CHALMERS
UNIVERSITY OF TECHNOLOGY

The Cooperative Sorting Strategy for Connected and Automated Vehicle Platoons

Online Processing A* Algorithm

Master's thesis in Infrastructure and Environmental Engineering

Xiangyu Kong

Department of Architecture and Civil Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Abstract

Along with the fast development of urbanization, the optimization of transportation plays an increasingly important role for solving traffic jam problem. Based on the assumption of fully connected and automated vehicles(CAVs), the optimal control and design of CAVs is one of the most effective ways for improving traffic efficiency as sorted CAVs platoons could help optimise the capacity of intersections. This research focus on optimizing the sequential actions of vehicle platoons for this given aim state. With the help of the perception and localization, multi-line vehicle platoons are abstracted into the permutation. The optimization object is the length of the trajectory from start permutation to the target permutation. A* algorithm wrote in Python could perform better with the help of the hash table and PyPy3. However, the time complexity problem still exists, which blocks it from extensive applications.

Model-free Reinforcement Learning (RL) method suffers from the large searching space and low sampling efficiency a lot and Mento Carlo Tree Search (MCTS) lacks suitable criteria. In this essay, online processing A* (OPA*) algorithm is original promoted for solving time complexity problem with the sacrification of the optimality. The comparison and evaluation of OPA* and A* methods mainly focus on the performance and time consumption. OPA* could give stable and scalable results which make it possible for industrial usage.

Keywords: sorting, hash table, heuristic algorithm, reinforcement learning, online processing A*

Acknowledgements

Thanks Jiaming Wu's generous help, thanks Xiaobo Qu offering me this oppotunity.

Xiangyu Kong, Gothenburg, 06 2020

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

Along with the fast development of urbanization, transportation engineers play increasingly important roles, especially in alleviate traffic jams. The traffic congestion has been a huge problem worldwide and lead to a 272 hour lose and a $2,291 cost per car at most.[28] There are two widely applied approaches for easing the traffic congestion, either extending road network or introducing new technologies.[4]

Final solutions for improving the performance of whole traffic system could include both these two methods. However, frequently occurred Braess' Paradox may lower down the effect of the extended, high-cost road network.[33] In contrast, new technologies do not need huge investment for eliminating traffic jam. So, the later approach could be a more cost-effective option than the former.

To enlarge the capacity of a intersection, several approaches are proposed. Among them, Kurt Dresner and Peter Stone found that reservation-based system or Autonomous Intersection Management (AIM) could improve the traffic intersection using efficiency for 200 to 300 times comparing to the conventional traffic lights.[7] Matthew etc. paper illustrate the multi-intersection AIM using time-based A star will have significant effect on improving traffic efficiency.[16] However, all these experience based on the random generated vehicle platoon and do not sorting the platoon before joining intersection. According to Weili Sun's research, the total improvement of the efficiency for a given vehicle platoon with different purposes could be maximized and the final solution is converged to the global optimal through grouping.[34] This research is to make the sorting or grouping occur in the shortest time or shortest trajectory. Most cases, the traffic light based methods are better than the reservation based intersection methods, and the main contribution for the time reduction mentioned in the reservation method is the shorten headway, which also cause a lot of problems.[22]

Based on the low latency communication between automated vehicles, this essay concentrates on shorting the decision sequence for vehicle platoons in control scenario. The solution also could be transferred to the unmanned storage problem due to the similarity of internal solving logic. Reducing actions that are not necessary will help to gain both ecological and economical benefits. Also, road safety will witness a clear increase as self-driving excludes human drivers, which are one of the

main contributors for road accidents.[11]

The cooperative sorting strategy for the vehicle platoons is a NP hard problem with numerous searching space. Traditional methods are focus on the heuristic algorithm like A* or IDA* because these algorithms could give global optimal solutions. However, they also have the exponential time complexity, and it is quite unacceptable when solving relatively big data.

Timeliness is critical for problems with big data. The value of the big data is fleeting, leaving little time for processing. With the linear increase of the problem's scale, the time for solving it increase exponentially.

1.2 Variants of A* Algorithm

For the sequential decision making problem, as the decision chain could be considered as trajectory. For the improvement of path-finding algorithms, common methods are basically from the perspective of software or the algorithm itself. The preprocessing algorithm adds a preprocessing step, and the jump point algorithm uses the characteristics of symmetry.

Moreover, there have been little research on sorting puzzle-like car platoons problem with implicit representations of huge search spaces. More widely, path-planning problems are similar to this problem and A* algorithm also has been widely explored. Main variants of A* in the path-planning domain are also included.

1.2.1 Jump Point Searching Algorithm

Jump Point Searching (JPS) algorithm is an extended version of the A* algorithm with the help oh the online pruning.[13] JPS could improve the efficiency of the A* algorithm to several orders of magnitude.[14] It focus on reducing the scale of the open list and the structure for this method is shown below:

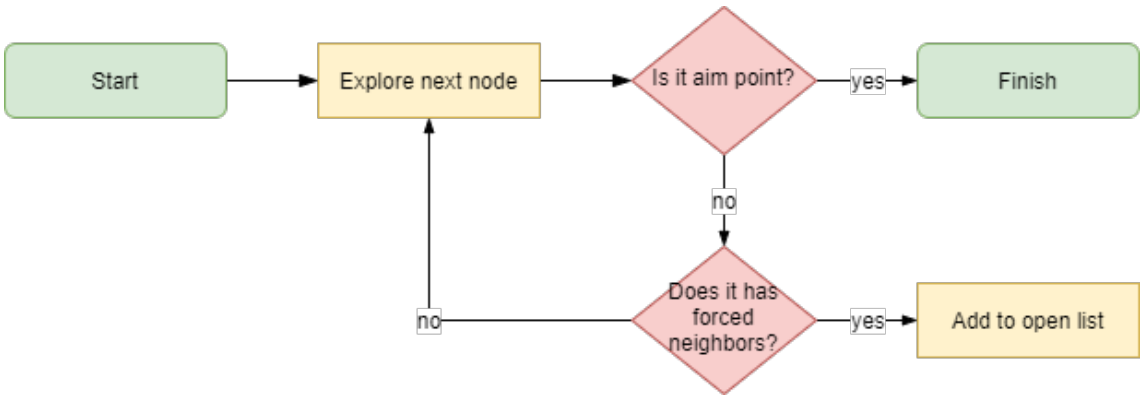


Figure 1.1: Structure of the Jump Point Searching Algorithm

This heuristics method has a strong assumption of the grid maps. In other words, it requires the symmetry property. However, in our problem, the child nodes are asymmetry because of the continuous changing space-action spaces.

1.2.2 Preprocessing Algorithms

Besides the JPS algorithm, preprocessing algorithms are used commonly. One of these is call ALT(A*, Landmark, Triangle Inequality) algorithm, it could improve the performance to more than one order of magnitude.[10] Based on the shortcuts, contraction hierarchies(CH) algorithm is promoted[8] before the hub-based labeling(HL). HL could be six orders of magnitude faster than Dijkstra's algorithm for random (long-range) queries.[1]

The structure of preprocessing algorithm is shown below:

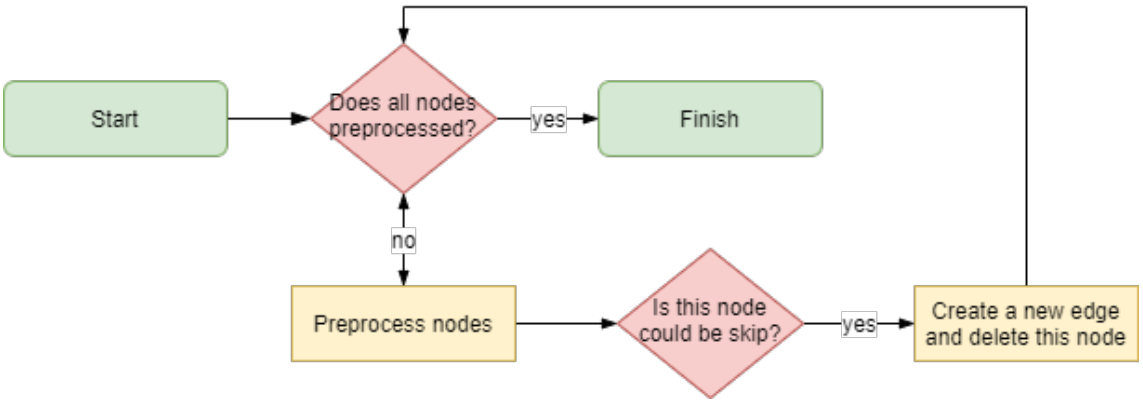


Figure 1.2: Structure of the Preprocessing Algorithm

As the name indicating, this method requires preprocessing for nodes. The time for one preprocessing ranging from minutes to hours.[1] This is not suitable for a dynamic scenario because the preprocessing has to be done online.

Also, as most of these methods' performance based on the static environment with a fixed number of nodes. In this essay, with the increasing search space, the Dijkstra algorithm is almost faster than the preprocessing algorithms with five orders of magnitude.[3]

1.3 Reinforcement Learning Approach

Naturally, in an age of machine learning, a graph-theory-related sequential decision-making question is primarily a problem that reinforcement learning has a potential to solve. Previous studies in the microscope longitudinal and lateral platoon control area showed few investigations for the interaction with the machine learning

method. However, Model-free RL method always shows good performance with expensive data gathering.[6] RL focuses on how to balance the relationship between exploration and utilization for the agent.[19] Most RL algorithm derived from Markov Decision Process (MDP).[35] The relationship between states and actions (value-based) is extremely important for the agent. RL methods without considering this relationship, in other words, policy-based method which considering whole decision-making process together, show little probability in solving this problem.

1.4 Monte Carlo Tree Search Approach

Similarly to this problem, Rubik's Cube also have large configuration to explore. With 44 hours' training, the method of the RL and MCTS could perform a 10-minutes median solve time comparing with the one using professional knowledge(within 1 second).[25]

By modifying the strategy of selection and backpropagation, the method Single-Player Monte-Carlo Tree Search (SP-MCTS) could solve problems similar with A* algorithm, and it could perform better when the environment does not have a specific and accurate and admissible evaluation function.[29] However, little results support SP-MCTS could perform better than A* when the heuristic function is given, in this essay, the Menhaden distance or norm 1.

In the Alpha Go, only the rollout itself could score 1457 with a time cost three milliseconds.[32] The performance without the MCTS strategy is also super good . The fast rollout algorithm may also be the key factor for the success of Alpha Go instead of the MCTS-based algorithm. Most importantly, this tailor-made fast rollout algorithm does not have the ability to generalize to other problem.

1.5 Improvements for the A* Algorithm

Suitable data structures and compiling processes also influence the speed a lot. All these solutions will be tested on a Linux distribution (Ubuntu) because in the real life, all these software will be performed on Linux as well. With the help of the hash table, the time for the operation of adding, deleting, indexing, etc. of a large amount of global data that A * needs to process every cycle is reduced to O(1). After the introduction of the hash table, the main problem of the algorithm is not the operation within the loop, but the times of the loop.

In addition to the data structure, the speed of the algorithm is also related to the language in which the algorithm is implemented. Under the same data structure, different high-level languages, the speed of the algorithm is also different. However, this improvement is less obvious than the previously mentioned approaches. Due to the existence of the Global Interpreter Lock (GIL), Python cannot implement

true multi-threaded operations. PyPy3 can use Just In Time (JIT) to improve the performance of Python several times by optimising the runtime. In this article, all scripts are based on PyPy3 except for the control group.

1.6 Online Processing A* Algorithm Approach

For a practical optimal solution, the speed of getting this solution is more important than getting a global optimal one. In this essay, a new method called online processing A*(OPA*) algorithm is promoted. It solve the time complexity problem with stable and acceptable solutions. OPA* focus on a online sorting strategy where A* is used for solving sub-problems.

2

Theory

2.1 Reinforcement Learning

Reinforcement learning objective:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}] \tag{2.1}$$

2.1.1 Markov Decision Process

Basic reinforcement learning, also known as approximate dynamic programming, or neuro-dynamic programming, is modeled as a Markov decision process:

- S: a set of environment and agent states;
- A: a set of actions;
- $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$: probability of transition at time t from a state s to another state s' under action a;
- $R_a(s, s')$: immediate reward after the transition from s to s' with action a;
- O: An agent's observability of the environment. (full observability or partial observability)

The goal of a reinforcement learning agent is to collect as much reward as possible. When comparing the performance of an agent with the best performing agent, the difference in performance leads to an increase in regret. In order for the conduct to be close to the best, although the direct compensation associated with it may be negative, the agent must reason for the long-term consequences of its actions (i.e., maximize future income).

In this research, this problem can not model as Markov decision process. The former node will determine the later node directly, which conflicts with the assumption of Markov property. So, the model-free RL method will be tested.

2.1.1.1 Conditional Independence

In probability theory, two random events, A and B, are conditionally independent in the case of a third event, C if A and B are stand-alone events in its conditional probability distribution. In other words, under the assumption of C occurs, the knowledge whether B occurs does not provide information about the likelihood of

occurrence of A. Formally:

$$(A \perp\!\!\!\perp B)|C \Leftrightarrow P[A \cap B|C] = P[A|C]P[B|C] \tag{2.2}$$

2.1.1.2 Markov Property

Given the present state, if it is conditional independent of the previous state, then it satisfy the Markov assumption.

MDP is the cornerstone of the current theory of intensive learning, for intensive learning, the decision-making process of Markov is generally used as a means of formality. That is to say, for the vast majority of current intensive learning algorithms, only can abstract the problem into MDP can ensure the performance of the algorithm (convergence, effect, etc.), for the problem that violates MDP does not necessarily ensure that the algorithm is effective, because its mathematical formula is based on MDP for derivation. Formally:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t] \tag{2.3}$$

2.1.2 Q Learning Algorithm

In Q learning, the agent's experience includes a series of episodes. In episode n, the agent using following algorithm to update[36]:

$$Q_n(x,a) = \begin{cases} (1\text{-}\alpha_n)Q_{n-1}(x,a) + \alpha_n[r_n + \gamma V_{n-1}(y_n)] & \text{if } x = x_n \text{ and } a = a_n, \\ \mathrm{Q}_{n-1}(x,a) & \text{otherwise,} \end{cases} \tag{2.4}$$

where

$$V_{n-1} \equiv \max_b Q_{n-1}(y,b) \tag{2.5}$$

2.1.3 Policy Gradient Algorithm

The policy gradient(PG) methods focus on modeling and optimizing the policy $J(\theta)$ directly. The probability of one decision sequence τ is:

$$p_\theta(\tau) = p(s_1)p_\theta(a_1|s_1)p(s_2|s_1,a_1)p_\theta(a_2|s_2)p(s_3|s_2,a_2)... = p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t) \tag{2.6}$$

The sum up of the reward is:

$$\overline{R}_\theta = \sum_\tau R(\tau)p_\theta(\tau) = E_{\tau\sim p_\theta(\tau)}[R(\tau)] \tag{2.7}$$

The gradient of the parameter is calculated by using the above two formulas to replace the following formula:

$$\triangledown f(x) = f(x)\triangledown log f(x) \tag{2.8}$$

The result is:

$$\nabla \overline{R}_\theta = \sum_\tau R(\tau) p_\theta(\tau) \tag{2.9}$$

Assume the sampling time is N, by using the average of N sampling to approximate mathematical expectations. The approximation for the expectations is:

$$\frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla log p_\theta(\tau^n) \tag{2.10}$$

With the help of the following formula:

$$\nabla log p_\theta(\tau^n) = \nabla log p(s_1) \prod_{t=1}^{T} p_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2.11}$$

By eliminate log symbol, the continued multiplication is changed into continued summation:

$$\nabla log p_\theta(\tau^n) = \nabla \sum_{t=1}^{T} [p(s_1) + p_\theta(a_t|s_t) + p(s_{t+1}|s_t, a_t)] \tag{2.12}$$

Items that are not related to the gradient sought are considered constant. Because the gradient of the constant item is zero, the formula can be simplified to the following formula:

$$\nabla log p_\theta(\tau^n) = \sum_{t=1}^{T} \nabla p_\theta(a_t|s_t) \tag{2.13}$$

The approximation for the expectations could be simplified as:

$$\nabla \overline{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla log p_\theta(a_t^n|s_t^n) \tag{2.14}$$

2.1.4 Proximal Policy Optimization Algorithms

One of the big disadvantages of the PG method is that parameter updates are slow, because re-sampling is inevitable for each parameter. Actually the strategy for PG's updating is on-policy and the agent being trained is the same agent that interacts with the environment. If the experience save by other agents could be used for training this agent, the efficiency could be improved a lot.

One big drawback of the Policy Gradient method is that the policy update could be very large.[31] To overcome this, TRPO limits the scope of policies that can be changed in each iteration through "surrogate" objective based on KL divergence. KL dispersion can be used to measure the degree of difference between two distributions.

$$D_{KL}(P||Q) = \mathbb{E}_x log \frac{P(x)}{Q(x)} \tag{2.15}$$

Instead of directly deploying KL divergence, PPO promotes clipped surrogate objective to simplify computation while maintaining the same functionality.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.16}$$

2.1.4.1 Importance Sampling

The expectation could be figured out by integration. However, if direct integration is difficult to achieve, using another simpler distribution to replace it could work. This is the original purpose of the importance sampling.

$$\begin{aligned} E_{x\sim p}[f(x)] &\approx \frac{1}{N}\sum_{i=1}^{N} f(x^i) \\ &= \int f(x)p(x)dx \\ &= \int f(x)\frac{p(x)}{q(x)}q(x)dx \\ &= E_{x\sim p}[f(x)\frac{p(x)}{q(x)}] \end{aligned} \tag{2.17}$$

where $\frac{p(x)}{q(x)}$ is called importance weight.

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) composed by 4 parts:[5]
1. Selection
2. Expansion
3. Simulation
4. Backpropagation

2.2.1 Upper Confidence bounds applied to Trees(UCT)

Through UCT, MCTS realizes the balance of node access times and values, and enhances the robustness of the algorithm.

$$\frac{w_i}{n_i} + c\sqrt{\frac{lnN_i}{n_i}} \tag{2.18}$$

where:
$\frac{w_i}{n_i}$ is the win ratio for noden_i;
C is a constant to balance the exploration and incorporates the $\sqrt{2}$ from the UCT formula;
N_i is the visited times of parent node;
n_i is he visited times of children node.

2.2.2 Single Player-Monte Carlo Tree Search

However, comparing with the traditional MCTS for 2 players, single player MCTS or SP-MCTS is modified accordingly. To begin with, for the selection part, the

conventional assumption of the UCT needs to be adjusted since the score range will no longer be negative. To fix this, the [-1,1] interval will be transferred to a range from o till a certain number of reward, or normalise the reward to the [-1,1].

$$\overline{X} + C\sqrt{\frac{int(N)}{tN_i}} + \sqrt{\frac{\sum x^2 - t(N_i)\overline{X}^2 + D}{t(N_i)}} \tag{2.19}$$

where:
$t(N)$ is the visited times for parent node N;
$t(N_i)$ is the visited times for children node N_i;
$\overline{X}$ is the mean value from playing game;
D is a constant to balance exploration.
The flow chart for SP-MCTS are shown below:

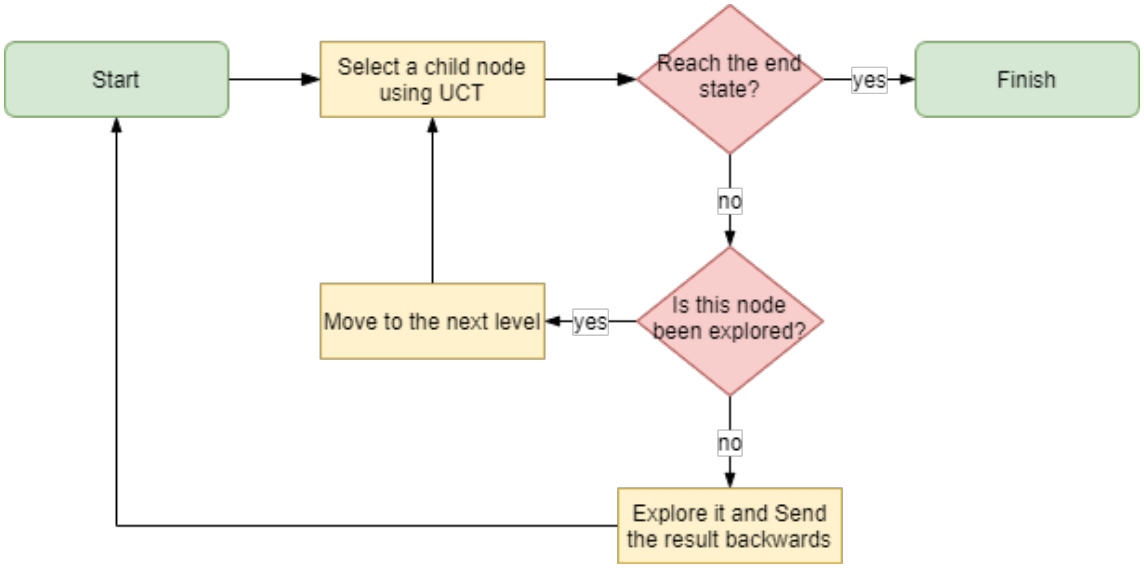


Figure 2.1: Structure of the SP-MCTS Algorithm

Other selection method also adapt heuristics algorithm like All-Move-As-First [17] or Rapid Action Value Estimation [9] to speed up the whole MCTS process. These method have a potential improvement for SP-MCTS as well. Also, instead of the win or loss directly obtain after the playout, the score calculation will be adjusted to the SP-MCTS according to the specific environment.Expanding strategy is not changed, also follows the principle, first encounter, first expand. Simulation strategy is a quick solving the problem. In the Alpha Go, the simulation algorithm itself performs pretty good with 1ms. In this research, the fast rollout could not be performed. Before reaching an end, all actions are selected based on the heuristics function and in this case, they are pseudo-randomly selected. [29] Backpropagation also continue using the average score. Not only the average score and the historical high score, but also the sum of the squared results corresponding to the modified

UCT mentioned before.

2.3 A* Algorithm

The vertical or horizontal movement are valued equal as 1 in this case for simplifying the problem. Each car is regarded as a single tile.

2.3.1 Consistency and Admissibility

The guiding heuristic function in A* is:

$$f(N) = g(N) + h(N) \tag{2.20}$$

where

- g(N) is the depth of node N
- h(N) is a heuristic value of node N

Two requirements for selecting heuristic functions are called admissibility and consistency. The conditions for $h(n)$ being consistent, or monotone, is:

$$h(N) \leq c(N, P) + h(P) \tag{2.21}$$

and

$$h(G) = 0 \tag{2.22}$$

where

- P is any subsequent node in the graph
- $c(N, P)$ is the cost from N to P

An admissible $h(N)$ means it will never overestimate the cost for reaching the goal G.[21] If reduce the cost to:

$$c'(N, P) = c(N, P) + h(P) - h(N) \tag{2.23}$$

The A* is equivalent to Dijkstra's algorithm.[15]

2.3.2 Complexity

The time and space complexity for A* are:

$$O(w^d) \tag{2.24}$$

where

- w is the maximum width of the A* searching tree
- d is the maximum depth of the A* searching tree

IDA* share the same time complexity as they both use the same heuristic functions, but in general IDA* will be slower as it ends up exploring the same nodes many times. The difference between A* and IDA* is the latter has a less space complexity:

$$O(wd) \tag{2.25}$$

The distance in the A* algorithm is using Menhaden distance. Among all distance measures, Menhaden distance, or known as norm 1, is the largest one. As A* algorithm could be faster with larger heuristics value, using Menhaden distance as heuristic function will fast the calculation process.

2.3.3 Bidirectional A* Algorithm

Bidirectional A* Algorithm will shorten the depth of the searching tree thus speed up the whole simulation process. This will probably not work if the meeting point for two A* algorithm is not locate in the middle. The worst case for the time complexity will be worse than original A*. Most importantly, it doesn't solve time complexity from the root.

2.4 Hash Table

The cooperation of algorithm with proper data structure largely determines the performance of the algorithm. Common data structures are arrays and list. Arrays are characterized by: easy to address, difficult to insert and delete; While the list is characterized by: difficult to address, easy to insert and delete. Is there a way to cover their pros and not their cons, like a data structure that could easily addressed, insert and edited? The answer is yes, that is the hash table.

Hash is to find a mapping relationship between data content and data storage address. The principle of the hash table is to convert the key into an integer number through a fixed algorithm function(hash function), and then take the number to the length of the array, and take the result as the subscript of the array. Store value in the array space with the number as the subscript.

Hash table uses hash technology to store records in a contiguous piece of storage space and it is used in data storage and retrieval applications to access data for a small and almost constant period of time per retrieval.

When querying using a hash table, it uses the hash function again to convert key to the corresponding array subscript and locate the space to get value, so that users can take full advantage of the positioning performance of the array for data.

Using the hash table, it is possible to search for n recorded files at once and the time is independent to n. Hash tables do not slow down due to the increase in the size of the problem.[24]

In Python, hash collision is avoided by open addressing (quadratic probing). The retrieval of data could enjoy a time complexity of O(1), that is delete, insert and select.

3
Methods

This research transforms the relationship between car platoons to the matrix for simplifying this microscope longitudinal and lateral control problem. Moving the tiles represents the moving of CAVs and using the cooperate strategy to denote the low-latency synchronized information exchange process. In the binary world, each car has its unique correspondence number. Finally, the multi -platoons turn to be a string or a list. Each state represents a unique node in the graph model, and the corresponding graphic edge is the selected cost function. To optimize the results of this graphic model, the study choosing both steps and time as the cost function.

3.1 Implementation of Reinforcement Learning

Assuming a finite Markov decision-making process (FMDP), Q-learning agent will find an optimal strategy which could converge to the expected maximum reward value.[26] With unlimited time for exploration and partially random strategies, Q-learning can determine the best action selection strategy for any given FMDP.[26]

RL approach mainly concentrate on making a balance between exploitation and exploration. Too much exploration will slow down the learning approach and too less exploration may lead to local optimal solution instead of global optimal solution. The converge for the highest reward is the counterpart of a global optimal solution in A* algorithm.

Generally speaking, the assumptions for the RL themselves are the most serious problem for the RL. Markov property is too strict for most of problems in real life, including the problem in this paper. Although the value-based reinforcement learning have the potential in getting an useful solution, the output trajectory will also not be so good due to the limitation of learning information. The agent only receives how useful its actions could be, but not the best action to take in any situation.[30]

3.1.1 Reward Function

This could be the most well-known problem in reinforcement learning, as well as how unstable the RL model could be. Several challenges are found in these problem:

1. If the punishment is to easy to get, the agent will grasp this punishment and end the game before learning the sorting algorithm, like always crashing.

2. The same situation like 1 will happen if the positive reward is too hard to get.(Or much harder than the punishment)

3. Menhaden distance is always lower than the length of the shortest trajectory, in some special cases, the next step must be negative. In other word, sometimes the increase of total Menhaden distance is unavailable. Thus, the negative value is not reasonable for actions that enlarging total Menhaden distance.

4. Average award seems to be a nice choice because when the reward is negative, the steps will be enlarged to learn more(increase the denominator of a negative number) and when the reward is positive, it will short the trajectory automatically(decrease the denominator of a positive number). However, the agent will try to learn as much steps as possible for enlarging the reward when using this adaptive method. When the searching space is 665280*24, it will be super hard to even finish a train and get a single reward.

5. Repeating actions (cheating) exists when the combination of actions gives a not really bad combined-reward.

6. If punish the agent too much, and it happen too often, the according to 1, agent just feel comfortable with this punishment.

A delicate and tailor-made reward function is necessary in this case. However, to get a reasonable solution, the reward function will always seems to be not good enough.

3.1.2 Implementation of Proximal Probability Optimization Algorithm

Deepmind and openAI promoting the Proximal Probability Optimization (PPO) method in 2017[31] In this paper, the PPO show its potential in solving hard task. PPO using probability approach to choosing the learning rate. When the method is too bad which often happens in the early stage, increasing the learning rate. And when the method is nearly the global optimal solution, constraining the learning rate in case of passing the solution.

The main idea in this method is the information entropy or KL divergence, or their close approximate, clip method. All these method providing an approach to restrict the changing rate of learning rate.

Instead of the Q-table, the PPO adapting a Neutral Network for storing training data. In practice, the training process focus on the parameters of the NN. Finally, the NN will gain the ability of choosing the best performance from all actions and

give a sequential decision making solution directly after the input original location.

The training time will be take into account, when the training time pass 10 mins, it is regarded as not possible in the real case. Because the start state could be regarded as different every time, so the training time should be take into account. As the real time limit will be 1s, including the time for giving a solution. So the training time should not overpass 1s in general. 10 mins limit is set for RL training. Also, it is not possible to determine if the RL algorithm works or not before converging. With time limits, if the time pass this limit, RL method could be regarded as a fail.

Through import PPO method, the agent should learn to change the learning rate according to the conditions. Like mentioned before, the clip method or KL divergence could smartly help the agent to choose better learning rate according to different conditions. When the condition seems to give more information, from the information theory perspective, the KL divergence will become larger. As the clip method is an approximate to the KL divergence, they should have a same results.

3.1.3 The Environment for Reinforcement Learning

Then, the state space for agent to learn could be counted through permutation formula:

$$A_{2n}^{n} = \frac{(2n)!}{n!} \tag{3.1}$$

The total available spaces are $2n$. The aim is to allocate n cars to only 1 aim locations. The action space is $4n$. For each car, 4 actions are available. Namely: acceleration(moving top), deceleration(moving down), turning left and turning right.

The problem of finding an optimal decision chain is transferred to choose the best action sequence from all possible action arrangements. The length of action sequence is unknown. The global optimal solution corresponding to the shortest length$x$. Assume the length of search is l and $x \leq l \leq +\infty$. Total sampling space for RL agent is $(4n)^l$. If the training time increase, before getting a solution, the sampling space is:

$$\lim_{l \to +\infty} (4n)^l \tag{3.2}$$

For $n > 0.25$, the sampling space is $+\infty$. Considering this continuous increasing sampling space is impossible for any agents to learn, RL methods need to limit the length of decision sequence to xn, the sampling space will be $(4n)^{xn}$. Then times needed to sample for 1 time is

$$\frac{(4n)^{xn}}{xn} \tag{3.3}$$

Assume $x = 10$ and $n = 6$, the number of parameters need to be approximate is$6.5 * 10^{82}$,in comparison, it is estimated that the there are between 10^{78} to 10^{82} atoms in the known, observable universe. The sampling time required for one complete sample is $1.08 * 10^{81}$ times. The implementation of RL is probably impossible.

MARL uses decentralised actor network for choosing actions and a centralised critic network to assign a global reward [23], is suffering from the curse of dimensionality and inefficient sampling [20]. In general, MARL is considered as a possible way to shrink the action space. QMIX discovers that the relationships between action-value function are informative and previous usage of this information is insufficient. Using NN to represent Q information from agent networks instead of directly sum-up, QMIX shows the superior ability to improve the leaning performance with the increase of state information. [27] Mean Field Multi-Agent RL simplifies the relationship between agents to 2 agents using Mean field theory. However, the curse of the dimensionality causing by the increasing number of agents in interactions also make the training process far away from convergence in MARL environments. [38] However, the value network becomes even more complex if the value network gives a approximate value for cooperative method. In this case, the problem of time complexity also exist.

Reinforcement Learning Upside Down(RLUD) provides a new structure for the RL problem by changing the reward to the actions, in other words, input a reward and output an action. [30] This solves the problem when RL-used approximate rewards are not good enough. Also, this change could not solve the time complexity problem as the scale of the state-action-reward space is still the same.

With around 18 million frames, DQN agent could reach the same level as human. [18] However, the fact is human could play this kind of easy game within minutes, but cannot solve NP hard problem within minutes. The super low sampling efficiency of RL largely brings down the probability of solving NP-hard problem within reasonable time.Considering a fully sharing-information environment, when the multi-agent reinforcement learning only consider the global cost, it becomes a cooperative multi-agent reinforcement learning and simplifying it as a single (global) Markov decision making (MDP) process in this particular case.

3.2 Implementation of SP-MCTS

The potential of solving this kind of problem with the help of MCTS by using SP-MCTS. Some problems exposed and hinder the success of these random approaches. Unlike the scores for games, the heuristics function here is Menhaden distance, however, the change of Menhaden distance can not be regarded as a good imitation for the reward. And the random method itself make the converging much more difficult as the 'better' node are not supposed to be selected. However, from the theorem of A star, all these node are supposed to be included in the shortest path. And if one of them are neglected, not only the best solution will be missed, but also missing even possible 'bad' solutions.

One way of the using Menhaden distance is to use it directly. By summing up the children's Menhaden distance and mark parent nodes with these information, and then using a modified UCT selecting method to select the node with a lower sum-

up.
However, sometimes the optimal node has an increase of Menhaden distance. A traversal for all solutions is inevitable when the criteria is dynamic. In other words, whether a node is good or not is determined by the solution instead of other criteria.

Instead of using Menhaden distance, the derivation of Menhaden distance, the sum-up of the children's Menhaden distance. the sum-up of children's Menhaden distance's derivations and even using the optimal child node's Menhaden distance's derivation are tested, and all of these criteria are not suitable for all nodes. In other words, there is no clear criteria that works for all nodes and the roll out for the problem is impossible, The MCTS is not able to perform as the lacking criteria for distinguishing between good and bad nodes.

3.3 Implementation of Hash Table and PyPy3

To speed up the heuristics searching process, the main problems that causing the low efficiency need to be identified.

First, the maintenance of the open list and the close list will repeatedly doing operations like adding node, selecting node and deleting node. Traditional data structure like adjacency matrix or adjacency linked list suffering from the average time complexity of $O(n^2)$, where n is the node number, e is the edge number. Binary heap could promote it to the level of $O(nlogn)$. Hash table could perform even better, with the time complexity of $O(1)$.

Second, the for loop used in the heuristics algorithm also takes a lot of time in a dynamic coding language like Python as the for language will be interpreted many times. To overcome this deficiency, either rewrite the code in a static language like go, or with the help of JIT(just in time) like pypy.

Each of these two measures will improve the performance to one order of magnitude at least. In total, especially facing large insert data, the performance of algorithm will be improved to a large extent.

With the help of heuristics function, the best path is found by continuous trial and error for all nodes. However, there is no clear feature for nodes on the path before getting the solution. In this case, the time complexity problem still exists.

3.4 Implementation of OPA* Algorithm

3.4.1 Basic Assumption

If there is no solution, finding a feasible solution has a higher priority than figuring out the global optimal solution. Based on this, this paper proposes OPA* algorithm. A general solution without more restriction also seems to be impossible. In this case, the adding limitations need to narrow the scope of the problem as well as allow a thorough regroup for all vehicles in case of any aim state. The assumption of this solution can be considered reasonable.

On the other hand, the final situation that requires frequent occurrence of excessive changes must not be the optimal situation. The optimal termination situation must depend on the initial situation, and the criteria for selecting these states is: the less changes, the better. So, the optimal selection conditions is also consistent with this assumption.

3.4.2 Specific Structure

To sort the car platoons before entering the intersection, OPA* method is promoted. No afterwards merging process is needed and the node after switching position is already sorted. Through online sorting process, the scale of the problem will be segmented into small time complexity problems. The solution could be divided into two parts, which can be done parallel:

1. Preparation. When cars are merging into car platoon with their aims, there could be one optimal start state and one optimal aim state for this certain platoon. However, in this essay, the preparation is made by random seed.

2. Online sorting. Every time OPA* algorithm only needs to consider the first three rows of cars except the last step. In the last step, 6 cars will be sorted together no matter how many rows they are. Because of the generation of empty rows caused by sorting process, automatically removing these empty rows are also important. With automatically shrinking of the vehicle platoons, it is possible to could bring the last part cars to the first part. Also, with the increase in computing power, four rows of cars are also possible.

Out of fairness and first-come-first-served rules, the car in front cannot be more than a certain number of rows in the back, and the car in the back cannot be more than the same number of rows in the front. In our example cases, the row number is set to 3 in every loop, so called sub state.

The flow chart for OPA* is shown below:

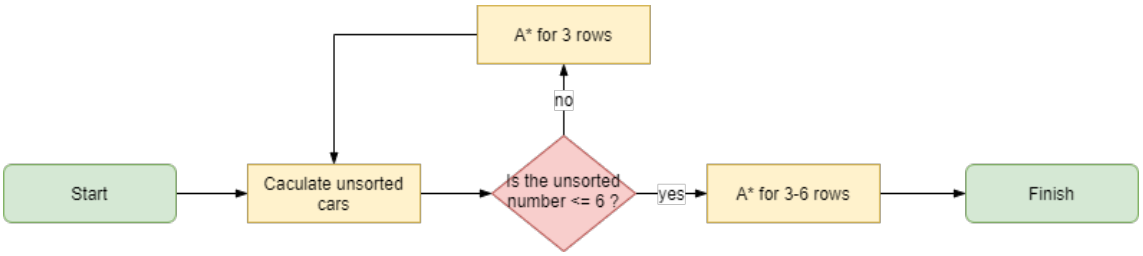


Figure 3.1: Structure of the OPA* Algorithm

Each time 3 to 6 cars are sorted online. The consuming time for this process determines the total consuming time. As the computing time of A* for a certain scale could vary a lot, the combination of small-scale A* also Assuming 3 lanes, the total problem will be sorted from front to back, each time 'eliminating' 3 cars or a row, and finally all 3*n cars will be sorted with only (n-1) times of sub-state. The problem of time complexity is avoided as even in the worst case, the biggest sub-problem is a sorting of 6 cars in 6 rows, the searching space is:

$$A_{12}^{6}$$

which is possible for A* algorithm with hash table to solve in a short time.

There are two expectations for OPA* algorithm:
1. The sorting is done with a time sequence, the calculation time could witness a linear increase.

2. As the A* algorithm is used with scale limitation, these part time could be seemed as a constant.

3.4.3 Pseudocode for OPA* Algorithm

3.4.3.1 A* Algorithm

Algorithm 1: A* Algorithm

Input: *startS*: Global start state
endS: Global aim state
hn: heuristic value for each state
preNode: Record precursor nodes
curS: Current state
newS: Next state
openDict[*start*]: Store unexplored nodes for future comparison
deepDict = []: Record the depth(g(n)) for every node

Output: *stepL*: A movement list that connect start list with end list

```
def getHn(startS,endS,gn):
    return hn;
def getChildren(curS):
    return newS;
def getPos(curS):
    return avaPos;
Initialize
    preNode[startS] = −1 ;
    deepDict[startS] = 0 ;
    openDict[startS] = getHn(startS, endS, deepDict[startS]) ;

while True do
    curS=MIN(openDict);
    DELETE(openDict(curS));
    if curS = endS then
        break ;
    end
    for i ∈getPos(curS) do
        newS = getChildren(curS);
        if newS ∉preNode then
            deepDict[newS] = deepDict[curS] + 1;
            openDict[newS] = getHn(newS, endS, deepDict[newS]) ;
            preNode[newS] = curS ;
        end
    end
    stepL.append(curS);
    while preNode[curS]! = −1 do
        curS = preNode[curS];
        stepL.append(curS);
    end
    return stepL;
```

3.4.3.2 OPA* Algorithm

Algorithm 2: Online Processing A* Algorithm

Input: *startL*: Global start state
endL: Global aim state
rowNum: Number of lanes
onlineStart: The start state for each sub-process
sort: The number of vehicles that have been sorted in a single sub-process
tail: The number of vehicles that have not been sorted in last sub-process
totalSort: Total number of vehicles that have been sorted
isTerminal: Judge if the sorted vehicle platoons are same with the global aim state
stepL: Get sub movement consequence from a star algorithm

Output: *totalStep* = []: A movement list that connect start list with end list

```
1  def getUnsorted(aList):
2      return unsorted;
3  Initialize
   onlineStart = startL[: 9] ;
   onlineEnd = endL[: 3] ;
   totalSort = [] ;
   totalStep = [] ;
   isTerminal = False ;

4  while True do
5      tail= DEEPCOPY(onlineStart);
6      for i ∈onlineEnd do
7          tail=tail.remove(i);
8          stepL=aStar(onlineStart,onlineEnd);
9          sort=onlineEnd ;
10         totalSort+=sort;
11         totalSort = FLATTEN(totalSort);
12     end
13     if isTerminal = True then
14         break ;
15     end
16     if getUnsorted(startL) = 2 ∗ rowNum then
17         isTerminal = True ;
18         onlineStart = startL[:] ;
19         onlineEnd=endL[loopTime ∗ rowNum : (loopTime + 2) ∗ rowNum];
20     else
21         onlineStart = startL[: 9] ;
22         onlineEnd=endL[loopTime ∗ rowNum : (loopTime + 1) ∗ rowNum];
23     end
24 end
```

3.4.3.3 Random Generation of Start State

Algorithm 3: Algorithm for random generating start state

Input:

$oneLeft$: One lines leaf to choose
$twoLeft$: Two lines leaf to choose
$loopTime$: count loop times
$randomRow$: Store random generated row
$seedNum$: The number of seed
$totVeh$: The number of vehilces
$childrenPool$: For store random children

Output:

$startL$: A random generated start state without empty row

```
1  Initialize
   startL = [] ;
   childrenPool = [] ;
   totVeh = 15 ;
   oneLeft = 15 ;
   oneLeft = 15 ;
   seedNum = 999 ;
   while True do
2      loopTime+=1;
3      if oneLeft!=0  twoLeft!=0 then
4          carNum=RANDOM([1,2],1);
5      else if oneLeft!=0 then
6          carNum=2;
7      else if oneLeft!=0 then
8          carNum=1;
9      else
10         break;
11     end
12     if carNum = 1 then
13         oneLeft-=1;
14     else
15         twoLeft-=1;
16     end
17     for i ∈ range(carNum) do
18         randomRow.append(childrenPool[0]);
19         childrenPool.remove(childrenPool[0]);
20         while LEN(randomRow)<3 do
21             randomRow.append('0');
22         end
23         random.shuffle(randomRow);
24         startL.append(randomRow);
25     end
26     return startL;
27 end
```

3.4.3.4 Random Generation of End States

Algorithm 4: Algorithm for random generating end states

Input: *startL*: A start state
rowNum: Number of lanes
poolDict: Store available car numbers for choose
subState: Substate of the whole state for choosing aim cars
avaOption: Available options for choosing from the pool
seedNum: The number of seed
nextRow: Row that is selected from pool

Output: *endL*: An aim state

```
1  def getNum(aList):
2      return cNum;
3  Initialize
     avaOption = 3 ;
     for i ∈range(startL) do
4      poolDict[i] = 0;
5  end
6  for i ∈ range(rowNum) do
7      subState = startL[: 9];
8      h
         for i ∈getNum(subState) do
9          poolDict[i]+ = 1;
10         if poolDict[i] = 2 then
11             subState.remove(poolDict[i]);
12             nextRow.append(poolDict[i]);
13             avaOption− = 1;
14         end
15     end
16 end
17 nextRow.append(random.sample(subState, avaOption));
18 for j ∈nextRow do
19     startL.remove(nextRow);
20 end
21 random.seed(seedNum);
22 random.shuffle(nextRow);
23 return endL;
24
```

3.4.4 Examples for Random Start and Aim State

The start states and end states are generated randomly and they meet the assumptions made before. For all situations that meet the requirement 'not passing through 3 rows with cars in each loop. Also, for the start state, no empty rows are included and for end state, first half rows are full and second half are empty to simulate a full usage of road.

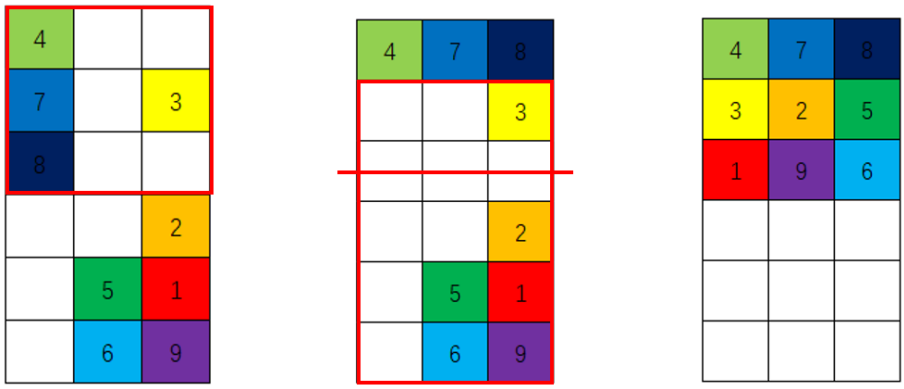


Figure 3.2: Example working flow the OPA* algorithm

From the figure above, the empty line is automatically eliminated cause the steps calculation is based on the aggressive multi-steps. When the empty line occur, it will be occupied by the following car platoons immediately. In other words, the last step of the former sub-state will happen at the same time with the first step pf the later sub-state.

3.4.5 Decision Tree Analysis for OPA* and A*

From the sample state mentioned in figure above, the decision trees are shown below. As the total decision tree is too long to show in A4, the decision trees are simplified to nodes and edges with the color temperature which represents the depth of the node in the decision tree.

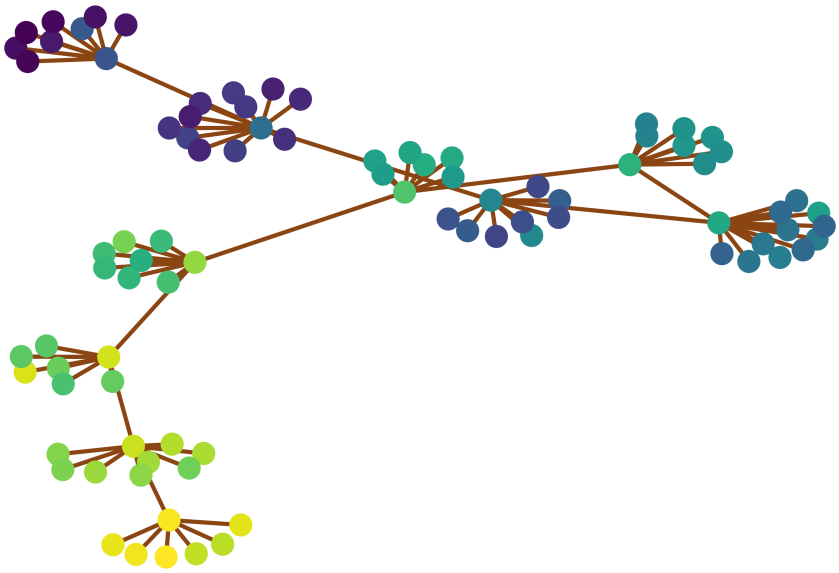


Figure 3.3: Decision tree for the first sub-state of OPA*

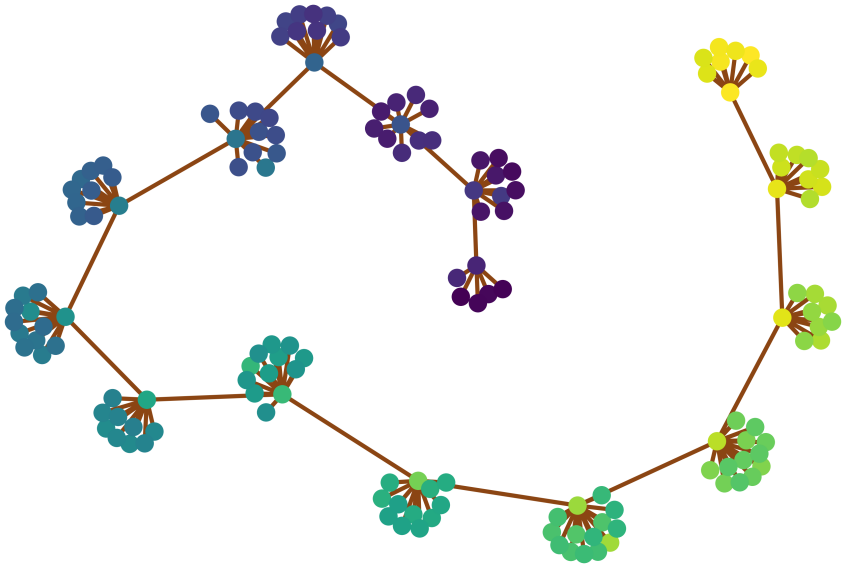


Figure 3.4: Decision for the second sub-state of OPA*

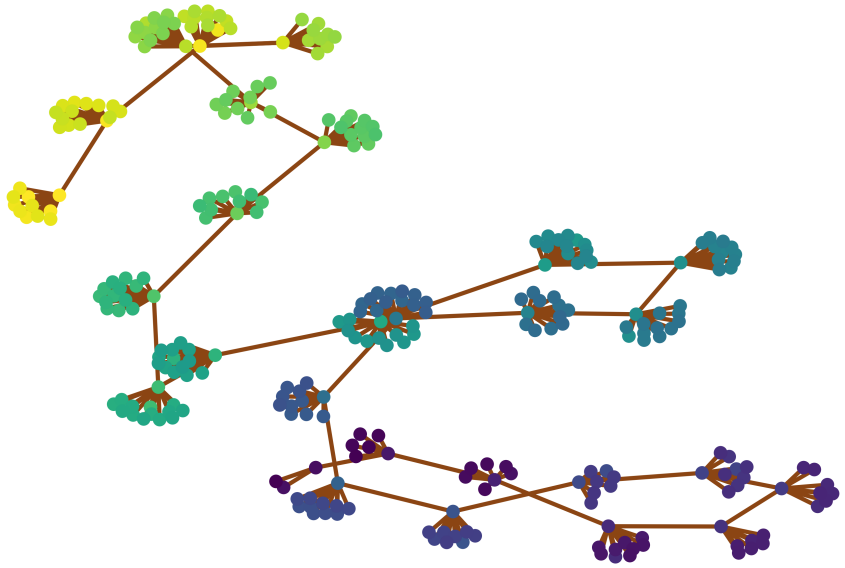


Figure 3.5: Decision tree for the A*

Also, these decision trees require a very precise expansion. Any wrong expansion will lead to the result of divergence rather than convergence.

The number of nodes on the decision tree for OPA* and A* is 243 (88+155) and 303 respectively. OPA* divides the decision tree into several sub-decision trees according to the number of the sub-states. In this approach, the depth for the decision tree is limited to a certain scale. However, for A*, the decision tree is much longer. The differences between decision trees also indicates that OPA* is more efficiency than

A*.

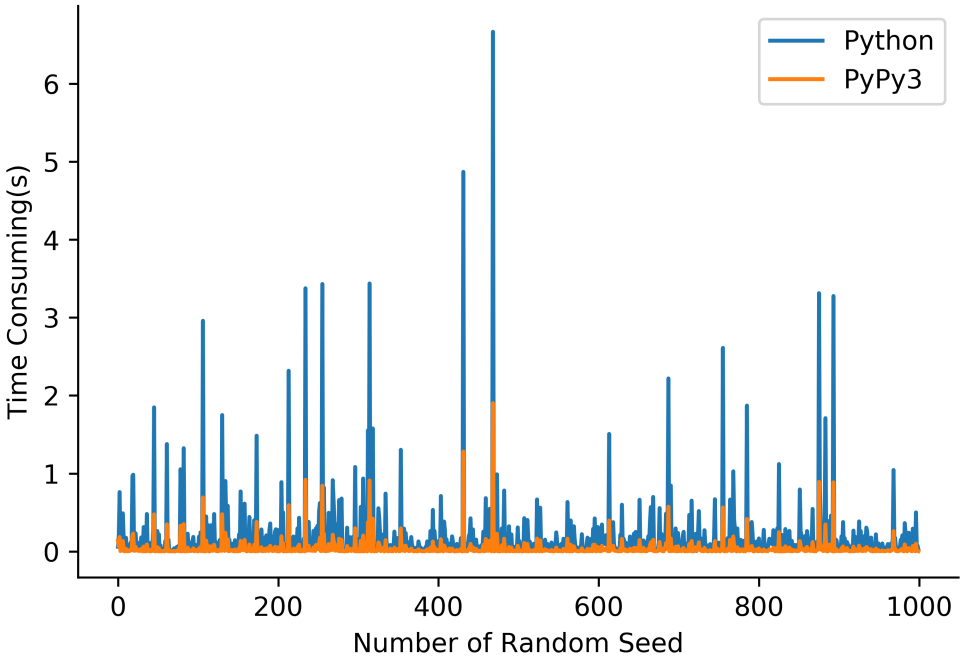
4

Results

4.1 Comparison between PyPy3 and Python

The traditional solution are written in Python and for the purpose of improving performance while saving calculation power, the whole permutations are processed by the hashable data type, string, and saved in a dictionary. Instead of using matrix calculation for state transition, in this essay it is formed by list replacement. All these approaches could speed up the whole process significantly from the average time complexity aspect. Moreover, instead of using Python directly, changing the interpreter to the PyPy3 will also improve the total performance. This comparison is performed on the CPU i7 6700. Through 1000 times random generate samples, results are from 6 cars scenario and shown below:

Figure 4.1: Time consumption comparison between the Python and PyPy3



Detailed information are shown in the table below: From the figures above, with 1000 random seeds, PyPy3 shows an around 17 times improvement.

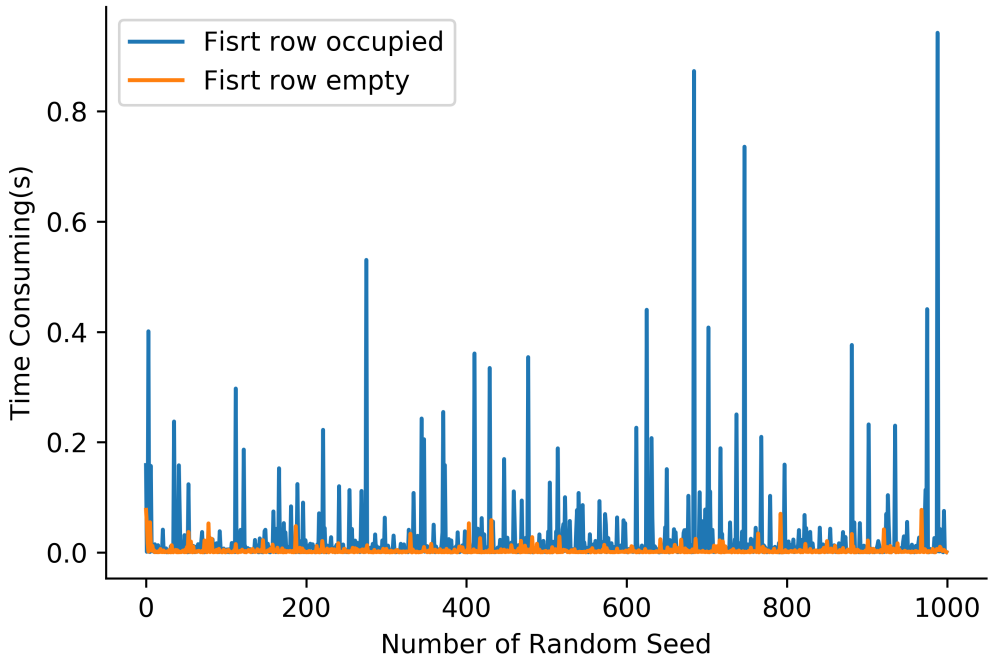
Table 4.1: Results for comparison between the Python and PyPy3

Performance	Median value
Python	0.003806
PyPy3	0.064481

4.2 The Relationship between Platoon's Arrangement and the Performance of A* Algorithm

Even for the same arrangement, whether the first row for the end state is occupied or not could be a influence factor for the time consumption of A* algorithm. The difference for this is neglected when using OPA* method as the empty line will be automatically eliminated. However, when the last step of A* method conflicts with the moving-up action, the total sorting steps should have one additional step. This step is not a stable add-on, it depends on whether there is a conflict in the end or not. The comparison between occupied first row and empty first row are shown below:

Figure 4.2: The comparison between the occupied first-row and empty first-row of the end state



Calculating time witnesses an significant decrease. This results indicates that the location of the aim state determines the performance of the sorting algorithm to a

large extent.

4.3 The comparison between Distributed Stochastic A* Algorithm and A* Algorithm

Distributed Stochastic A* Algorithm(DSA*) Algorithm is proposed for solving the problem of massive exploration with a modified edge cost. [37]

$$\hat{H}_{DSA*} = \hat{H} - \varepsilon C_{min}/exp(\hat{H}), \varepsilon \in (exp(-C_{min}, 1)) \tag{4.1}$$

where C_{min} is the minimum edge cost, which is 1.
The DSA* algorithm and A* algorithm are implemented with the hash table, the results are shown below:

Figure 4.3: The comparison between the DSA* algorithm and A* algorithm

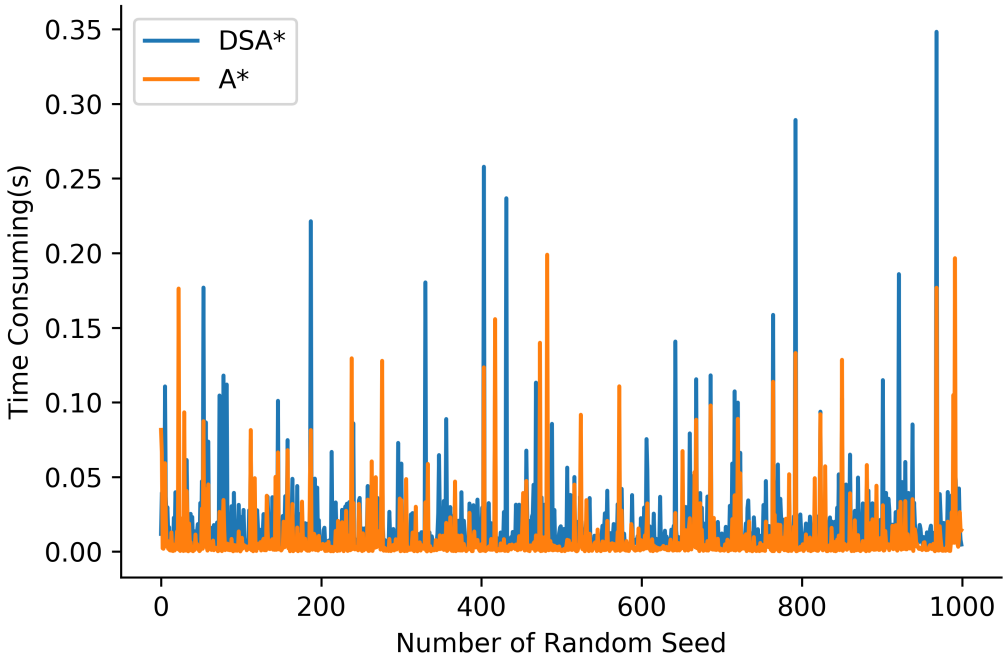


Table 4.2: Results for comparison between the DSA* algorithm and A* algorithm

Performance	Median value	Mean value
A*	0.00132	0.003343
DSA*	0.002451	0.008835

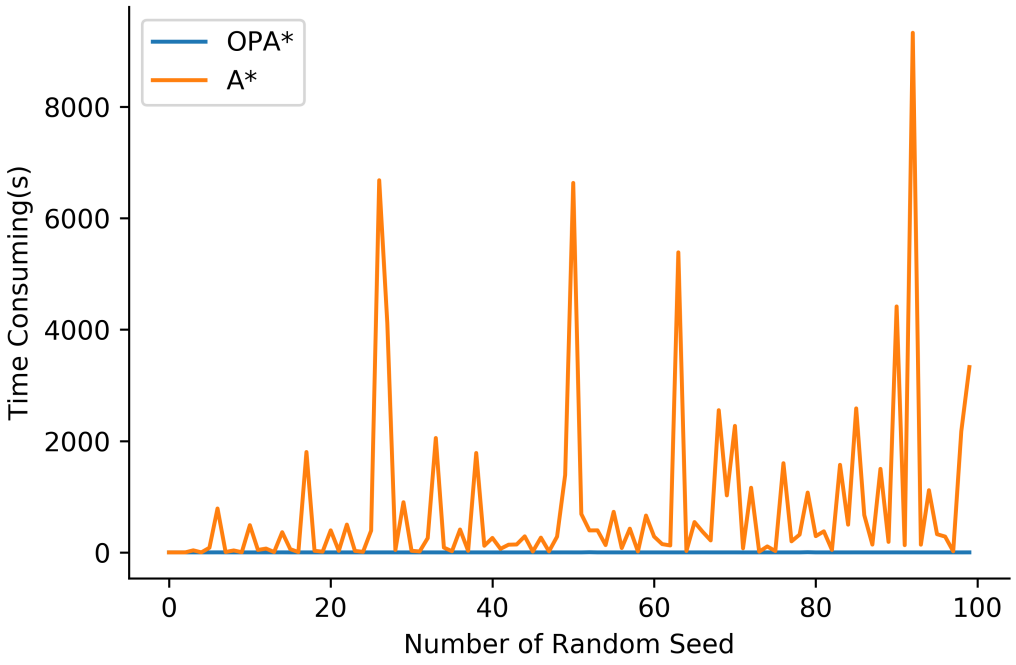
The impact of DSA* algorithm seems to be negative. It maybe because the using data structure, Hash table, has an $O(1)$ time complexity during the exploration.

Then, because of the increase calculation load caused by the equation above, DSA* is slightly slower than the A*.

4.4 Comparison between OPA* Algorithm and A* Algorithm

After swishing the aim state for A* one row backwards, the performance comparison are shown in the boxplots below:

Figure 4.4: The time consumption comparison between the OPA* algorithm and A* algorithm



OPA* is clearly fast than A*, however, for the optimality, A* performs better. Detail information for the comparison are shown below:

Table 4.3: Results for the time consumption comparison between the OPA* algorithm and A* algorithm

Methods	Median value (s)	Mean value (s)
OPA*	0.027528	0.135837
A*	258.297169+	814.256094+

Figure 4.5: The performance comparison between the OPA* algorithm and A* algorithm

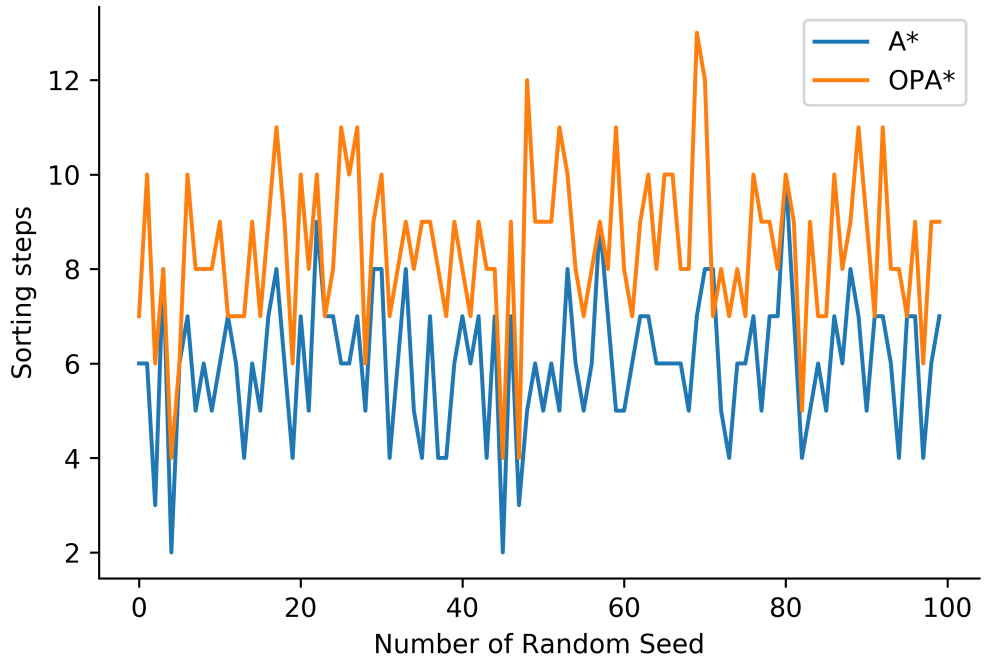


Table 4.4: The performance comparison between the OPA* algorithm and A* algorithm

Methods	Median value	Mean value
OPA*	8.0	8.42
A*	6.0(+1.0)	6.01(+1.0)

There is 1 step difference between the aim state of the A* and OPA*. In the cases where the rear cars are aligned and waiting for the front cars, that is, the last steps of the traditional a star method involve the displacement of the front rows of cars. All rear cars need to wait for the front cars to complete the entire process. Because the empty row is used by the preceding vehicle until the last moment, this empty first row needs to be reserved. A* should to take this 1 step into total steps.

From figures and tables above, the performance for the new algorithm is relatively good comparing to the old algorithm, or the global optimal solution. The sorting steps for 9 cars is 8 when the global optimal is 7(6+1). As for the computing time, OPA* shows significant advantage.

4.5 Extended Scenario for the OPA* Algorithm

To test OPA* in more complex scenarios, another 5 random cases are implemented, namely 21-33 cars. Results are shown in the box plots:

Figure 4.6: The time consumption of the OPA* algorithm for extended scenario

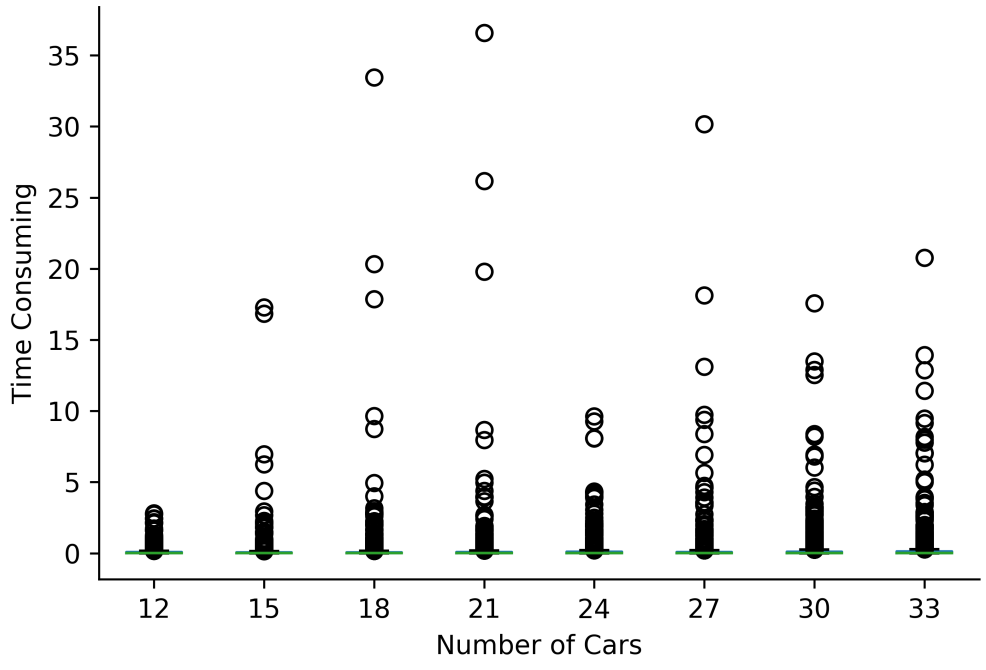
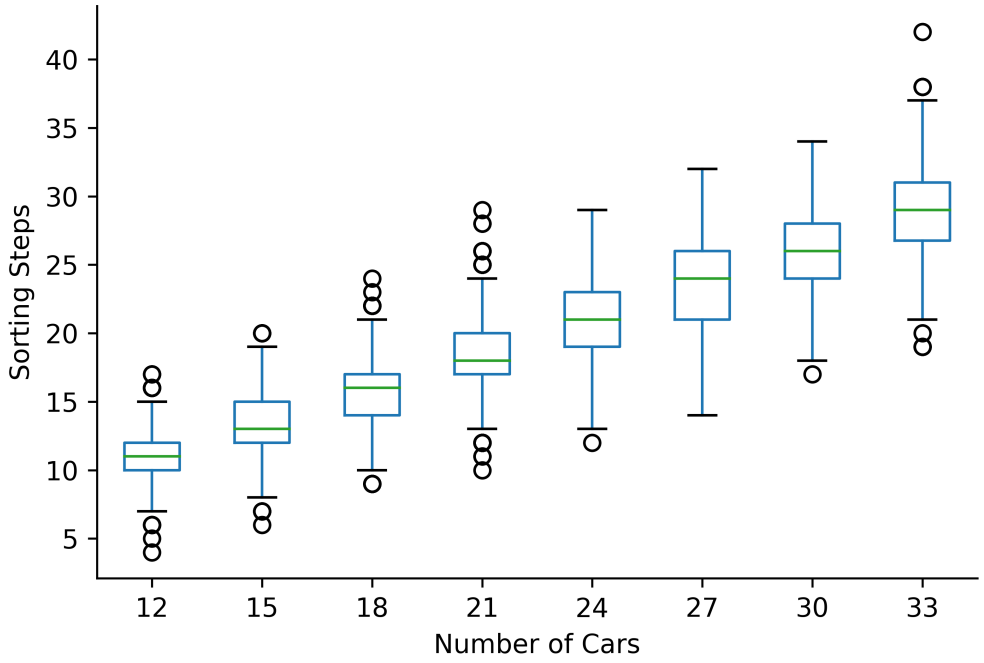


Figure 4.7: The performance of the OPA* algorithm for extended scenario



As for the consuming time, the results are all close to 0. Moreover, for steps, there is a linear increase response to the linear increase of the scale of the problem. In this case, the optimality is probably the same with 9 cars. There could be a slight difference between OPA*'s results and global optimal solutions. All in all the optimality for OPA* (below 33 cars) is acceptable. The detailed results are shown below:

Table 4.5: Results of the time consumption of the OPA* algorithm for extended scenario

Number of cars	Median value (s)	Mean value (s)
12 cars	0.021531	0.095694
15 cars	0.018866	0.136395
18 cars	0.018051	0.208853
21 cars	0.019348	0.209775
24 cars	0.021530	0.177012
27 cars	0.019873	0.222824
30 cars	0.023136	0.248363
33 cars	0.025651	0.303072

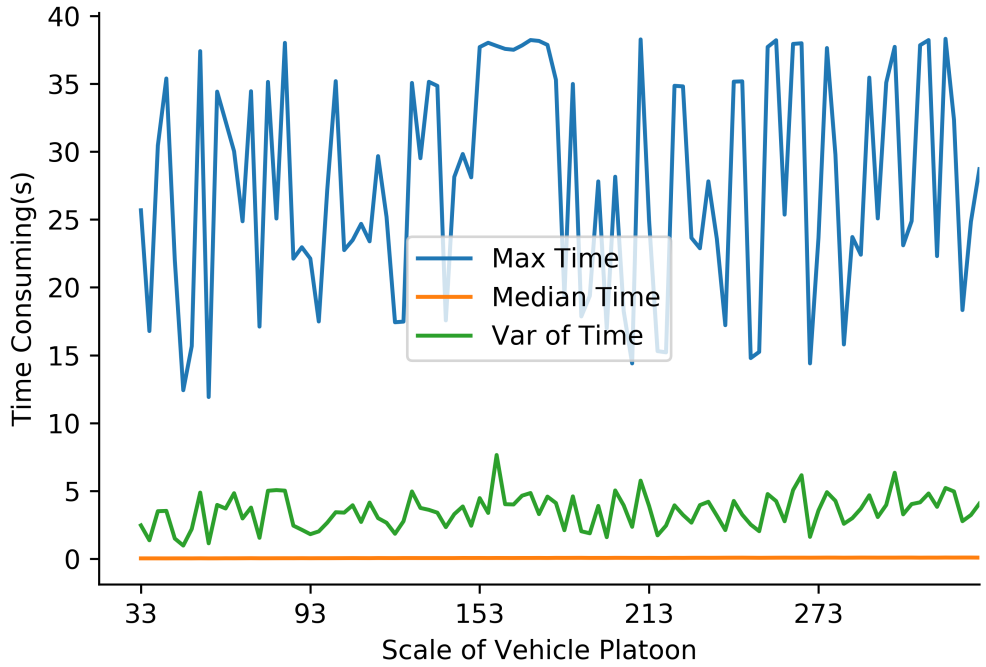
Table 4.6: Results of the performance of the OPA* algorithm for extended scenario

Number of cars	Median value (steps)	Mean value (steps)
12 cars	11.0	10.986
15 cars	13.0	13.336
18 cars	16.0	15.880
21 cars	18.0	18.502
24 cars	21.0	21.045
27 cars	24.0	23.527
30 cars	26.0	26.027
33 cars	29.0	28.698

From the data above, results witness a fluctuate trend for median value in the time consumption with the increase of the car number. However, for the mean value, an slightly increase is found.

To analyze a more clear trend, from 36 cars to 333 cars are performed. The results are shown in the line chart below: As the median time is equal to the mean value of computing time and the lowest time consuming also overlap with median time from the scale of the max time consumption. So only median value are shown in the figure below:

Figure 4.8: The time consumption analysis for the OPA* algorithm

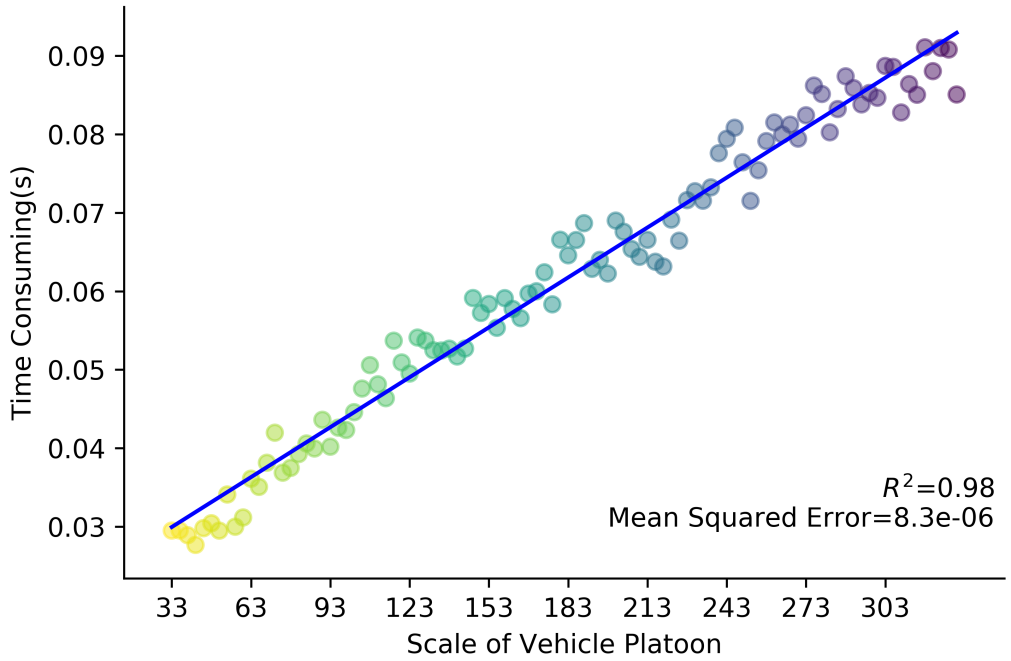


From the figure above, the median computing time is negligible when considering

the maximum value. However, in the cases that all start states and end states are designed for an optimal usage of the intersection, the median value weights more importance.

Mean value for the time is not shown as the mean value overlaps with the median value. The median computing time for all random seeds is shown below:

Figure 4.9: The linear regression of the time consumption



The linear relationship is expected. The reason for the fluctuation mainly caused by the unstable CPU usage when performing these random seeds. The overall trend for the time illustrate an average time complexity of $O(n)$. Also, the average time complexity for A* in the cases where cars are below 6 could be regarded as $O(1)$. When breaking the total vehicle platoons into sub-platoons, the time complexity for sorting them back are shown in the TableA-1 in the Appendix. Comparing with these algorithm, OPA* has the lowest average time complexity. Moreover, the lowest time complexity for a global sort for n cars is $O(n)$. In this case, OPA* could be one of the fastest algorithms for this problem.

Detail information are shown in the table below:

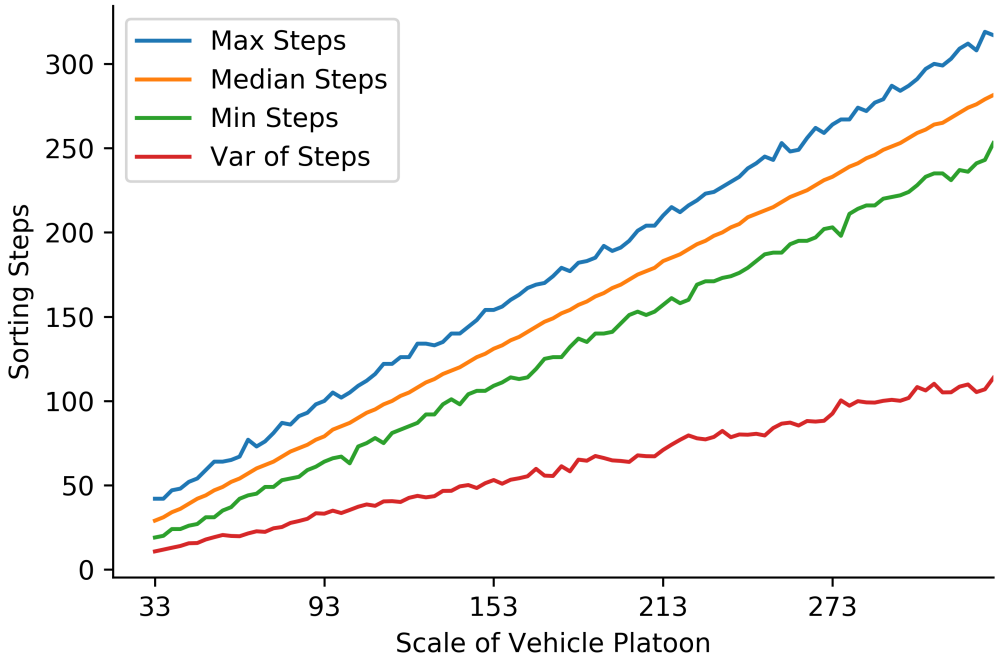
Table 4.7: Results for the time consumption analysis for the OPA* algorithm

Number of cars	Median value	Mean value
Max Time	27.809735	27.399988
Median Time	0.062404	0.060826
Var of Time	3.529809	3.471107

Even considering the unlikely happened worst cases, the time is still within 30s, which is also an acceptable value.

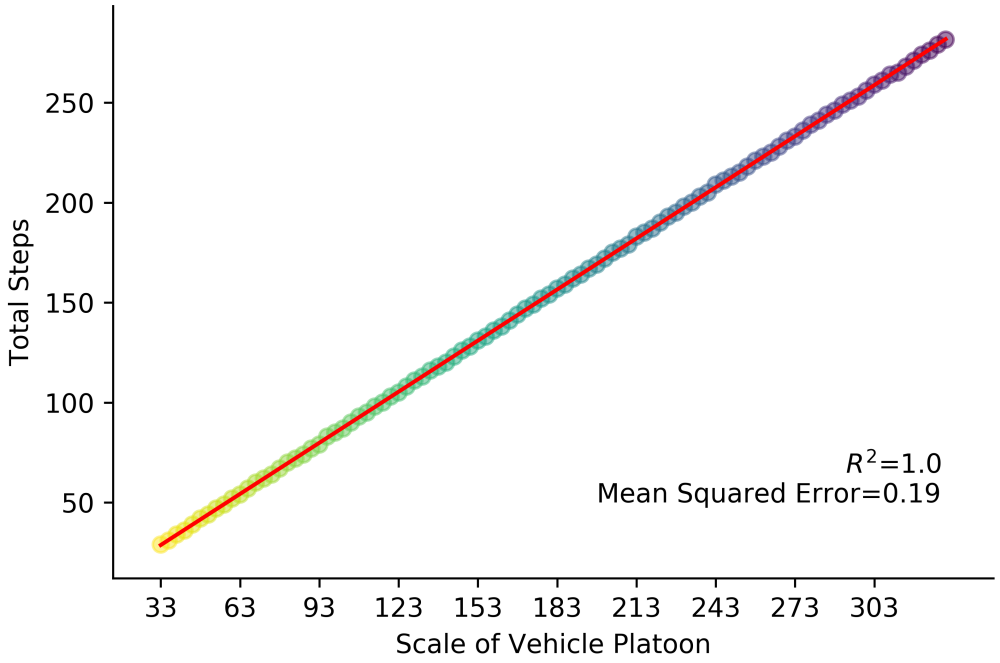
As the optimal solution is sacrificed, the criteria for evaluating the results are also properly relaxed. As long as the solution is stable enough and within acceptable limits, the algorithm is considered as feasible. For example, when sorting 30 cars, 30 steps is reasonable.

Figure 4.10: The performance (steps) analysis for the OPA* algorithm



As the median steps overlap with mean steps from the scale of the maximum number of steps. So only median value are shown in the figure. From the figure, the variant of steps becomes larger. However, the variance increase slighter than the steps. Even considering maximum steps, the results are considerable, especially when there have been no other solution for these kind of scale problem.
In this case, the median value for the steps is also analyzed, the results are shown below:

Figure 4.11: The linear regression of the computing steps for the OPA* algorithm



A perfect linear relationship is found between sorting steps and the scale of the vehicle platoon. This result is very stable and it proves the scalability of the OPA* algorithm.

Overall, the OPA* could provide stable and fast solutions with the compensation of reducing optimality.

5

Conclusion

With the acceleration of urbanization, traffic congestion has become an increasing concern. The traditional method generally refers to the passing efficiency through road construction, and the emergence of new technologies brings more possibilities. With the development of self-driving technology, intelligent networked cars have gradually entered people's vision. There are more and more intersection optimization schemes under the assumption of intelligent connected cars, and it is possible to greatly improve the efficiency of traffic without repairing roads. Therefore, how to achieve fast and efficient fleet sequencing before entering the intersection has become an indispensable problem. The fast sorting algorithm is based on real-time changing data from driverless cars. The value of the data is fleeting, and the algorithm must run fast. The traditional heuristic algorithm is helpless for the slightly larger sorting problem.

5.1 Reinforcement Learning

The PPO method tends to shrink the exploration area and limits to the trust region of previous sampling.[12] In their research, even with a simple quadratic objective, the converge demonstrates a clearly variance with the global optimal solution, not to mention a $4 * the number of cars$ times objective. All in all, RL related method is probably not a good solution for vehicle sorting problem.

Comparing the outcome of A* and reinforcement learning, the A* shows an absolute advantage in solving this kind of problem and reinforcement learning does not even have the possibility of getting stable optimal solutions with unlimited calculation power.

Two major deficiencies for RL are identified:

1. The RL method is not admissible. The RL agent only grasp the relationship between states and actions but it does not know which action is best.[30]Comparing with the clear heuristic value function using in the A* method, the efficiency for skipping useless nodes and edges is significantly slowed down.

2. The RL method is not consistent. The Markov property prevents the possibility of skipping repeated states. In comparison, in A* algorithm, the repeated states are removed directly after they are explored. Without this function, even A* algorithm

will have serious problem in getting a simple solution without step limitation, not to mention an optimal solution.

Above 2 main reasons are established if methods adapts Markov property. Both reinforcement learning methods and multi agents reinforcement learning methods are literally not supposed to get a good enough solution for NP-hard sequential decision making that requires consistent or admissible.

As the stochasticity, instability, and non-generalization of reinforcement learning, the searching tree is hard to reach an end, not to mention for a good enough solution. All in all, if the problem could be modeled with MDP, then RL could probably solve it, otherwise, whether RL works or not need to be analyzed according to the specific problem. And in this case, it does not work.

5.2 Improvements for A* Algorithm

In this article, the path planning is minimised from the path of roads to path of switching car locations to promote an optimal usage for the intersection.
A star algorithm is proved to be useful for the path planning problem and there are various extension of it to speed up the process.[2] However, all preprocessing algorithms are not okay for an nodes-unknown environment as incomplete node preprocessing can increases the calculation time. The complexity of the preprocessing structure itself cost a lot of time and a few preprocessed nodes only improve very little. These two factor could sum up as a negative contribution.
Instead of using matrix or vectors for storing global data, hash table is introduced as the O(1) operation time complexity it brings. The using of hash table could accelerate the A* algorithm clearly. PyPy3 could also improve the performance. However, after these improvements, A* algorithm still have a time complexity problem as the search space fast expands with the increasing dimension.

5.3 Advantages of Online Processing A* Algorithm

Final solution focus on OPA* algorithm. By dividing the vehicle platoon dynamically, OPA* eventually solve this problem with one assumption. Following are the advantages of OPA* algorithm:

5.3.1 Speed

Online processing A*(OPA*) algorithm could be seemed as one approach of limiting the depth and width of the searching tree from dividing real vehicle platoons dynamically. Also other approaches could exist. Both this approach will scarify the optimality. From 1000 random seeds simulation, OPA* algorithm could meet the daily scenario as the time consuming is lowering under 0.1 second even for 33 cars platoons. And there are no limit for the largest solvable scale. Considering 33 cars

is sufficient for a single traffic light interval, the largest scale limits to 33 cars.

5.3.2 Flexibility

No matter when the sorting process is ended, the already sorted vehicle platoon are available directly after it being sorted. Comparing with OPA*, A* will finish the whole process after a thorough sorting process and any break in this process will lead to a stagnation for the whole vehicle platoons.

5.3.3 Scalability

Even for the worst case, the computing time is within 30 seconds. With improving computing power, the upper limit could be limits to 3 second by using atomic operation to achieve a 10 cores parallel computing for example. For 9 car with aim state relaxations, A* shows a upper limit at the level about 10000 seconds(2.78h), which is unacceptable for real life scenario.

5.3.4 Reusability

Or known as generalization. With a time limits of 1 minutes, for 1000 random seeds, results are 100 percent got. For 9 cars with aim state relaxations, the results are 0 percent got. The time for the A* is not usable in the real life, not to mention the reusability.

5.4 Limitations of Online Processing A* Algorithm

Despite the advantages, OPA* also has limitations:

5.4.1 Assumption

The prerequisite for these algorithm is a 3-row moving limitation for every sub-state. This could increase the workload for the design of the aim state. However, a well defined aim state also should not change the platoon too much from the environmental and ecological perspective. Also for the global state, the total movement could be at least half the length of the origin state. The 3-row moving limitation could expand to 4-row and even more row limitation, these ability is determined by the calculation power. The more rows involved, the more relaxation it gives to the design of aim state. How much limitation will this assumption caused to the aim state need to be figured out in the future.

5.4.2 Optimality

As the A* algorithm could not give the global optimal solution for the increasing number of car platoons, the degree of optimality could not be calculated directly.

For 9 cars, the difference is 1 2 steps. Assuming this also works for the following extended scenarios, the results are acceptable and stable. Moreover, for each substate in the OPA*, the result is calculated by A*, and this result is guaranteed to be global optimal result.

5.5 Summary

A star algorithm is proved to be useful for the path planning problem and there are various extension of it to speed up the process.[2] In this article, the path planning is minimised from the path of roads to path of switching car locations to promote an optimal usage for the intersection. All preprocessing algorithms are not okay for an nodes-unknown environment as incomplete node preprocessing not only does not shorten the time but increases it. With the increasing scale of the problem, the depth for the searching tree expends. The exponential expansion of the searching space consists of both the increase of the width and the depth of the searching tree. Either reducing the width or limiting the depth or both could compensate the computing speed with the cost of the optimality.

The relationship between the hash table and the A * algorithm is rarely mentioned. For the A * algorithm, there is an exponentially large open list that needs to be maintained. And hash table can eliminate the problem of time complexity. From this perspective, for large-scale problems, hash tables and A * could be an excellent cooperation. What is more, the using of PyPy3 could fast up all Python scripts by optimizing its runtime. Even if the hardware performance is improved, it is difficult to get an acceptable solution within an acceptable time range. So, software-based solutions are needed.

In this case, a hypothesis is made, that is, in the sorting of the subspace, a car cannot move forward or backward more than three rows. Under this assumption, OPA* algorithm is made. In real life scenario, the initial state and the end state are the optimal conditions that determined by integer programming. For these optimal cases, the difficulty of the arrangement could be much smaller than the random cases. So, the performance could be expected with a further improvement. Under the reasonable assumption, the OPA* algorithm exhibits many excellent features.

When there are several start state and end state combination with the same optimality, in other words, the optimal intersection has several choices. In this case, all choices will be compared according to the Menhaden distance. The least one will be selected to speed up the whole process.

All in all, OPA* shows fast, scalable and stable performance, and have the migrating ability. It could be one of the fastest solutions with sacrificing optimality.

A

Appendix 1

Table A.1: Time complexity comparison for sorting algorithms

Name	Average case	Worst case	Stable
Bubble sort	$\Theta(n^2)$	$O(n^2)$	Yes
Selection sort	$\Theta(n^2)$	$O(n^2)$	No
Insertion sort	$\Theta(n^2)$	$O(n^2)$	Yes
Merge sort	$\Theta(nlogn)$	$O(nlogn)$	Yes
Quick sort	$\Theta(nlogn)$	$O(n^2)$	No
Bucket sort	$\Theta(d(n+k))$	$O(n^2)$	Yes
Heap sort	$\Theta(nlogn)$	$O(nlogn)$	No

References

[1] Abraham, I., Delling, D., Goldberg, A. V., & Werneck, R. F. (2011). A hub-based labeling algorithm for shortest paths in road networks. In *International symposium on experimental algorithms* (pp. 230–241).

[2] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., ... others (2019). Self-driving cars: A survey. *arXiv preprint arXiv:1901.04407*.

[3] Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., ... Werneck, R. F. (2016). Route planning in transportation networks. In *Algorithm engineering* (pp. 19–80). Springer.

[4] Biswas, D., Su, H., Wang, C., Stevanovic, A., & Wang, W. (2019). An automatic traffic density estimation using single shot detection (ssd) and mobilenet-ssd. *Physics and Chemistry of the Earth, Parts A/B/C*, *110*, 176–184.

[5] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

[6] Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., & Abbeel, P. (2018). Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214*.

[7] Dresner, K., & Stone, P. (2004). Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems-volume 2* (pp. 530–537).

[8] Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International workshop on experimental and efficient algorithms* (pp. 319–333).

[9] Gelly, S., & Silver, D. (2011). Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, *175*(11), 1856–1875.

[10] Goldberg, A. V., & Harrelson, C. (2005). Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual acm-siam symposium on discrete algorithms* (pp. 156–165).

[11] Guanetti, J., Kim, Y., & Borrelli, F. (2018). Control of connected and automated vehicles: State of the art and future challenges. *Annual Reviews in Control*, *45*, 18–40.

[12] Hämäläinen, P., Babadi, A., Ma, X., & Lehtinen, J. (2018). Ppo-cma: Proximal policy optimization with covariance matrix adaptation. *arXiv preprint arXiv:1810.02541*.

[13] Harabor, D. D., & Grastien, A. (2011). Online graph pruning for pathfinding on grid maps. In *Twenty-fifth aaai conference on artificial intelligence.*

[14] Harabor, D. D., & Grastien, A. (2014). Improving jump point search. In *Twenty-fourth international conference on automated planning and scheduling.*

[15] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

[16] Hausknecht, M., Au, T.-C., & Stone, P. (2011). Autonomous intersection management: Multi-intersection optimization. In *2011 ieee/rsj international conference on intelligent robots and systems* (pp. 4581–4586).

[17] Helmbold, D. P., & Parker-Wood, A. (2009). All-moves-as-first heuristics in monte-carlo go. In *Ic-ai* (pp. 605–610).

[18] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second aaai conference on artificial intelligence.*

[19] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237–285.

[20] Khan, A., Zhang, C., Lee, D. D., Kumar, V., & Ribeiro, A. (2018). Scalable centralized deep multi-agent reinforcement learning via policy gradients. *arXiv preprint arXiv:1805.08776*.

[21] Korf, R. E. (2000). Recent progress in the design and analysis of admissible heuristic functions. In *International symposium on abstraction, reformulation, and approximation* (pp. 45–55).

[22] Levin, M. W., Boyles, S. D., & Patel, R. (2016). Paradoxes of reservation-based intersection controls in traffic networks. *Transportation Research Part A: Policy and Practice*, *90*, 14–25.

[23] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems* (pp. 6379–6390).

[24] Maurer, W. D., & Lewis, T. G. (1975). Hash table methods. *ACM Computing Surveys (CSUR)*, *7*(1), 5–19.

[25] McAleer, S., Agostinelli, F., Shmakov, A., & Baldi, P. (2018). Solving the rubik's cube without human knowledge. *arXiv preprint arXiv:1805.07470*.

[26] Melo, F. S. (2001). Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, 1–4.

[27] Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*.

[28] Reed, T. (2019). Inrix global traffic scorecard.

[29] Schadd, M. P., Winands, M. H., Van Den Herik, H. J., Chaslot, G. M.-B., & Uiterwijk, J. W. (2008). Single-player monte-carlo tree search. In *International conference on computers and games* (pp. 1–12).

[30] Schmidhuber, J. (2019). Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*.

[31] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[32] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... others (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484.

[33] Steinberg, R., & Zangwill, W. I. (1983). The prevalence of braess' paradox. *Transportation Science*, *17*(3), 301–318.

[34] Sun, W., Zheng, J., & Liu, H. X. (2017). A capacity maximization scheme for intersection management with automated vehicles. *Transportation research procedia*, *23*, 121–136.

[35] Van Otterlo, M., & Wiering, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement learning* (pp. 3–42). Springer.

[36] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

[37] Wu, J., Ah, S., Zhou, Y., Liu, P., & Qu, X. (2020). The cooperative sorting strategy for connected and automated vehicle platoons. *arXiv preprint arXiv:2003.06481*.

[38] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., & Wang, J. (2018). Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*.