Multi-Agent Path Finding with Priority for Cooperative Automated Valet Parking

Ayano Okoso[1*], **Keisuke Otaki**[1], **Tomoki Nishi**[1]
[1] Toyota Central R&D Labs., Inc.
{okoso, otaki, nishi}@mosk.tytlabs.co.jp

Abstract

Shortage of parking lots has been a serious problem due to the rapid population growth in urban areas. Cooperative Automated Valet Parking (Co-AVP) is a promising approach to mitigate the problem. Co-AVP system would realize high density and efficient parking by automatically navigating vehicles to vacant parking spaces. Cooperative path planning has been extensively studied as Multi-Agent Path Finding (MAPF). In Co-AVP, vehicles would be often prioritized. For instance, vehicles would be able to spend more time to park in vacant spaces than moving from parking spaces to waiting drivers. Previous MAPF settings, however, cannot treat such priorities. In this paper, we formulate MAPF considering each agent's priority. We also develop the optimal method based on Conflict-Based Search (named CBS-Pri) and the heuristic method based on Cooperative A* (named CA*-Pri). We verified that higher prioritized vehicles tended to preferentially arrive at waiting drivers by our methods in numerical experiments. We also found that CBS-Pri always searched better solutions than or equal to CA*-Pri but requires more computational time.

1 Introduction

Rapid population growth in urban areas is getting worse to social problems related to transportation (e.g., traffic congestion, air pollution, and loss of public space) [Rodrigue *et al.*, 2016]. Shortage of parking lots is one of the problems. Drivers often spend much time to find vacant parking spaces in crowded areas. Automated Valet Parking (AVP) [Banzhaf *et al.*, 2017], where vehicles automatically find vacant spaces and park, is a promising approach to address the problem in various studied approaches such as the smart parking system [Shaheen, 2005]. It will enable to reduce drivers' travel time and park with high density because the space where drivers get off and walk is unnecessary. Previous work reported that AVP realized to reduce necessary parking spaces by up to 50% [Banzhaf *et al.*, 2017].

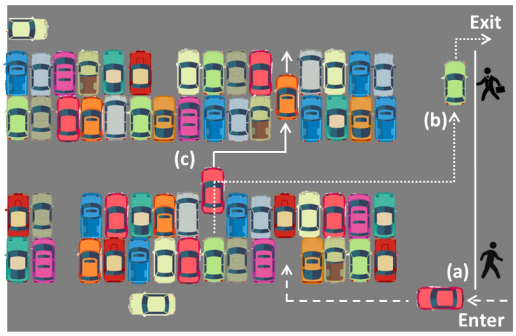

Figure 1: Overview of Cooperative Automated Valet Parking

Cooperative Automated Valet Parking (Co-AVP), where multiple vehicles move and park simultaneously, would help to reduce waiting time to arrive at drivers from parking spaces. Figure 1 shows an instance of Co-AVP; (a) a vehicle moves to a suitable parking space after its driver get off, (b) a vehicle travels at its waiting driver from a parking space, and (c) a vehicle relocates to a more suitable space. Previous AVP studies for path planning has been focused on the generation of trajectories of a single vehicle to a parking space under kinematic constraints [Conner *et al.*, 2007; Song, 2013]. In our best knowledge, however, no cooperative path planning among vehicles has been studied to realize Co-AVP.

Cooperative path planning is well studied as *Multi-Agent Path Finding (MAPF)* (see, e.g., [Standley, 2010; Yu and LaValle, 2013a]). The purpose of MAPF is to find collision-free paths for agents (e.g., robots or vehicles) to reach their destinations (called "task") with minimum cost such as travel time or distance. Most conventional settings in MAPF assume that all tasks are unprioritized. In Co-AVP, however, prioritization of tasks is often natural. For instance, vehicles have to arrive at their drivers as soon as possible to save the drivers' time when they travel somewhere, but could spend more time to park after drivers get off. In some cases, it might be desirable that upper grade members receive their vehicles earlier than others as a member bonus.

In this paper, we propose MAPF with task priorities, named MAPF with Priority, and develop two solvers. The first solver, named CBS-Pri, finds optimal paths of agents to reach their destinations based on Conflict-Based Search (CBS) [Sharon *et*

*Contact Author

al., 2015]. Whereas the second solver, named CA*-Pri, finds paths using a heuristics based on Cooperative A* (CA*) [Silver, 2005]. Both solvers have their pros and cons. For instance, the CBS-Pri can always find the optimal paths if given instances are solvable but needs more computational time than CA*-Pri. On the other hand, CA*-Pri can find paths faster than CBS-Pri but the optimality of the paths is not guaranteed.

The main contributions of this paper are as follows: (1) As far as we know, we are the first to apply MAPF to the Co-AVP problem. (2) We formulate MAPF for prioritized tasks, named MAPF-Pri. (3) We propose two solvers: an optimal (CBS-Pri) and heuristics (CA*-Pri) solvers. (4) We evaluated the methods proposed in this paper using synthetic grid graphs simulated Co-AVP environments. We verified that higher prioritized vehicles tended to preferentially arrive at waiting drivers by our methods in numerical experiments. We also found that CBS-Pri yield better solutions than or equal to CA*-Pri but required more computational time as the number of agents increased.

The rest of this paper is organized as follows. Section 2 presents a description of related work, while Section 3 presents preliminaries. Section 4 explains the formulation and proposed methods for MAPF with Priority, while experimental results are presented in Section 5. Section 6 draws a conclusion of this paper.

2 Related Work

AVP has been studied from various points of view such as sensors, mapping, localization, monitoring, motion planning, and platforms [Löper *et al.*, 2013; González *et al.*, 2016; Chirca *et al.*, 2015]. Research topics which have been explored extensively in this direction include technologies for estimating the parking area or optimal trajectory to parking positions [Al-Absi *et al.*, 2010; Oentaryo and Pasquier, 2004; Buehler and Wegener, 2003]. However, most of these researches on path/route/trajectory planning focus on a single vehicle and do not consider cooperative path planning for multiple vehicles. In this paper, we apply MAPF approaches to cooperative path planning for AVP (Co-AVP).

Various extensions has been studied for MAPF to fit real-world applications. Ma et al. propose a life-long setting assumed logistics in a warehouse that pickup and delivery goods repeatedly [Ma *et al.*, 2017]. Besides this, some researchers presented the setting in which specific agents can execute specific tasks [Ma and Koenig, 2016], the setting whereby adjacent agents can exchange packages that they carry as tasks [Ma *et al.*, 2016]. Ma et al. introduced a setting that maximizes the number of agents that reach within the time constraints [Ma *et al.*, 2018]. However, in our best knowledge, MAPF to treat prioritized tasks has not been developed.

It is well-known that MAPF is NP-hard [Yu and LaValle, 2013b], moreover its various solvers have been studied extensively [Felner *et al.*, 2017]. These solvers can be classified as optimal solvers [Sharon *et al.*, 2015; Surynek, 2012; Wagner and Choset, 2011] or heuristics-based solvers [Silver, 2005; Latombe, 2012]. Both types of solvers have pros and cons. For instance, optimal solvers can always find the optimal paths but comparatively need more computational time. On the other hand, heuristics solvers can find paths faster than optimal solvers but the optimality of the paths is not guaranteed. In this paper, we propose optimal and heuristics solvers respectively based on the methods: Conflict-Based Search [Sharon *et al.*, 2015] and Cooperative A* [Silver, 2005].

3 Preliminaries

First, we introduce a formulation of Multi-Agent Path Finding (MAPF). Let $G(V, E)$ be a simple graph with the set V of vertices and the set $E \subseteq V \times V$ of edges, where (u, v) corresponds to the move from $u \in V$ to $v \in V$. $\mathcal{A} \coloneqq \{a_1, \dots, a_m\}$ denotes a set of m agents. Each agent a_i has a given unique task $\mathcal{T}_i \coloneqq (s_i, g_i) \in \mathcal{T}$ to travel from the origin $s_i \in V$ to the destination $g_i \in V$, where $\mathcal{T}$ is the set of tasks. An agent can either move to an adjacent vertex or wait at its current vertex at each time step $t \in \{1, 2, \dots, \infty\}$. The cost of each action is one. Let T_i be a travel time of a_i until the agent arrives at its destination g_i. A path π_i of the agent a_i is defined as a sequence of vertices $\pi_i := \langle v_1^i, v_2^i, \dots, v_t^i, v_{t+1}^i, \dots, v_{T_i}^i \rangle$ satisfying $v_1^i = s_i$ and $v_{T_i}^i = g_i$, where v_t^i is the vertex that a_i visits at time t. The set $\boldsymbol{\pi} := \{\pi_1, \dots, \pi_m\}$ consists of paths for all agents.

A *solution* of a MAPF instance is a set of valid paths of all agents. A path is *valid* if its path has no collisions. A *collision* between a_i and a_j is either a vertex collision (i, j, v, t) (two agents a_i and a_j can never occupy the same vertex v at the same time step t), that is $v = v_t^i = v_t^j$, or an edge collision (i, j, v_1, v_2, t) (two agents a_i and a_j can never switch positions with the same edge (v_1, v_2) from time step t to $t + 1$), that is $v_1 = v_t^i = v_{t+1}^j$ and $v_2 = v_{t+1}^i = v_t^j$.

The objective of MAPF is to find a set of valid paths $\boldsymbol{\pi}^*$ with minimum total travel cost as follows:

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} \sum_{i \in \mathcal{A}} T_i. \tag{1}$$

4 MAPF with Priority

4.1 Formulation

In this section, we introduce a formulation of MAPF with Priority (MAPF-Pri). Let $\mathbf{p} \coloneqq \{p_1, \dots, p_m\}$ be the set of priorities, where $p_i \in \mathbb{R}_+$ is a priority of a_i. $p_i > p_j$ means that a_i has the higher priority task than a_j.

The objective of MAPF-Pri is to find the set $\boldsymbol{\pi}^*$ with minimum total cost as follows:

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} \sum_{i \in \mathcal{A}} p_i \cdot T_i. \tag{2}$$

The valid paths computed by solvers would reflect each agent priority because each agent travel cost is weighted by its priority. MAPF-Pri is obviously *NP-hard* because MAPF, which is a particular case of MAPF-Pri, is *NP-hard* [Yu and LaValle, 2013b].

4.2 Conflict-Based Search for MAPF with Priority

In this subsection, we present Conflict-Based Search for MAPF with Priority (CBS-Pri) as an optimal solver. CBS is a well-known solver guaranteed to return optimal paths in

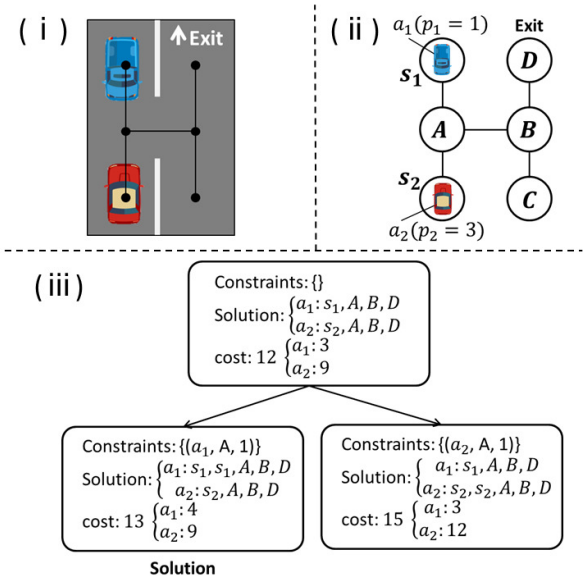

Figure 2: (i) Co-AVP example. (ii) Graph structure of Co-AVP example. (iii) Constraint tree.

problems with valid solutions. In order to accomplish this, CBS adds a set of constraints for each agent and finds paths satisfying the constraints. If there are collisions among these paths, CBS adds a new constraint and resolve the collisions one at a time. While CBS performs a best-first search with the travel cost, CBS-Pri, on the other hand, employs the travel cost weighted based on priority.

The search process of CBS-Pri, like CBS, consists of two hierarchies: the low-level search and the high-level search. In the low-level search, CBS-Pri updates the agent's paths satisfying the constraints imposed by the high-level search. Whereas, in the high-level search, CBS-Pri validates if paths by the low-level search have collisions and if so, CBS-Pri adds a new constraint to resolve the collisions. CBS-Pri stores the set of constraints on a *constraint-tree* (CT). We can straightforwardly prove that CBS-Pri always returns an optimal solution when the instance has valid solutions by following the proofs of the original CBS (see [Sharon *et al.*, 2015]). We describe the processes in detail using the example illustrated in Fig. 2 as below.

Example

Figure 2 (i) and (ii) respectively show a simple example of Co-AVP. In Fig. 2, the problem instance illustrated in diagram (i) can be interpreted by a graph structure as in diagram (ii). The blue and red vehicles are agents a_1 and a_2, respectively; besides, the red vehicle has higher priority ($p_1 = 1, p_2 = 3$). Both vehicles travel to the same goal vertex ($g_1 = g_2 = D$) from each start vertex (s_1 and s_2). The cost of moving and waiting at one timestep is set as one.

In the high-level search, CBS-Pri searches a constraint tree (CT) as shown in Fig. 2 (iii). Each node N in the CT contains the following data: (1) A set of constraints ($N.constraints$), (2) a solution ($N.solution$), and (3) the total cost ($N.cost$) of the current solution ($N.solution$). At the root node of the CT, $N.constraints$ is the empty set. $N.solution$ has the set of paths for all agents by the low-level search. Furthermore, the path π_i for agent a_i must satisfy the constraints of a_i in $N.constraints$. $N.cost$ is calculated by Eq. (2). In the example shown in Fig. 2, $N.solution$ of the root node is $\langle s_1, A, B, D\rangle$ for a_1 and $\langle s_2, A, B, D\rangle$ for a_2. $N.cost$ of the root node is 12 summed up each travel cost weighted by priority p_i. This root node is then push in a *priority queue* (named *OPEN*) and will be expanded next. The expanded node is deleted from *OPEN*. $N.solution$ is not a valid solution because the paths have collisions. Two child nodes are generated to resolve it. CBS-Pri adds constraints $(a_1, A, 1)$ for the left child node and $(a_2, A, 1)$ for the right child node. Furthermore, (a_i, v, t) means "agent a_i does not exist on node v at time t". In the left child node, the low-level search updates the path for a_1 to avoid A at timestep one. The cost of the left child node is now 13. In the right child node, similarly, $N.solution$ is generated with cost 15. The both child nodes are inserted into *OPEN*. Finally, the left child node is expanded because of the lowest cost in *OPEN* and validated if its solution has collisions. Since no collisions exist, the solution of the left child node is returned as an optimal solution.

Algorithm 1 the high-level search of CBS-Pri

Input: MAPF-Pri instance: $G, \mathcal{A}, \mathcal{T}, \mathbf{p}$
Output: $\boldsymbol{\pi}^*$

```
1:  R.constraints ← ∅
2:  R.solution ←low-level(∀a_i ∈ A)
3:  R.cost ← COST(R.solution)
4:  OPEN ← ⟨⟩
5:  insert R to OPEN
6:  while OPEN not empty do
7:      P ← pop the node with the lowest cost from OPEN
8:      for each time step t ∈ max(T_i) do
9:          if collision(a_i, a_j, v, t) exists then
10:             C ← (a_i, a_j, v, t)
11:             for each agent a_i included in C do
12:                 A ← new node
13:                 A.constraints ← P.constraints ∪ (a_i, v, t)
14:                 A.solution ← P.solution
15:                 Update A.solution by low-level(a_i)
16:                 A.cost ← COST(A.solution)
17:                 Insert A to OPEN
18:         else
19:             π* ← P.solution
20:             return π*
```

The high-level search

Algorithm 1 shows the procedure of the high-level search. In this algorithm, *COST*() denotes the function that calculates the cost value by Eq. (2), and low-level (a_i) represents the low-level search for a_i. At lines 1-5, CBS-Pri sets the root node containing initial conditions and insert the node into *OPEN*. At lines 6-20, CBS-Pri expands the node with the lowest cost from *OPEN* and check if the solution is valid (line 9). If the solution has no collisions, CBS-Pri returns it as an optimal solution (lines 19-20). On the other hand, if a collision (a_i, a_j, v, t) or (a_i, a_j, e, t) occurs, CBS-Pri adds new nodes

Algorithm 2 CA*-Pri

Input: MAPF-Pri instance: $G, \mathcal{A}, \mathcal{T}, \mathbf{p}$
Output: $\boldsymbol{\pi}^*$

```
1:  while True do
2:      ORDER ← sort(A, p)
3:      initialize(A, reservation-table)
4:      π* ← ∅
5:      for each agent a_i ∈ ORDER do
6:          π_i ← A*(a_i, reservation-table)
7:          if π_i exists then
8:              reserve(π_i, reservation-table)
9:              π* ← π* ∪ {π_i}
10:         else
11:             break
12:     if |π*| = m then
13:         return π*
```

with new constraints (line 13) and updates the solution and the cost by involving the low-level search (lines 14-16). For simplicity purpose, we describe only node collisions (a_i, a_j, v, t) in Algorithm 1. CBS-Pri inserts the node into *OPEN* (line 17) and continues to expand the next node.

The low-level search

In the low-level search, CBS-Pri finds the optimal path for agent a_i that satisfy all constraints of a_i by the high-level search. In this paper, we use A* algorithm which searches on the space-time map consisting of vertices and time.

4.3 Cooperative A* for MAPF with Priority

Here, we propose a heuristics solver for MAPF-Pri based on Cooperative A* (CA*-Pri). CA* sequentially finds paths of each agent randomly ordered. Each agent searches its collision-free path on the graph by the space-time A* algorithm using a *reservation table*. In the reservation table, each agent writes its path: vertex v at timestep t and/or edge e at timestep from t to $t+1$. When subsequent agents search their paths, they avoid the vertices or edges that have already written in the reservation table. CA*-Pri also searches the path for a_i by space-time A* algorithm using the reservation table but sorts agents based on the priority instead of ordering randomly.

Algorithm 2 shows the procedure. In the algorithm, sort($\mathcal{A}$, $\mathbf{p}$) represents the function to sort agents $a_i \in \mathcal{A}$ by priority $\mathbf{p}$, while initialize($\mathcal{A}$, *reservation-table*) is the function to reserve initial positions of all agents on the reservation table. A*(a_i, *reservation-table*) is the function to search the optimal path for a_i by space-time A* algorithm using the reservation-table, whereas reserve(π_i, *reservation-table*) is the function that writes the path into the reservation-table. CA*-Pri repeats searching and reserving until paths for all agents are calculated. First, CA*-Pri sorts agents by priority and reserves initial positions of all agents (lines 2-3). At lines 4-11, CA*-Pri performs path generation and reservation successively. Besides, CA*-Pri searches the path for a_i by space-time A* (line 6). If the valid path of a_i exists, CA*-Pri writes the path into the reservation table (lines 7-9). Otherwise, CA*-Pri randomly re-orders the agents with the same priority as a_i and repeats the path generation. If the valid paths for all agents are computed, CA*-Pri returns them as solutions (lines 12-13).

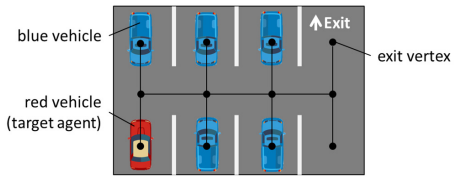

Figure 3: Example of Co-AVP environment. The red vehicle is the target agent. All vehicles move to the exit.

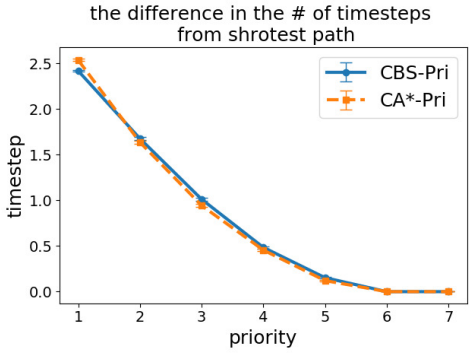

Figure 4: Difference in the number of target agent's travel timesteps compared to the case of the shortest path.

5 Experiments

We evaluated the methods proposed in this paper in synthetic grid graphs simulated Co-AVP environments, named Co-AVP instance. Furthermore, the evaluation of CBS-Pri and CA*-Pri is based on the following aspects: (1) The value of priority, (2) the number of agents, and (3) the number of priority variations. We utilized a PC with an Intel Core i7-3770 CPU at 3.40GHz with 32GB of memory and implemented our code by python. We set a timeout of five minutes for each instance.

For the purpose of simplicity, we assumed in the experiments that all tasks for vehicles are to arrive at exits preassigned for each agent ((a) in Fig. 1). No vehicles moved to vacant spaces and relocated other vacant spaces ((b) and (c) in Fig. 1).

Also, all agents synchronously move to their adjacent vertices or wait at their current vertices at each timestep. We further assumed that agents disappeared immediately after arriving at their destination vertices. Therefore, Co-AVP instances in the experiments always had valid solutions even when some agents shared the exits.

5.1 Experiment 1

We evaluated path length computed by CBS-Pri or CA*-Pri when changing the priorities of the vehicles. We used the simple setting as shown in Fig. 3. The number of agents was six: a red vehicle and five blue vehicles. The black circles show the vertices. All vehicles move on the vertices along the edges of the graph to the common exit vertex. We changed the priorities of the red vehicles from one to seven and the ones of

Table 1: Success rate and average computational time.

method	success rate	average computational time [sec.]
CBS-Pri	1.0	0.13
CA*-Pri	0.96	0.0012

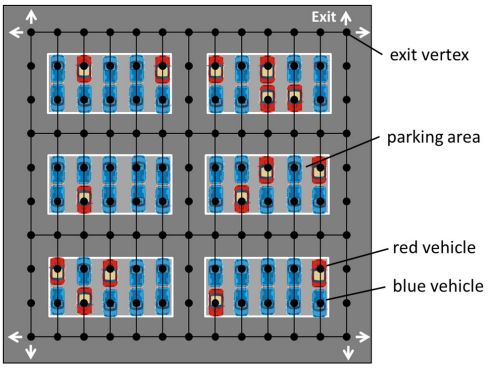

Figure 5: Example environment. The red vehicles move to pre-assigned exits. The blue ones are parked.

the blue vehicles from one to five. We tested the path length in all combinations of the priorities. That is, the number of the tests were 21,875 (7×5^5). Figure 4 shows the mean difference of the red vehicle's path length from its shortest path length with respect to the priorities. The blue solid line and the orange dashed line respectively represent CBS-Pri and CA*-Pri, and the error bars denote the standard error. The results show that the paths computed by our methods converged to the shortest path.

We also evaluated computational time and success rates of the proposed methods with respect to the red vehicle priorities. The computational time represents the running time per instance, and the success rate means the ratio of the number of instances solved within timeout to the total number of instances. Table 1 shows the mean of the computational time and the success rates. The computational time of CA*-Pri was 10^3 times shorter than CBS-Pri. CA*-Pri, however, could not always return valid paths because the paths of the agents with higher priority were precomputed and fixed. On the other hand, the success rate of CBS-Pri was 1.0 because CBS-Pri was theoretically guaranteed to return the valid paths in the instances with feasible solutions if the timeout is long enough.

5.2 Experiment 2

We evaluated our methods in large Co-AVP environments to validate their scalability. We used a 13×10 grid graph as shown in Fig. 5. The number of the red and the blue vehicles was 60. Each red vehicle, which is an agent, had a task traveling to the randomly pre-assigned exit vertex of the four corners. The blue vehicles had no task and stayed in their initial positions as the parking vehicles. The number of the agents was changed from 5 to 40 at intervals of five vehicles. The priority of each agent was assigned an integer value from one to five randomly. We tested for 1,500 times of each condition.

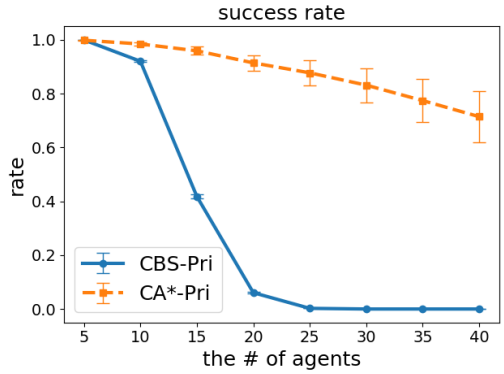

Figure 6: Ratio of agents that successfully reach their goals.

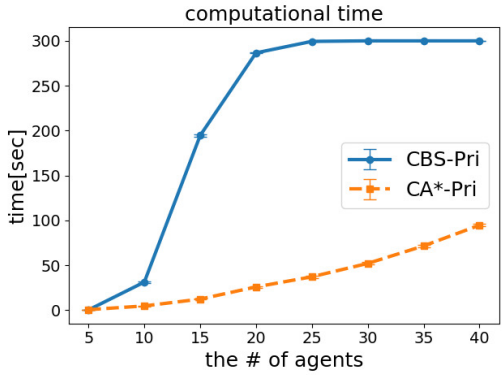

Figure 7: Average computational time.

First, we evaluated the success rates and the computational time of our methods for the number of the agents (Fig. 6 and Fig. 7). The results showed that the success rate of CBS-Pri more drastically decreased with increasing the number of the agents compared to the one of CA*-Pri because the many instances were terminated due to timeout. CA*-Pri had 70% of success rate even when the number of agents was 40, but CBS-Pri had almost 0% when the number of agents exceeded 20. The computational time of the both methods increased for the number of the agents but the increase of CA*-Pri was much more moderate than the one of CBS-Pri. That is, CA*-Pri would be more scalable method than CBS-Pri.

Second, we compared CBS-Pri to CA*-Pri for the travel cost computed by Eq. (2). Figure 8 illustrates the mean travel cost per vehicle for the number of agents. The green dashed and dots line represents the cost in the case that all agents reach their destinations with the shortest paths. That is, the cost is the lower bound, but the paths are not always valid. CBS-Pri could obtain the better solutions than CA*-Pri but the difference was just less than 10% of the travel cost. That is, CA*-Pri could find near optimal solutions compared to the optimal solutions returned CBS-Pri.

Finally, Fig. 9 shows the result of success rates when the number of priority variations was changed. The success rate of CBS-Pri is almost constant while that of CA*-Pri decreases as the number of priority variations increases. CBS-Pri searches

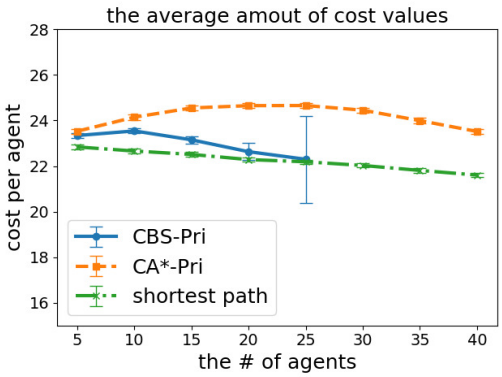

Figure 8: Average amount of cost values.

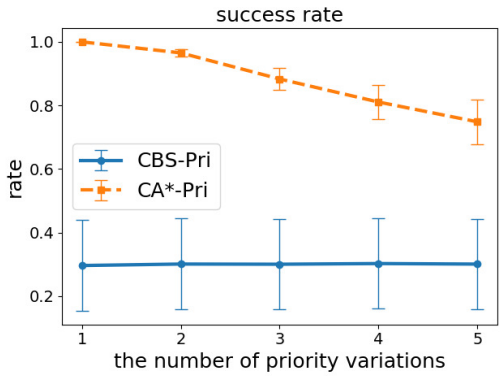

Figure 9: Ratio of agents that successfully reach their goals.

all paths simultaneously, on the other hand, CA*-Pri searches each path sequentially. In CA*-Pri, when all paths are not valid, the order of searching is re-sorted within the same priority only. Therefore CA*-Pri could return no valid solutions when the agents with lower priority needed to be prioritized than the ones with higher priority. This would be why the success rate of CA*-Pri decreased with increasing the number of the priority variations.

6 Conclusion

We formalized MAPF with Priority and applied it to Co-AVP. We proposed two methods: CBS-Pri and CA*-Pri. CBS-Pri is a method based on Conflict-Based Search and it theoretically guaranteed to always return the optimal solutions. On the other hand, CA*-Pri is a heuristic search method based on Cooperative A*. We evaluated our methods using synthetic grid graphs simulated Co-AVP environments. We verified that the high prioritized vehicles can preferentially reach the waiting drivers by our methods. We also evaluated our methods for computational time and the success rate. The results showed that the both methods had pros and cons. CBS-Pri always yield more efficient solutions than or equal to CA*-Pri when returning paths but achieved much lower success rate due to the computational time in the instances with larger number of agents. Thus, CBS-Pri could be employed when the number of agents is small enough but otherwise one has to resort to CA*-Pri.

Our methods for MAPF-Pri is, indeed, promising for other ITS applications such as logistics in a warehouse and traffic management at junctions because we use no assumption for Co-AVP to develop the methods. In the future, we plan the following directions: (1) development of more scalable methods for the number of agents and the size of environment, (2) development of algorithms to treat not only moving to exits but also moving to vacant parking spaces or relocating more suitable spaces, and (3) application to other domains such as logistics.

References

[Al-Absi *et al.*, 2010] Hamada RH Al-Absi, Justin Dinesh Daniel Devaraj, Patrick Sebastian, and Yap Vooi Voon. Vision-based automated parking system. In *ISSPA*, pages 757–760. IEEE, 2010.

[Banzhaf *et al.*, 2017] Holger Banzhaf, Dennis Nienhüser, Steffen Knoop, and J Marius Zöllner. The future of parking: A survey on automated valet parking with an outlook on high density parking. In *IV*, pages 1827–1834. IEEE, 2017.

[Buehler and Wegener, 2003] Oliver Buehler and Joachim Wegener. Evolutionary functional testing of an automated parking system. In *Proceedings of CCCT and ISAS*, 2003.

[Chirca *et al.*, 2015] Mihai Chirca, Roland Chapuis, and Roland Lenain. Autonomous valet parking system architecture. In *ITSC*, pages 2619–2624. IEEE, 2015.

[Conner *et al.*, 2007] David C Conner, Hadas Kress-Gazit, Howie Choset, Alfred A Rizzi, and George J Pappas. Valet parking without a valet. In *IROS*, pages 572–577. IEEE, 2007.

[Felner *et al.*, 2017] Ariel Felner, Roni Stern, E Shimony, Eli Boyarski, M Goldenerg, Guni Sharon, NR Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *SoCS*, 2017.

[González *et al.*, 2016] David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles. *T-ITS*, 17(4):1135–1145, 2016.

[Latombe, 2012] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[Löper *et al.*, 2013] Christian Löper, Claas Brunken, George Thomaidis, Stephan Lapoehn, Paulin Pekezou Fouopi, Henning Mosebach, and Frank Köster. Automated valet parking as part of an integrated travel assistance. In *ITSC*, pages 2341–2348. IEEE, 2013.

[Ma and Koenig, 2016] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. In *Proceedings of AAMAS*, pages 1144–1152. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[Ma *et al.*, 2016] Hang Ma, Craig A Tovey, Guni Sharon, TK Satish Kumar, and Sven Koenig. Multi-agent path

finding with payload transfers and the package-exchange robot-routing problem. In *AAAI*, pages 3166–3173, 2016.

[Ma *et al.*, 2017] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of AAMAS*, pages 837–845. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[Ma *et al.*, 2018] Hang Ma, Glenn Wagner, Ariel Felner, Jiaoyang Li, TK Kumar, and Sven Koenig. Multi-agent path finding with deadlines. *arXiv preprint arXiv:1806.04216*, 2018.

[Oentaryo and Pasquier, 2004] Richard Jayadi Oentaryo and Michel Pasquier. Self-trained automated parking system. In *ICARCV*, volume 2, pages 1005–1010. IEEE, 2004.

[Rodrigue *et al.*, 2016] Jean-Paul Rodrigue, Claude Comtois, and Brian Slack. *The geography of transport systems*. Routledge, 2016.

[Shaheen, 2005] Susan Shaheen. Smart parking management field test: A bay area rapid transit (bart) district parking demonstration. *Institute of Transportation Studies*, 2005.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Silver, 2005] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[Song, 2013] B Song. Cooperative lateral vehicle control for autonomous valet parking. *International Journal of Automotive Technology*, 14(4):633–640, 2013.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. 2010.

[Surynek, 2012] Pavel Surynek. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*, pages 564–576. Springer, 2012.

[Wagner and Choset, 2011] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *IROS*, pages 3260–3267. IEEE, 2011.

[Yu and LaValle, 2013a] Jingjin Yu and Steven M LaValle. Multi-agent path planning and network flow. pages 157–173, 2013.

[Yu and LaValle, 2013b] Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 2013.