoming more greedy, allowing a better balance between exploration and exploitation.

Chapter 4

Experiments

To analyze the behavior of all nine algorithms, we conducted an experiment recording 30 runs for each of the three groups (*Baseline*, *Medium Extensions*, *Full Extensions*) under 30 different random seeds, using the same seed for each corresponding run across all groups. The maze layout was kept constant throughout all runs. The CSV files contain information about the algorithm used, score and time remaining, termination reason, number of steps (for time complexity), and the (x, y) coordinates visited at each step.

We ran each of the nine algorithms 30 times because a single run is not sufficient to evaluate overall behavior. This is especially important for local search methods, which are highly sensitive to randomness, meaning that only a few runs cannot represent the true performance of the algorithm.

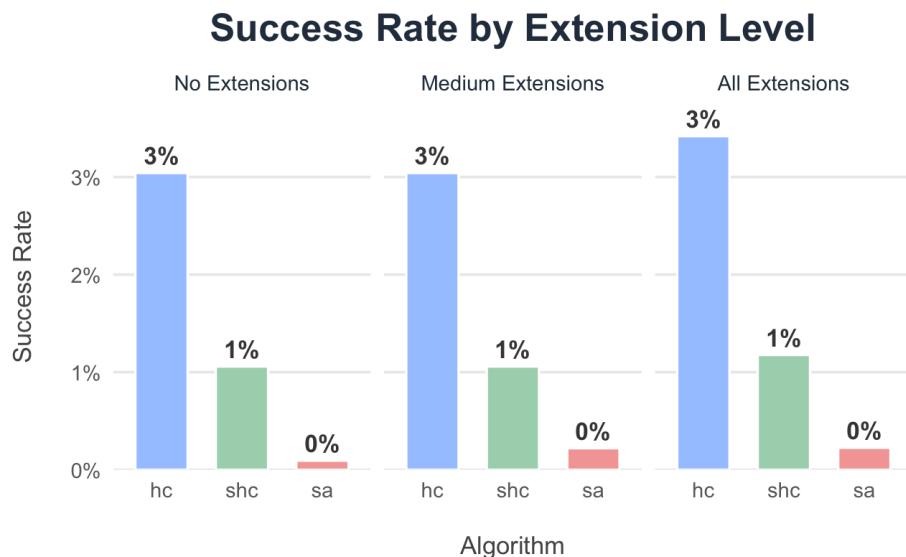


Figure 4.0.1: Success rate of each algorithm across all extension groups

The first metric of our analysis is the success rate of each algorithm, measured as the percentage of successful runs. Overall, Hill Climbing performs the best across all three extension groups. Its behavior aligns well with our environment, where wandering, backtracking, or random exploration are highly penalized. Hill Climbing always selects the best available move and avoids states that worsen the score. In contrast, other algorithms may waste time or score while moving sideways or backwards.

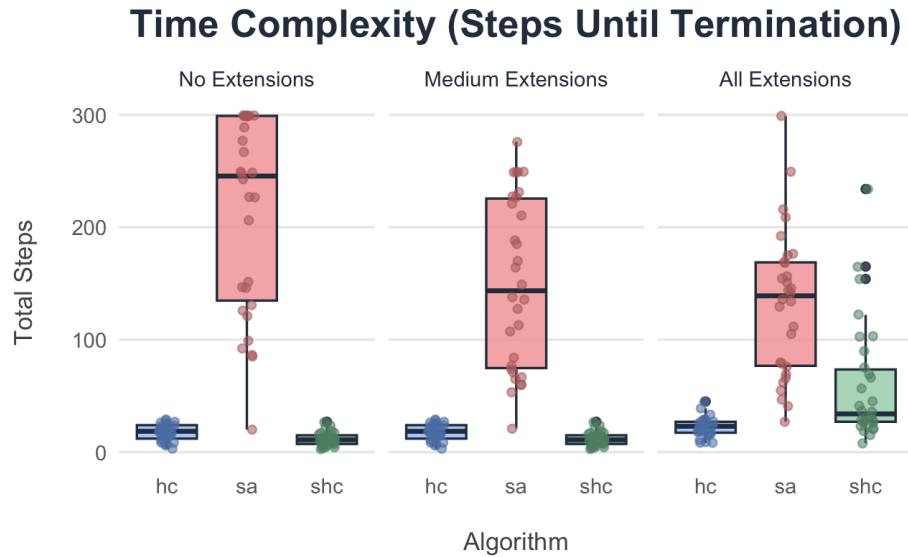


Figure 4.0.2: Number of termination steps for each algorithm

Here we get a more detailed picture of Hill Climbing, which typically terminates in fewer steps than other algorithms. Its greedy strategy ensures that it always chooses the best move and does not explore worse states. Interestingly, Stochastic Hill Climbing sometimes performs even better at the beginning; this can be attributed to its ability to choose randomly among the equally good moves.

Simulated Annealing, on the other hand, typically requires more steps to terminate, especially considering that its success rate is nearly zero. This behavior is expected, as Simulated Annealing occasionally accepts worse moves in hopes of escaping local optima.

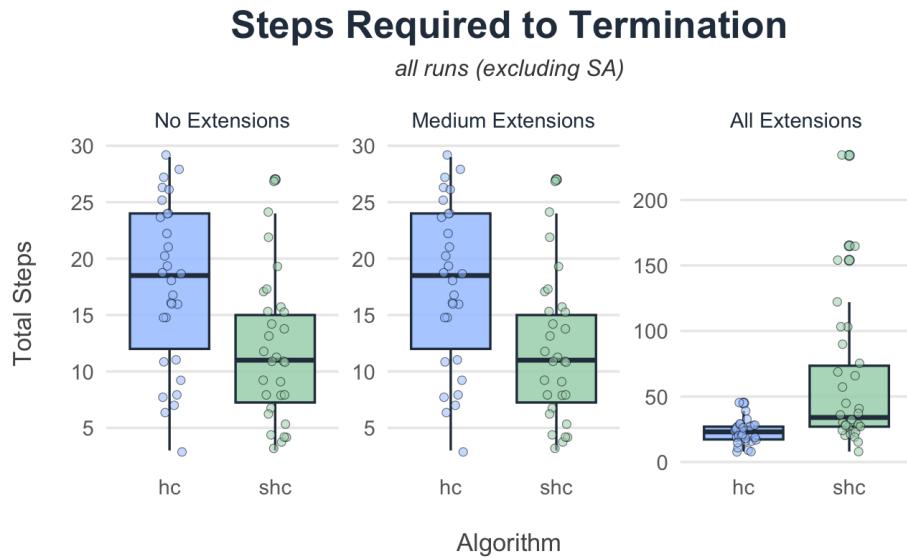


Figure 4.0.3: Termination steps without Simulated Annealing

Now let us examine the same graph without Simulated Annealing, allowing clearer comparison between Hill Climbing and Stochastic Hill Climbing. As expected, both algorithms take significantly more steps when all extensions are applied. This aligns with the slight improvement observed earlier in the success rate of Hill Climbing. Note that sideways moves were added to Hill Climbing.

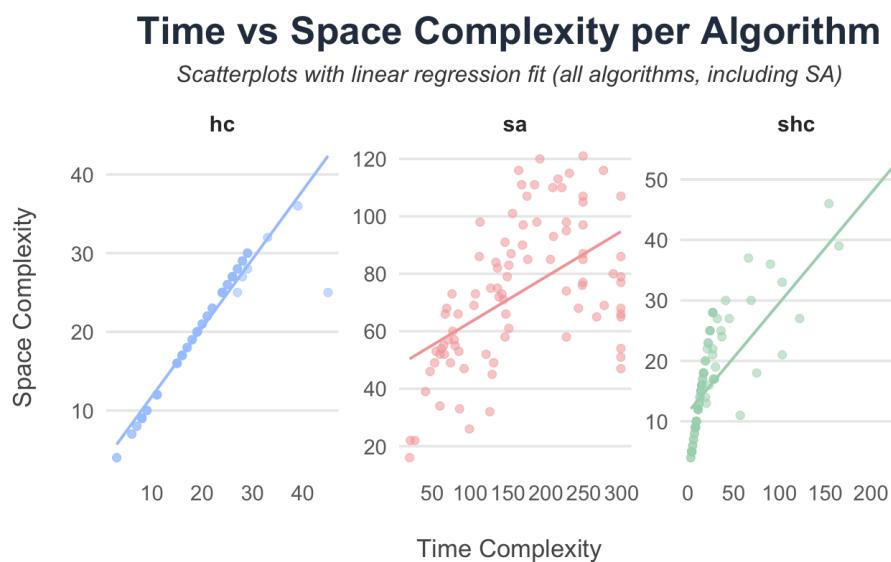


Figure 4.0.4: Time vs Space Complexities

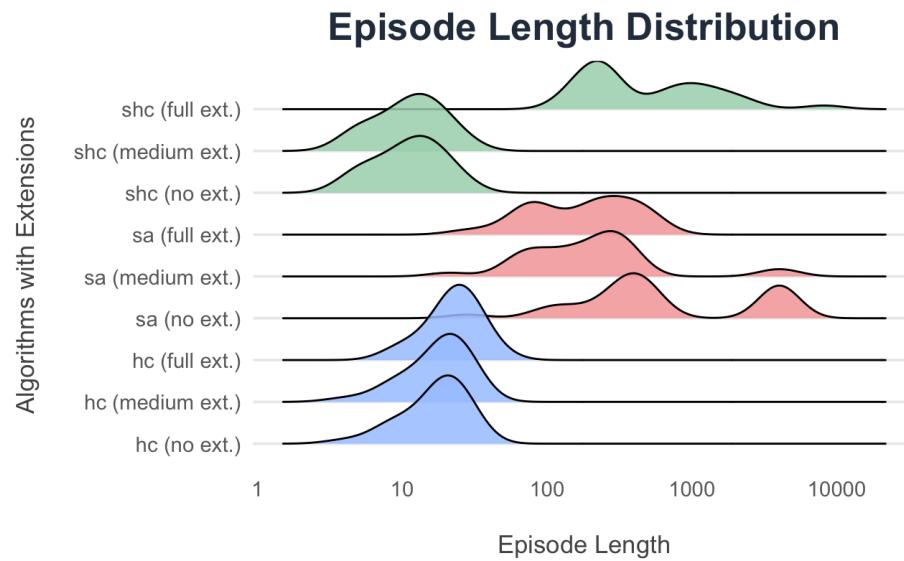


Figure 4.0.5: Episode Length Distribution

The episode length shows the time distribution, showing how much time each algorithm takes to terminate. Simulated Annealing and Stochastic Hill Climbing with full extension takes the longer time to complete their runs.

Chapter 5

Conclusion

During this project, we aimed to understand the behaviour of several local search algorithms within a partially observable and heavily penalized environment that resembles a real navigation challenge at the American University of Armenia. By simulating the Office Hour Race—where three agents, each equipped with different decision-making strategies, attempt to locate a professor’s office under time and score constraints—we were able to analyze how algorithmic design choices influence success, efficiency, and stability. What first appeared to be a simple task of moving from point A to point B quickly revealed itself as a complex problem in which exploration is costly and a single wrong turn can severely damage an agent’s chances.

After running 30 experiments for each algorithm across three extension configurations, one of the most important lessons we learned is that the structure of the world matters just as much as the structure of the algorithm. Every exploratory move in our environment carries a high risk—either entering an angry professor’s office or wasting valuable time. As a result, algorithms that rely heavily on exploration, especially Simulated Annealing (SA), suffered substantially. Although SA is theoretically strong and designed to escape local optima by occasionally accepting worse states, in our grid this behaviour became counterproductive. SA wandered broadly, visited large portions of the map, lost score rapidly, and almost never reached the goal. Its episode length distributions and time–space scatterplots revealed extreme variability and inefficiency.

In contrast, Hill Climbing (HC) emerged as the most consistent and reliable performer across nearly all metrics. Its greedy strategy—always selecting the best available move—fits perfectly in an environment where avoiding risk is the safest possible approach. HC explored only a small number of states, terminated in short episodes, and achieved the highest success rate across all extension groups. Even when we introduced sideways moves or random restarts, HC remained stable and predictable.

This demonstrates that in a world that punishes exploration, the simplest algorithm can in fact be the strongest.

Stochastic Hill Climbing (SHC) occupied an interesting middle ground. On one hand, SHC occasionally outperformed HC in early termination steps because its ability to choose among multiple improving moves provided some flexibility. On the other hand, when we added more extensions, SHC sometimes became unstable. In the “All Extensions” setup, its episode lengths and runtime increased significantly, showing that randomness—while beneficial in moderation—can become harmful in environments with high penalties. This made SHC a “sometimes brilliant, sometimes chaotic” algorithm: capable of strong performance but prone to instability.

Across all analyses—success rate, time complexity, space complexity, and episode length—we consistently observed that our environment rewards stability, penalizes exploration, and favours algorithms that commit to immediate improvement. This pattern aligns with findings in the literature: maze navigation tasks often benefit from greedy, incremental strategies, whereas algorithms designed for large theoretical state spaces, such as SA, are less suited for tightly constrained, penalty-driven environments.

Ultimately, this project highlighted how differently algorithms behave when placed in realistic, physically inspired settings rather than purely theoretical ones. It also emphasized that an algorithm considered “better” in theory is not necessarily better for every problem. Methods that excel in optimization or scheduling can fail dramatically in fast-paced navigation scenarios with harsh penalties. Through the Office Hour Race simulation, we learned not only how to design and run experiments, but also how to evaluate algorithms in terms of context, constraints, and practical consequences.

For future work, it would be interesting to explore hybrid approaches—such as combining HC’s greedy stability with controlled doses of SHC’s randomness—or incorporating learning mechanisms that allow agents to gradually build partial maps of the environment. However, within the scope of this project, one conclusion remains clear: although Hill Climbing is the simplest algorithm, it matched our environment the best and emerged as the most effective strategy for the Office Hour Race.

References

- [1] Kelly Easton, George L Nemhauser, and Michael A Trick. Solving the traveling tournament problem: A combined integer programming and local search approach. *Operations Research*, 59(1):58–71, 2001.
- [2] Wai-Tat Fu and John R Anderson. Climbing the mazes: A cognitive model of spatial planning. *Psychological Review*, 113(4):934–960, 2006.