Analyzing AI: A Comparative Study of A* and Q-Learning

Ayden Shine
Wilfrid Laurier University
Dept. of Science
Waterloo, Canada
nair5190@mylaurier.ca

Jeremy An
Wilfrid Laurier University
Dept. of Science
Waterloo, Canada
anxx9640@mylaurier.ca

Costantinos Boyiatzis
Wilfrid Laurier University
Dept. of Science
Waterloo, Canada
boyi6700@mylaurier.ca

Ilyas Ozcelik
Wilfrid Laurier University
Dept. of Science
Waterloo, Canada
ozce7100@mylaurier.ca

Shawn Phung
Wilfrid Laurier University
Dept. of Science
Waterloo, Canada
phun4180@mylaurier.ca

***Abstract*—Traffic management is a continuously pressing issue in modern cities, in which the desired goal is to efficiently reduce the travel time and vehicle emissions. This report will investigate solutions that aim to address the issues of traffic management in self-driving vehicles. Due to the complexity of the problem, we will investigate potential solutions from other pieces of related literature. By analyzing similar problems, considering their pros and cons, and comparing them against each other, we create a basis for us to formulate alternative ways of approaching this problem. We explore the usage of the A* Search and Q-Learning algorithms, using the information provided by the Central Traffic Control Unit (CTCU), and go over the performance, benefits, and drawbacks of each. Going through simulations of both algorithms, paired with the research of existing results, our proposed solution ends up being a mixture of the two. We conclude that both algorithms have their respective strengths and weaknesses, which can be made up for by employing methods from both in a mixed algorithm that utilizes known information given by the CTCU, and the adaptive learning ability from machine learning techniques.**

***Index Terms*—CTCU : Central Traffic Control Unit**
BNART : Best Neighbor Algorithm for Routing Traffic

I. Introduction

In the BNART problem that was given to us states that it needs a CTCU that will regulate traffic within a certain area. This will send signals to self driving cars within the area and will direct them all to their destinations within the fastest time. It takes into account intersections, traffic and other roadblocks that will potentially increase the travel time. This could be used in real life metropolitan areas where navigation can be very difficult with the high density of people and traffic. Since it will look for the most optimal path for all cars connected to the network. When coming up with a solution for the problem we realized the complexity of the problem itself so we decided to use other articles for information on how to approach this problem. We initially started with the use of A * search formula. This formula would find the distance between the starting node and the ending node with the least cost, which in this scenario would be travel time.

II. Related Work

1) A. A Comprehensive study for robot navigation techniques: Rahiman, W. (2014) attempts to solve a problem that closely resembles the BNART problem that we are dealing with. The research aims to have a robot navigate a cluttered space, from a given start and end position while following a safe path and producing efficient path lengths. This technology has various applications, from material handling, disaster relief, patrolling, and of course, transportation. This is somewhat similar to the BNART problem, which also attempts to have a vehicle navigate through a complex environment while aiming for a safe and efficient path. The problem that this article talks about covers a much broader topic, but has many overlapping elements in regards to our solution. For example, as part of their navigation methods, the article explores the use of Djikstra's algorithm, which is an algorithm that is used to find the shortest path in an environment, by exploring all paths equally. In our solution, we have implemented the A*Search algorithm, which is similar to Djikstra's algorithm in the sense that both algorithms aim to find the shortest path from a starting location to an end destination, but differ in the methods that they use to achieve this result. The article studies many different methods, and much like us, comes to a conclusion that dynamic, hybrid solutions with heuristic approaches would demonstrate better results on average, while acknowledging that implementing hybrid algorithms comes with many challenges such as compatibility and uncertainty in control performance.

B. BNART: A Novel Centralized Traffic Management Approach for Autonomous Vehicles An optimal Routing Algorithm which can adapt into any dynamic environment is a crucial aspect for an effective traffic management. This article presents us with a comprehensive study on traffic

management for self-driving vehicles by utilizing the Central Traffic Control Unit (CTCU). Similar to us, this system is responsible for path planning and providing with real-time information on other cars and the environment conditions and so on. Main focus being to reduce congestion and optimize traffic flow through the Best Neighbor Algorithm for Routing Traffic (BNART). The article emphasizes on ensuring efficient routing to minimize congestion and maximize on-time arrivals. This article focuses on using performance metrics such as average travel time and distance, evaluating BNART against Breadth First Search (BFS) which is a popular method used for search to emphasize how effective BNART is for their approach. Meanwhile our approach focuses on using the A* search algorithm for optimal path-finding. Our method is an alternate way from the article's proposed approach which also guarantees to find a complete solution under certain conditions such as sufficient memory and a finite space.

III. System Model and Problem Description

A. System Model

We consider the road structure shown in Fig. 1 with a graph $G = (V, E)$, where V is a set of nodes (vertices) and E is a set of edges connecting two nodes. To simplify the process of calculating travel time and traffic, we divide each edge (from one vertex to another) into equal segments. Hence, the set of nodes shown in Fig. 1 is divided into 3 subsets: a set of buildings (shown with grey nodes), roads (shown with blow color), and intersections (shown with orange color). These nodes are numbered 0 to 24 in a left-to-right, top-down manner. The starting and destination nodes of a vehicle are given the colors green and red, respectively, as placeholders. We can consider each vehicle $v \in V$ in the network is registered and reports to the CTCU when it plans on entering the network at time T_s^v and setting a route from starting node $S_v$to a destination node D_v. An example is shown with the purple line, where vehicle 1 starts from location S_1 (located on node 0, link $< 0, 1 >$), traversing multiple roads and intersections until it reaches its destination location at node 22, located on link $< 14, 22 >$. Note that any link $< x, y >$ represents the traversal between nodes 0 to 1, and is not equal to $< y, x >$. We let T_{Total} represent the total time taken to traverse between the starting link and the link the vehicle is currently at, and is going at the speed limit. When the vehicle arrives/departs from an intersection node, we would add a delay to T_{Total}, to simulate stopping/slowing at an intersection or an overall delay in traffic in that area.

B. Problem Description

The problem that we are trying to solve here is to find the best path for each vehicle v in the system to go from its source S_v to its destination D_v by considering multiple vehicles at a time to minimize traffic congestion and optimize travel time of all vehicles. It should be noted that we are not trying to find the shortest distance possible, as that may cause congestion, leading to more time spent in traffic and a higher travel time. As shown in Fig. 1, we can see that even though the shortest

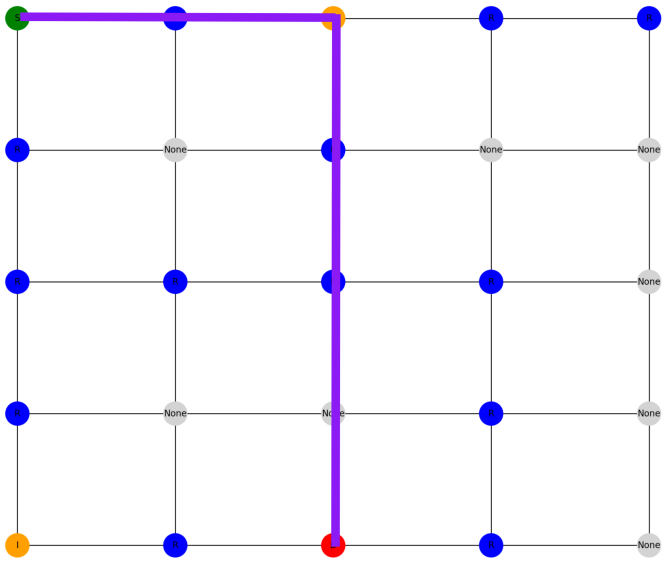

Fig. 1. Illustration of the system model

distance is highlighted by the purple path, it may not be the best path routing because the vehicle traverses through the intersection node 2 (links $< 1, 2 >$ and $< 2, 7 >$). Consider that there is a second vehicle, V_2, which wishes to traverse through the link $< 2, 7 >$ at the same time: to avoid congestion links when possible, the algorithm should consider rerouting one of the two vehicles such that they would not traverse the same link. We can calculate the location and travel time of each vehicle by taking its starting time (starting time at source node) and the time each vehicle v spends in traffic. For the sake of consistency and simplicity, we assume all vehicles travel at the maximum speed each road segment allows for, and when they reach or leave an intersection node, there is a fixed delay added to T_Total. When a new vehicle enters the system, the problem is solved again while taking into consideration the current locations of all vehicles in the network.

IV. Proposed Approach

Our first approach depends heavily on the A* search to find the shortest path in an environment which is represented by a 5x5 graph. Our approach uses three classes namely Node, Graph, and Vehicle classes. We use the node class which uses an ID and a flag which indicates if we're at an intersection. The vehicle class represents all the vehicles in the environment that have a start state, goal state, and the time taken as their parameters. Our heuristic function plays a very important role in determining an estimate of the cost (distance) from the current node to our goal state. We use this estimate to prioritize which nodes need to be explored to reach the goal state faster. The minimum distance value which is returned is the path that we explore first using A star search.

The intersections (denoted by the orange squares in Fig 2) are considered to be unique situations where they represent an intersection that might cause any kind of delay ie. traffic, accidents, or construction. The algorithm can choose to avoid

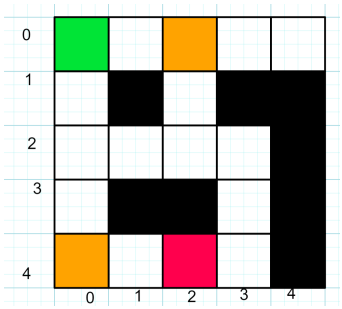

Fig. 2. 5x5 Grid indicating our environment

it or add a delay to the travel time when a vehicle passes through these intersections. The components used for our A* search include:

- *g(n)* which is the exact cost of the path from the start node to any node n.
- *h(n)* which is the heuristic estimated cost from node n to the goal. We obtain this value from the heuristic function explained earlier.
- at last, we have *f(n)* which is the total estimated cost of the cheapest path.

Our search algorithm runs in a loop, extracting the node with the lowest f(n) value in the open set which contains all the open nodes. The loop finishes after we have attained the goal state. After we've reached our goal, the code retraces the path it took and returns it as the solution. Through each iteration through the loop, the cost updates by 1 at each intersection to eventually return the cost it took to get from the initial state to the goal state. The efficiency of our code heavily depends on the effectiveness of our heuristic function.

For our second approach, we decided to use a q-learning algorithm. Q learning algorithm uses a q table and will update that q table based on different rewards or punishments received from each action. In our scenario actions will be choosing which road to take and for rewards it will decrease the amount of time it takes the car to reach the end while for punishments it would increase the amount of time. This is suited for scenarios where the transitions and rewards are not known because it allows for the program to learn and adapt, meaning it would fit in with our scenario. After all, in real life, the world is always changing and the flow of traffic is always different.

V. Performance Evaluation

When evaluating the performance of A* search and Q learning algorithm for traffic management and path finding there is a clear difference in their efficiencies and adaptability depending on the environments tested. For small scale routes or less complex problems A* is much more efficient. It found the optimal route with a computational execution of 0.001 seconds. To contrast, Q learning solved the same problem in 0.011 seconds. The difference highlights the efficiency of A* in simpler route scenarios. However A* search's efficiency diminishes quickly as the route gets more complicated. As more variables get introduced the complexity increases slowing down the processing speed. This makes Q learning more efficient in more complex scenarios as it adapts to the environment unlike A*.

On the other hand, Q learning, despite its slower initial execution time, shines in more complex and longer scenarios or when a route is taken multiple times. This is because of its ability to learn from repeated routes. It is best in frequently navigated routes, especially more dynamic ones such as traffic. While A* is best suited for short term optimization Q learning excels in dynamic environments.

VI. Discussion and conclusion

Since there are multiple solutions to this problem and a lot of different approaches. Since we only tested two of them it is plausible that there are better algorithms for this solution. From the knowledge that we gained about these two algorithms, we learned that while they each have their own strengths it would be beneficial to merge both algorithms for better results. Because it then will be able to estimate the most optimal path while also taking note of all the surroundings within the q table. Which can lead to finding even more optimized paths.

References

[1] Anonymous. (n.d.). BNART: A Novel Centralized Traffic Management Approach for Autonomous Vehicles.

[2] Gul, F., Rahiman, W., & Alhady, S. S. N. (n.d.). Full article: A Comprehensive Study for Robot Navigation Techniques. https://www.tandfonline.com/doi/full/10.1080/23311916.2019.1632046