A Planning and Scheduling System for Car Pooling

Max Camilleri
Faculty of Artificial Intelligence
79287257
max.camilleri.20@um.edu.mt

Dr Josef Bajada
Faculty of Artificial Intelligence
23402506
josef.bajada@um.edu.mt

ABSTRACT

With today's advancements in autonomous vehicles, this project aims to evaluate rooting techniques that could be used to autonomously direct a fleet of transportation vehicles alone a network. A series of AI and metaheuristic techniques such as genetic algorithms and Tabu search where evaluated. This was done using a series of graphs and metrics to determine the advantages and disadvantages of certain algorithms in the given situations. These techniques where embedded into a front facing UI, aimed at aiding the future research of this project.

Keywords

Tabu Search, Genetic Algorithms, Metaheuristic, User Interface

1. INTRODUCTION

With technological advancements leading us ever closer to an autonomous world, it is not extravagant to imagine a fleet of autonomous ride sharing vehicles being a common transportation method. This project aims to evaluate multiple artificial intelligence techniques with regards to the routing of these autonomous vehicles. A combination of common artificial intelligence metrics, and graphs will be used to evaluate and compare the different configurations.

2. LITERATURE REVIEW

The outlined problem is a prominent research area with countless new developments emerging each day. The task follows the Euclidean traveling salesman problem, attempting to compute an optimal path through all the states. Outlines below are a few techniques used to approximate the traveling salesman problem.

2.1 Tabu Search

Tabu search is a metaheuristic local search algorithm used for optimization. Tabu search follows the methodology of the hill climbing algorithm with a few modifications. Local search methods, like hill climbing, have a tendency of getting stuck in local maximums. This is a region in the solution space which has no immediate improvement values in ether direction, as can be seen in figure 1. Tabu search enhances the performance of such techniques by prohibiting already visited solutions from being visited again during a set time frame. Tabu search is used to solve np-hard problems. Some examples problems tabu search has been applied to are: finding low energy structures of proteins [1], the optimal placement of power distribution centres [2], and of course, the traveling salesman problem [3] [4].

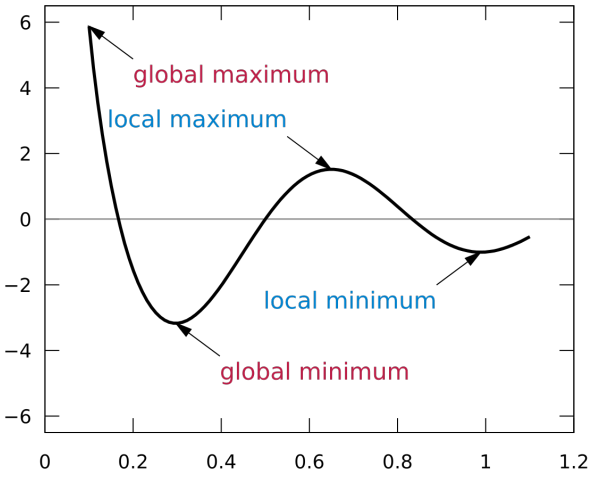

Figure 1: local Maximum and Minimum

To accomplish this Tabu Search makes use of a tabu list, in which it stores the last *'x'* actions that where taken. The size of the tabu list is predefined before the algorithm starts. It greatly impacts the exploratory properties of the algorithm. A large list would force the algorithm to explore away from a local maximum, even if this may lead to worse evaluations in the meantime. This list gives the algorithm its name, as trough it some actions are considered *'tabu'*.

2.2 Nearest Neighbour

Nearest neighbour algorithm compares the distribution of distances that occur from a data point to its nearest neighbour in each data set. Consequently, the NN algorithm is a greedy algorithm that follows a very simple procedure [5].

The algorithm follows the following procedure:

1. A vertex *'x'* is picked at random as the current location
2. The lightest edge is chosen form vertex *'x'* to the nearest unvisited node.
3. Vertex *'x'* is marked as visited
4. Once all vertices are visited, the algorithm is complete.

This relatively simple algorithm was one of the first to be applied to the TSP [5]. Although nowadays its effectiveness has been out-shined by other, more expensive algorithms, its simplicity can prove superior in certain situations [6] [7].

2.3 Ant Colony Optimization

The ant colony optimization algorithm is an agent-based metaheuristic algorithm that is used to approximate solutions. The algorithm was modelled after the behaviour of ants and combined with other swarm inspired algorithms to create a distinct family of optimization metaheuristics [8].

Ants have a unique method of alerting others on the presence of food. Once an ant encounters a food source, they will carry a piece of it to the desired destination, dropping a trail of pheromones on the way. This allows other ants find and follow the pheromone trail to the food source and aid in the collection process. As more ants transport the food back, the pheromone trail increases in intensity, alerting more ants. The pheromones evaporate at a constant rate with respect to time. This means that the shortest path from the food source to the desired location will have the strongest sent.

ACO works in a similar manner to this, where the current node in a TSP problem acts as the delivery location, and the rest of the node represent possible food sources. This is done through the following formula.

$$p_{ij} = \frac{(r_{ij})^{\alpha}(s_{ij})^{\beta}}{\sum(r_{iz})^{\alpha}(s_{iz})^{\beta}}$$

Where the *'r'* represents the pheromone consent ration and *'s'* represents the static probability wait of choosing weight *'ij'*.

By changing the alpha, beta, and *'s'* parameters, the algorithm can be tuned for exploratory, or greedy operation. While still a relatively new algorithm, its effectiveness in optimizing the TSP problem has been proven. This can be seen in a handful of papers such as, Benhra Jamal's *'A Performance Comparison of GA and ACO Applied to TSP'* [8], which concluded that ACO outperformed the widely used GA due to its greedy approach, and Daoziong Gong's *'A hybrid approach of GA and ACO for TSP'* [9], where both algorithms are combined to provide an improvement on the already established genetic algorithm.

2.4 Genetic Algorithms

A Genetic algorithm is a search heuristic inspired by Charles Darwin's theory of evolution. The algorithm reflects the process of natural selection by allowing better performing evaluations to be used in future generations.

The algorithm works off a population of predetermined size. This population is initially created at random. Using some evaluation criteria, the fitness of the population is calculated. A traditional genetic algorithm will take the top performing members of the population and always them to combine with one another. This process create a new population of the same size as the last based off of the top performers of the last generation. A mutation criterion is also applied to the new population. This allows unexplored strategies to be explored.

This project focuses on the implementation of two of the above algorithms, those being Tabu Search and Genetic Algorithms

3. METHODOLOGY

3.1 Front End Implementation

The project was produced using the python, JavaScript, HTML, and CSS programming languages. The back-end of the system uses python, and communicates to the front-end using the flask package. The user Interface of the system is separated into two distinct areas, as can be seen in figure 2. The left area is a map of Malta. The resulting trip distribution will be displayed here through the use leaflet [3.3.4] markers, where blue markers represent pickups, red markers represent drop-offs, and the cars represent the current location of the autonomous vehicle. The Car markers also have the ability to follow there designated route, providing a visual representation of the paths. This feature is not yet refined, as the rotation aspect of the moving vehicle does not always accurately represent the direction of the car. This subject is discussed in further detail is section 5.

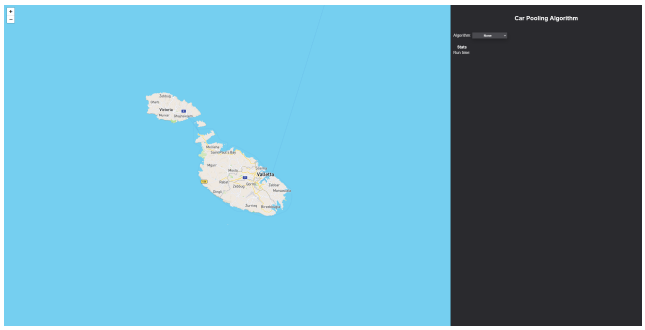

Figure 2: Base Website

The right side of the UI allows the user to input a series of parameters, as well as show some statistics about the algorithm's performance. The uses can select two different algorithms to test, these being, Genetic Algorithms and Tabu Search. Each algorithm takes four parameters, the iteration count, trips preset, passengers per car, and population size/ tabu size respectively for each algorithm. These can be seen in figures 3 and 4.

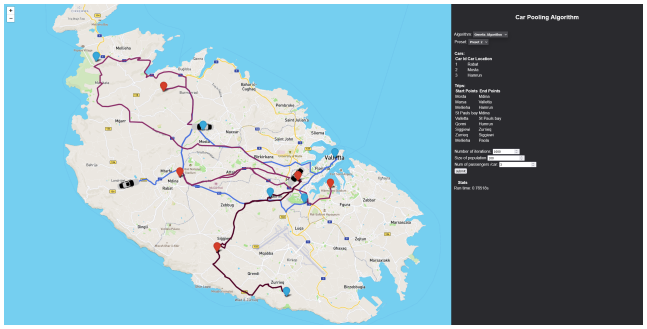

Figure 3: Website with Genetic Algorithm

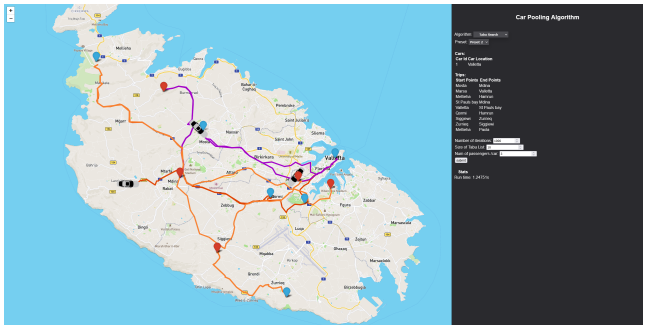

Figure 4: Website with Tabu Search

The user interface section of the UI also displays the time taken to run the algorithm on the given settings. The goal was to add more information about the running of the algorithms alongside this, but due to time restrictions this was not feasible. Section 5 goes into further detail about website limitations.

3.2 Environment Setup

The environment was modelled as a graph as it organizes the complex system of cars, pickups, and drop-offs into an efficient data structure. The nodes of the graph can represent either the start location of a car, a pickup location, or drop-off location, while the edges in the graph represent possible paths a car can take. There are 3 conditions edges follow when being created. These are:

- Always create an edge between the cars start locations and all the pickup points. This is done as any given car has the ability to execute any trip.
- Create an edge between all the pickup locations and all the drop-off locations. Note that there are extra conditions that are checked later. These ensure that the cars can not go to an end point if the respective pickup has not been performed yet.
- Create an edge between all the drop-offs and all the pickup locations, except the pickup location associated with the given drop-off.

Each edge stores three additional pieces of information all calculated using the OSRM package. These are: the distance between the nodes, the duration it takes to travel from node *'a'* to node *'b'*, and a list of geographical points indicating the roads that need to be taken to traverse from node *'a'* to node *'b'*. OSRM returns a JSON document, which was retrieved through the use of the requests python package.

3.3 Featured Software and Packages

3.3.1 Docker

Docker is an open-source platform that gives developers the ability to create and share containerized applications. Using docker, a developer can create an isolated environment inside of a container, making it easy to share the project.

A docker container will include all dependencies (frameworks, libraries, etc.) to run an application in an efficient and bug-free manner. This makes it ideal for sharing in development applications with a testing team, as well as sharing finished projects, like OSRM *(seciton 3.3.3)*, with other developers who would like to use it in their systems.

At the basic level, a docker container acts similarly to a virtual Machine (VM), creating a sub environment on the host machine. However, there are some differences. Firstly, docker containers share the kernel of the host OS. This allows for multiple docker containers to fun on a singe OS. On the other hand, VMs establish a separate operating system based on a portion of the hardware available in the system. This means that for multiple workloads, multiple VMs are needed. Furthermore, since docker is running on the host operating system it has a negligible start-up time. A VM would first need to wait for the operating system to start up before executing the environment, making it slower overall. For more differences, please refer to footnote [1]

3.3.2 Open Street Maps

OSM is an open-source database of geodata around the world. Most commonly, OSM is used for its extensive mapping of roads around the world. The community behind OSM is extremely active, often updating changes in road networks ahead of competing services such as google maps. It achieves this through, user contributions, surveys, and satellite and government data.

While trivial in nature, the system has been used in countless scientific applications. For example, road data was used to research the remaining road-less areas worldwide. This data was then used to update the Forest landscape integrity index [2].

In this project, OSM was used in Combination with OSRM *(section 3.3.3)* and docker *(3.3.1)* to estimate the efficiency of different routing possibilities.

[1] https://www.simplilearn.com/tutorials/docker-tutorial/docker-vs-virtual-machine
[2] https://www.forestintegrity.com/

3.3.3 OSRM

The open-Source Routing Machine (OSRM) is a C++ implementation of a high-performance routing engine aimed at calculating the shortest paths in road networks [3]. It combines its routing algorithm with the OpenStreetMap's project to compute routs anywhere in the world.

Although being created in C++, OSRM can accessed through a docker image [4]. OSRM gives users the ability to query for a multitude of operations, such as, shortest route between two coordinates, nearest road, and even approximate the Traveling Salesman Problem. It does all this while also giving options for different modes of transport such as cars and bikes. Above all this, the system returns the duration of the trip, the distance, and the points needed to complete the trip following the road network.

For this project the OSRM package was set up using a docker image on an Ubuntu Linux subsystem. The system was set up to calculate routes being traversed using the cars preset. Furthermore, a simple shell script was set up to automatically run the docker image with the selected preset on the OSM map of Malta.

3.3.4 Leaflet

Leaflet is a map representation tool that allows for simplicity, performant maps to be embedded in web and mobile applications. It works in conjunction with other mapping services (like OSM used in the project *(section 3.3.2)*, to construct maps on given metadata. Above this, it allows its community to create plugins for the service, some of which are 'MovingMarker' [5] and 'RotateMarker' [6] used in this project.

3.3.5 Packages

The implementation of this project made use of various packages to aid in its creation. Given most of the implementation was done using the python programming language, below are a list of python packages used.

Table 1: Packages

Package Name	Related Documentation
Math	https://docs.python.org/3/library/math.html
Copy	https://docs.python.org/3/library/copy.html
Time	https://docs.python.org/3/library/time.html
Random	https://docs.python.org/3/library/random.html
Flask	https://flask.palletsprojects.com/en/2.1.x/
Json	https://docs.python.org/3/library/json.htm#module-json
Requests	https://pypi.org/project/ requests/

Firstly, the math package was used to simplify the evaluation of the fitness functions. The implementation made use of a few features of the package, including the computation of square root. Similarly to the math package, the copy and time packages provided small but helpful utilities. The copy package provided the *'deep copy'* function, allowing for nested arrays to be copied. This was required as the base copy function in python acts as a *'shallow copy'* only copying the top level of a set of nested arrays. The time function was used the evaluate the time taken for a given algorithm to run. The result of this is presented to the user through the front UI [**??**].

Next, the random package is a self-contained method of pseudo randomly generating numbers. Given its utility, this package was used all throughout the project where randomization was required. Some instances of this are, randomly initializing a population for a genetic algorithm, and creating an initial solution for the tabu search.

Finally, the Flask and Json packages were used in tandem to communicate between the back-end python script, and front-end JavaScript. Flask is a web framework that allows development of web applications trough the use of python. Given that the majority of the project's technicalities were created in python, flask provides an easy way of communicating the calculated data to the website. The receiving end of the flask call is a JavaScript program expecting a Json response. For this reason, the Json package was first used to convert the python arrays into Json objects. This combination of techniques was used to send over run times and geographical points.

The Json package was also used in connection with the Requests package. This module allows for quick HTTP requests. Therefore, it was used to retrieve Json data created by an OSRM instance *(section 3.3.3)*. Given the data is in Json format, the Json module was used to convert data to a readable format.

Furthermore, the matplotlib.pyplot [7] packages were used for evaluation *(section 4)*. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in python. Specifically, the *'.pyplot'* section of matplotlib was used, aimed at creating plots.

4. EVALUATION

4.1 Fitness Function

To begin the evaluation process, the subject we wish to evaluate needed to be determined. Given the nature of the problem, there are multiple schools of thought that can be applied. For example, the algorithms could be configured to optimize greedily in favour of the operator, doing everything possible to reduce distance travelled, or in favour of the users, minimizing the wait time.

For this implementation, three fitness metrics where considered. Note that for representational purposes the network in figure 5 was used. This network consists of two cars, and six trips. Where each trips respective pickup and drop off locations are found horizontally to one another. For the purpose of this test, each fitness function was run on a genetic algorithm of population size one hundred, over a thousand iterations.

[3]https://en.wikipedia.org/wiki/Open_Source_Routing_Machine
[4]https://hub.docker.com/r/osrm/osrm-backend
[5]https://github.com/ewoken/Leaflet.MovingMarker
[6]https://github.com/bbecquet/Leaflet.RotatedMarker
[7]https://pypi.org/project/matplotlib/

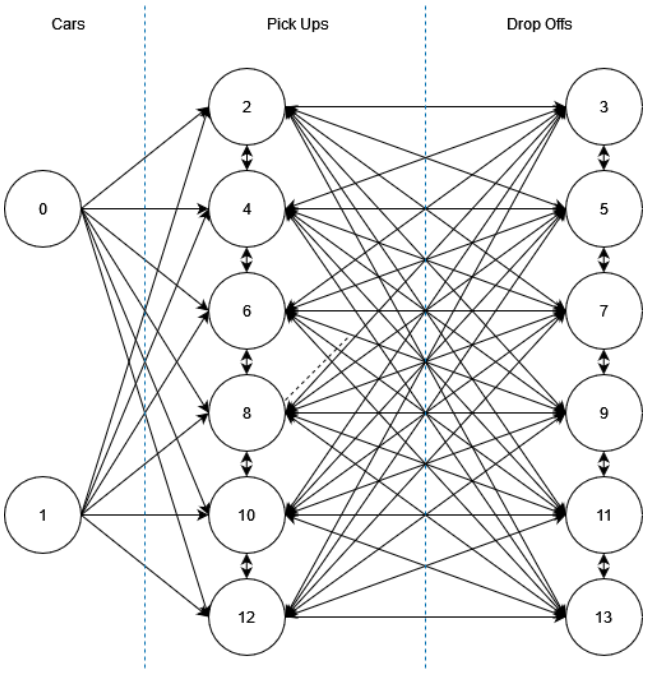

Figure 5: Test Network

The first metric optimizes for the least overall distance. This algorithm determines the fitness of a solution depending on the total distance travelled, across all vehicles in the solution.

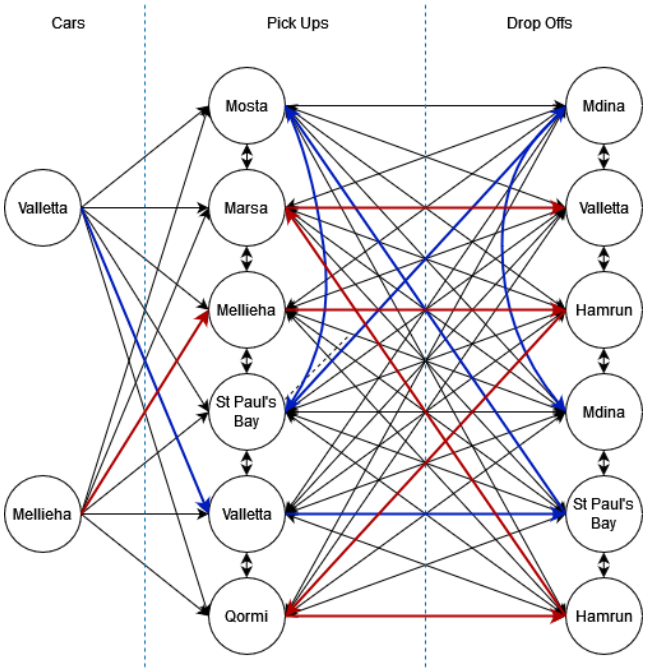

Figure 6: Optimal solution for Distance fitness Function

The second fitness criterion explored optimizes greedy in favour of the user, attempting to reduce wait times across all users. The fitness of a solution under this criterion is the sum of squares of all the pickup wait times. The pickup times in this evaluation were squared to greater distinguish between a user who has been waiting, for example, five minutes (as their cost would be 25), and someone who has been waiting ten minutes (as their cost would evaluate to 100). Once the sum of squares is computed, the square root of the result is taken. This is done to contain the result in the same time space as the duration parameter.

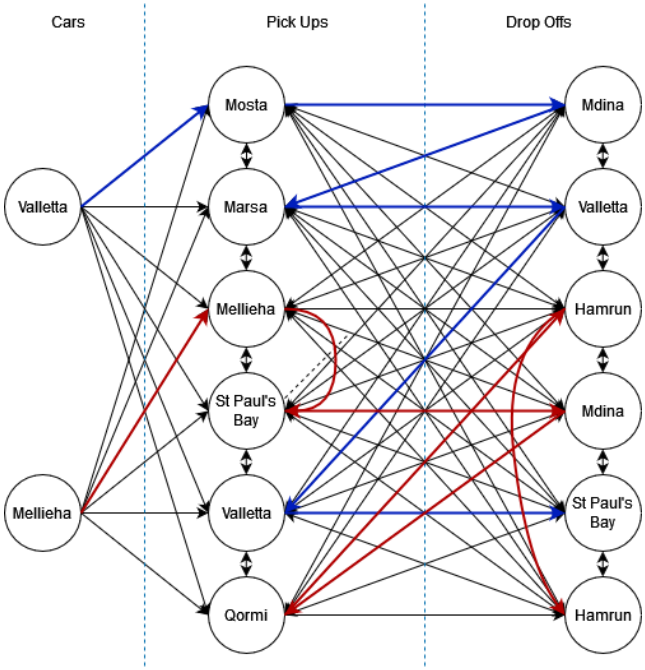

Figure 7: Optimal solution for Duration fitness Function

Finally, a combination of the above two fitness methods was also evaluated. This model weighed the minimum overall time travelled and minimum wait time each at a value of 0.5. In this evaluation the time for a solution to complete was taken instead of the overall distance. Time and distance are directly proportional to one another, and therefore can be used interchangeably. Time was chosen for this implementation as it follows the same units as the wait time criteria, making it easy to combine results. Note this is not necessarily the optimal distribution of overall time and wait time. This will be explored in greater detail soon.

Comparing figures 6, 7, and 8, one can see that given the same network, different fitness functions will not all produce the same traversal. It is interesting to note that the combination fitness model behaves very similarly to the minimum wait time model. This may indicate that the weighted distribution between the two sub functions is skewed towards the minimum wait time model.

An attempt was made at rectifying this. Using a genetic algorithm, the system attempted to find the optimal weights for a specified wait time. The algorithm proved unrefined at best. Given an environment with an insufficient number of cars for the time goal, the algorithm would not be able to converge to the desired time. This led to heavily skewed, and unrealistic answer, where the wait time value approached 1, and the distance weight approached 0. Given a situation where the number of cars was adequate for the specified

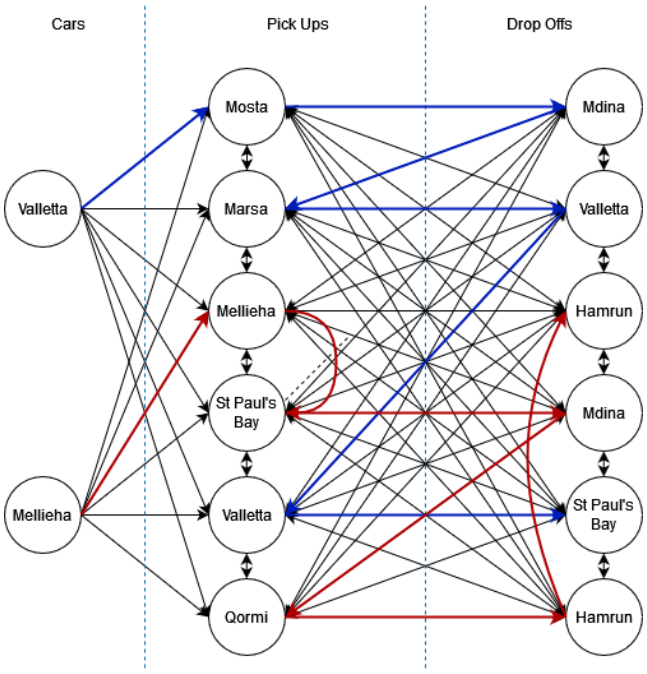

Figure 8: Optimal solution for Composite fitness Function

time, the algorithm produced values within a three second variance.

The algorithm uses the previously mentioned genetic algorithm *(section 2.4)* as the fitness function for its population. This means that for each member of the population a separate genetic algorithm must be run. This drastically increased the run time of the application making it infeasible to run. Combining this with the realization that there is no one set of weights that would match every solution and time frame, this approach did not prove effective.

In future iterations of this project, I would like to revisit this and improve on this idea. Considering the success of weight optimization by genetic algorithms in other areas [10] [11], this approach may prove effective given the right changes.

4.2 Variable Iteration Count

One of the main factors that affects the performance of an AI algorithm is the iteration count. This refers to the number of times a given algorithm is run before providing an answer. The number of iterations an algorithm performs should provide a balance between run time and the optimal solution. In this section we investigate how different iteration counts effect the performance over various algorithms and networks.

Firstly, the Tabu Search algorithm was tested with the network in figure 5. Over the different iteration counts the algorithm had a constant tabu size of twenty and started from the same initial solution.

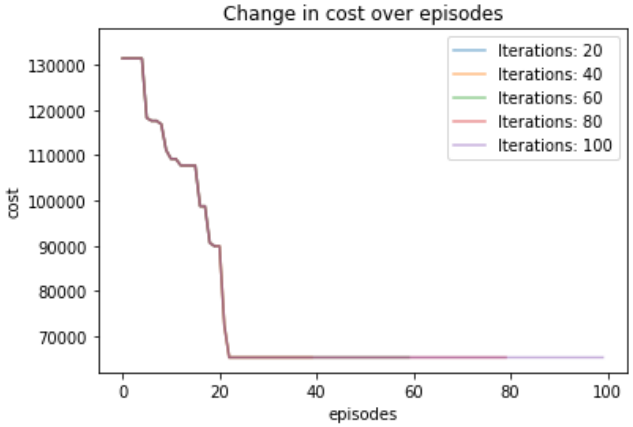

Figure 9: Variable Iteration Tabu Search 1

Given the deterministic nature of the tabu search selection process, and since all instances where run with the same initial solution, all the instances in this test followed the same path. The line for each instance was set to an opacity value *'0.5'*, giving the ability to identify the stopping points of each instance. With this configuration it appears that any iterations over the twenty two mark do not contribute anything to the final solution. To further reinforce this, the test was rerun to reach a maximum of a thousand iterations, with a delta value of one hundred.

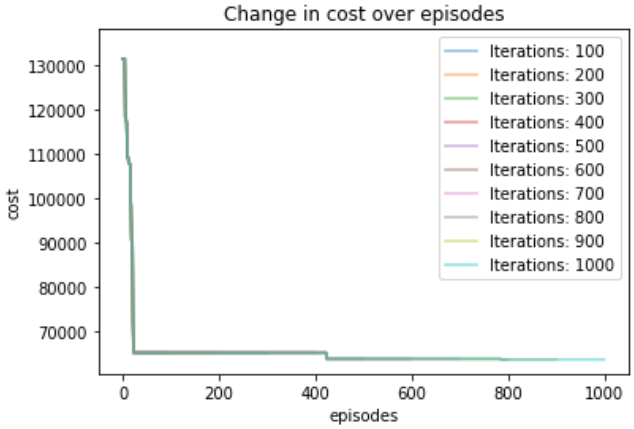

Figure 10: Variable Iteration Tabu Search 2

As can be seen in figure 10, the instances around the four hundred mark make a small improvement on the previous iterations. However, the size of this improvement does not justify the increase in computation time needed. Therefore, for this network the point of diminishing returns is around twenty iterations.

A similar test was repeated with a larger environment containing ten cars and thirty trips. For full environment refer to Appendix B. The instances where run for a maximum of one thousand iterations, with iteration intervals of one hundred, and a tabu size of fifty. The results are reported in figure 11

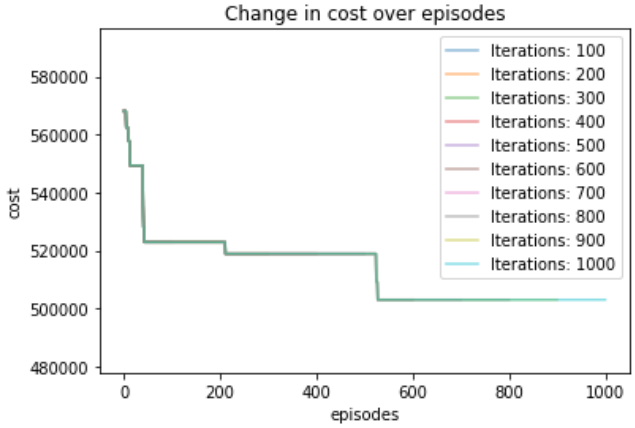

Figure 11: Variable Iteration Tabu Search 3

The results of figure 11 show a greater distribution variance over the one thousand episodes than the results in figure 10. Comparing the two, one can identify that the number of iterations needed to converge on the point of diminishing returns is proportional to the size of the environment. This is reinforced by the percentage change over the first twenty iterations in both environments. In the first, smaller environment, the percentage change over twenty iterations was 53.84%, while the larger environment have a percentage change of only 1.66%. These values where computed using Equation 4.2.

$$p = \frac{V_2 - V_1}{|V_1|}$$

The same two graph networks where run using the genetic algorithm. The tests where run with a population size of one hundred and a max iteration count of 500. The results of these two networks can be seen in figures 12 and 13 below.

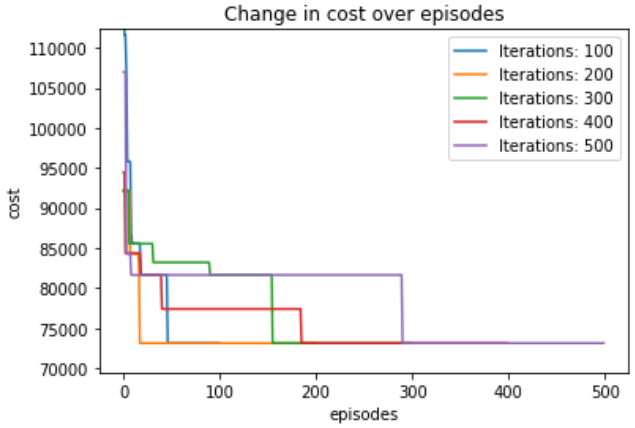

Figure 12: Variable Iteration Genetic Algorithm 1

Unlike the tabu search, the genetic algorithms population is randomly generated at the beginning of each instance. This means that there is a random chance that one instance converges faster than another given the same environment. This can be observed in both figures as the start points of the different instances vary. Note that because of this the

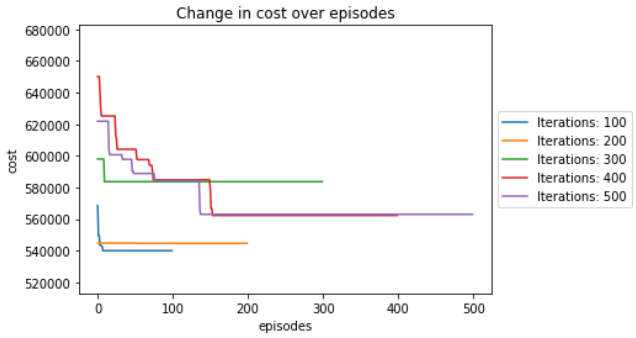

Figure 13: Variable Iteration Genetic Algorithm 2

convergence range is rather large and does not provide any pattern in relation to iteration count.

The stochastic nature of the genetic algorithms initial solution leads it to benefit from a larger number of iterations when compared to tabu search. This however does not mean that there is no point of diminishing returns. The environment in figure 12 can be seen run bellow over the course of two thousand iterations.

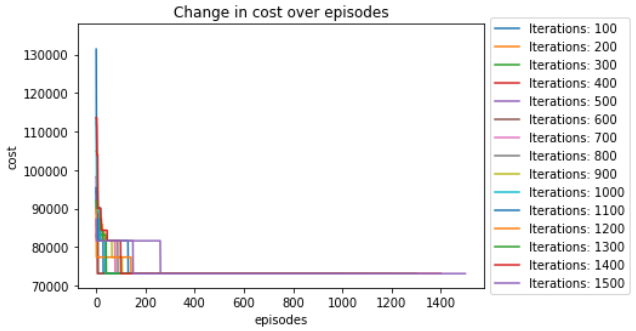

Figure 14: Variable Iteration Genetic Algorithm 3

Note that after approximately three hundred iterations there are no improvements worth the extra iteration cost.

4.3 Variable Population Size with Genetic Algorithms

Similarly, to changing the number of iterations, changing the population size of a genetic algorithm will increase processing time and performance. This is as the algorithm will have to simulate more solutions. To start, environment one *(Appendix A)* was run with a variable population size maxing out at one hundred, over the course of five hundred episodes. Note, this is the point of diminishing returns determined for this environment in *'Variable Iteration Count'* [4.2].

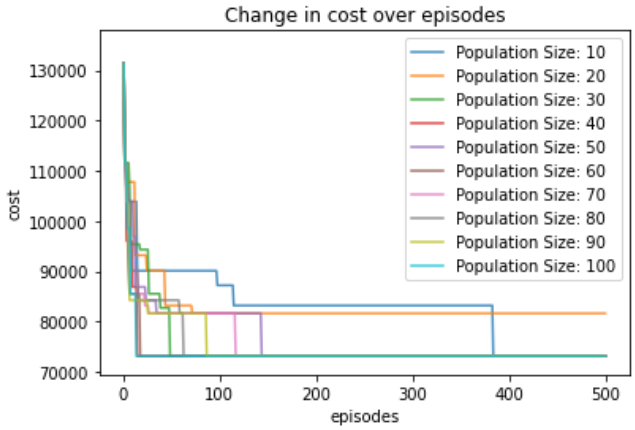

Figure 15: Variable Population Size in Genetic Algorithms 1

From figure 15 one can observe the population sizes of ten and twenty take significantly longer to converge, if at all. The rest of the population sizes all converge between episodes zero and two hundred. Note that in some cases the order of improvement is not proportional to the size of the population. This is due to the stochastic population initialization mentioned in [4.2].

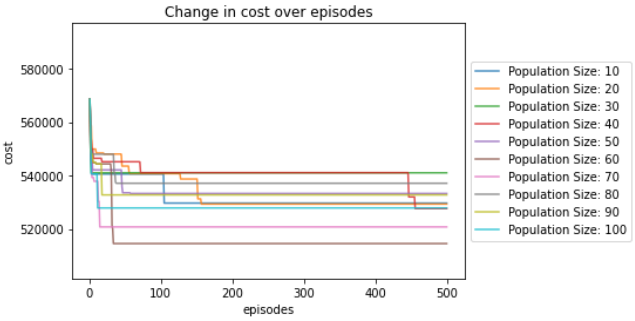

Figure 16: Variable Population Size in Genetic Algorithms 2

Comparing the results of environment one to environment two *(Appendix B)* with the same pre-set, as can be seen in figure 16, one can see that the population size seems to have little to no effect on outcome of the instances. Furthermore, none of the lines converge to the same points. This indicates a lack of iterations and difference in population size.

4.4 Variable Tabu List Size

Unlike changing the population size in a genetic algorithm, changing the tabu size will not increase processing time. The tabu list is an array of swaps that have been recently performed. Therefore, changing the tabu size will increase the amount of exploration done by the algorithm, increasing the probability of exiting a local maximum. A test was run using the first test environment *(Appendix A)* over the course of 100 iterations. The algorithm was run five time, each with the same starting point. Each instance had a unique tabu size, starting form five and ending at 25, with each increasing by a factor of five. The results of this test can be seen in figure 17.

The graph clearly shows an increase in performance with a larger tabu size. As previously mentioned, this is as it

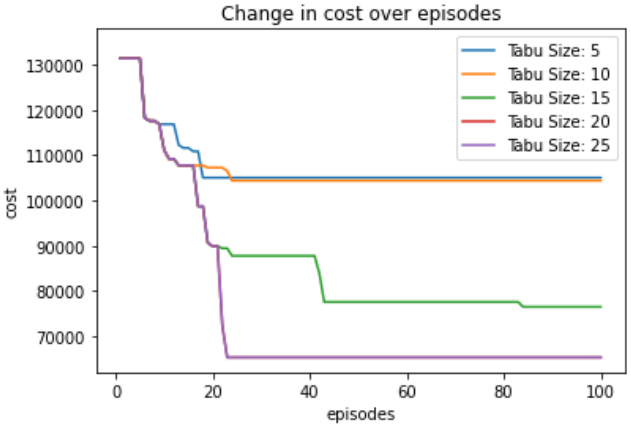

Figure 17: Variable Tabu Size 1

allows the algorithm to exit any local maximums it might encounter. Note, the red line does not appear on the graph as it follows the exact path as the purple line. This indicates that a tabu size of 20 is enough for the given problem. Furthermore, comparing the instance with a tabu size of fifteen, to that of size 25, shows that the larger tabu size converges to its final state faster than lower values.

The test was repeated using the larger test environment *(Appendix B)*. Given the larger environment the test was run for a thousand iterations, with tabu size varying from twenty to one hundred. The results in figure 18 show a similar story. The larger tabu sizes provide more improvements in the long term.

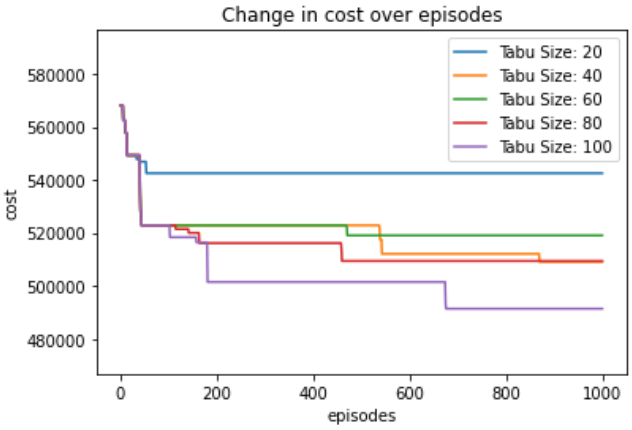

Figure 18: Variable Tabu Size 2

5. CONCLUSIONS AND FUTURE IMPROVEMENTS

Overall, the project provides a basic system for evaluating artificial intelligence techniques with regards to autonomous ride sharing vehicles. It outlines two key techniques that could be used to approximate such a problem. Following are a set of ideas that could help improve such a system given a wider development scope.

5.1 Front End Development

The frond end of this project acts a prototype of what is possible with a system of this nature. While its features are useful, given more time the statistical representation section would provide a more detailed description of the path the algorithms are taking. Above what is currently present, the section intended to house a series of collapsible performance graphs, tables representing the routes of each vehicle, and features allowing comparisons of multiple tests.

Furthermore, the vehicle path visualization mention in section 3.1 requires further tuning in the rotations of the cars. Another feature that would be implemented in future iterations of the project is only showing a vehicles path when the chosen vehicle is clicked on. This would aid in complex networks, as the vehicle paths would be easier to follow.

5.2 Improvement on fitness functions

As mentioned in section 4 above, the fitness function can vary depending on what optimization factor is desired. Assuming a balanced system for both operators and users, the aforementioned genetic algorithm stores potential as a solution. This could be combined with neural networks [10] [12] to solve such problems.

5.3 Other algorithms of note

While this project only focused on two possible techniques, its important to note that these are not the only possible solutions to a problem of this calibre. As mentioned in section 2, other algorithms have been applied to similar projects, yielding promising results. Some examples of these are nearest neighbour, ant colony optimization, and neuro fuzzy, as can be seen in Stanley Glenn E. Brucal, and Elmer P.Dadios' paper on a *'Comparative analysis of solving traveling salesman problem using artificial intelligence algorithms'* [13]. Future iterations of this project aim to implement such techniques and embed their functionality into the front facing UI.

6. REFERENCES

[1] J. Błażewicz, P. Łukasiak, and M. Miłostan, "Application of tabu search strategy for finding low energy structure of protein," *Artificial Intelligence in Medicine*, vol. 35, no. 1-2, pp. 135–145, 2005.

[2] K. Nara, Y. Hayashi, K. Ikeda, and T. Ashizawa, "Application of tabu search to optimal placement of distributed generators," in *2001 IEEE power engineering society winter meeting. Conference proceedings (Cat. No. 01CH37194)*, vol. 2, pp. 918–923, IEEE, 2001.

[3] J. E. Knox, *The application of tabu search to the symmetric traveling salesman problem.* PhD thesis, University of Colorado at Boulder, 1989.

[4] M. Zachariasen and M. Dam, "Tabu search on the geometric traveling salesman problem," in *Meta-heuristics*, pp. 571–587, Springer, 1996.

[5] B. A. AlSalibi, M. B. Jelodar, and I. Venkat, "A comparative study between the nearest neighbor and genetic algorithms: A revisit to the traveling salesman problem," *International Journal of Computer Science and Electronics Engineering (IJCSEE)*, vol. 1, no. 1, pp. 110–123, 2013.

[6] J. Sankaranarayanan, H. Samet, and A. Varshney, "A fast all nearest neighbor algorithm for applications involving large point-clouds," *Computers & Graphics*, vol. 31, no. 2, pp. 157–174, 2007.

[7] Y. Chen and P. Zhang, "Optimized annealing of traveling salesman problem from the nth-nearest-neighbor distribution," *Physica A: Statistical Mechanics and its Applications*, vol. 371, no. 2, pp. 627–632, 2006.

[8] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[9] D. Gong and X. Ruan, "A hybrid approach of ga and aco for tsp," in *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No. 04EX788)*, vol. 3, pp. 2068–2072, IEEE, 2004.

[10] B. Freisleben and M. Härtfelder, "Optimization of genetic algorithms by genetic algorithms," in *Artificial Neural Nets and Genetic Algorithms*, pp. 392–399, Springer, 1993.

[11] S. Y. SERT, Y. Ar, and G. E. BOSTANCI, "Evolutionary approaches for weight optimization in collaborative filtering-based recommender systems," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 3, pp. 2121–2136, 2019.

[12] D. Whitley *et al.*, "Genetic algorithms and neural networks," *Genetic algorithms in engineering and computer science*, vol. 3, pp. 203–216, 1995.

[13] S. G. E. Brucal and E. P. Dadios, "Comparative analysis of solving traveling salesman problem using artificial intelligence algorithms," in *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–6, IEEE, 2017.

APPENDIX

A. ENVIRONMENT ONE

Following is Environment One used in testing:

Table 2: Environment One: Cars

Car Index	Location
1	Valletta
2	Mellieha

Table 3: Environment One: Trips

Trip Index	Start Location	End Location
1	Mosta	Mdina
2	Marsa	Valletta
3	Mellieha	Hamrun
4	St Paul's Bay	Mdina
5	Valletta	St Paul's Bay
6	Qormi	Hamrun

B. ENVIRONMENT TWO

Following is Environment two used in testing:

Table 4: Environment two: Cars

Car Index	Location
1	Selmun
2	St Pauls Bay
3	Mgarr
4	Mdina
5	Had-Dingli
6	Haz-Zebbug
7	Balzan
8	Pieta
9	Tarxien
10	Hal Safi

Table 5: Environment two: Trips

Trip Index	Start Location	End Location
1	Mosta	Mdina
2	Haz-Zabbar	zejtun
3	Gzira	Msida
4	Birkirkara	Attard
5	Marsa	Valletta
6	Naxxar	Pembroke
7	Mellieha	Hamrun
8	Iklin	Lija
9	Sqieqi	Paceville
10	Pembroke	Gzira
11	St Pauls bay	Mdina
12	Sliema	Kappara
13	Saint Julians	Mqabba
14	Valletta	St Pauls bay
15	Kalkara	Qrendi
16	Zejtun	Smart City
17	Qorm	Hamrun
18	Kirkop	zurrieq
19	Marsaskala	naxxar
20	Siggiewi	Zurrieq
21	Gudja	Marsaxlokk
22	Sliema	Mellieha
23	Zurrieq	siggiewi
24	San Gwann	paola
25	Siggiewi	Attard
26	Mellieha	Paola
27	Santa Lucija	Marsa
28	Mosta	Iklin
29	Buggibba	Mgarr
30	Mellieha	Mgarr