Planning and Acting on a Partially Observable Taxi Environment

Sambit Sahoo
Project Code
University of Maryland
ssahoo@umd.edu

1 Introduction

The project involves designing and implementing a grid-based environment to solve the problem of transporting passengers to their dropoff locations via taxi. By leveraging GT-Pyhop, a planner can be created to enable a taxi to navigate a 10x10 grid to pickup and dropoff passengers. The main goal of this project is to develop a planner that can compute the optimal path to passengers and dropoff locations using a hierarchical task planner. By utilizing simple tasks like *drive*, *pickup*, and *dropoff*, the taxi can complete compound tasks like *passenger_pickup* and *passenger_dropoff*. The taxi can optimally decide the best sequence to complete these tasks. However, some bushes exist in the environment that the planner does not know of. The taxi is unaware of the bushes until it is next to them. Due to this partially observable environment, there is a cycle of planning and acting. There are a few problems that are investigated:

1. Navigating a Partially Observable Environment: The taxi must navigate the grid, while avoiding bushes. Since the bushes are not observed by the planner initially, the planner recomputes the optimal path as the taxi navigates the environment.
2. Hierarchical Task Network: Primitive tasks are the building blocks to achieving complex objectives. The taxi cannot simply just pickup a passenger as it must navigate to the location of the passenger first.
3. Different Search Algorithms: Many variations of search algorithms can be used for pathfinding problems. It is important to compare the results and evaluation metrics from different methods to understand their efficiency. The two main approaches explored are *A** and *MCTS*.
4. Different Acting Approaches: The two acting approaches explored are *Run-Lazy-Lookahead* and *Run-Lookahead*. There are key differences that are explored through this project.

2 Background

Planning Domain Definition Language (PDDL) is a language that is used to describe the components of planning problems in a standardized way (Ghallab, Nau, and Traverso to appear), Figure 2.6. There are many components, such as actions, states, and goals. This is a stepping stone for creating planning systems to generate actions to achieve a specified objective.

GTPyhop is an automated planning system that uses hierarchical planning techniques to create plans to achieve objectives (Nau 2021). Given a list of tasks, it can create a sequence of actions to accomplish these goals. There are many features, including utilizing rigid relations, actions, methods, and commands. Rigid relations are fixed relationships that do not change during execution, such as types or relations between entities. Actions are primitive operations that modify the state of the environment. Methods are strategies that comprise of subtasks to complete a compound task based on the state. Commands are executable actions used by the actor to update the state based on a given plan.

A graph traversal algorithm used in the project is *A**, (Ghallab, Nau, and Traverso to appear), Algorithm 3.3. *A** computes the shortest path between two points using the sum of the actual and estimated costs, the heuristic function. The actual cost is the cost of reaching the current node from the starting node. The estimated cost is calculated using the *ManhattanDistance* heuristic, the sum of the absolute difference between the current point and goal coordinates. Two important properties of *A** are completeness and optimality. The algorithm must guarantee to find a solution if one exists to achieve completeness. Since this heuristic is admissible, meaning it does not overestimate the true cost, the solution will be optimal. The nodes are selected based on the minimal cost, discovering neighboring nodes and calculating their costs, and then selecting the next smallest cost. This process repeats until it finds the goal.

Another search algorithm is Monte Carlo Tree Search (*MCTS*), (Ghallab, Nau, and Traverso to appear) Algorithm 9.26. The premise of the algorithm is that it builds a search tree incrementally using random sampling to explore potential future states. It selects actions that are likely to lead to favorable outcomes, and then selects the best action based on the simulations. *MCTS* can handle the uncertainty of exploring different paths and evaluate the best sequence of actions to take. While *A** uses a deterministic approach with a fixed heuristic, *MCTS* is more flexible and capable of dealing with uncertainty.

This project uses *Run-Lookahead*, (Ghallab, Nau, and Traverso to appear), Algorithm 2.3. This is an acting algorithm that calls *Lookahead* to get a plan and perform the first action of that plan in each iteration of acting. This process repeats until the current state matches the goal state. This algorithm is useful when the environment changes in unpredictable ways that would cause the planner to fail, but unnecessary if plan failures are not frequent.

Furthermore, this project uses *Run-Lazy-Lookahead*, (Ghallab, Nau, and Traverso to appear), Algorithm 2.4. This algorithm is similar to *Run-Lookahead*, but there is one significant difference. Rather than calling *Lookahead* immediately after the action, it will simulate the current plan until there are no actions left in the plan or an action fails. Once this occurs, *Lookahead* is called to create a new plan. This process also repeats until the current state matches the goal state. This algorithm is typically superior to *Run-Lookahead* because it eliminates the overhead of planning after every action.

3 Approach

3.1 Environment

The environment reflects a layout similar to the Toy Text Taxi environment (OpenAI 2024). The key features are:

1. Grid: The world consists of a 10x10 grid.
2. Taxi: There is one taxi that is at a random, distinct location on the grid. The taxi can carry all the passengers at the same time, if necessary.
3. Passengers: There are three passengers at random, distinct locations on the grid. Each passenger has a designated dropoff location.
4. Bushes: Bushes are obstacles in this environment. The taxi is unable to drive into a bush space. Bushes occupy 2x2 spaces and never overlap.

The environment will be partially observable. The planner is unaware of the amount and locations of the bushes. This means the planner will most likely have to adjust the path of the taxi to get to certain locations during the acting phase as it encounters bushes. For the trials, there are 0 to 5 bushes in the environment. The goal in this domain is to transport the three passengers to their designated dropoff locations, while minimizing the runtime and total moves of the taxi.

Figure 1 and 2 in Section 7 illustrates an example of the environment setup. The first figure portrays the initial information that the planner is aware of, and the second figure shows the environment with knowledge of the bushes.

3.2 Planning Approach

The first step in the planning approach was to sketch out the Taxi environment and identify the moving and static pieces involved. A PDDL was utilized to visualize the structure and actions required for the planning phase. The representation includes predicates for relationships and states and operators for actions. The source code can be referenced in the GitHub repository.

Next, the GTPyhop domain was configured. This involved defining the rigid relations of the Taxi environment, such as the taxi, passengers, dropoff areas, and bushes. Additionally, there are dynamic components, such as the actions. The three primitive actions in this domain are *drive*, *pickup*, and *dropoff*. With these three simple actions, the planning problem can be broken down into segments. By connecting these segments, compound tasks form to complete a subgoal. Bringing this all together, the planner can effectively compute a solution for the taxi to transport the passengers to their desired locations. For the *drive* action, there are two preconditions. The taxi location must be one space from the desired destination, vertically or horizontally. Also, the desired location can not be a bush space. If these conditions are met, the taxi drives to the new location. For *pickup*, the taxi and passenger must be in the same move space. If this condition is met, the passenger's new location is the taxi. For *dropoff*, the passenger must be in the taxi and the taxi must be at the designated dropoff location for the passenger. If these conditions are met, the passenger is successfully dropped off at the designated location.

Furthermore, TOHTN methods are implemented for completing two compound tasks: *passenger_pickup* and *passenger_dropoff*. Additionally, a helper function is heavily utilized, *navigate*. The *navigate* method facilitates the movement of the taxi from its current location to the desired location, either a passenger or dropoff location. There were three implementations of navigate: naive approach, *A**, and *MCTS*.

The most basic approach is naive. This approach involves moving the taxi to the desired location using the *ManhattanDistance* to determine the shortest path. However, this design only works when there are no bushes in the environment. If the taxi encounters a bush, it cannot reroute, causing the plan to fail. This method is ineffective in most cases. The next approach utilizes *A** to compute a path to the desired location. Using actual and estimated costs, the algorithm can dynamically adjust to move around bushes. The estimated cost is calculated using the *ManhattanDistance* heuristic. This makes the algorithm more effective than the naive approach and ensures optimality in finding the shortest path. Lastly, the third implementation employs *MCTS*. This approach explores various paths by simulating trajectories and picking the path with the best outcome, balancing exploration, trying untested routes, and exploitation, favoring promising routes. Unlike *A**, it can handle uncertainty and dynamic changes with ease. There are advantages to both *A** and *MCTS* based on the scenario. With the varying amounts of bushes, the limits of each algorithm are explored.

For the first task, *passenger_pickup*, the objective is to get a specified passenger into the taxi. There are three cases to consider. The base case is that the passenger is already in the taxi, so there is no action. The next case is if the taxi and passenger are in the same move space. This means the *pickup* action can be called. The most complex case is navigating to the passenger and picking them up. The *navigate* helper function is called and the passenger can be picked up. Similarly, the *passenger_dropoff* method objective is to dropoff the passenger at the designated dropoff location. The base case is that there is no passenger in the taxi to drop off. The second case is that the taxi is already at the dropoff area for the specified passenger and the only action to commit is *dropoff*. The last case is when the taxi needs to navigate to the dropoff area, and then dropoff the passenger. Now, that the planning approach is established, it is time to use it in the domain and confront some obstacles to achieve the goal.

Due to the partially observable nature of the environment, the best course of action to decide the order is to start by selecting the closest passenger to the taxi. After picking up the passenger, it will calculate the *ManhattanDistance* and find the closest passenger to pickup or passenger dropoff location for passengers in the taxi. This process repeats until all passengers are at their designated dropoff location.

3.3 Acting Approach

The first step of the acting approach is to implement the commands, which are the executable actions that update the state during the acting phase. These commands are similar in structure to the primitive actions. Using the plan, the actor executes primitive actions to move the taxi, pickup a passenger, or drop off the passenger.

However, since the planner is unaware of the location of the bushes due to the partially observable environment, the actions may not go according to plan. If the taxi encounters a bush, the current plan fails, prompting the planner to reevaluate the situation and create a new plan. Then, the actor can begin executing the new set of actions. There are two different acting approaches explored: *Run-Lazy-Lookahead* and *Run-Lookahead*.

The purpose of using two actors is to compare how many times the actor would call the planner. For *Run-Lazy-Lookahead*, the planner is only invoked when the current plan fails. In contrast, *Run-Lookahead* calls the planner at every step, regardless of failure. The difference is explored further in Section 5.

4 Evaluation Plan

The evaluation aims to assess the effectiveness of planning and acting within the partially observable Taxi environment. Specifically, it explores the differences between the navigation approaches, *A** and *MCTS*, and acting strategies, *Run-Lazy-Lookahead* and *Run-Lookahead*.

The first question explored is: how often is the planner being called *Run-Lazy-Lookahead* and *Run-Lookahead* as the complexity of the environment increases? This is measured by increasing amount of bushes. Since *Run-Lazy-Lookahead* only calls the planner when an action fails, the planner is expected to be called more often as more bushes are in the environment. For *Run-Lookahead*, the amount of times the planner is called should also increase, but not by a substantial amount. The difference between the number of times a new planner is made should be more apparent for *Run-Lazy-Lookahead* as the complexity increases. As complexity increases, the difference between times planner is called should narrow.

The second question is how does the choice of navigation influence the optimality and runtime of the plans generated? The plans generated using *A** are expected to be more efficient in terms of path length, especially when the environment does not have many bushes. However, it may be difficult for *A** to adapt to unforeseen obstacles due to the fixed *ManhattanDistance* heuristic. For *MCTS*, it is expected to perform well in highly dense environments due to the exploratory nature of the algorithm. For simpler environments, the paths may be less efficient, giving *A** should have an edge.

Independent variables One independent variable is the acting method used, comparing the differences between *Run-Lazy-Lookahead* and *Run-Lookahead*. Another independent variable is using *A** versus *MCTS* for the navigation method. The number of bushes in the environment is also an independent variable. The environment has 0 to 5 bushes in a given simulation.

Dependent variables The dependent variable is the number of times the planner is called during execution for the two acting methods. For the varying navigating algorithms, the number of steps to pickup and dropoff all the passengers is the dependent variable. For the varying amount of bushes in the environment, the total distance traveled by the taxi versus the optimal path and the number of times the planner is called due to obstacles are measured.

To elaborate, the first set of trials has the starting positions of the taxi, passengers, and dropoff locations in the same place. The variable will be the number of bushes in the environment, ranging from 0 to 5 bushes. The number of drives is tracked to see how the bushes affect the path of the taxi during acting. Additionally, it will track the number of times the planner is recalled with *Run-Lazy-Lookahead*. This tracks the number of times the planner fails due to the bush being in the planned path.

The second set of trials will use different environments, where the taxi, passengers, dropoff locations, and bushes will be random places. The purpose is to understand how many times *Run-Lookahead* is called versus *Run-Lazy-Lookahead*. The number of bushes is preset to five.

5 Results

The images of the environments are available in the GitHub repository, in the TrialsLayout folder.

The first experiment was to investigate the number of times *drive* was called with *A** and *MCTS* using *Run-Lazy-Lookahead*. The data for *A** is available in Figure 3. For *A**, as the number of bushes increased, the number of drives and amount of bush spaces encountered also increased. When there were no bushes, it took 39 drives to pickup and dropoff all passengers. There were 0 bushes encountered since there were 0 bushes in the environment. When a bush was added, the number of drives increased to 41 with 3 bush encounters. For two bushes, the added bush did not affect the route of the taxi, but it saw an extra bush space. For three bushes, it took 43 drives to transport the passengers and there was a large increase of 10 bush encounters. For four bushes, the number of drives was 47 and there were 14 bushes encountered. Lastly, when all five bushes were in the environment, the number of drives greatly increased to 63 and the bush encounters were 15. The results for the number of bushes and number of drives are in Figure 5. The results from the number of bushes and the number of bush encounters can be viewed in Figure 6.

For *MCTS*, there were five trials as the results varied every run. The data for *MCTS* is available in Figure 4. When there were 0 bushes in the environment, the number of drives was 1997, 1253, 1417, 1215, and 1303. The number of bush encounters was zero across the board. When there was 1 bush, the number of drives decreased. There were 845, 753, 717, 957, and 949 drives and 7, 4, 5, 9, and 7 bush encounters, respectively. There were more bush encounters than the number of bushes because it would go toward the same bush multiple times. When there were 2 bushes, the number of drives was 967, 803, 807, 855, and 755. The bush space encounters were 6, 5, 7, 8, and 8. For 3 bushes, there were 611, 569, 585, 691, and 581 drives and 11, 11, 14, 14, and 11 bush encounters, respectively. There were 621, 679, 439, 483, and 661 drives and 16, 15, 14, 14, and 15 bush encounters for 4 bushes. The approach was unable to find a solution for when there were 5 bushes. The results for the number of bushes and number of drives are in Figure 7. The results from the number of bushes and the number of bush encounters can be viewed in Figure 8.

Next, there are the results from comparing *Run-Lazy-Lookahead* and *Run-Lookahead* using the *A** algorithm, which is available in Figure 9 and Figure 10. On average, the planner would formulate a plan 8 times per trial for *Run-Lazy-Lookahead*, while it was about 26.3 times per trial for *Run-Lookahead*. The problem was not severely complex in most cases. One of the most difficult examples was illustrated earlier, through the *A** and *MCTS* comparison.

5.1 Discussion of Tradeoffs and Limitations

For *A**, the results aligned with the expectations. As the number of bushes increased, the actor would be more prone to running into it since the planner was unable to plan around it due to partial observability. *A** performed well in maneuvering around obstacles once they were found as there was no major increase in drives until there were five bushes. For the case of five bushes, the problem was significantly harder to complete. There was a dead-end that would force the planner to have to go around a collective of bushes. The figure for the environment with 5 bushes is available in Figure 5 in Section 7. Due to the difficulty of completing the goal when there were 5 bushes, the increase in drives does not deviate significantly from the expectation. The graph looks nearly exponential at a slow rate, but the difficulty of handling the dead-end is the cause. In a similar scenario with five bushes in different locations, the number of drives did not increase significantly. *A** is a strong algorithm for solving problems in the Taxi environment. It utilizes the *ManhattanDistance* heuristic to efficiently find the shortest path to the goal, even with obstacles blocking its original path plan. The performance of this algorithm worsens as the amount of obstacles increases. In a fully observable environment, this algorithm would excel. In the future, there could be more bushes in the environment or a larger grid space to examine results further.

For *MCTS*, the results did not align with the expectations. The number of drives was high in comparison to *A**. As the number of bushes increased, the number of drives decreased. One reason for this is that the bushes eliminated some spaces the actor would've driven into, making the number of potential paths more limited as there were fewer move spaces. One supporting detail of this is that as the number of bush encounters increased, the number of drives decreased. The runtime for this algorithm took significantly longer than *A**. There aren't metrics to support this, but it was an interesting detail to note. This indicates that *MCTS* is more computationally expensive due to the simulations and random sampling. With the sparse amount of information given to the *MCTS* algorithm, it was not expected to excel in this environment.

The results from the comparison of *Run-Lazy-Lookahead* and *Run-Lookahead* were as expected. For *Run-Lazy-Lookahead*, the planner was run less often because it would only have to run when the current plan failed. This only occurs when the actor follows the plan, but wants to take the taxi through a bush. Since the taxi can not do this, it must replan to move around the bush. Meanwhile, *Run-Lookahead* replans at every step, regardless of whether the action succeeds or not. This explains the high values of the planner being called. The values map fairly well to each other, suggesting that there are unnecessary planner calls with *Run-Lookahead* in these scenarios. *Run-Lazy-Lookahead* is more efficient in terms of computational resources for this Taxi environment.

6 What I learned

- When starting this project, I realized the importance of visualizing the planning problem at hand. It was much easier to digest the components after sketching out a few examples of the domain.
- I realized the importance of starting with the base case and building up when designing the TOHTN methods as if it were a proof. There were some subtle mistakes in my code that took some time to debug due to missing a switch of variable value leading to failed preconditions or unexpected effects.
- I learned more about *A** and *MCTS* and how it can affect the system in terms of search. I knew the difference based on the definition, but it was interesting to explore this domain. *MCTS* did not work as planned. Due to the uncertainty of the location of the bushes, I think it made the algorithm less effective. Also, the *drive* action could go in four different directions, making the branching factor very high.
- *A** seems to be a good method for when the environment is fully observable. It functioned decently in this environment, but not as well as I was originally expecting.
- For a simple domain like this, the runtime didn't impact my results adversely. I expected this to an extent. In this scenario, it was not apparent how much better *Run-Lazy-Lookahead* can be compared to *Run-Lookahead*. However, I can see how it would affect more complex domains and the unnecessary overhead *Run-Lookahead* could cause in certain instances.
- Originally, I didn't plan to make a partially observable environment. It was interesting to see how the complexity

increased from the original version and made the results from *A** and *MCTS* more interesting to explore.

- *MCTS* was a late addition to the project. It was interesting to see how it fared in this environment, even though I didn't have much time to explore it.

7 Summary Figures and Tables

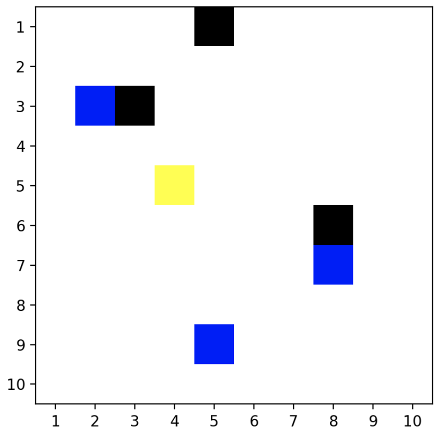


Figure 1: This figure illustrates the domain that the planner sees initially. There are no bushes visible as there are none next to the taxi currently. The passengers are labeled in blue, the dropoff areas are labeled in black, and the taxi is labeled in green. The blue at (8,7) is Passenger 1 and wants to be delivered to (3,3). Passenger 2 is at (2,3) and wants to be dropped off at (8,6). Lastly, Passenger 3 is at (5,9) and wants to be dropped off at (5,1).

References

Ghallab, M.; Nau, D.; and Traverso, P. to appear. *Acting, Planning, & Learning*. Cambridge University Press.

Nau, D. 2021. GTPyhop: Hierarchical Task Network (HTN) planning in Python. https://github.com/dananau/GTPyhop.

OpenAI. 2024. Taxi Environment.

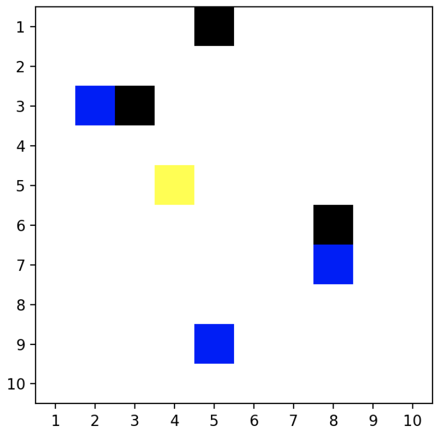
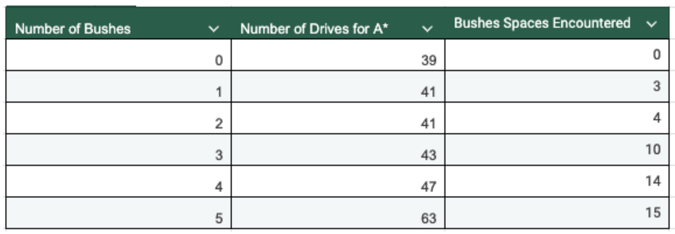
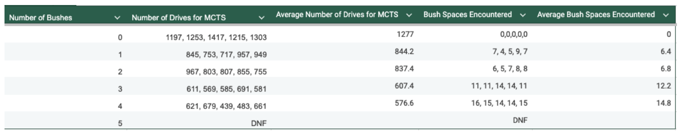


Figure 2: This figure illustrates the domain that the planner sees initially. Unlike the first figure, the bushes are visible. This is the environment if it is fully observable. There are two bushes that the actor must navigate around while picking up and dropping off passengers. If the actor runs into a bush, the planner will reroute the taxi.



Number of Bushes	Number of Drives for A*	Bushes Spaces Encountered
0	39	0
1	41	3
2	41	4
3	43	10
4	47	14
5	63	15

Figure 3: This table shows the number of drives and bush space encounters for the *A** algorithm. This is also available in the GitHub repository. Tables/A*Trials.png



Number of Bushes	Number of Drives for MCTS	Average Number of Drives for MCTS	Bush Spaces Encountered	Average Bush Spaces Encountered
0	1197, 1253, 1417, 1215, 1303	1277	0,0,0,0,0	0
1	845, 753, 717, 957, 949	844.2	7, 4, 5, 9, 7	6.4
2	967, 803, 807, 855, 755	837.4	6, 5, 7, 8, 8	6.8
3	611, 569, 585, 691, 581	607.4	11, 11, 14, 14, 11	12.2
4	621, 679, 439, 483, 661	576.6	16, 15, 14, 14, 15	14.8
5	DNF		DNF	

Figure 4: This table shows the number of drives and bush space encounters for the *MCTS* algorithm. This is also available in the GitHub repository. Tables/MCTSTrials.png

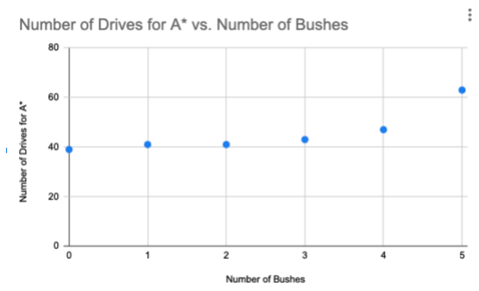


Figure 5

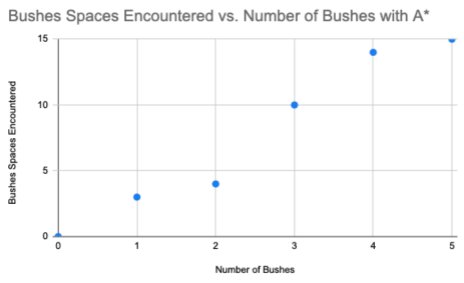


Figure 6

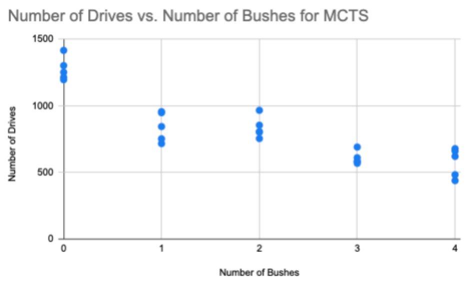


Figure 7

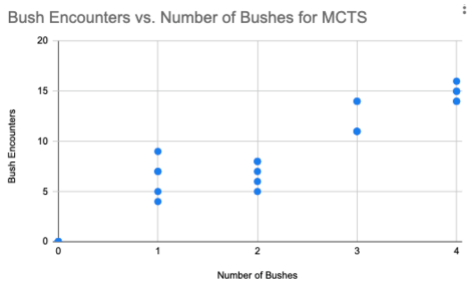
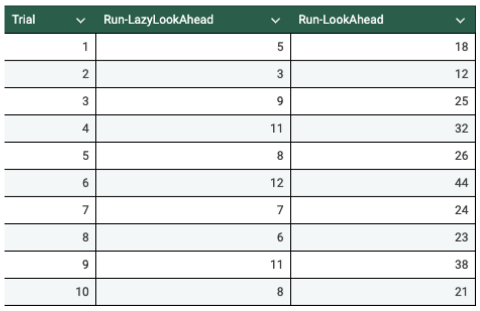


Figure 8



Trial	Run-LazyLookAhead	Run-LookAhead
1	5	18
2	3	12
3	9	25
4	11	32
5	8	26
6	12	44
7	7	24
8	6	23
9	11	38
10	8	21

Figure 9

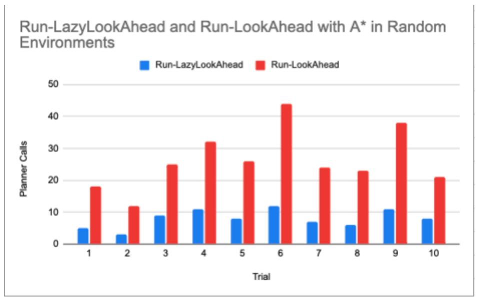


Figure 10