Exact Multi-Agent Pickup and Delivery Using Branch-and-Cut-and-Price

Edward Lam, Peter J. Stuckey, Daniel Harabor

Monash University, Australia

Abstract

Given a set of co-operating agents and a set of pickup-delivery requests located on a 2-dimensional grid, the Multi-Agent Pickup and Delivery (MAPD) problem assigns the requests to the agents such that every agent moves from its start position to the positions of its assigned requests and finally to its end position without colliding into other agents and that the sum of arrival times is minimized. This paper proposes an exact branch-and-cut-and-price algorithm that performs a three-level search. A high-level integer programming problem is solved using a branch-and-bound tree search to select an optimal subset of paths from a large collection of paths, a mid-level routing problem is solved to find the sequence of requests assigned to each agent, and a low-level path finding problem is solved to find the movements that navigate each agent to the locations of its assigned requests. A small preliminary experiment indicates that the algorithm, believed to be the first exact method, can solve instances with up to 25 agents and 50 requests.

Introduction

The Multi-Agent Pickup and Delivery (MAPD) problem is an abstraction of the problem of controlling robots in automated warehouses. The problem is defined on a discrete time horizon and a 2-dimensional grid divided into square cells. A set of co-operating agents is situated on the grid, each initially at its start cell. At every timestep, an agent can move north, south, east or west, or wait at its current cell. Every agent begins at its start cell and must arrive at its designated end cell before the end of the time horizon.

The problem includes a set of pickup-delivery pairs. Each pair comprises two requests called the pickup and the delivery. Each request is located at a given cell and is associated with a time window. The problem assigns every request pair to exactly one agent. Every agent must depart its start location, visit the cells of its assigned requests within their time windows and then arrive at its end location. Once an agent completes a pickup, the agent must then complete the associated delivery before another pickup can be attempted.

While navigating the grid, agents must not collide into each other. At most one agent can be at a cell at any given time, called the vertex conflict condition, and agents cannot cross over each other into opposite cells, called the edge conflict condition. The time that an agent reaches its end location after completing all its requests and waits there indefinitely (because other agents no longer need to pass through) is called its finish time. The problem minimizes the sum of finish times, i.e., the so-called sum-of-costs objective.

This paper introduces a branch-and-cut-and-price algorithm for MAPD called BCP-MAPD. Branch-and-cut-and-price is an advanced technique from mathematical programming that computes a strong lower bound within a branch-and-bound tree search by dynamically building a tight linear relaxation. BCP-MAPD performs a three-level search. A high-level integer programming problem is solved using a branch-and-bound tree search to select from a database a sequence of requests and a path on the map for each agent, a mid-level routing problem is solved to generate better request sequences to add into the database, and a low-level path finding problem is solved to find better movements that complete the sequence of requests assigned to each agent. BCP-MAPD is believed to be the first exact algorithm for MAPD. Preliminary experimental results show that the algorithm can optimally solve instances with up to 25 agents and 50 pickup-delivery requests. The remainder of this paper presents the algorithm and the results in detail.

Background and Literature Review

Liu et al. (2019) recognized that MAPD combines parts of the Multi-Agent Path Finding (MAPF) problem (Stern et al. 2019) and the Pickup and Delivery Problem with Time Windows (PDPTW) from the family of Vehicle Routing Problems (VRPs) (Vigo and Toth 2014).

The PDPTW is a sequencing problem of assigning pickup-delivery tasks to agents. Each task is broken up into a pickup request and delivery request. The PDPTW assigns the pickup-delivery tasks to the agents such that every request is completed within its time window and the total travel distance of all agents is minimized. All agents start at a central depot, visit the locations of their allocated requests and then return to the depot. Collisions are not considered in the PDPTW because the network is defined at a higher-level, for example, a road network. All current state-of-the-art exact algorithms for the PDPTW are based on branch-and-cut-and-price. Dumas, Desrosiers, and Soumis (1991) developed the first branch-and-price algorithm for the PDPTW, which did not include any valid inequalities, and used a branching rule that generated many children at each node. Røpke and Cordeau (2009) extended this algorithm with several families of valid inequalities and compared two shortest path algorithms for generating paths. The constraint that a path can visit each customer at most once is enforced in both the high-level and low-level in one shortest-path variant, and this constraint is only enforced at the high-level in the other variant. Baldacci, Bartolini, and Mingozzi (2011) later added sophisticated lower bounds.

MAPF considers a set of agents, each with a start cell and end cell, on a 2-dimensional grid. The problem finds a path for every agent from its start cell to its end cell such that the agents do not collide into each other and that the sum of finish times is minimized. MAPF does not contain a set of requests but simply attempts to navigate every agent from its start cell straight to its end cell without vertex and edge conflicts. The current state-of-the-art for exact MAPF are three tree search algorithms: (1) CBS, which performs a high-level tree search and solves low-level path finding problem at every node of the search tree (Li et al. 2021), (2) Lazy CBS, which adds conflict-driven clause learning from Boolean satisfiability (SAT) and constraint programming to CBS (Gange, Harabor, and Stuckey 2019), and (3) a branch-and-cut-and-price algorithm (Lam et al. 2022).

The MAPD problem can be roughly divided into a PDPTW component that constructs a sequence of requests for each agent and a MAPF component that finds a path for every agent to perform their assigned requests. The main difference between the PDPTW and MAPD is that the travel distances in the PDPTW is given as a look-up matrix, whereas in MAPD, they are computed as the solution to a MAPF problem, which varies due to the collisions, if any.

Problem Definition

The MAPD problem is defined on a 2-dimensional grid with width $W \in \mathbb{Z}_+$ and height $H \in \mathbb{Z}_+$. Define $\mathcal{L} = \{0, \dots, W-1\} \times \{0, \dots, H-1\}$ as the set of locations. A location $l = (x, y) \in \mathcal{L}$ is a pair of a horizontal coordinate x and a vertical coordinate y on the grid. Define $\mathcal{O} \subset \mathcal{L}$ as the set of obstacles. Agents cannot move into an obstacle. A location $l_2 = (x_2, y_2) \in \mathcal{L}$ is a neighbor of location $l_1 = (x_1, y_1) \in \mathcal{L}$ in the

- north direction if $x_2 = x_1$ and $y_2 = y_1 - 1$,
- south direction if $x_2 = x_1$ and $y_2 = y_1 + 1$,
- west direction if $x_2 = x_1 - 1$ and $y_2 = y_1$, and
- east direction if $x_2 = x_1 + 1$ and $y_2 = y_1$.

Under this definition, the north-west corner of the grid is the origin $(0, 0)$.

Let $\mathcal{T} = \{0, \dots, T-1\}$ be the set of discrete timesteps where $T \in \mathbb{Z}_+$. The problem is defined on a time-expanded directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{L} \times \mathcal{T}$ is the set of vertices and $\mathcal{E} = \{(((x_1, y_1), t_1), ((x_2, y_2), t_2)) \in \mathcal{V} \times \mathcal{V} : |x_2 - x_1| + |y_2 - y_1| \leq 1 \wedge t_2 = t_1 + 1 \wedge (x_2, y_2) \notin \mathcal{O}\}$ is the set of edges. A vertex $v \in \mathcal{V}$ is a location-timestep pair. An edge $e \in \mathcal{E}$ represents a movement from a location at some timestep to a neighbor location (a move action) or a movement to the same location in the next timestep (a wait action). The reverse $e' = ((l_2, t_1), (l_1, t_1 + 1))$ of an edge $e = ((l_1, t_1), (l_2, t_1 + 1))$ is a movement in the opposite direction.

Let $\mathcal{A}$ be the set of agents. Each agent $a \in \mathcal{A}$ has a start location $s_a \in \mathcal{L}$ and a goal location $g_a \in \mathcal{L}$. While the start location and goal location of an agent may be identical, all start locations are unique and all goal locations are unique. A path p of length $k \in \{1, \dots, T\}$ for agent a is a sequence of k locations $(l_0, l_1, l_2, \dots, l_{k-1})$ such that $l_0 = s_a$, $l_{k-1} = g_a$ and $((l_t, t), (l_{t+1}, t+1)) \in \mathcal{E}$ for all $t \in \{0, \dots, k-2\}$. Path p visits the vertex (l_t, t) for all $t \in \{0, \dots, k-1\}$ and (g_a, t) for all $t \in \{k, \dots, T-1\}$ because the agent remains at its goal location after the path ends. Path p traverses the edges $((l_t, t), (l_{t+1}, t+1))$ for all $t \in \{0, \dots, k-2\}$ and the edges $((g_a, t), (g_a, t+1))$ for all $t \in \{k-1, \dots, T-2\}$. It has a cost $c_p = k - 1$ equal to the number of actions required to reach the goal location (and wait there indefinitely).

Let there be $R \in \mathbb{Z}_+$ pickup-delivery tasks. Let $\mathcal{R}^\uparrow = \{0, \dots, R-1\}$ and $\mathcal{R}^\downarrow = \{R, \dots, 2R-1\}$ be the set of pickup and delivery requests respectively. For all $i = 0, \dots, R-1$, the pickup $r^\uparrow = i \in \mathcal{R}^\uparrow$ and the delivery $r^\downarrow = R + i \in \mathcal{R}^\downarrow$ are paired. Every request $r \in \mathcal{R}^\uparrow \cup \mathcal{R}^\downarrow$ is located at $L_r \in \mathcal{L}$ and must occur between $\underline{T}_r \in \mathcal{T}$ and $\bar{T}_r \in \mathcal{T}$ inclusive, where $\underline{T}_r \leq \bar{T}_r$.

The MAPD problem assigns every pickup-delivery pair $(r^\uparrow, r^\downarrow)$, $r^\uparrow \in \mathcal{R}^\uparrow$, to an agent and assigns a path to every agent. If a pickup-delivery pair $(r^\uparrow, r^\downarrow)$ is assigned to an agent, then the path assigned to the agent must visit the vertices $(L_{r^\uparrow}, t_{r^\uparrow}) \in \mathcal{V}$ and $(L_{r^\downarrow}, t_{r^\downarrow}) \in \mathcal{V}$ at some time $t_{r^\uparrow} \in \{\underline{T}_{r^\uparrow}, \dots, \bar{T}_{r^\uparrow}\}$ and $t_r^\downarrow \in \{\underline{T}_r^\downarrow, \dots, \bar{T}_r^\downarrow\}$, where $t_r^\uparrow \leq t_r^\downarrow$. At any given time, an agent can carry at most one item, i.e., $t_{r_1^\downarrow} \leq t_{r_2^\uparrow}$ or $t_{r_2^\downarrow} \leq t_{r_1^\uparrow}$ for any two different requests $r_1^\uparrow, r_2^\uparrow \in \mathcal{R}^\uparrow$ with $r_1^\uparrow \neq r_2^\uparrow$.

The paths assigned to the agents must be free of vertex collisions and edge collisions, i.e., if an agent is assigned the path $p_1 = (l_0^{p_1}, \dots, l_{k_1-1}^{p_1})$ and a different agent is assigned the path $p_2 = (l_0^{p_2}, \dots, l_{k_2-1}^{p_2})$, the conditions $l_t^{p_1} \neq l_t^{p_2}$ and $l_t^{p_1} \neq l_{t+1}^{p_2} \vee l_t^{p_2} \neq l_{t+1}^{p_1}$ must hold for all $t \in \mathcal{T}$.

A feasible solution consists of paths that satisfy all these conditions. An optimal solution is a feasible solution that minimizes the sum of path costs.

The Algorithm

This section briefly reviews the branch-and-cut-and-price technique and presents a branch-and-cut-and-price algorithm for MAPD named BCP-MAPD.

Overview

Branch-and-cut-and-price is a method for solving a difficult combinatorial optimization problem via a sequence of easier subproblems (Desrosiers and Lübbecke 2010; Lübbecke and Desrosiers 2005). The intuition behind a branch-and-cut-and-price algorithm for MAPF, named BCP-MAPF, is given by Lam (2019).

BCP-MAPD consists of four main components. The master problem selects a fractionally-optimal subset of paths from a huge but incomplete database of paths. The master problem is explicitly allowed to select multiple paths for each agent subject to the requirement that the proportions of the paths selected for each agent sum to 100%. The pricer finds lower-cost paths for each agent to add into the database. The separators resolve conflicts in solutions to the master problem and add extra reasoning by prohibiting certain combinations of selections in the master problem. The branching rules remove the fractionalities in the master problem by building a search tree that guesses whether an agent does or does not take an edge, incrementally forcing the master problem to select fewer and fewer paths for each agent until eventually it finds a solution that selects one path with 100% proportion for each agent. A complete exploration of the search tree will yield an optimal solution if one exists.

Let the navigation graph $\mathcal{G}^{\text{nav}} = (\mathcal{V}^{\text{nav}}, \mathcal{E}^{\text{nav}}) = \mathcal{G}$ explicitly denote the time-expanded graph $\mathcal{G}$, which is used for navigating the agents on the map. Define the sequencing graph $\mathcal{G}^{\text{seq}} = (\mathcal{V}^{\text{seq}}, \mathcal{E}^{\text{seq}})$ with vertices $\mathcal{V}^{\text{seq}} = \mathcal{R}^{\uparrow} \cup \mathcal{R}^{\downarrow} \cup \{\top, \bot\}$ where $\top$ and $\bot$ are source and sink vertices representing the start and goal location of an agent. The edges

$$\begin{aligned}\mathcal{E}^{\text{seq}} = &\{(\top, \bot)\} \cup \\ &\{(\top, r^{\uparrow}) : r^{\uparrow} \in \mathcal{R}^{\uparrow}\} \cup \\ &\{(r^{\uparrow}, r^{\downarrow}) : r^{\uparrow} \in \mathcal{R}^{\uparrow}\} \cup \\ &\{(i, j) : i \in \mathcal{R}^{\downarrow}, j \in \mathcal{R}^{\uparrow}, i \neq R + j\} \cup \\ &\{(r^{\downarrow}, \bot) : r^{\downarrow} \in \mathcal{R}^{\downarrow}\}\end{aligned}$$

are partitioned into five subsets, which respectively represent moving from the start location to the goal location without completing any request, from the start location to a pickup, from a pickup to its corresponding delivery, from a delivery to a different pickup, and from a delivery to the goal location. The main idea behind BCP-MAPD is to simultaneously search for paths on the sequencing graph and the navigation graph. To avoid ambiguity, we refer to vertices and edges in the sequencing graph as requests and legs.

The Master Problem

Define a request sequence $s = (\top, r_1, \ldots, r_{\cdot}, \bot)$ as a path on $\mathcal{G}^{\text{seq}}$ from the source $\top$ to the sink $\bot$, where $r_1, \ldots, r_n \in \mathcal{R}$ are not necessarily unique. For every path p, associate it with a request sequence s containing the requests completed in the path. Note that a request r does not necessarily need to be completed on a path even if the path visits the location of r because, e.g., another agent is completing the request. Define a plan $i = (s, p)$ as a pair of a request sequence and a path, which together define an order of completing the requests and a path on the map to complete these requests.

The master problem uses linear programming to minimize the sum-of-costs of selecting a set of plans for every agent such that the selected proportions of each plan sum to 100%, the paths within the selected plans are fractionally free of conflicts and every request is completed with 100% proportion within all selected plans (across all agents). Let $\Lambda_a = \{(s_{a,1}, p_{a,1}), \ldots, (s_{a,\cdot}, p_{a,\cdot})\}$ be the set of plans of agent $a \in \mathcal{A}$.

For all $a \in \mathcal{A}$ and $i = (s, p) \in \Lambda_a$, let $\lambda_{a,i} \in [0, 1]$ be a decision variable indicating the proportion of selecting plan i, and let $\alpha^r_{a,i} \in \{0, 1\}$ be a constant indicating the number of times that request $r \in \mathcal{R}^{\uparrow} \cup \mathcal{R}^{\downarrow}$ appears in the request sequence of plan i.

The master problem begins as the linear program:

$$\min \sum_{a \in \mathcal{A}} \sum_{i=(s,p) \in \Lambda_a} c_p \lambda_{a,i} \tag{1a}$$

subject to

$$\sum_{i \in \Lambda_a} \lambda_{a,i} = 1 \qquad \forall a \in \mathcal{A}, \tag{1b}$$

$$\sum_{a \in \mathcal{A}} \sum_{i \in \Lambda_a} \alpha^r_{a,i} \lambda_{a,i} = 1 \qquad \forall r \in \mathcal{R}^{\uparrow}, \tag{1c}$$

$$\lambda_{a,i} \geq 0 \qquad \forall a \in \mathcal{A}, i \in \Lambda_a. \tag{1d}$$

Objective Function (1a) minimizes the total cost of the chosen paths. Constraint (1b) requires the proportions of the selected plans for every agent to sum to 100%. Constraint (1c) requires every pickup request to be completed with 100% proportion once across all selected plans. By the definition of $\mathcal{G}^{\text{seq}}$, the corresponding delivery will be completed next after a pickup. Constraint (1d) are the non-negativity constraints standard in linear programming. Constraints (1b) and (1d) together ensure that $\lambda_{a,i} \in [0, 1]$. Constraints enforcing vertex conflicts and edge conflicts are initially omitted and added dynamically as necessary. BCP-MAPD incrementally builds the sets Λ_a and the vertex and edge conflict constraints.

Resolving Vertex and Edge Conflicts

Additional constraints prohibiting vertex conflicts and edge conflicts are dynamically added to the master problem whenever the selected paths have conflicts. Separators check the solution to the master problem and add constraints resolving conflicts whenever they occur. We use the same separators and constraints for vertex and edge conflicts as BCP-MAPF (Lam et al. 2022).

Generating Sequences and Paths

The sets Λ_a of plans are exponential in the instance size. Therefore, only a small but sufficient number of plans are generated on-demand. The pricer generates the plans for every agent by solving a two-level shortest path problem that finds a request sequence and a path that completes that sequence.

The high-level shortest path problem finds a request sequence by searching for a path on $\mathcal{G}^{\text{seq}}$. The current implementation runs a forward search starting from the source $\top$. For every extension of the partial path along an edge/leg in $\mathcal{G}^{\text{seq}}$, a low-level shortest path problem is solved on $\mathcal{G}^{\text{nav}}$ to navigate the agent from its current location to the next location of the leg. This two-level shortest path problem is excruciatingly difficult to solve.

The pricer needs to find sequences and paths that minimize the *reduced cost* instead of the regular cost function. The reduced cost bounds the maximum improvement to the cost of the current solution to master problem if the new sequence-path pair is added. The reduced cost is calculated by subtracting the dual variables of the constraints in the master problem, as in standard column generation apporaches.

Enforcing Integrality

Because the master problem can select paths with fractional proportion, it is embedded in a branch-and-bound tree search to remove these fractionalities. Nodes in the search tree correspond to guesses as to whether an edge in the sequencing graph or the navigation graph is used or not used. Branching rules are subroutines that make these choices.

After the pricer reports that no more columns can potentially improve the cost of the master problem solution and all separators report that all conflicts are resolved in the master problem solution, then the master problem has been solved at the node and the branching rules are executed to find and remove a fractionality in the master problem solution by creating children nodes.

Two branching rules are used. The first removes fractionalities in the request sequences. It selects a leg from the sequencing graph that is used by more than one agent and creates two children nodes in the search tree. The first child forces the agent to take the leg and the second child node forbids the leg. This branching rule is called until a point when all legs in the sequencing graph are used by at most one agent.

The second branching rule is then called to remove fractionalities in the paths. This branching rule selects an edge on the navigation graph that is used by more than one agent within a leg on the sequencing graph and then creates three children nodes. The first child says that the leg is used and the edge must be used during the leg. The second child says that the leg is used and the edge cannot be used during the leg. The third child says that leg cannot be used. If decisions about the leg are incompatible with earlier decisions, then the new node is infeasible and not considered further. These three children nodes partition the search space, so that any solution will appear in exactly one subtree. These decisions are also enforced in the pricer, to ensure that all new paths respect these decisions.

While it is possible to branch on an edge (i.e., requiring the agent to take or avoid an edge at any time during its plan), it is not easy to force an agent to take an edge because it is not yet decided during which leg this edge occurs. The three-way branching scheme avoids searching all legs to find optimal paths that satisfy the branching decision.

Other Constraints

Typical in mathematical programming approaches, the master problem is tightened with other constraints that provide additional reasoning. We implement a range of constraints (Lam et al. 2022; Lam and Le Bodic 2020) for MAPF as well as the subset row constraints (Jepsen et al. 2008) for VRP. Note that a huge range of constraints have been developed for the VRPs but most are theoretically (Letchford and Salazar-González 2006) and/or experimentally (Costa, Contardo, and Desaulniers 2019) shown to be ineffective within a branch-and-cut-and-price setting due to the extremely tight formulation.

Results

We conduct a small experiment to show that this exact approach is viable. Each instance is run for 2 hours. Due to the huge computation times, results for only one map are currently available. The experiments are run on the `31x79-w5` warehouse map commonly seen in MAPF studies and with up to 50 agents and 50 pickup-delivery pairs. Figure 1 shows the success rate over six instances for each pair of number of agents and number of requests. These preliminary results demonstrate that computing optimal solutions to this challenging problem is possible, even if the run times are currently excessive.

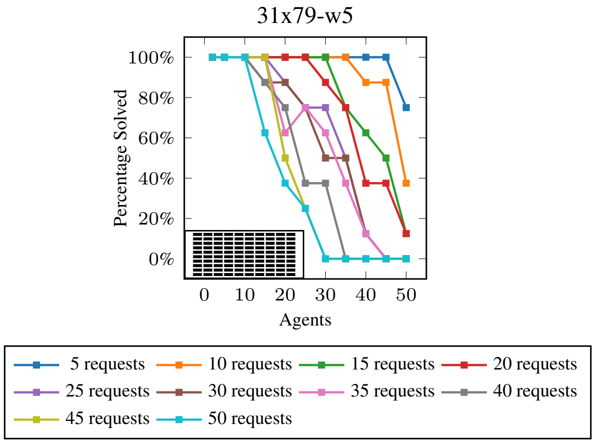

Figure 1: Success rate. Higher is better.

Conclusion and Future Directions

This paper presents a branch-and-cut-and-price algorithm for MAPD named BCP-MAPD. BCP-MAPD is believed to the first exact algorithm for MAPD. Early results indicate that it can solve small instances with up to 25 agents and 50 requests.

Over seven decades of work, algorithms for the family of VRPs are pushing towards hundreds of customers (Costa, Contardo, and Desaulniers 2019). Modern branch-and-cut-and-price algorithms can be described as a "bag of tricks" because they contain a large library of subroutines, each for tackling one part of the overall problem. Future work on BCP-MAPD should follow in this direction, so that suboptimal nodes can be pruned earlier and ultimately improving run time.

References

2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 1289–1296. International Joint Conferences on Artificial Intelligence Organization.

Baldacci, R.; Bartolini, E.; and Mingozzi, A. 2011. An Exact Algorithm for the Pickup and Delivery Problem with Time Windows. *Operations Research*, 59(2): 414–426.

Costa, L.; Contardo, C.; and Desaulniers, G. 2019. Exact Branch-Price-and-Cut Algorithms for Vehicle Routing. *Transportation Science*, 53(4): 946–985.

Desrosiers, J.; and Lübbecke, M. E. 2010. Branch-Price-and-Cut Algorithms. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Dumas, Y.; Desrosiers, J.; and Soumis, F. 1991. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1): 7–22.

Gange, G.; Harabor, D.; and Stuckey, P. 2019. Lazy CBS: Implict Conflict-Based Search Using Lazy Clause Generation. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS)*, volume 29, 155–162.

Jepsen, M.; Petersen, B.; Spoorendonk, S.; and Pisinger, D. 2008. Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows. *Operations Research*, 56(2): 497–511.

Lam, E.; and Le Bodic, P. 2020. New Valid Inequalities in Branch-and-Cut-and-Price for Multi-Agent Path Finding. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS-20)*, volume 30, 184–192.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144: 105809.

Letchford, A. N.; and Salazar-González, J.-J. 2006. Projection results for vehicle routing. *Mathematical Programming*, 105(2): 251–274.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301: 103574.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (in print).

Lübbecke, M. E.; and Desrosiers, J. 2005. Selected topics in column generation. *Operations Research*, 53(6).

Røpke, S.; and Cordeau, J.-F. 2009. Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 43(3): 267–286.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 151–158.

Vigo, D.; and Toth, P., eds. 2014. *Vehicle Routing: Problems, Methods, and Applications*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, second edition.