

自動化された衣服の効率的な物理的・アルゴリズム的設計に向けて

Teng Guo

Jingjin Yu

概要- 大都市圏への駐車は、都市の造園に影響を与える交通パターンに対して、さらに示唆を与える、時間のかかる作業であることが多い。駐車場に必要なプレミアムスペースを削減することで、自動機械式駐車場システムの開発につながった。自動化されたガレージは、各島に1~2列の車両を持つ通常のガレージと比較して、複数列の車両を積み重ねることで、より高い駐車需要をサポートすることができる。この複数列のレイアウトは駐車スペースを縮小するが、駐車と検索がより複雑になる。本研究では、100%近い駐車密度をサポートする自動ガレージ設計を提案する。駐車場の駐車と複数車両の回収の問題をマルチロボット経路計画問題の特殊なクラスとしてモデル化し、自動化されたガレージのすべての一般的な操作を扱うための関連アルゴリズムを提案する。(1)駐車/回収を同時に行う実現可能で効率的な解を求める最適アルゴリズムと最適に近い方法、(2)ラッシュアワーでのスケジュール検索を容易にするために車両を再配置する新しいシャッフル機構、などである。提案手法が大規模かつ高密度の実世界駐車アプリケーションに有望であることを示す徹底的なシミュレーション研究を実施する。

I. INTRODUCTION

自動駐車システム(ガレージ)の発明は、都心部や人口の多い地域など、スペースに大きなプレミアムがある地域の駐車問題を解決するのに役立つ。現在、駐車場の面積はますます不足し、高価になってきている。マンハッタンのスポットは、簡単に200,000米ドルを超える可能性がある。高密度駐車場をサポートし、スペースを節約し、より便利なガレージを開発することは、経済的/効率的な理由から非常に魅力的である。

自動化されたガレージでは、人間のドライバーは、駐車プロセスを気にすることなく、特定のI/O(入出力)ポートで車両を落下(ピックアップ)させるだけでよい。このようなシステムでは、ドアを開けるために周囲のスペースを必要としないため、駐車がより近くなる可能性がある。駐車場や港に車両を移動させることは、このようなシステムにとって重要な機能である。解決策の一つは、ロボットのバレットを使って車両を移動させることである。このようなシステムはすでに市販されており、例えば香港のHKSTP [1]がある。このようなシステム[2]では、車両は互いにブロックし合うことができるように駐車されており、特定の車両を検索するために複数の再配置が必要である。残念ながら、これらのシステムがどの程度機能しているか、例えば駐車/回収効率に関する情報はほとんどない。その他のソリューションは、自動運転車のための駐車に焦点を当てている。このような自動化されたガレージでは、車両は駐車枠や港まで自らを運転することができる。となり

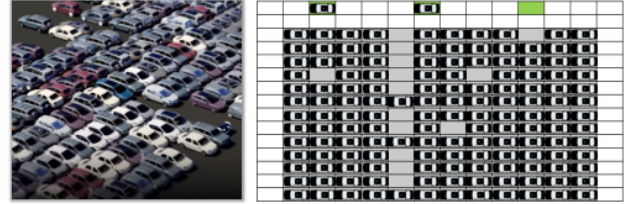


図1. 左:想定される自動ガレージシステムの密度レベルの説明図。右: グリッドベースの抽象化で、3つのI/Oポートで車両をドロップオフし、検索を行う。柔軟性はあるが、限られた省スペースしか提供しない

the system more flexible but provides limited space-saving advantages, besides requiring autonomy from the vehicles.

近年、効率的なマルチロボット経路計画アルゴリズムが数多く提案され、複数のロボットバレットを用いた駐車・回収コストの低減が可能となっている。本研究では、マルチロボットに基づく駐車・検索問題を研究し、完全な自動ガレージ設計を提案し、ほぼ100%の駐車密度をサポートし、ガレージを効率的に動作させるための関連アルゴリズムを開発する。

結果と貢献。我々の研究の主な結果と貢献は以下の通りである。自動ガレージの設計において、このようなガレージが必要とする主要な操作をモデル化するバッチ式車両駐車・検索(BVPR)と連続式車両駐車・検索(CVPR)の問題を導入し、自動ガレージシステムの将来の理論的・アルゴリズム的研究を促進する。

アルゴリズム面では、 $m_1 \times m_2$ の格子地図上で $(m_1 - 2)(m_2 - 2)/m_1 m_2$ という高い駐車密度に対応し、大型ガレージの駐車密度が100%に近づく複数車両の駐車・回収が可能なシステムを研究する。グリッドのようなシステムの規則性を利用して、最適なILPベースの手法と、逐次計画に基づく高速な準最適アルゴリズムを提案する。我々の準最適アルゴリズムは、良好な解の質を維持しながら、非常にスケーラブルであり、大規模なアプリケーションに適している。さらに、検索順序が予想できる場合、ラッシュアワーに高速な車両検索を行うために、オフピーク時に車両を並べ替えるシャッフル機構を導入する。我々の再配置アルゴリズムは、全ガレージ密度付近で $O(m_1 m_2)$ の総時間コストでこのようなシャッフルを実行する。

関連研究。研究者は、効率的な高密度駐車ソリューションに向けた多様なアプローチを提案している。自動運転車のための多くのシステムが研究されており[3]-[5]、車両は中央制御装置を使って駐車され、複数の列に積み重ねられ、互いにブロックすることができる。これらの設計は、駐車容量を最大50%増加させる。

G. Teng, J. Yuは、米国ニュージャージー州ピスカタウェイのニュージャージー州立大学ラトガース校コンピュータサイエンス学部に所属している。電子メール {teng.guo, jingjin.yu}@rutgers.edu。この研究は、NSF賞IIS-1845888とAmazon Research Awardの一部支援を受けている。

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 toward 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 in 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 the parking and retrieval more complicated. In this work, we propose an automated garage design that supports near 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

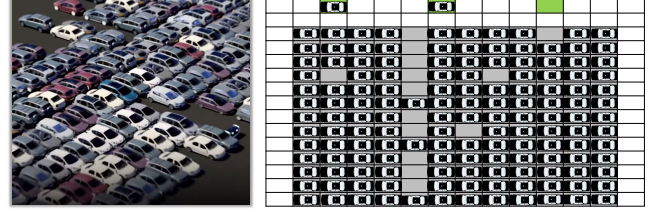


Fig. 1. Left: an illustration of the density level of the envisioned automated garage system. Right: The grid-based abstraction with three I/O ports for vehicle dropoff and retrieval.

the system more flexible but provides limited space-saving advantages, besides requiring autonomy from the vehicles.

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.

しかし、検査は閉塞により非常に複雑になり、自動運転車の操縦性に大きく影響される。

ほとんどの車両が自動運転できないため、ロボット・バレットベースの高密度駐車システムの方がより適切な選択となりうる。パズルベースドストレージ(PBS)システムまたはグリッドベースシャトルシステムは、もともと[6]によって提案されたもので、最も有望な高密度ストレージシステムの一つである。このようなシステムでは、AGVやシャトルなどの蓄電ユニットは4つの枢軸方向に可動である。空のセル(エスコート)が少なくとも1つなければならない。車両を回収するためには、エスコートを利用して、希望する車両をI/Oポートに移動させなければならない。これは、NP困難であることが知られている15パズルに似ています[7]。単一のエスコートと複数のエスコートを持つ1台の車両を検索するための最適なアルゴリズムが[6], [8]で提案されている。しかし、これらの方法は一度に1台の車両を検索することしか考慮していない。また、平均検索時間は従来の通路ベースのソリューションよりもはるかに長くなる可能性がある。容量需要と検索効率のトレードオフを達成するために、最近の先進的なマルチロボット経路計画(MRPP)アルゴリズム[9]を利用することで、複数の車両の検索と駐車を同時に可能にする、より多くのエスコートとI/Oポートを使用することを提案する。

MRPPは広く研究されている。静的またはワンショット設定[10]では、グラフ環境と、各ロボットがユニークな開始位置とゴール位置を持つ多数のロボットが与えられたとき、タスクは、開始からゴールまでの全てのロボットの衝突のない経路を見つけることである。ワンショットMRPPをメイクスパンかコスト和のどちらかを最小化するという観点から最適に解くことはNP困難であることが証明されている[11], [12]。MRPPのソルバーは最適と準最適に分類できる。最適ソルバーは、MRPPをILP [13]、SAT [14]、ASP [15]などのよく研究された問題に還元するか、探索アルゴリズムを用いて最適解を見つけるために結合空間を探索する [16], [17]。NPhardnessのため、最適ソルバーは大きな問題を解くのに適していない。境界付き準最適ソルバー[18]は、強い最適性保証を持ちながら、より良いスケラビリティを達成する。しかし、特に高密度環境では、まだスケールが小さい。大規模MRPPを解くための多項式時間アルゴリズム[19]があるが、これは解の質を犠牲にしている。他の0(1)時間最適多項式時間アルゴリズム[20]–[23]は、主にメイクスパンの最小化に焦点を当てており、連続的な設定にはあまり適していない。

組織である。本稿の残りの部分は以下のように構成されている。第II章では、ガレージ設計を含む前置きを説明する。第III–第IV章では、自動化されたガレージを操作するためのアルゴリズムを示す。第V章では、ガレージシステムの徹底的な評価と考察を行い、第VI章で結論を述べる。

II. PRELIMINARIES

A. ガレージ設計仕様

本研究では、自動格子ベースのガレージは、4連結 $m_1 \times m_2$ 格子 $G(V, E)$ である(図1参照)。グリッドの最上縁には、駐車車両や特定の駐車車両を検索するための n_o 個のI/Oポート(ここでは単にポートと呼ぶ)が分散している。ポートは、検索または駐車のいずれかにのみ使用できる。

車両は駐車場(下部中央 $(m_1 - 2) \times (m_2 - 2)$ サブグリッドのセル)に駐車しなければならない。空き地が駐車したら、それは移動可能な障害物となる。 $O = \{o_1, \dots, o_{n_o}\}$ は港の集合、 $P = \{p_1, \dots, p_{l_p}\}$ は駐車場の集合である。

B. バッチ車両駐車・検索(BVPR)

バッチ式車両駐車・回収(BVPR)は、バッチ式駐車・回収の最適化を目指す。1バッチで、駐車する車両は n_p 台、回収する車両は n_r 台、駐車する車両は n_l 台である。 $C = C_p \cup C_r \cup C_l$ を全車両の集合とする。すなわち、 $|C| < |P|$ である。時間はタイムステップに離散化され、AGV/シャトルが運ぶ複数の車両が同時に移動することができる。各タイムステップにおいて、各車両は現在の位置で左、右、上、下、または待機することができる。車両間の衝突は避けるべきである:

- 1) Meet collision. Two vehicles cannot be at the same grid point at any timestep: $\forall i, j \in C, v_i(t) \neq v_j(t)$;
- 2) Head-on collision. Two vehicles cannot swap locations by traversing the same edge in the opposite direction: $\forall i$ によって、 $(v_i(t), \Delta v_i(t))$ (their + (see 1) 反対側の else;
- 3) P移動 = $\text{argmax}_{j \in C} (v_j(t))$ のとき、次の=同じv reacher j (t + edg vehicle collisions 1)に従う。2). 方向: 1 = 偽; veare 垂直。ions at タイムステップtにおける車両iの移動方向ベクトルを $\hat{e}_i(t)$ the $m(t) = v_i(t+1) - v_i(t)$ とすると、 $\forall i, j \in C, (v_i(t+1) - v_j(t)) \wedge \hat{e}_i(t) \neq \hat{e}_j(t) = \text{false}$.

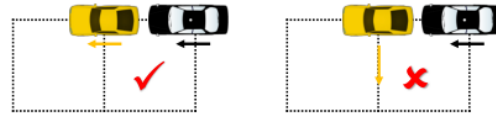


図2. 同じ方向への車両の平行移動(左)は許容されるが、車両の垂直追従(右)は禁止される。
ある種のMRPP定式化[10]とは異なり、垂直な追従衝突を考慮する必要があり、問題を解くのが難しくなる。タスクは、各車両の無衝突経路を見つけ、現在位置から所望の位置に移動させることである。具体的には、車両が回収されるためには、その目標が指定された港である。他の車両については、所望の位置は駐車場のいずれか1つである。解決策の質を評価するために、以下の基準が用いられる:

- 1) 平均駐車/検索時間(APRT): 車両の回収または駐車に必要な平均時間。
- 2) Makespan (MKPN): すべての車両を希望する位置に移動させるのに必要な時間。
- 3) Average number of moves per task (ANM): the sum of the distance of all vehicles divided by the total number of vehicles in $C_p \cup C_r$.

一般に、これらの目的はパレートフロント[12]を作り出し、これら2つを同時に最適化することは必ずしも不可能である。

C. 連続車両駐車・検索(CVPR)

CVPR は、駐車場と検索問題の連続バージョンである。BVPRの構造の大部分を継承しているが、いくつかの重要な違いがある。

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. $\mathcal{O} = \{o_1, \dots, o_{n_o}\}$ is the set of ports and $\mathcal{P} = \{p_1, \dots, p_{|\mathcal{P}|}\}$ 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: $\forall i, j \in \mathcal{C}, (v_i(t) = v_j(t+1) \wedge v_i(t+1) = v_j(t)) = \text{false}$;
- 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 $\forall i, j \in \mathcal{C}, (v_i(t+1) = v_j(t) \wedge \hat{e}_i(t) \perp \hat{e}_j(t)) = \text{false}$.

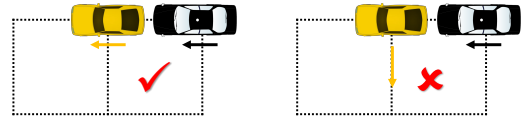


Fig. 2. While parallel movement of vehicles in the same direction (left) is allowed, *perpendicular following* of vehicles (right) is forbidden.

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,

この定式化では、以下の仮定を置く。車両 $i \in C_r$ が所望の港に到着すると、環境から除去される。港に(容量内で)駐車する必要がある新しい車両が出現し、車両の回収に新たな要求が発生する。また、ある港が車両の回収に使用されている場合、他のユーザは回収タスクが終了するまで、その港に車両を駐車することができない。時間地平が無大または固定である場合を除き、解の質を評価するために3つの基準を使用することができる。

D. 車両シャッフル問題(VSP)

実際のガレージでは、システムが容量に達するオフピーク時(朝のラッシュアワーなど)にいることが多く、回収要求も少ない。後の時間帯(例えば、午後のラッシュアワー)における車両の検索順序がわかっている場合、その情報を利用して車両を再シャッフルし、後での検索を容易にすることができる。この問題はワンショットMRPPとして定式化される。開始構成 X_I とゴール構成 X_G が与えられたとき、再構成を達成するために衝突のない経路を見つける必要がある。ゴールの構成は、車両の検索時間順序に従って決定される。先に検索されると予想される車両は、後で出発する他の車両に遮られないように、港の近くに駐車されるべきである。

III. SOLVING BVPR

A. 整数線形計画法(ILP)

[13]、[24]のネットワークフローに基づくアイデアを基に、BVPRを多商品マックスフロー問題に還元し、整数計画法を用いて解く。 C_r の車両は特定のゴール位置(港)を持っており、異なる商品として扱われなければならない。一方、 $C_p \triangle C_l$ の車両は駐車枠のどれかに駐車できるので、1つの商品と見なすことができる。インスタンスが T ステップで解けると仮定して、図3に示すような T ステップの時間拡張ネットワークを構築する。

T ステップの時間拡張ネットワークは、有向ネットワーク $N_T = (V_T, E_T)$ で、有向、単位容量のエッジを持つ。ネットワーク N_T は、元のグラフの頂点 V の $T + 1$ 個のコピーを含む。タイムステップ t における頂点 $u \in V$ のコピーを u_t とする。タイムステップ t において、 $(u, v) \in E$ または $v = u$ の場合、エッジ (u_t, v_{t+1}) が E_T に追加される。 $i \in C_r$ に対して、頂点 s_{i0} (s_i は i の開始頂点)に商品タイプ i の1ユニットの供給を与えることができる。車両 i がポートに到着することを保証するために、 T におけるゴール頂点 g_i とソースノード s_{i0} を結ぶフィードバックエッジを追加する。駐車が必要な車両については、補助ソースノード α と補助シンクノード β を作成する。各 $i \in C_p \triangle C_l$ に対して、ノード α とその開始ノード s_{i0} を結ぶ単位容量の辺を追加する。車両は駐車スロットのどれかに駐車できるので、任意の頂点 $u \in P$ に対して、 u_T と β を結ぶ容量が1の辺を追加する。 $C_p \sim C_l$ の車両に対する $n_p + n_l$ 単位の商品の供給は、ノード α で与えることができる。

ILPを用いて多商品マックスフローを解くために、バイナリ変数 $X = \{x_{iuvt}\}$, $i = 0, \dots, n$

$\{r\}$, $(u, v) \in E$ または $u = v$, $0 \leq t \leq T$; 変数を真に設定すると、対応する辺が最終解に使われることを意味する。ILPの定式化は以下のように与えられる。

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

$$\text{subject to} \quad \forall t, v, i \quad \sum_u x_{iuv(t-1)} = \sum_w x_{iuvw} \quad (2)$$

$$\forall t, i, v \quad \sum_u x_{iuvt} \leq 1 \quad (3)$$

$$\forall t, i, (u, v) \in E \quad \sum_i (x_{iuvt} + x_{ivut}) \leq 1 \quad (4)$$

$$\forall t, i, (u, v) \perp (v, w) \quad \sum_i (x_{iuvt} + x_{ivwt}) \leq 1 \quad (5)$$

$$\sum_{i=0}^{n_r-1} x_{ig_i s_i T} + \sum_{u \in P} x_{n_r u \beta T} = n_p + n_r + n_l \quad (6)$$

$$x_{iuvt} = \begin{cases} 0 & i \text{ が } t \text{ で 辺 } (u, v) \text{ を 横切らない場合} \\ 1 & \text{if } i \text{ traverses edge } (u, v) \text{ at } t \end{cases} \quad (7)$$

式(1)において、時間地平 T 内の全車両の移動回数を最小化する。式(2)は、各格子点における流れの保存制約を指定する。式(3)は、meet-collisionsを避けるための頂点制約を指定する。式(4)において、車両は同じエッジを反対方向に通過することは許されない。式(5)は、以下のコンフリクトを垂直に禁止する制約を指定する。 T ステップの時間拡張ネットワークのプログラミングが実行可能であれば、解を求めることができる。そうでなければ、実行可能な解が得られるまで T を段階的に増加させる。整数計画法が解を持つ最小の T は、最小のメイクスパンである。その結果、ILPは二次目的として、総手数を最小化するメイクスパン最適解を求める。

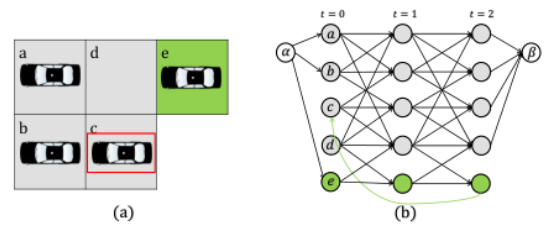


図3 (a) 4つの駐車場と1つの港を持つBVPRインスタンスの例。赤い長方形内の車両は回収する必要があり、他の車両は灰色の領域に駐車する必要がある。(b) BVPRインスタンスから削減された2ステップフローネットワーク。

B. 高密度プランニングのための効率的なヒューリスティック

ILPはメイクスパン最適解を見つけることができるが、スケールが小さい。我々は、解の質の小さなコストで密なBVPRインスタンスを素早く解くことができる高速なアルゴリズムを求めている。我々が提案するアルゴリズムは、検索と駐車の一車両運動プリミティブに基づいて構築されている。

1) 単一車両の駐車/回収のためのモーションプリミティブ: 解の質に関係なく、BVPRは $C_r \triangle C_p$ の各車両を逐次計画することで解くことができる。具体的には、各ラウンドにおいて、与えられた車両 $i \in C_r \triangle C_p$ に対して、 i をその港に移動させるか、

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-expanded network as shown in Fig. 3.

A T -step time-expanded 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 $X = \{x_{iuvt}\}, i =$

$0, \dots, n_r, (u, v) \in \mathcal{E}$ or $u = v, 0 \leq t \leq T$; a variable set to true means that the corresponding edge is used in the final solution. The ILP formulation is given as follows.

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

$$\text{subject to} \quad \forall t, v, i \quad \sum_u x_{iuv(t-1)} = \sum_w x_{iuvt} \quad (2)$$

$$\forall t, i, v \quad \sum_v x_{iuvt} \leq 1 \quad (3)$$

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

$$\forall t, i, (u, v) \perp (v, w) \quad \sum_i (x_{iuvt} + x_{ivwt}) \leq 1 \quad (5)$$

$$\sum_{i=0}^{n_r-1} x_{ig_i s_i T} + \sum_{u \in \mathcal{P}} x_{n_r, u \beta T} = n_p + n_r + n_l \quad (6)$$

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

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.

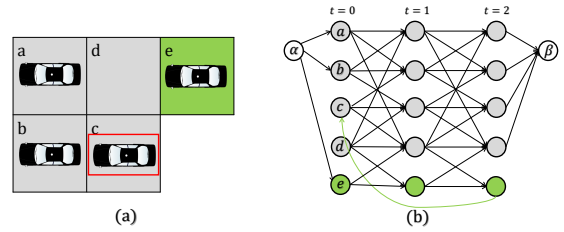


Fig. 3. (a) A illustrative BVPR instance with 4 parking spots and one port. The vehicles within the red rectangle need to be retrieved while other vehicles should be parked in the gray areas. (b) The 2-step flow network reduced from the BVPR instance.

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

駐車スポットのいずれかに移動する1つのタスクを完了することのみを考える。車両*i*が目的地に到着した後、次のタスクが解決される。図4と図5の例では、最大容量に達しているが、この手法の動作を示している。

図4の検索シナリオでは、赤い四角で示された車両が検索される。左の破線で示した直感的な経路を実現するためには、経路を遮断する車両を途中で排除する必要がある、そのためには、遮断する車両を2つの空の列を利用して左または右に1ステップ移動させることで容易に実現できる。このようなモーションプリミティブは、常にデッドロックのない車両の取得に成功する。

図5の駐車シナリオでは、車両をグリーンポートに駐車する必要がある。まず、空点(エスコート)を貪欲に探索することで、これを行う。車両の平行移動のメカニズムを用いると、まず1タイムステップで駐車車両の柱に移動させ、次に1タイムステップで車両の直下の位置に移動させることができる。その後、車両は目的地であるエスコートに直接移動することができる。

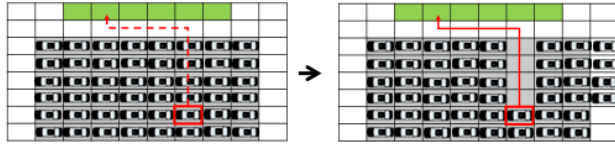


図4. 車両を検索するためのモーションプリミティブ。

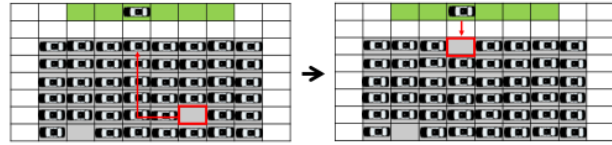


図5. 車両駐車のためのモーションプリミティブ。

逐次計画では、解があれば常に戻るが、複数の駐車場や検索要求を実行する場合、逐次解はMKPNやARPTで測定される性能が低下する。

2) MCPによる単一動作プリミティブの結合(CSMP): 最小通信政策(MCP)[25]は、影響を受けていないロボットを止めることなく、予期せぬ遅延を処理するためのロバスタなマルチロボット実行政策である。実行中、MCPは元の計画と同様にロボットが各頂点を訪問する順序を保持する。ロボット*i*が移動動作を行い、頂点*v*を入力しようとしているとき、MCPはロボット*i*が次にその頂点に入るかどうかを元の計画によってチェックする。次に別のロボット*j*が*v*に入るように計画されている場合、*i*は*j*が*v*を離れるまで現在の頂点で待機する。

MCPを用いて、逐次計画によって発見された計画に並行性を導入する。アルゴリズムはAlg. 1とAlg. 2に記述されている。1はCSMPのフレームワークを記述している。まず、 $C_r \sim C_p$ の全ての車両の初期計画を一つずつ求める(4-6行目)。パスを取得した後、すべての待ち状態を削除し、各頂点の車両訪問順序をキューのリストに記録する(7行目)。次に、すべての車両がタスクを終了し、目的地に到達するまで、MCPを使用して計画を実行するループを入力する(8-15行目)。Alg. 2において、*i*が元の順序に従って頂点*v_i*に入る次の車両である場合、現在*v_i*にいる車両があるかどうかをチェックする。

もしそうでなければ、*i*を*v_i*に入れる。現在、別の車両*j*が*v_i*を占有している場合、次のステップで*j*が次の頂点*v_j*に移動しているかどうかを、再帰的に関数MCPMoveと呼ぶことで調べる。次のステップで*j*が*v_j*に移動し、*i*, *j*の移動方向が垂直でない場合、車両*i*を頂点*v_i*に入力する。そうでなければ、*i*は*u_i*で待つべきである。このアルゴリズムは構成上デッドロックフリーであるが、ページ数の関係上、比較的簡単な証明は省略する。

命題III.1. CSMPはデッドロックフリーであり、有限時間で常に実行可能解が存在する場合、その解を求める。

Algorithm 1: CSMP

```

1 Function CSMP ():
2   foreach  $v \in \mathcal{V}$ ,  $VOrder[v] \leftarrow Queue()$ 
3    $InitialPlans \leftarrow \{\}$ 
4   for  $i \in C_p \cup C_r$  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
9        $(u_i, v_i) \not\subset (u_j, v_j)$ ) then
10      move  $i$  to  $v_i$ 
11       $VOrder[v_i].popfront()$ 
12       $mcpMoved[i]=true$ 
13      return true
14   let  $i$  wait at  $u_i$ 
15    $mcpMoved[i]=false$ 
16   return false

```

3) 優先順位付け: 逐次計画では、計画順序に関係なく、常に解を見つけることができる。しかし、優先順位はソリューションの質に影響を与える。ランダムな優先順位(Alg. 1 4-6行目)で計画を立てる代わりに、単車駐車は2ステップしかかからないので、可能な限りまず駐車計画を立てることができる。 C_p 内の全ての車両が駐車された後、SingleMPを適用して車両を検索する。 C_r 内の車両のうち、まず、より早く目標に到達し、下段の車両をブロックしないように、港に近い車両にSingleMPを適用する。

C. 複雑さ解析

本節では、CSMPの時間複雑性と解のメイクスパン上限を解析する。SingleMPでは、1台の車両を駐車/回収するために、 n_p 台の車両が閉塞の原因となり、移動の必要性があると仮定する。 $n = n_p + n_r + n_l$ したがって、 $C_r \Delta C_p$ の全車両に対してSingleMPを用いて経路を計算する複雑さは、 $(n_p + n_r)n$ で境界される。

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.

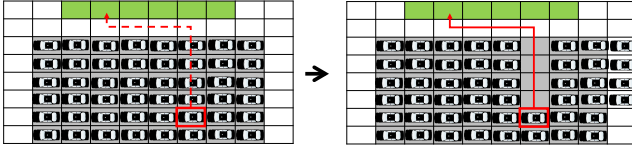


Fig. 4. The motion primitive for retrieving a vehicle.

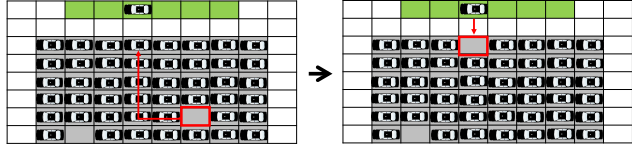


Fig. 5. The motion primitive for parking a vehicle.

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
9        $(u_i, v_i) \not\perp (u_j, v_j))$  then
10        $move\ i\ to\ v_i$ 
11        $VOrder[v_i].popfront()$ 
12        $mcpMoved[i] = true$ 
13       return  $true$ 
14    $let\ i\ wait\ at\ u_i$ 
15    $mcpMoved[i] = false$ 
16   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

SingleMPによって計算された各単一車両経路の経路長は、 $m_1 + m_2$ 以下である。すべての単一車両経路を連結して得られる経路のメイクスパンは、 $n_r(m_1 + m_2) + 2n_p$ で境界される。これは、MCPが $n_r(m_1 + m_2) + 2n_p$ 回以上の反復を要しないことを意味する。したがって、解のメイクスパンは $n_r(m_1 + m_2) + 2n_p$ で上界される。MCPの各ループにおいて、基本的に $n = n_p + n_r + n_l$ 個のノードを持つグラフ上でDFSを実行し、全てのノードを横断する。したがって、CSMPの時間複雑度は $O(n(n_r m_1 + n_r m_2 + 2n_p))$ となる。まとめると、CSMPの時間複雑度は $O(n(n_r m_1 + n_r m_2 + 2n_p))$ で境界があり、メイクスパンは $n_r(m_1 + m_2) + 2n_p$ で上限がある。

D. CSMPのCVPRへの拡張

CSMPはCVPRを解くのに容易に適応できる。BVPR版と同様に、各タイムステップで各車両に対してMCPMoveを呼び出す。新しいリクエストがあるタイムステップに来到、SingleMPを使用して関連する車両のパスを計算し、頂点の訪問順序の情報を更新する。すなわち、車両 $i \in C_p \Delta C_r$ にSingleMPを適用するとき、車両 j がタイムステップ t で頂点 u を訪問する場合、 i をキュー $V_Order[u]$ にプッシュする。このようにして、MCPは訪問順序を維持したまま計画を実行する。以前に計画された車両は、新しい要求の影響を受けず、当初の計画を実行し続ける。この方法の主な欠点は、訪問順序が固定されているため、通常、再計画よりも解の質が悪いことである。

IV. ライブ・ケーブルによる対照解答

VSPは本質的に静的/ワンショットMRPPを解いている。 $m_1 \times m_2$ グリッド上では、 $2(m_2 - 2)$ 列シャッフルと $(m_1 - 2)$ 行シャッフルを超えないルービックテーブルアルゴリズム[26]を適用することで解くことができる。図6に示す例として、近傍の2つの列を使って、与えられた列の車両をかなり効率的にシャッフルすることができ、 $O(m_1)$ ステップしか必要としない[20]。行シャッフルも同様である。駐車車両の数に応じて、1つ以上の複数の行/列シャッフルを同時に実行することができる。となる(スペースが限られているため、ストレートフォワード証明は省略)。

命題IV.1. ガレージ容量がフルの場合、 $O(m_1 m_2)$ のメイクスパンと、ガレージに $\Theta(m_1 m_2)$ の空きスポットと $\Omega(m_1 m_2)$ の駐車車両がある場合の $O(m_1 + m_2)$ のメイクスパンを用いてVSPを解くことができる。一方、 $\Omega(m_1 m_2)$ の駐車車両では、VSPを解くために必要なメイクスパンは $\Omega(m_1 + m_2)$ である。

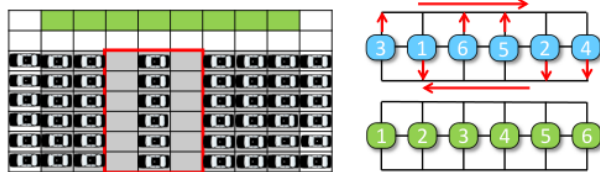


図6. 1行/1列を「シャッフル」する仕組みの説明図。

V. EVALUATION

本節では、提案アルゴリズムを評価する。すべての実験はIntel® Core™ i7-9700 CPU、3.0GHzで行った。各データポイントは、特に断りのない限り、ランダムに生成されたインスタンスに対する20回の実行の平均である。ILPはC++で実装され、その他のアルゴリズムはCPythonで実装されている。シミュレーションの動画は <https://youtu.be/XPp0B5f7CzA> にある。

A. BVPRにおけるアルゴリズム性能

グリッドサイズを変化させる。最初の実験では、グリッドの辺の長さを変化させた $m \times m$ グリッドで、最も密なシナリオの下で提案アルゴリズムを評価する: システムには $(m - 2)^2$ 台の車両があり、すべてのポートは駐車または検索に使用される ($n_p + n_r = n_o$)。その結果は図7にある。CONCATは、単純に単車経路を連結する手法である。rCSMPでは、優先順位がランダムな車両に対してSingleMPを適用し、pCSMPでは、優先順位付け戦略を適用する。

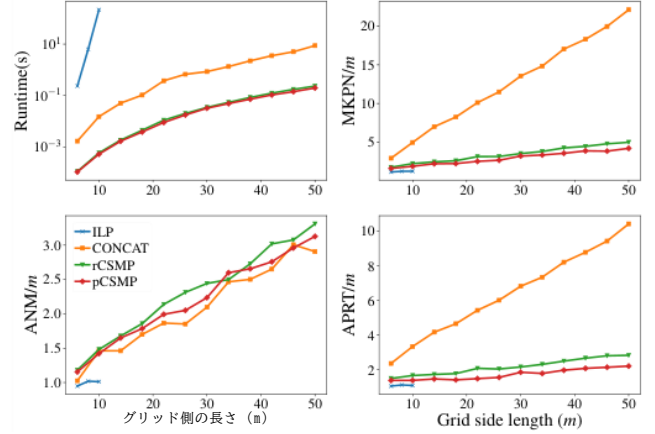


図7. 提案手法の $m \times m$ グリッドにおけるランタイム、MKPN、APRT、AVNデータ。

手法の中で、ILPはMKPN、ANM、APRTの点で最も優れた解の質を持っており、その最適性が保証されていることから予想される。しかし、ILPは $m \leq 10$ 、 $n \leq 64$ の極限に達し、スケーラビリティが最も低い。CONCAT、rCSMP、pCSMPはよりスケーラブルで、 50×50 グリッド、2304台のインスタンスを数秒で解くことができる。CONCATは単車経路を連結しているだけなので、rCSMPやpCSMPに比べて非常に長い経路になる。MCP手順は同時実行性を大幅に改善し、より優れた解の質につながるが観察される。CONCATで得られた経路のMKPNとAPRTは10m-20m、rCSMPとpCSMPで得られた経路のMKPNとAPRTは2m-4mである。優先順位付け戦略を用いたpCSMPのMKPNとAPRTは、rCSMPのそれよりも約20%低い。

車両密度の影響。2つ目の実験では、グリッドサイズを 20×20 に固定し、車両密度の変化に対するアルゴリズムの挙動を調べる。なお、 $n_p + n_r = n_o$ とする。その結果を図8に示す。前回と同様に、ILPは密度が20%以下のインスタンスを合理的な時間でしか解くことができないが、

complexity of computing the paths using `SingleMP` for all vehicles in $C_r \cup C_p$ 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 $i \in C_p \cup C_r$, 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 $\Omega(m_1 + m_2)$.*

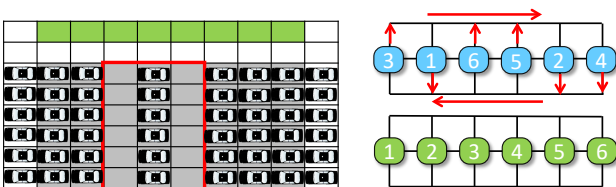


Fig. 6. Illustration of the mechanism for “shuffling” a single row/column.

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/XPpOB5f7CzA>.

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.

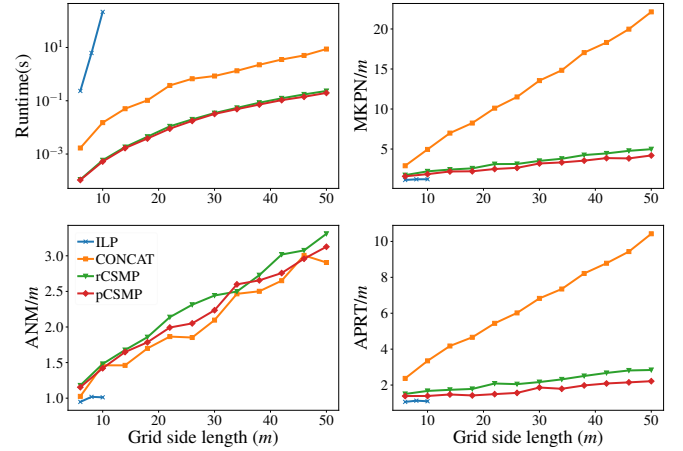


Fig. 7. Runtime, MKPN, APRT, AVN data of the proposed methods on $m \times m$ grids under the densest scenarios.

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 density below 20% in

他の3つのアルゴリズムは全て最も密度の高いシナリオに取り組むことができる。すべてのアルゴリズムにおいて、 C_r の車両密度はMKPNとAPRTに限定的な影響しか与えず、 r CSMPとpCSMPはCONCATよりもはるかに品質が良い。低密度シナリオでは、移動に必要な車両が少なく済むため、車両の回収/駐車に支障をきたす可能性がある。その結果、車両密度が高くなるにつれてANMが増加する。

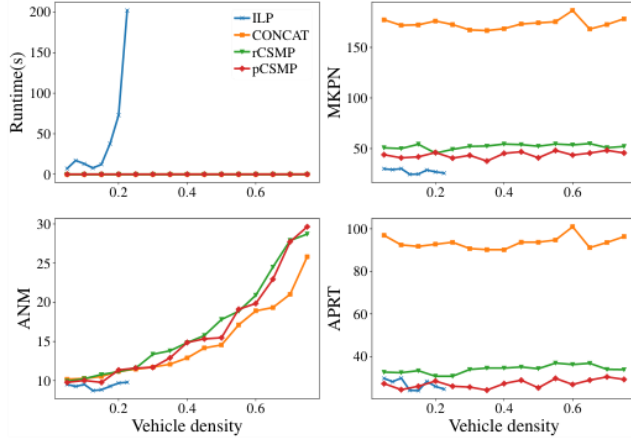


図8. 提案手法の20×20グリッド、車両密度を変化させた場合の実行時間、MKPN、APRT、ANMデータ。

B. CVPRにおけるアルゴリズム性能

ランダム検索とパーキング。連続CSMPを10ポートの12×12グリッドでテストする。各時間ステップにおいて、ポートが利用可能であれば、容量を超えなければ確率 p_p でこのポートに駐車する必要がある新車両が存在することになる。そして、確率 p_r で、このポートは、ランダムな駐車車両がある場合、その車両を検索するために使用される。以下の3つのシナリオをシミュレートする：

(i). 朝のラッシュアワー。当初は駐車していない。 $p_p = 0.6$, $p_r = 0.01$ 。

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

(iii). 夕方のラッシュアワー。最初はガレージは満杯。検索要求が駐車場を支配する: $p_p = 0.01$, $p_r = 0.6$. タイムステップの最大数は500とした。これらのシナリオの下で、平均検索時間、平均駐車時間、総移動回数を評価する。その結果を図9に示す。オンラインCSMPは、検索回数が少ないため、朝最高の性能を達成している。一方、3つのシナリオの平均検索時間は2m以下、駐車時間はm以下である。これは、最も密なシナリオとラッシュアワーでも、アルゴリズムが良好な解の質でパスを計画できることを示している。

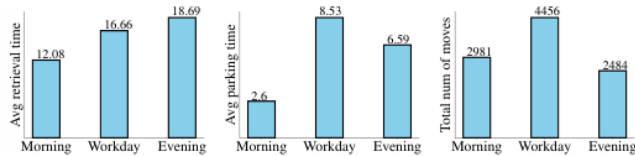


図9. 3つのガレージトラフィックパターンにおけるCVPR性能統計。

シャッフリングの利点。本実験では、シャッフリングの効果(VSPを解くため)を検証する。各車両には、ある人が他の人より早く帰宅する夕方のラッシュアワーに期待される検索優先順序が割り当てられていると仮定する。検索を容易にするために、車両に対して列シャッフル操作を行う。図10(a)~(c)に、最も密な設定における、グリッドサイズの異なる $m \times m$ グリッドに対する列シャッフル演算のメイクスパン、平均移動回数、計算時間を示す。シャッフルの経路は1秒以内に計算できるが、シャッフル完了のメイクスパンは m^2 に対して線形にスケールし、平均移動回数は m に対して線形にスケールする。2500 120

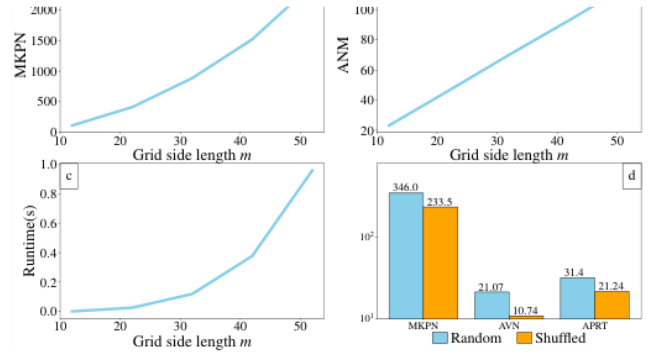


図10. グリッドサイズを変化させた $m \times m$ グリッドに対して列シャッフル操作を行った統計量。

シャッフル後、 $p_r = 1$, $p_p = 0$ の連続CSMPを適用して全車両を取得し、シャッフルを行わない場合と比較する。結果のメイクスパン、1台あたりの平均移動回数、1台あたりの平均検索時間を図10(d)に示す。シャッフルなし構成と比較して、CSMPは30%~50%少ない移動回数と検索時間ですべての車両を検索することができ(すべてのデータに適合させるために対数スケールが使用されていることに注意)、ラッシュアワー検索を予期して車両を再配置することが大きな利点を提供することを示している。

VI. 結論と考察

本研究では、大都市圏における高速/高効率での車両の高密度駐車を可能にすることを目的とした、自動化されたガレージシステムの完全な物理的かつアルゴリズム的な設計を紹介する。我々は、検索と駐車問題をマルチロボットの経路計画問題としてモデル化し、我々のシステムがほぼ100%の車両密度をサポートすることを可能にする。提案するILPアルゴリズムはメイクスパン最適解を提供できるが、CSMPアルゴリズムは高いスケーラビリティを持ち、解の品質も良い。また、後で順序付けされた検索操作のために、非ラッシュ時間中に車両再配置を実行することは非常に有益であることが明確に示されており、これは我々の自動化されたガレージ設計のユニークな高実用性の特徴である。

今後の課題として、MRPP/MAPF研究の最新の進歩を活用し、CSMPのスケーラビリティと柔軟性をさらに向上させる予定である。また、自動化されたガレージの設計を2Dから3Dに拡張し、複数のレベルの駐車をサポートする予定である。最後に、物理的・アルゴリズム的設計を実現する小規模なテストベッドを構築したい。

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.

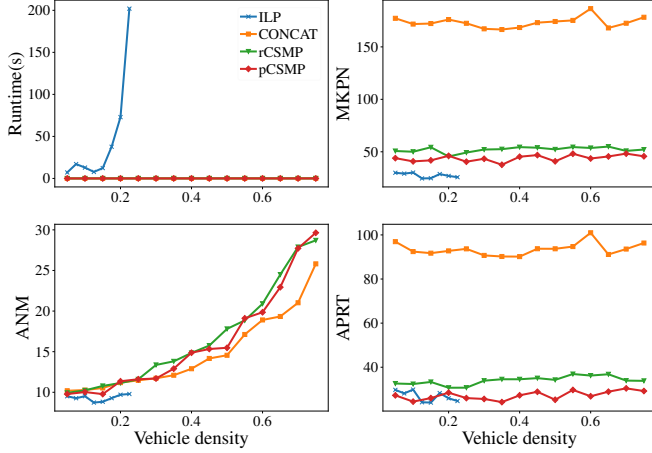


Fig. 8. Runtime, MKPN, APRT, ANM data of the proposed methods on 20×20 grids with varying vehicle density.

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.

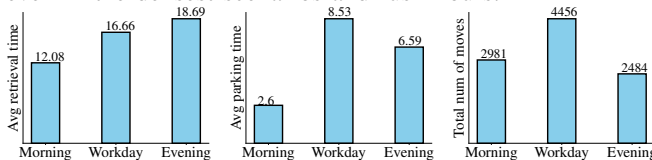


Fig. 9. CVPR performance statistics under three garage traffic patterns.

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 .

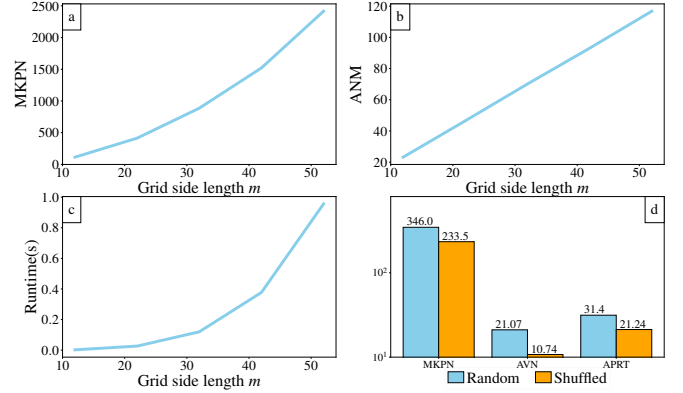


Fig. 10. Statistics of performing the column shuffle operations on $m \times m$ grids with varying grid size.

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.

VII. APPENDIX

命題IV.1の証明。全密度に対して、Rubik Tableアルゴリズムを適用することができる。列をシャッフルするには、2つの空列を利用することでハイウェイモーションプリミティブを適用し、1つの空列を利用することでラインマージソートモーションプリミティブを適用することができます[20]。各列シャッフルには $0(m_1)$ ステップが必要であり、そのような列は m_2 個あり、全ての列は $1/2$ 個の空列を用いて逐次シャッフルできる。行のシャッフルにも同様のことができる。したがって、どのような再構成も $0(m_1 m_2)$ のメイクスパンを使って行うことができる。

次に、 $\Theta(m_1 m_2)$ のエスコートがある場合、ルービックテーブルアルゴリズムを用いて、再構成に $0(m_1 + m_2)$ のメイクスパンが必要であることを証明する。重要な点は、すべての行/列を $0(m_1 + m_2)$ でシャッフルする方法を見つけることである。

$\lambda m_1 m_2$ ($0 < \lambda < 1$) 車両と $(1 - \lambda)m_1 m_2$ エスコートがあると仮定する。そのために、まずラベルなしMRPを適用して、開始とゴールの構成を、複数のブロックから構成され、各ブロックが少なくとも1つの空の列または行を持つブロック構成に変換する(図11(d))。簡単のため、ブロックの各黒セルは車両で満たされていると仮定する。そうでない場合は、仮想車両で埋める。ここで、 $W = \frac{1}{1-\lambda} = 0(1)$ はブロックの幅であり、異なるブロックのシャッフルは並列に実行できる。行シャッフルも同様に $0(m_1)$ で行うことができる。

そして、ラベルなし変換が $0(m_1 + m_2)$ で完了できることを証明すればよい。変換は、図11に示すように、以下のように行うことができる。まず、初期設定から始めて、すべての車両をできるだけ右方向に動かす。(図11(a)~図11(b))。第2ステップでは、車両を上方に移動させる(図11(b)~図11(c))。構成(c)から構成(d)を得るためには、より多くの車両を持つ行の追加車両は、より少ない車両を持つ行まで列に沿って下方に移動することができる。各列が同じ数の車両を持つ後、すべての車両は左向きの行に沿って移動し、構成(d)を形成することができる。全体として、以下のすべての会議、正面衝突、または垂直衝突が発生する。

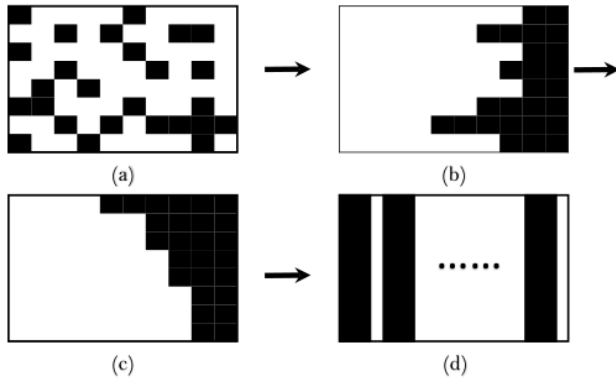


図11. $0(m_1 + m_2)$ ステップで任意の構成を「ブロック」構成にするステップ。(a) 初期設定。(b) (a)から全車両を右方向に移動させた場合の構成。(c) (b)から全車両を上方に移動させて得られる構成。(d) 望ましい通常の「ブロック」構成。

vehicles always move in one direction, and there would be no

したがって、ラベルなし変換は $0(m_1 + m_2)$ で完了する。これらを合わせると、 $\Theta(m_1 m_2)$ 個の車両がある場合、VSP の再構成には $0(m_2 + m_1)$ かかると結論づけられる。

下限は、スタートとゴールの間のマンハッタン距離の最大値を計算することで求めることができる。 $\Omega(m_1 m_2)$ 車両のランダムなスタートとゴールの場合、下界は $m_1 + m_2 - o(m_1 + m_2)$ [20] である。したがって、 $\Omega(m_1 m_2)$ 個の車両がある場合、再構成に必要なステップ数は $\Omega(m_1 + m_2)$ となる。

□

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 $n \times n$ 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.

VII. APPENDIX

Proof of Proposition IV.1. For the full density, Rubik Table algorithm can be applied here. To shuffle a column, one can apply the high-way motion primitive by utilizing two empty columns or the line-merge-sort motion primitive by utilizing one empty column [20]. Each column shuffle requires $O(m_1)$ steps and there are m_2 such columns, and all the columns can be shuffled *sequentially* using the one/two empty columns. The same can be applied to shuffle a row. Thus, any reconfiguration can be done using $O(m_1 m_2)$ makespan.

Next, we prove that when there are $\Theta(m_1 m_2)$ escorts, it requires $O(m_1 + m_2)$ makespan for the reconfiguration, using the Rubik Table Algorithm. The key point is to find a method to shuffle all rows/columns in $O(m_1 + m_2)$.

Assume that we have $\lambda m_1 m_2$ ($0 < \lambda < 1$) vehicles and $(1 - \lambda)m_1 m_2$ escorts. To do that, first, we first apply unlabeled MRPP to convert the start and goal configurations to *block configurations* (Fig. 11 (d)) which are composed of several blocks, and each block has at least one empty column or row. For simplicity, we assume that each black cell of the block is filled with a vehicle. If not, we fill in with virtual vehicles. It is not hard to see that shuffling all the columns in each block could be done in $O(W m_2)$ steps, where $W = \frac{1}{1-\lambda} = O(1)$ is the width of the block and shuffles in different blocks can be performed *in parallel*. Row shuffles can be done similarly in $O(m_1)$.

Then we only need to prove that the unlabeled conversion can be completed in $O(m_1 + m_2)$. The conversion can be done in the following way shown in Fig. 11. First, starting from the initial configuration, we move all the vehicles rightwards as much as possible. (Fig. 11(a) to Fig. 11(b)). In the second step, vehicles are moved upwards (Fig. 11 (b) to Fig. 11(c)). To get configuration (d) from configuration (c), the extra vehicles in the rows that have more vehicles can move downwards along the column to the rows that have fewer vehicles. After each row has the same number of vehicles, all the vehicles can move along the rows leftwards to form the configuration (d). In the whole process, all the

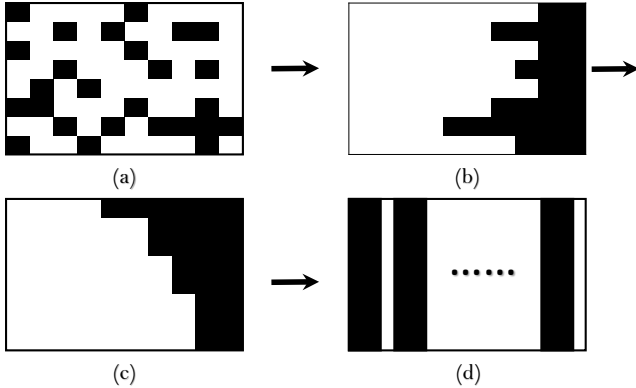


Fig. 11. Steps taking an arbitrary configuration to a “block” configuration in $O(m_1 + m_2)$ steps, on which Rubik Table results can be applied. (a) Initial configuration. (b) Configuration obtained by moving all vehicles rightwards from (a). (c) Configuration obtained by moving all vehicles upwards from (b). (d) The desired regular “block” configuration.

vehicles always move in one direction, and there would be no

meet, head-on, or perpendicular following conflicts happening. Therefore, the unlabeled conversion can be completed in $O(m_1 + m_2)$. Combining all together, we conclude that when there are $\Theta(m_1 m_2)$ vehicles, the reconfiguration of VSP takes $O(m_2 + m_1)$.

The lower bound can be obtained by computing the maximum Manhattan distance between the starts and goals. For random starts and goals with $\Omega(m_1 m_2)$ vehicles, the lower bound is $m_1 + m_2 - o(m_1 + m_2)$ [20]. Thus, the required number of steps for the reconfiguration is $\Omega(m_1 + m_2)$ when there are $\Omega(m_1 m_2)$ vehicles. □

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 $n \times n$ 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.

- [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 path-finding 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.