2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Toward Efficient Physical and Algorithmic Design of Automated Garages

Teng Guo Jingjin Yu

Abstract— **Parking in large metropolitan areas is often a time-consuming task with further implications for traffic patterns that affect urban landscaping. Reducing the premium space needed for parking has led to the development of automated mechanical parking systems. Compared to regular garages having one or two rows of vehicles on each island, automated garages can have multiple rows of vehicles stacked together to support higher parking demands. Although this multi-row layout reduces parking space, it makes parking and retrieval more complicated. In this work, we propose an automated garage design that supports nearly 100% parking density. Modeling the problem of parking and retrieving multiple vehicles as a special class of multi-robot path planning problem, we propose associated algorithms for handling all common operations of the automated garage, including (1) optimal algorithm and near-optimal methods that find feasible and efficient solutions for simultaneous parking/retrieval and (2) a novel shuffling mechanism to rearrange vehicles to facilitate scheduled retrieval at rush hours. We conduct thorough simulation studies showing the proposed methods are promising for large and high-density real-world parking applications.**

I. INTRODUCTION

The invention of automated parking systems (garages) helps solve parking issues in areas where space carries significant premiums, such as city centers and other heavily populated areas. Nowadays, parking space is becoming increasingly scarce and expensive; a spot in Manhattan could easily surpass $200,000$ USD. Developing garages supporting high-density parking that save space and are more convenient is thus highly attractive for economic/efficiency reasons.

In automated garages, human drivers only need to drop off (pick up) the vehicle in a specific I/O (Input/Output) port without taking care of the parking process. Vehicles in such a system do not require ambient space for opening the doors, and can thus be parked much closer. Moving a vehicle to a parking spot or a port is the key function for such systems. One of the solutions is to use robotic valets to move vehicles. Such systems are already commercially available, such as HKSTP [1] in Hong Kong. In such systems [2], vehicles are parked such that they may block each other, requiring multiple rearrangements to retrieve a specific vehicle. Unfortunately, little information can be found on how well these systems function, e.g., their parking/retrieval efficiency. Other solutions focus on parking for self-driving vehicles. In such an automated garage, vehicles are able to drive themselves to parking slots and ports. This makes the system more flexible but provides limited space-saving advantages, besides requiring autonomy from the vehicles.

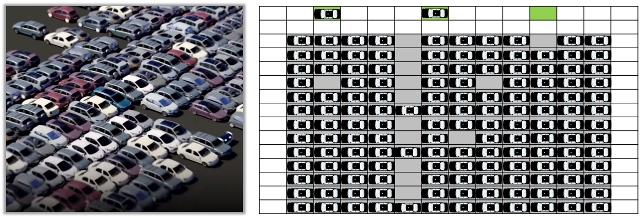


Fig. 1. Left: an illustration of the density level of the envisioned automated garage system. Right: The grid-based abstraction with three I/O ports for vehicle dropoff and retrieval.

Recently, many efficient multi-robot path planning algorithms have been proposed, making it possible for lowering parking and retrieval cost using multiple robotic valets. In this work, we study multi-robot based parking and retrieval problem, proposing a complete automated garage design, supporting near 100% parking density, and developing associated algorithms for efficiently operating the garage.

Results and contributions. The main results and contributions of our work are as follows. In designing the automated garage, we introduce *batched vehicle parking and retrieval* (BVPR) and *continuous vehicle parking and retrieval* (CVPR) problems modeling the key operations required by such a garage, which facilitate future theoretical and algorithmic studies of automated garage systems.

On the algorithmic side, we study a system that can support parking density as high as $(m_1-2)(m_2-2)/m_1m_2$ on a $m_1 \times m_2$ grid map and allow multi-vehicle parking and retrieving, which approaches 100% parking density for large garages. Leveraging the regularity of the system, which is grid-like, we propose an optimal ILP-based method and a fast suboptimal algorithm based on sequential planning. Our suboptimal algorithm is highly scalable while maintaining a good level of solution quality, making it suitable for large-scale applications. We further introduce a shuffling mechanism to rearrange vehicles during off-peak hours for fast vehicle retrieval during rush hours, if the retrieval order can be anticipated. Our rearrangement algorithm performs such shuffles with total time cost of $O(m_1m_2)$ at near full garage density.

Related work. Researchers have proposed diverse approaches toward efficient high-density parking solutions. Many systems for self-driving vehicles have been studied [3]–[5], where vehicles are parked using a central controller and may be stacked in several rows and can block each other. These designs increase parking capacity by up to 50%. However, the retrieval becomes highly complex due to blockages and is heavily affected by the maneuverability

---

G. Teng, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. Emails: {`teng.guo, jingjin.yu`}`@rutgers.edu`. This work is supported in part by NSF award IIS-1845888 and an Amazon Research Award.

979-8-3503-2365-8/23/$31.00 ©2023 IEEE

of self-driving vehicles.

With most vehicles being incapable of self-driving, robotic valet based high-density parking systems could be a more appropriate choice. The Puzzle Based Storage (PBS) system or grid-based shuttle system, proposed originally by [6], is one of the most promising high-density storage systems. In such a system, storage units, which can be AGVs or shuttles, are movable in four cardinal directions. There must be at least one empty cell (escort). To retrieve a vehicle, one must utilize the escorts to move the desired vehicles to an I/O port. This is similar to the 15-puzzle, which is known to be NP-hard to optimally solve [7]. Optimal algorithms for retrieving one vehicle with a single escort and multiple escorts have been proposed in [6], [8]. However, these methods only consider retrieving one single vehicle at a time. Besides, the average retrieval time can be much longer than conventional aisle-based solutions. To achieve a trade-off between capacity demands and retrieval efficiency, we suggest using more escorts and I/O ports that allow retrieving and parking multiple vehicles simultaneously by utilizing recent advanced Multi-Robot Path Planning (MRPP) algorithms [9].

MRPP has been widely studied. In the static or one-shot setting [10], given a graph environment and a number of robots with each robot having a unique start position and a goal position, the task is to find collision-free paths for all the robots from start to goal. It has been proven that solving one-shot MRPP optimally in terms of minimizing either makespan or sum of costs is NP-hard [11], [12]. Solvers for MRPP can be categorized into *optimal* and *suboptimal*. Optimal solvers either reduce MRPP to other well-studied problems, such as ILP [13], SAT [14] and ASP [15] or use search algorithms to search the joint space to find the optimal solution [16], [17]. Due to the NP-hardness, optimal solvers are not suitable for solving large problems. Bounded suboptimal solvers [18] achieve better scalability while still having a strong optimality guarantee. However, they still scale poorly, especially in high-density environments. There are polynomial time algorithms for solving large-scale MRPP [19], which are at the cost of solution quality. Other $O(1)$ time-optimal polynomial time algorithms [20]–[23] are mainly focusing on minimizing the makespan, which is not very suitable for continuous settings.

Organization. The rest of the paper is organized as follows. Sec. II covers the preliminaries including garage design. In Sec. III-Sec. IV, we provide the algorithms for operating the automated garage. We perform thorough evaluations and discussions of the garage system in Sec. V and conclude with Sec. VI.

II. PRELIMINARIES

A. Garage Design Specification

In this study, the automated grid-based garage is a four-connected $m_1 \times m_2$ grid $\mathcal{G}(\mathcal{V}, \mathcal{E})$ (see Fig. 1). There are n_o I/O ports (referred simply as *ports* here on) distributed on the top border of the grid for dropping off vehicles for parking or for retrieving a specific parked vehicle. A port can only be used for either retrieving or parking at a given time. Vehicles must be parked at a *parking spot*, a cell of the lower center $(m_1 - 2) \times (m_2 - 2)$ subgrid. Once a vacant spot is parked, it becomes a movable obstacle. $\mathcal{O} = \{o_1, ..., o_{n_o}\}$ is the set of ports and $\mathcal{P} = \{p_1, ..., p_{|\mathcal{P}|}\}$ is the set of parking spots.

B. Batched Vehicle Parking and Retrieval (BVPR)

Batched vehicle parking and retrieval, or BVPR, seeks to optimize parking and retrieval in a *batch* mode. In a single batch, there are n_p vehicles to park, and n_r vehicles to retrieve, n_l parked vehicles to remain. Denote $\mathcal{C} = \mathcal{C}_p \cup \mathcal{C}_r \cup \mathcal{C}_l$ as the set of all vehicles. At any time, the maximum capacity cannot be exceeded, i.e., $|\mathcal{C}| < |\mathcal{P}|$. Time is discretized into timesteps and multiple vehicles carried by AGVs/shuttles can move simultaneously. In each timestep, each vehicle can move left, right, up, down or wait at the current position. Collisions among the vehicles should be avoided:

1) Meet collision. Two vehicles cannot be at the same grid point at any timestep: $\forall i, j \in \mathcal{C}, v_i(t) \neq v_j(t)$;
2) Head-on collision. Two vehicles cannot swap locations by traversing the same edge in the opposite direction: $\forall i, j \in \mathcal{C}, (v_i(t) = v_j(t+1) \wedge v_i(t+1) = v_j(t)) =$ false;
3) Perpendicular following collisions (see Fig. 2). One vehicle cannot follow another when their moving directions are perpendicular. Denote $\hat{e}_i(t) = v_i(t+1) - v_i(t)$ as the moving direction vector of vehicle i at timestep t, then $\forall i, j \in \mathcal{C}, (v_i(t+1) = v_j(t) \wedge \hat{e}_i(t) \perp \hat{e}_j(t)) =$ false.

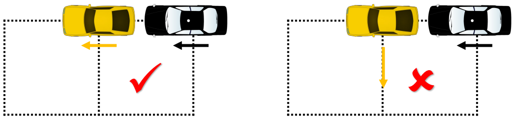


Fig. 2. While parallel movement of vehicles in the same direction (left) is allowed, *perpendicular following* of vehicles (right) is forbidden.

Unlike certain MRPP formulations [10], we need to consider perpendicular following collisions, which makes the problem harder to solve. The task is to find a collision-free path for each vehicle, moving it from its current position to a desired position. Specifically, for a vehicle to be retrieved, its goal is a specified port. For other vehicles, the desired position is any one of the parking spots. The following criteria are used to evaluate the solution quality:

1) Average parking/retrieving time (APRT): the average time required to retrieve or park a vehicle.
2) Makespan (MKPN): the time required to move all vehicles to their desired positions.
3) Average number of moves per task (ANM): the sum of the distance of all vehicles divided by the total number of vehicles in $\mathcal{C}_p \cup \mathcal{C}_r$.

In general, these objectives create a Pareto front [12] and it is not always possible to simultaneously optimize any two of these objectives.

C. Continuous Vehicle Parking and Retrieval (CVPR)

CVPR is the continuous version of the vehicle parking and retrieval problem. It inherits most of BVPR's structure, but with a few key differences. In this formulation, we make the following assumptions. When a vehicle $i \in \mathcal{C}_r$ arrives at its desired port, it would be removed from the environment. There will be new vehicles appearing at the ports that need to be parked (within capacity) and there would be new requests for retrieving vehicles. Besides, when a port is being used for retrieving a vehicle, other users cannot park vehicles at the port until the retrieval task is finished. Except for MKPN when the time horizon is infinite or fixed, the three criteria can still be used for evaluating the solution quality.

D. *Vehicle Shuffling Problem (*VSP*)*

In real-world garages, there are often off-peak periods (e.g., after the morning rush hours) where the system reaches its capacity and there are few requests for retrieval. If the retrieval order of the vehicles at a later time (e.g., afternoon rush hours) is known, then we can utilize the information to reshuffle the vehicles to facilitate the retrieval later. This problem is formulated as a one-shot MRPP. Given the start configuration X_I and a goal configuration X_G, we need to find collision-free paths to achieve the reconfiguration. The goal configuration is determined according to the retrieval time order of the vehicles; vehicles expected to be retrieved earlier should be parked closer to the ports so that they are not blocked by other vehicles that will leave later.

III. SOLVING BVPR

A. *Integer Linear Programming (*ILP*)*

Building on network flow based ideas from [13], [24], we reduce BVPR to a multi-commodity max-flow problem and use integer programming to solve it. Vehicles in $\mathcal{C}_r$ have a specific goal location (port) and must be treated as different commodities. On the other hand, since the vehicles in $\mathcal{C}_p \cup \mathcal{C}_l$ can be parked at any one of the parking slot, they can be seen as one single commodity. Assuming an instance can be solved in T steps, we construct a T-step time-extended network as shown in Fig. 3.

A T-step time-extended network is a directed network $\mathcal{N}_T = (\mathcal{V}_T, \mathcal{E}_T)$ with directed, unit-capacity edges. The network $\mathcal{N}_T$ contains $T+1$ copies of the original graph's vertices $\mathcal{V}$. The copy of vertex $u \in \mathcal{V}$ at timestep t is denoted as u_t. At timestep t, an edge (u_t, v_{t+1}) is added to $\mathcal{E}_T$ if $(u,v) \in \mathcal{E}$ or $v = u$. For $i \in \mathcal{C}_r$, we can give a supply of one unit of commodity type i at the vertex s_{i0} where s_i is the start vertex of i. To ensure that vehicle i arrives at its port, we add a feedback edge connecting its goal vertex g_i at T to its source node s_{i0}. For the vehicles that need to be parked, we create an auxiliary source node α and an auxiliary sink node β. For each $i \in \mathcal{C}_p \cup \mathcal{C}_l$, we add an edge of unit capacity connecting node α to its starting node s_{i0}. As the vehicles can be parked at any one of the parking slots, for any vertex $u \in \mathcal{P}$ we add an edge of unity capacity connecting u_T and β. A supply of $n_p + n_l$ unit of commodity of the type for the vehicles in $\mathcal{C}_p \cup \mathcal{C}_l$ can be given at the node α.

To solve the multi-commodity max-flow using ILP, we create a set of binary variables $X = \{x_{iuvt}\}, i = 0, ..., n_r, (u,v) \in \mathcal{E}$ or $u = v$, $0 \le t \le T$; a variable set to true means that the corresponding edge is used in the final solution. The ILP formulation is given as follows.

$$\text{Minimize} \quad \sum_{i,t,u \neq v} x_{iuvt} \tag{1}$$

$$\text{subject to} \quad \forall t, v, i \quad \sum_{u} x_{iuv(t-1)} = \sum_{w} x_{ivwt} \tag{2}$$

$$\forall t, i, v \quad \sum_{v} x_{iuvt} \le 1 \tag{3}$$

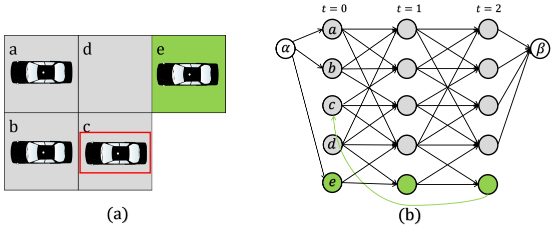
$$\forall t, i, (u,v) \in \mathcal{E} \quad \sum_{i} (x_{iuvt} + x_{ivut}) \le 1 \tag{4}$$

$$\forall t, i, (u,v) \perp (v,w) \sum_{i} (x_{iuvt} + x_{ivwt}) \le 1 \tag{5}$$

$$\sum_{i=0}^{n_r - 1} x_{i g_i s_i T} + \sum_{u \in \mathcal{P}} x_{n_r u \beta T} = n_p + n_r + n_l \tag{6}$$

$$x_{iuvt} = \begin{cases} 0 & \text{if } i \text{ does not traverse edge } (u,v) \text{ at } t \\ 1 & \text{if } i \text{ traverses edge } (u,v) \text{ at } t \end{cases} \tag{7}$$

In Eq. (1), we minimize the total number of moves of all vehicles within the time horizon T. Eq. (2) specifies the flow conservation constraints at each grid point. Eq. (3) specifies the vertex constraints to avoid meet-collisions. In Eq. (4), the vehicles are not allowed to traverse the same edge in opposite directions. Eq. (5) specifies the constraints that forbid perpendicular following conflicts. If the programming for T-step time-expanded network is feasible, then the solution is found. Otherwise, we increase T step by step until there is a feasible solution. The smallest T for which the integer programming has a solution is the minimum makespan. As a result, the ILP finds a makespan-optimal solution minimizing the total number of moves as a secondary objective.



(a) (b)

Fig. 3. (a) A illustrative BVPR instance with 4 parking spots and one port. The vehicles within the red rectangle need to be retrieved while other vehicles should be parked in the gray areas. (b) The 2-step flow network reduced from the BVPR instance.

B. *Efficient Heuristics for High-Density Planning*

ILP can find makespan-optimal solutions but it scales poorly. We seek a fast algorithm that is able to quickly solve dense BVPR instances at a small cost of solution quality. The algorithm we propose is built on a single-vehicle motion primitive of retrieving and parking.

1) Motion Primitive for Single-Vehicle Parking/Retrieving: Regardless of the solution quality, BVPR can be solved by sequentially planning for each vehicle in $\mathcal{C}_r \cup \mathcal{C}_p$. Specifically, for each round we only consider completing one single task for a given vehicle $i \in \mathcal{C}_r \cup \mathcal{C}_p$, which is either moving i to its port or one of the parking spots. After vehicle i arrives at its destination, the next task is solved. The examples in Fig. 4 and Fig. 5, where the maximum capacity is reached, illustrate the method's operations.

For the retrieval scenario in Fig. 4, the vehicle marked by the red rectangle is to be retrieved. For realizing the intuitive path indicated by the dashed lines on the left, vehicles blocking the path should be cleared out of the way, which can be easily achieved by moving those blocking vehicles one step to the left or to the right, utilizing the two empty columns. Such a motion primitive can always successfully retrieve a vehicle without deadlocks.

For the parking scenario in Fig. 5, we need to park the vehicle in the green port. We do so by first searching for an empty spot (escort) greedily. Using the mechanism of parallel moving of vehicles, the escort can first be moved to the column of the parking vehicle in one timestep and then moved to the position right below the vehicle in one timestep. After that, the vehicle can move directly to the escort, which is its destination.

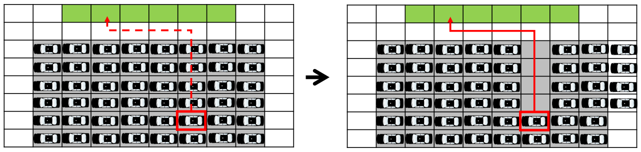


Fig. 4. The motion primitive for retrieving a vehicle.

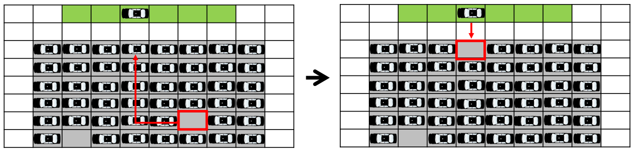


Fig. 5. The motion primitive for parking a vehicle.

Sequential planning always returns a solution if there is one; however, when multiple parking and retrieval requests are to be executed, sequential solutions result in poor performance as measured by MKPN and ARPT.

2) Coupling Single Motion Primitives by MCP *(*CSMP*):* Minimal Communication Policy (MCP) [25] is a robust multi-robot execution policy to handle unexpected delays without stopping unaffected robots. During execution, MCP preserves the order by which robots visit each vertex as in the original plan. When a robot i is about to perform a move action and enter a vertex v, MCP checks whether robot i is the next to enter that vertex by the original plan. If a different robot, j, is planned to enter v next, then i waits in its current vertex until j leaves v.

We use MCP to introduce concurrency to plans found through sequential planning. The algorithm is described in Alg. 1 and Alg. 2. Alg. 1 describes the framework of CSMP. First, we find initial plans for all the vehicles in $\mathcal{C}_r \cup \mathcal{C}_p$ one by one (Line 4-6). After obtaining the paths, we remove all the waiting states and record the order of vehicle visits for each vertex in a list of queues (Line 7). Then we enter a loop executing the plans using MCP until all vehicles have finished the tasks and reach their destination (Line 8-15). In Alg. 2, if i is the next vehicle that enters vertex v_i according to the original order, we check if there is a vehicle currently at v_i. If there is not, we let i enter v_i. If another vehicle j is currently occupying v_j, we examine if j is moving to its next vertex v_j in the next step by recursively calling the function `MCPMove`. If j is moving to v_j in the next step and the moving directions of i, j are not perpendicular, we let vehicle i enter vertex v_i. Otherwise, i should wait at u_i. The algorithm is deadlock-free by construction; we omit the relatively straightforward proof due to the page limit.

Proposition III.1. CSMP *is dead-lock free and always finds a feasible solution in finite time if there is one.*

Algorithm 1: CSMP

```
1  Function CSMP ():
2      foreach $v \in \mathcal{V}$, $VOrder[v] \leftarrow Queue()$
3      $InitialPlans \leftarrow \{\}$
4      for $i \in \mathcal{C}_p \cup \mathcal{C}_r$ do
5          SingleMP($i, InitialPlans$)
6      Preprocess($InitialPlans, VOrder$)
7      while True do
8          for $i \in \mathcal{C}$ do
9              $mcpMoved \leftarrow Dict()$
10             MCPMove ($i$)
11         if AllReachedGoal()=true then
12             break
```

Algorithm 2: MCPMove

```
1  Function MCPMove ($i$):
2      if $i$ in $mcpMoved$ then
3          return $mcpMoved[i]$
4      $u_i \leftarrow$ current position of $i$
5      $v_i \leftarrow$ next position of $i$
6      if $i = VOrder[v_i].front()$ then
7          $j \leftarrow$ the vehicle currently at $v_i$
8          if $j$=None or (MCPMove ($j$)=true and
             $(u_i, v_i) \not\perp (u_j, v_j)$) then
9              move $i$ to $v_i$
10             $VOrder[v_i].popfront()$
11             $mcpMoved[i]$=true
12             return true
13     let $i$ wait at $u_i$
14     $mcpMoved[i]$=false
15     return false
```

3) Prioritization: Sequential planning can always find a solution regardless of the planning order. However, priorities will affect the solution quality. Instead of planning by a random priority order (Alg. 1 Line 4-6), when possible, we can first plan for parking since single-vehicle parking only takes two steps. After all the vehicles in $\mathcal{C}_p$ have been parked, we apply `SingleMP` to retrieve vehicles. Among vehicles in $\mathcal{C}_r$, we first apply `SingleMP` for those vehicles that are closer to their port so that they can reach their targets earlier and will not block the vehicles at the lower row.

C. Complexity Analysis

In this section, we analyze the time complexity and solution makespan upper bound of the CSMP. In `SingleMP`, in order to park/retrieve one vehicle, we assume n_b vehicles may cause blockages and need to be moved out of the way. Clearly $n_b < n$ where $n = n_p + n_r + n_l$. Therefore, the complexity of computing the paths using `SingleMP` for all vehicles in $\mathcal{C}_r \cup \mathcal{C}_p$ is bounded by $(n_p+n_r)n$. The path length of each single-vehicle path computed by `SingleMP` is no more than $m_1 + m_2$. The makespan of the paths obtained by concatenating all the single-vehicle paths is bounded by $n_r(m_1+m_2)+2n_p$. This means that MCP will take no more than $n_r(m_1+m_2)+2n_p$ iterations. Therefore, the makespan of the solution is upper bounded by $n_r(m_1 + m_2) + 2n_p$. In each loop of MCP, we essentially run DFS on a graph that has $n = n_p + n_r + n_l$ nodes and traverse all the nodes, for which the time complexity is $O(n)$, Therefore the time complexity of CSMP is $O(n(n_r m_1 + n_r m_2 + 2n_p))$. In summary, the time complexity of CSMP is bounded by $O(n(n_r m_1 + n_r m_2 + 2n_p))$, while the makespan is upper bounded by $n_r(m_1 + m_2) + 2n_p$.

D. *Extending* CSMP *to* CVPR

CSMP can be readily adapted to solve CVPR. Similar to the BVPR version, we call `MCPMove` for each vehicle at each timestep. When a new request comes at some timestep, we compute the paths for the associated vehicles using the `SingleMP` and update the information of vertex visit order. That is, when we apply `SingleMP` on a vehicle $i \in \mathcal{C}_p \cup \mathcal{C}_r$, if vehicle j will visit vertex u at timestep t', then we push i to the queue $VOrder[u]$, where the queue is always sorted by the entering time of u. In this way, MCP will execute the plans while maintaining the visiting order. The previously planned vehicles will not be affected by the new requests and keep executing their original plan. The main drawback of this method is that it usually has worse solution quality than replanning since the visiting order is fixed.

IV. SOLVING VSP VIA RUBIK TABLES

VSP is essentially solving a static/one-shot MRPP. On an $m_1 \times m_2$ grid, it can be solved by applying the Rubik Table algorithm [26], using no more than $2(m_2-2)$ column shuffles and (m_1-2) row shuffles. As an example shown in Fig. 6, we may use two nearby columns to shuffle the vehicles in a given column fairly efficiently, requiring only $O(m_1)$ steps [20]. The same applies to row shuffles. Depending on the number of parked vehicles, one or more multiple row/column shuffles may be carried out simultaneously. We have (straightforward proofs are omitted due to limited space)

Proposition IV.1. VSP *may be solved using* $O(m_1 m_2)$ *makespan at full garage capacity and* $O(m_1+m_2)$ *makespan when the garage has* $\Theta(m_1 m_2)$ *empty spots and* $\Omega(m_1 m_2)$ *parked vehicles. In contrast, with* $\Omega(m_1 m_2)$ *parked vehicles, the required makespan for solving* VSP *is* $\Omega(m_1 + m_2)$.

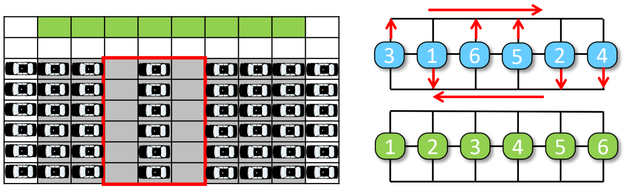


Fig. 6. Illustration of the mechanism for “shuffling” a single row/column.

V. EVALUATION

In this section, we evaluate the proposed algorithms. All experiments are performed on an Intel® Core™ i7-9700 CPU at 3.0GHz. Each data point is an average over 20 runs on randomly generated instances unless otherwise stated. ILP is implemented in C++ and other algorithms are implemented in CPython. A video of the simulation can be found at `https://youtu.be/CZTaxnAS7TU`.

A. *Algorithmic Performance on* BVPR

Varying grid sizes. In the first experiment, we evaluate the proposed algorithms on $m \times m$ grids with varying grid side length, under the densest scenarios: there are $(m-2)^2$ vehicles in the system and all the ports are used for either parking or retrieving $(n_p+n_r = n_o)$. The result can be found in Fig. 7. CONCAT is the method that simply concatenates the sing-vehicle paths. In rCSMP, we apply `SingleMP` on vehicles with random priority order, while in pCSMP, we apply the prioritization strategy.

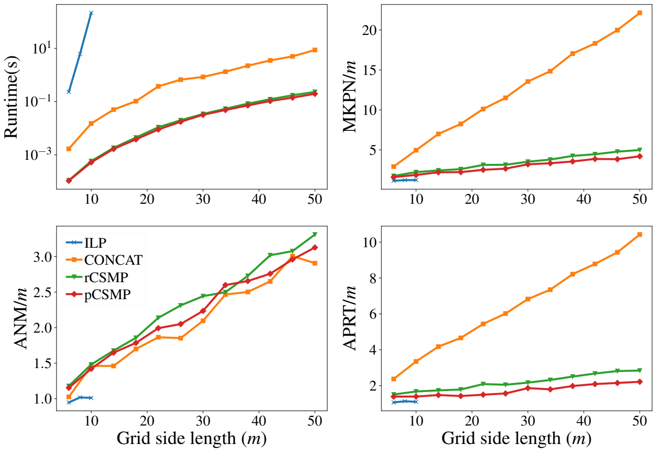


Fig. 7. Runtime, MKPN, APRT, AVN data of the proposed methods on $m \times m$ grids under the densest scenarios.

Among the methods, ILP has the best solution quality in terms of MKPN, ANM, and APRT, which is expected since its optimality is guaranteed. However, ILP has the poorest scalability, hitting a limit with $m \leq 10$ and $n \leq 64$. CONCAT, rCSMP, and pCSMP are much more scalable, capable of solving instances on 50×50 grids with 2304 vehicles in a few seconds. Since CONCAT just concatenates sing-vehicle paths, this results in very long paths compared to rCSMP and pCSMP; we observe that the MCP procedure greatly improves the concurrency, leading to much better solution quality. MKPN and APRT of paths obtained by CONCAT can be $10m$-$20m$ while the MKPN and APRT of the paths obtained by rCSMP and pCSMP are $2m$-$4m$. MKPN and APRT of pCSMP with prioritization strategy is about 20% lower than these of rCSMP.

Impact of vehicle density. In the second experiment, we examine the behavior of algorithms as vehicle density changes, fixing grid size at 20×20. We still let $n_p+n_r = n_o$. The result is shown in Fig. 8. As in the previous case, ILP can only solve instances with a density below 20%

in a reasonable time, while the other three algorithms can all tackle the densest scenarios. For all algorithms, vehicle density in $\mathcal{C}_l$ has limited impact on MKPN and APRT, where rCSMP and pCSMP have much better quality than CONCAT. In low-density scenarios, fewer vehicles need to move, which may cause blockages for retrieving/parking a vehicle. As a result, ANM increases as vehicle density increases.

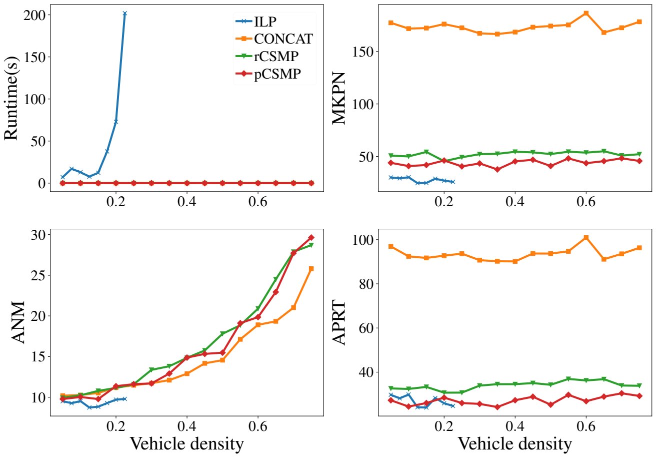


Fig. 8. Runtime, MKPN, APRT, ANM data of the proposed methods on 20×20 grids with varying vehicle density.

B. *Algorithmic Performance on* CVPR

Random retrieving and parking. We test the continuous CSMP on a 12×12 grid with 10 ports. In each time step, if a port is available, there would be a new vehicle that need to be parked appearing at this port with probability p_p if it does not exceed the capacity. And with probability p_r this port will be used to retrieve a random parked vehicle if there is one. We simulate the following three scenarios:

(i). Morning rush hours. Initially, no vehicles are parked. There are many more requests for parking than retrieving: $p_p = 0.6, p_r = 0.01$.

(ii). Workday hours. Initially, the garage is full. Request for parking and retrieval are equal: $p_p = p_r = 0.05$.

(iii). Evening rush hours. Initially, the garage is full. Retrieval requests dominate parking: $p_p = 0.01, p_r = 0.6$.

The maximum number of timesteps is set to 500. We evaluate the average retrieval time, average parking time, and total number of moves under these scenarios. The result is shown in Fig. 9. Online CSMP achieves the best performance in the morning due to fewer retrievals. On the other hand, the average retrieval time in all three scenarios is less than $2m$ and parking time is less than m. This shows that the algorithm is able to plan paths with good solution quality even in the densest scenarios and rush hours.

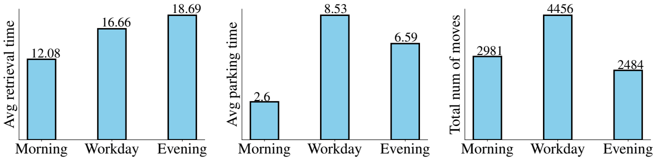


Fig. 9. CVPR performance statistics under three garage traffic patterns.

Benefits of shuffling. In this experiment, we examine the effect of the shuffling (for solving VSP). We assume that each vehicle is assigned a retrieval priority order, as to be expected in the evening rush hours when some people go home earlier than others. We perform the *column* shuffle operations on the vehicles to facilitate the retrieval. The makespan, average number of moves, and computation time of the column shuffle operations on $m \times m$ grids with different grid sizes under the densest settings are shown in Fig. 10(a)-(c). While the paths of shuffling can be computed in less than 1 second, the makespan of completing the shuffles scales linearly with respect to m^2 and the average number of moves scales linearly with respect to m.

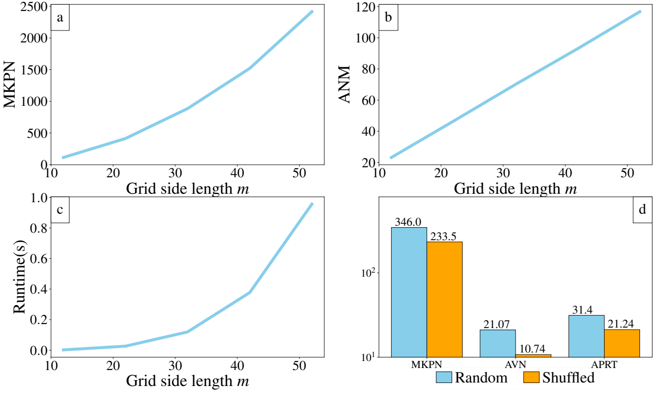


Fig. 10. Statistics of performing the column shuffle operations on $m \times m$ grids with varying grid size.

After shuffling, continuous CSMP with $p_r = 1, p_p = 0$ is applied to retrieve all vehicles and compared to the case where no shuffling is performed. The outcome makespan, average number of moves per vehicle, and average retrieval time per vehicle are shown in Fig. 10(d). Compared to unshuffled configuration, CSMP is able to retrieve all the vehicles with 30%-50% less number of moves and retrieval time (note that logarithmic scale is used to fit all data), showing that rearranging vehicles in anticipation of rush hour retrieval provides significant benefits.

VI. CONCLUSION AND DISCUSSIONS

In this work, we present the complete physical and algorithmic design of an automated garage system, aiming at allowing the dense parking of vehicles in metropolitan areas at high speeds/efficiency. We model the retrieving and parking problem as a multi-robot path planning problem, allowing our system to support nearly 100% vehicle density. The proposed ILP algorithm can provide makespan-optimal solutions, while CSMP algorithms are highly scalable with good solution quality. Also clearly shown is that it can be quite beneficial to perform vehicle rearrangement during non-rush hours for later ordered retrieval operations, which is a unique high-utility feature of our automated garage design.

For future work, we intend to further improve CSMP's scalability and flexibility, possibly leveraging the latest advances in MRPP/MAPF research. We also plan to extend the automated garage design from 2D to 3D, supporting multiple levels of parking. Finally, we would like to build a small-scale test beds realizing the physical and algorithmic designs.

REFERENCES

[1] Yeefung Automation, “First AGV Robotic Parking System in Hong Kong developed by Yeefung,” 2021. [Online]. Available: https://www.youtube.com/watch?v=M225gM7rplw

[2] A. K. Nayak, H. Akash, and G. Prakash, “Robotic valet parking system,” in *2013 Texas Instruments India Educators' Conference*. IEEE, 2013, pp. 311–315.

[3] M. Ferreira, L. Damas, H. Conceicao, P. M. d'Orey, R. Fernandes, P. Steenkiste, and P. Gomes, “Self-automated parking lots for autonomous vehicles based on vehicular ad hoc networking,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 472–479.

[4] J. Timpner, S. Friedrichs, J. Van Balen, and L. Wolf, “K-stacks: high-density valet parking for automated vehicles,” in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 895–900.

[5] M. Nourinejad, S. Bahrami, and M. J. Roorda, “Designing parking facilities for autonomous vehicles,” *Transportation Research Part B: Methodological*, vol. 109, pp. 110–127, 2018.

[6] K. R. Gue and B.-S. Kim, “Puzzle-based storage systems,” *Naval Research Logistics (NRL)*, vol. 54, 2007.

[7] D. Ratner and M. K. Warmuth, “Finding a shortest solution for the n× n extension of the 15-puzzle is intractable.” in *AAAI*, 1986, pp. 168–172.

[8] H. Yu, Y. Yu, and M. de Koster, “Optimal algorithms for scheduling multiple simultaneously movable empty cells to retrieve a load in puzzle-based storage systems,” *Available at SSRN 3506480*, 2016.

[9] A. Okoso, K. Otaki, S. Koide, and T. Nishi, “High density automated valet parking via multi-agent path finding,” in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 2146–2153.

[10] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar, *et al.*, “Multi-agent pathfinding: Definitions, variants, and benchmarks,” in *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[11] P. Surynek, “A novel approach to path planning for multiple robots in bi-connected graphs,” in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3613–3619.

[12] J. Yu and S. M. LaValle, “Structure and intractability of optimal multi-robot path planning on graphs,” in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[13] J. Yu and S. M. LaValle, “Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics,” *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[14] P. Surynek, “An optimization variant of multi-robot path planning is intractable,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010.

[15] E. Erdem, D. G. Kisa, U. Oztok, and P. Schüller, “A general formal framework for pathfinding problems with multiple agents,” in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[16] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, “Conflict-based search for optimal multi-agent pathfinding,” *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[17] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, “The increasing cost tree search for optimal multi-agent pathfinding,” *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[18] M. Barer, G. Sharon, R. Stern, and A. Felner, “Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem,” in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[19] R. J. Luna and K. E. Bekris, “Push and swap: Fast cooperative pathfinding with completeness guarantees,” in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[20] T. Guo and J. Yu, “Sub-1.5 time-optimal multi-robot path planning on grids in polynomial time,” in *Robotics: Sciences and Systems*, 2022.

[21] T. Guo, S. W. Feng, and J. Yu, “Polynomial Time Near-Time-Optimal Multi-Robot Path Planning in Three Dimensions with Applications to Large-Scale UAV Coordination,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[22] S. D. Han, E. J. Rodriguez, and J. Yu, “Sear: A polynomial-time multi-robot path planning algorithm with expected constant-factor optimality guarantee,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[23] J. Yu, “Constant factor time optimal multi-robot routing on high-dimensional grids,” *2018 Robotics: Science and Systems*, 2018.

[24] H. Ma and S. Koenig, “Optimal target assignment and path finding for teams of agents,” in *AAMAS*, 2016.

[25] H. Ma, T. S. Kumar, and S. Koenig, “Multi-agent path finding with delay probabilities,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[26] M. Szegedy and J. Yu, “On rearrangement of items stored in stacks,” in *The 14th International Workshop on the Algorithmic Foundations of Robotics*, 2020.