Task and Path Planning for Multi-Agent Pickup and Delivery

Minghua Liu
Tsinghua University
liumh413@gmail.com

Hang Ma, Jiaoyang Li, Sven Koenig
University of Southern California
(hangma,jiaoyanl,skoenig)@usc.edu

ABSTRACT

We study the offline Multi-Agent Pickup-and-Delivery (MAPD) problem, where a team of agents has to execute a batch of tasks with release times in a known environment. To execute a task, an agent has to move first from its current location to the pickup location of the task and then to the delivery location of the task. The MAPD problem is to assign tasks to agents and plan collision-free paths for them to execute their tasks. Online MAPD algorithms can be applied to the offline MAPD problem, but do not utilize all of the available information and may thus not be effective. Therefore, we present two novel offline MAPD algorithms that improve a state-of-the-art online MAPD algorithm with respect to task planning, path planning, and deadlock avoidance for the offline MAPD problem. Our MAPD algorithms first compute one task sequence for each agent by solving a special traveling salesman problem and then plan paths according to these task sequences. We also introduce an effective deadlock avoidance method, called "reserving dummy paths." Theoretically, our MAPD algorithms are complete for well-formed MAPD instances, a realistic subclass of all MAPD instances. Experimentally, they produce solutions of smaller makespans and scale better than the online MAPD algorithm in simulated warehouses with hundreds of robots and thousands of tasks.

KEYWORDS

agent coordination; multi-agent path finding; path planning; pickup and delivery task; task assignment; traveling salesman problem

ACM Reference Format:
Minghua Liu and Hang Ma, Jiaoyang Li, Sven Koenig. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION AND BACKGROUND

In many real-world applications of multi-agent systems, agents have to operate in a common environment, continuously attend to new tasks one by one, and plan collision-free paths to execute the tasks. Examples include autonomous aircraft-towing vehicles [17], office robots [28], video game characters [15], robot teams that have to change formations [9], and robots for automated warehouses [31] that have to move inventory pods from their storage locations to inventory stations or other locations. This problem is called the Multi-Agent Pickup-and-Delivery (MAPD) problem [13], where a team of agents has to execute a batch of tasks in a known environment. Each task is characterized by a pickup location, a delivery location, and a release time. To execute a task, the agent has to move first from its current location to the pickup location of the task (it has to be at the pickup location at or after the release time of the task) and then to the delivery location of the task, without colliding with other agents. Multiple tasks can be assigned to each agent.

Ma et al. [13] studied the online version of the MAPD problem, where each task becomes known only after its release time. However, the tasks and their release times are often known a priori. For example, packages in automated warehouses might be prepared at specific times that are known a priori, and a robot can pick up a package only after its preparation. Similarly, the takeoff and landing times of aircraft might be known a priori for autonomous aircraft towing. We therefore study the offline version of the MAPD problem, where all tasks and their release times are known a priori. Online MAPD algorithms can be applied to the offline MAPD problem but then do not utilize all available information and may thus not be effective. We therefore present two novel offline MAPD algorithms that improve a state-of-the-art online MAPD algorithm by using more information (and in other ways) and evaluate them with respect to makespan (for effectiveness) and runtime (for efficiency).

1.1 Related Work

The MAPD problem is related to the generalized target assignment and path finding problem, that Nguyen et al. [18] solve with answer set programming. They propose an approach for a simplified warehouse variant (where the number of tasks is no larger than the number of agents) that operates in three phases, which leads to unnecessary waiting of agents between phases, and scales only to 20 agents or tasks.

The task-assignment aspect of the MAPD problem is related to multi-robot task-allocation problems, which have been widely studied, see Nunes et al. [19] for a survey. Most closely related are the traveling salesman problem (TSP), the vehicle routing problem, and their constrained versions [1, 20, 24, 32]. Their pickup and delivery versions have also received attention. Calvo and Colorni [3] propose a heuristic algorithm for the dial-a-ride problem, where a fleet of vehicles without fixed routes and schedules transports people from their pickup locations to their delivery locations, with the objective of maximizing the number of people served. Das et al. [4] propose an approximation algorithm for the multi-vehicle minimum latency problem with point-to-point requests, with the objective of minimizing the total latency while serving all requests. However, none of these problems are a perfect match for the task-assignment aspect of the MAPD problem.

The path-planning aspect of the MAPD problem is related to the Multi-Agent Path Finding (MAPF) and Anonymous Multi-Agent Path Finding (AMAPF) problems. The MAPF problem is to plan collision-free paths for agents from given start locations to given goal locations in a known environment. The MAPF problem is NP-hard to solve optimally with regard to some objectives [14, 35] and sometimes even NP-hard to approximate within given factors [14].

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, and 1837779 as well as a gift from Amazon. Minghua Liu performed his research while being an undergraduate exchange student at the University of Southern California as part of the USC Viterbi/Tsinghua Summer Research Program. We would like to thank Eli Boyarski for helpful comments and Zecong Hu for editorial help.
Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

It can be solved with dedicated MAPF algorithms [7, 10, 21–23, 25, 29, 30] or reductions to other well-studied combinatorial problems [5, 26, 34], see [6, 12] for a survey. The AMAPF problem is a version of the MAPF problem where "anonymous" agents can swap their goal locations. The AMAPF problem is thus to plan a one-to-one assignment of agents to goal locations and collision-free paths for the agents from given start locations to their assigned goal locations in a known environment. The AMAPF problem can be solved in polynomial time using max-flow algorithms [11, 33] or graph-theoretic algorithms [16]. We explain later how a MAPD instance (where each agent has to visit multiple locations instead of a single goal location) can be divided into several MAPF/AMAPF instances.

1.2 Contributions

We present two offline MAPD algorithms, called Task Assignment and Prioritized path planning (TA-Prioritized) and Task Assignment and Hybrid path planning (TA-Hybrid). Both MAPD algorithms first assign the tasks to agents. They compute one task sequence for each agent by solving a special TSP, which ignores collisions and thus uses estimated travel times between locations. Its purpose is to minimize the makespan according to the estimated travel times. Both MAPD algorithms then plan collisions-free paths for the agents that visit pickup and delivery locations in the order of the tasks in their task sequences, while keeping the makespan according to the actual travel times small. Both MAPD algorithms use the same task-planning method. They also use the same deadlock-avoidance method during path planning, called "reserving dummy paths," which guarantees their completeness. (A dummy path of an agent is a path with minimal travel time to the parking location of the agent.) They differ in their path-planning methods and thus in their efficiency-effectiveness trade-offs.

Ma et al. [13] present three online MAPD algorithms that can also be applied to the offline MAPD problem. CENTRAL is the most effective one. It iteratively assigns tasks to agents with the Hungarian algorithm in the outer loop and then plans their paths by solving MAPF instances with Conflict-Based Search (CBS) [21] in the inner loop. Our MAPD algorithms improve CENTRAL with respect to task planning, path planning, and deadlock avoidance for the offline MAPD problem. They produce solutions of smaller makespans and scale better than CENTRAL in simulated warehouses with hundreds of robots and thousands of tasks.

2 PROBLEM DEFINITION

A MAPD problem consists of a set of M agents $A = \{a_1, \ldots, a_M\}$, a set of N tasks $T = \{t_1, \cdots, t_N\}$, and an undirected connected graph $G = (V, E)$ whose vertices V correspond to locations and whose edges E correspond to connections between the locations that the agents can move along. Each task $t_j \in T$ is characterized by a pickup location $s_j \in V$, a delivery location $g_j \in V$, and a non-negative integer release time r_j. Each agent a_i has a unique parking location $p_i \in V$ assigned to it and starts in it at time step 0. At each discrete time step, the agent either executes a wait action to stay in its current location or a move action to move to an adjacent location. Agents have to avoid collisions with other agents: (a) Two agents cannot be in the same location at the same time step (vertex collision), and (b) two agents cannot move along the same edge in opposite directions at the same time step (edge collision). An agent that is not executing a task is called a free agent and can be assigned any unassigned task t_j. To execute its task, the agent has to move from its current location via pickup location s_j to delivery location g_j without colliding with other agents. It has to be at the pickup location at or after the release time r_j. It then starts to execute its task and is called a task agent. When it reaches the delivery location, it finishes to execute its task and is called a free agent again. The objective of the MAPD problem is to minimize the makespan, that is, the earliest time step when all tasks have been executed.

The pickup and delivery locations of tasks are called task endpoints, and the parking locations of agents are called non-task endpoints. Although not every MAPD instance is solvable, well-formed MAPD instances are always solvable [13]. A MAPD instance is well-formed iff (a) the number of tasks is finite, (b) the parking location of each agent is different from all task endpoints, and (c) there exists a path between any two endpoints that traverses no endpoints. In well-formed MAPD instances, agents can always stay in their parking locations for as long as necessary to avoid collisions with other agents. Well-formed MAPD instances are a realistic subclass of all MAPD instances since many real-world MAPD instances are well-formed, including for automated warehouses.

3 TASK ASSIGNMENT

The task-assignment part, shared by both of our MAPD algorithms, computes one task sequence for each agent. The task sequence of agent a_i specifies which tasks are assigned to the agent and in which order the agent should execute them. The execution time $\mathcal{M}_i$ of the task sequence is the number of time steps required for the agent to execute all tasks in the task sequence in the given order. Even though task assignment ignores collisions and the agent thus does not have to wait to avoid collisions with other agents, the execution time can be different from its travel distance since it might have to wait for the release times of tasks. The primary objective of our MAPD algorithms is to minimize the makespan $\max(\mathcal{M}_i)$, which is the largest execution time of all task sequences. Their secondary (tie-breaking) objective is to minimize the sum of the execution times of all task sequences $\sum \mathcal{M}_i$. Our MAPD algorithms first construct a directed weighted graph for a MAPD instance using a similar idea as in [20, 32]. They then solve a special TSP on it to compute good task sequences for these objectives.

3.1 Constructing the Graph

Our MAPD algorithms first construct a directed weighted graph $G' = (V', E')$ with $V' = \mathcal{A} \cup \mathcal{T}$, where vertex $\alpha_i \in \mathcal{A}$ represents agent a_i and vertex $\tau_i \in \mathcal{T}$ represents task t_i. There are four types of edges $(u, v) \in E'$, each of which has an integer weight $w(u, v)$:

$$w(u, v) = \begin{cases} \max(dist(p_i, s_j), r_j) & u = \alpha_i, v = \tau_j \\ dist(s_i, g_i) + dist(g_i, s_j) & u = \tau_i, v = \tau_j \\ dist(s_i, g_i) & u = \tau_i, v = \alpha_j \\ 0 & u = \alpha_i, v = \alpha_j, \end{cases}$$

where $dist(u, v)$ is the distance from u to v in G. The first row computes the edge weight from α_i to τ_j as the travel time of agent a_i from its parking location p_i to the pickup location s_j of its first task t_j plus, if needed, the wait time of the agent for the release time

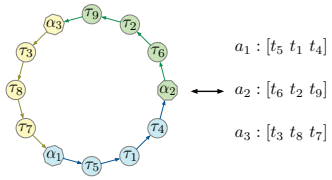


Figure 1: Left: A Hamiltonian cycle, which contains three agent vertices and nine task vertices. Right: The corresponding task sequences for the agents.

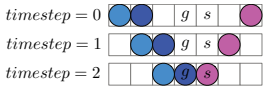


Figure 2: Paths for three agents. Although another agent can move from location s to location g at time step 0, it cannot stay in location g or move to other locations afterward.

r_j of the task. The second row computes the edge weight from τ_i to τ_j as the travel time of the agent from the pickup location s_i of task t_i via the delivery location g_i of the same task to the pickup location s_j of its next task t_j. The third row computes the edge weight from τ_i to α_j as the travel time of the agent from the pickup location s_i of its last task to the delivery location g_i of the same task. Finally, the fourth row computes the edge weight from α_i to α_j as zero since agent a_i is assigned no tasks and thus incurs no travel time.

Since G' is a complete graph, it contains Hamiltonian cycles. Since a Hamiltonian cycle visits each agent vertex exactly once, it can be partitioned into M parts, where each part consists of an agent vertex, a sequence of task vertices, and another agent vertex (in this order). Since a Hamiltonian cycle also visits each task vertex exactly once, the M parts can be converted to M task sequences, one for each agent. Figure 1 shows an example. The sum of the edge weights of each part is a lower bound on the execution time of the corresponding task sequence since the edge weights do not model the release times of the tasks except for the first task of each task sequence.

3.2 Solving a Special TSP

Our MAPD algorithms use the LKH-3 TSP solver [8] to plan a good Hamiltonian cycle on G' for their objectives given earlier. The TSP solver is able to solve a variety of constrained TSPs and vehicle routing problems by transforming them to the standard symmetric TSP and handling their constraints with penalty functions. We let the TSP solver compute the execution time of each task sequence during each of its iterations so that it can take the release times of all tasks in the task sequence into account. Consider the task sequence $[t_{i_1}, t_{i_2}, \ldots, t_{i_{l_i}}]$ of agent a_i and the time step $start(t_{i_k})$ when the agent starts to execute task t_{i_k}. For the first task t_{i_1}, we have $start(t_{i_1}) = w(\alpha_i, \tau_{i_1})$ since the edge weights of G' take the release time of the first task into account. For the following tasks t_{i_k}, we have $start(t_{i_k}) = \max(start(t_{i_{k-1}}) + w(\tau_{i_{k-1}}, \tau_{i_k}), r_{i_k})$ for all $k = 2, \ldots, l_i$ since the edge weights of G' do not take the release times of these tasks into account. The execution time $\mathcal{M}_i$ of the task sequence is $start(t_{i_{l_i}}) + w(\tau_{i_{l_i}}, \alpha_{i+1})$, where α_{i+1} is the next agent vertex in the Hamiltonian cycle. The TSP solver appears to plan good task sequences even though it has to solve a non-Euclidean Hamiltonian cycle problem for uncommon objectives.

4 PRIORITIZED PATH PLANNING

The path-planning part of TA-Prioritized uses an improved version of prioritized planning [27] to plan collision-free paths for the agents to execute all of their tasks according to their task sequences. Van den Berg and Overmars [27] solve the MAPD problem by planning paths for the agents, one by one, in decreasing order of the estimated execution times of their task sequences, that is, giving priority to agents with larger estimated execution times. After a path has been planned for an agent, the paths of all remaining agents are not allowed to collide with it. This way, agents with larger estimated execution times have fewer constraints, which may result in a smaller makespan. This technique has already chosen the next agent based on the estimated execution times before it plans a path for an agent. TA-Prioritized improves on this technique by choosing the next agent only after it has planned a path for an agent. This way, it can choose the next agent based on the actual execution times (that take the paths of the previous agents into account). For each remaining agent, it tentatively assumes that it chooses this agent next and plans a path for it. It then chooses the agent next whose path has the the largest execution time, and the procedure repeats. TA-Prioritized obtains collision-free paths for all agents after M iterations, during each of which it plans paths for at most M remaining agents.

The path of an agent is a concatenation of several sub-paths according to its task sequence, namely, from its start location to the pickup location of its first task, to the delivery location of its first task, to the pickup location of its second task, and so on, ending at the delivery location of its last task. TA-Prioritized constructs the path sub-path by sub-path, where a sub-path moves the agent from its current location to its goal location, which is either the pickup location of a task at or after its release time or the delivery location of a task. All sub-paths have to avoid collisions with the paths of all previous agents and cannot contain the parking locations of all remaining agents. They can be found via an A* search for a shortest path in the space of location-time pairs (x, t). TA-Prioritized plans them without backtracking to achieve efficiency, which is nontrivial since there might not exist a collision-free sub-path for an agent from its current location to its goal location. Figure 2 shows an example where TA-Prioritized would have to backtrack to plan a different sub-path or risk being incomplete. TA-Prioritized addresses this issue by using "reserving dummy paths" as deadlock-avoidance method.

TA-Prioritized implements "reserving dummy paths" by changing the goal-test function of the A* search for each sub-path: An A* node (x, t) is a goal node iff (1) location x is the goal location of the agent, (2) time step t is at or after the release time of the task if the goal location is the pickup location of the task, and (3) the A* search is able to plan a "dummy path" from location x at time step t to the parking location of the agent. This dummy path has to "hold" the parking location (that is, allow the agent to stay there forever), avoid collisions with the paths of all previous agents (and their final dummy paths, see below), and cannot contain the parking locations of all remaining agents. If the goal test fails, then the A* search

continues. An agent never moves along its dummy path, except for its last one (the "final" dummy path), that moves it from the delivery location of its last task to its parking locations and holds it, since the purpose of a dummy path is only to guarantee that the subsequent sub-path for the agent (and its dummy path), that replace this dummy path, exists. All sub-paths also have to avoid collisions with the final dummy paths of all previous agents. The objective of TA-Prioritized is to minimize the makespan by minimizing the execution time of each sub-path (without its final dummy path).

Whenever TA-Prioritized plans a sub-path for an agent (and its dummy path) for well-formed MAPD instances, it is guaranteed to find collision-free ones (and is thus complete) since they exist, which can be proved by induction. Each agent starts in its parking location. Assume that the agent is in its parking location. It can stay there until all previous agents have moved along their paths (and their final dummy paths) to their parking locations. Then, it can first move to its goal location, then move back to its parking location, and finally stay there for as many time steps as needed. This path does not collide with the paths of the previous agents because it avoids the parking locations of the previous agents and their paths avoid its parking location. Such a collision-free path exists since the MAPD instance is well-formed, which implies that paths can avoid all parking locations. This proof continuous to work (and TA-Prioritized is thus still complete) if the definition of a well-formed MAPD problem is changed to require only that there exists a path between any two endpoints that traverses no parking locations rather than no endpoints.

Completeness cannot only be guaranteed by "reserving dummy paths" but also by "holding the goal location" [13]. However, this existing deadlock-avoidance method can result in sub-paths with large execution times in case the paths of some previous agents pass through the goal location of an agent. This is so because the agent then has to wait until the other agents have passed through its goal location before it can move to it and stay there for as many time steps as needed.

5 HYBRID PATH PLANNING

TA-Hybrid is similar to TA-Prioritized but uses a different path-planning method to achieve a different efficiency-effectiveness trade-off. The path-planning part of TA-Hybrid uses MAPF-based path planning to plan the paths of the new task agents and AMAPF-based path planning to plan the paths of the free agents. We first give overviews of TA-Hybrid in Section 5.1, path planning for the new task agents in Section 5.2, and path planning for the free agents in Section 5.3. Then, we describe in detail in Section 5.4 how the min-cost max-flow algorithm is used for path planning for the free agents.

5.1 Framework

TA-Hybrid considers two groups of agents for path planning and uses a different path-planning method for each group:

- **Group 1: New task agents**. TA-Hybrid plans sub-paths for them from their current locations to the delivery locations of their current tasks. The agents cannot swap their delivery locations. TA-Hybrid thus uses Improved Conflict-Based Search (ICBS) [2], a recent version of CBS, to perform MAPF-based path planning for them. The resulting paths remain unchanged until the new task agents reach their delivery locations since TA-Hybrid plans paths only for new task agents but not the other task agents.
- **Group 2: Free agents**. TA-Hybrid plans sub-paths for them from their current locations to the pickup locations of the next tasks in their task sequences. It may have planned such paths for them before. However, the paths can be improved while the agents follow them, for example, by the agents swapping their pickup locations. TA-Hybrid thus uses a polynomial-time min-cost max-flow algorithm to perform AMAPF-based path planning for them at every time step where the set of free agents has changed.

Algorithm 1: TA-Hybrid

Input: G, *agents* (= A), and *tasks* (= T).
Output: none.

```
1  timestep ← 0 ;
2  sequences ← TaskAssignment(G, agents, tasks) ;
3  while not all tasks have been executed do
4      for i = 1, . . . , M do
5          if sequences[i].size() > 0 then
6              task ← sequences[i].front();
7              if agents[i].location = task.delivery_location then
8                  sequences[i].delete(task);
9      PlanPathsToDelivery(G, agents, sequences, timestep);
10     if the set of free agents has changed or timestep = 0 then
11         PlanPathsToPickup(G, agents, sequences, timestep);
12     Move all agents for one time step ;
13     timestep ← timestep + 1;
```

Algorithm 1 shows the pseudo-code of TA-Hybrid. It first calls Procedure TaskAssignment to compute the task sequences of all agents (Line 2). At each time step, it checks whether one or more task agents have arrived at the delivery locations of their current tasks (Line 7). If so, then it turns each such agent into a free agent and removes the task from its task sequence (Line 8). This way, the first task in the task sequence of each agent is always its current task. Then, TA-Hybrid calls Procedure PlanPathsToDelivery to perform MAPF-based path planning for all agents in Group 1 (Line 9). Afterward, if the set of free agents has changed or it is the first time step (Line 10), TA-Hybrid calls Procedure PlanPathsToPickup to perform AMAPF-based path planning for all agents in Group 2 (Line 11). Finally, all agents move for one time step (Line 12), the current time step is incremented (Line 13), and the procedure repeats until all tasks have been executed (Line 3).

TA-Hybrid uses "reserving dummy paths" for both Procedures PlanPathsToDelivery and PlanPathsToPickUp to guarantee completeness. Like for TA-Prioritized, the agents never move along their dummy paths, except for the final one of each agent, since the purpose of a dummy path is only to guarantee that the subsequent sub-path for the agent (and its dummy path) for the agent, that replace this dummy path, exists. Unlike for TA-Prioritized, all dummy paths are stored since collisions with them have to be avoided, for the following reason: After TA-Prioritzed has planned a (non-final) dummy path for an agent, it immediately plans the next sub-path for the agent (and its dummy path), that replace this dummy

Algorithm 2: PLANPATHSTODELIVERY

Input: G, *agents* (= A), their (task) *sequences*, and the current *timestep*.
Output: *agent*.path for all agents *agent* in *group1*.

```
1  group1 ← ∅;
2  for i = 1, . . . , M do
3      if sequences[i].size() > 0 then
4          task ← sequences[i].front() ;
5          if timestep ≥ task.release_time and not task.executing then
6              if agents[i].location= task.pickup_location then
7                  group1.append(agents[i]);
8  if group1.size() > 0 then
9      constraints ← {agent.path | agent ∉ group1};
10     ICBS(G, group1, constraints, timestep);
```

Algorithm 3: PLANPATHSTOPICKUP

Input: G, *agents* (= A), their (task) *sequences*, and the current *timestep*.
Output: *agent*.path for all agents *agent* in *group2* and their (task) *sequences*.

```
1  group2 ← ∅ ;
2  for i = 1, . . . , M do
3      if sequences[i].size() > 0 then
4          task ← sequences[i].front();
5          if not task.executing then
6              group2.append(agents[i]);
7  if group2.size() > 0 then
8      Partition group2 into subgroups;
9      for each subgroup do
10         constraints ← {agent.path | agent ∉ subgroup};
11         MINCOSTMAXFLOW(G, subgroup, constraints, timestep);
12         DUMMYPATHPLANNING(G, subgroup, constraints, timestep);
```

path, and thus does not have to avoid collisions with this dummy path. On the other hand, after TA-Hybrid has planned the dummy path for an agent, it does not immediately replace it and thus has to avoid collisions with it.

Whenever TA-Hybrid plans a sub-path for an agent (and its dummy path) for well-formed MAPD instances, it is guaranteed to find collision-free ones (and is thus complete) since they exist, as we show separately for Groups 1 and 2 in the following. This proof continues to work (and TA-Hybrid is thus still complete) if the definition of a well-formed MAPD problem is changed to require only that there exists a path between any two endpoints that traverses no pickup and parking locations rather than no endpoints.

5.2 Group 1: Paths to Delivery Locations

Algorithm 2 shows the pseudo-code of Procedure PLANPATHSTODELIVERY for all agents in Group 1. TA-Hybrid checks whether one or more free agents are at the pickup locations of their current tasks at or after their release times and are not executing them yet (Lines 5-6). If so, then each such agent turns into a task agent and is part of Group 1 (Line 7). Then, TA-Hybrid uses MAPF-based path planning with ICBS to plan the next sub-paths for all agents in Group 1 (and their dummy paths) simultaneously (Line 10). These sub-paths (and their dummy paths) have to avoid collisions with the sub-paths of all agents not in Group 1 (and their dummy paths) (Line 9). The objective of ICBS is to minimize the makespan by using, for each sub-path, the sum of the execution time of the sub-path (without its dummy path) and the estimated execution time for the remaining tasks in the task sequence (as provided by the task-assignment part).

TA-Hybrid implements "reserving dummy paths" by changing the A* search on the low-level of ICBS so that it plans a sub-path from the current location of an agent to its delivery location and a dummy path from there to its parking location. The dummy path has to hold the parking location. Both paths have to avoid collisions with the sub-paths of all agents not in Group 1 (and their dummy paths), in addition to the collisions already avoided by the standard A* search on the low-level of ICBS.

Whenever Procedure PLANPATHSTODELIVERY plans sub-paths for all agents in Group 1 (and their dummy paths), it is guaranteed to find collision-free ones since they exist. All agents can move along their sub-paths (and their dummy paths) to their parking locations. Then, all agents in Group 1, one by one, can first move to their delivery locations, then move back to their parking locations, and finally stay there for as many time steps as needed. Such collision-free paths exist for the same reason as for TA-Prioritized.

5.3 Group 2: Paths to Pickup Locations

Algorithm 3 shows the pseudo-code of Procedure PLANPATHSTO-PICKUP for all agents in Group 2. TA-Hybrid checks whether one or more agents do not yet execute their current tasks (Lines 5). If so, then each such agent is a free agent and part of Group 2 (Line 6). Then, TA-Hybrid could use ICBS to perform MAPF-based path planning in order to plan the next sub-paths for all agents in Group 2 (and their dummy paths). However, it uses a min-cost max-flow algorithm for AMAPF-based path planning instead because it is faster and can improve the task sequences and thus the resulting makespan - but it can plan only for subsets of agents in Group 2 simultaneously whose current pickup locations are pairwise different and can plan their sub-paths only to their current pickup locations. Each such sub-path has to hold its pickup location to guarantee that the subsequent dummy path for the agent, that replaces holding the pickup location, exists. TA-Hybrid plans dummy paths for the agents from their assigned pickup locations to their parking locations afterward.

First, TA-Hybrid partitions the agents in Group 2 into subgroups so that agents whose current tasks have the same pickup location are in different subgroups (Line 8). It assigns integers to all agents with the same pickup locations, starting with 1, and then puts agents with the same integer into the same subgroup. Second, for each subgroup, TA-Hybrid uses a min-cost max-flow algorithm to plan a one-to-one mapping from the agents in the subgroup to the current pickup locations of these agents as well as collision-free sub-paths for them that arrive at the pickup locations after the release times of their tasks (Line 11). These sub-paths have to hold their pickup locations and avoid collisions with the sub-paths of all agents not in the subgroup (and their dummy paths) (Line 10). If an agent is assigned the current pickup location of a different agent, TA-Hybrid replaces its current task sequence with the task sequence of this different agent, which can improve the resulting makespan. The objective of the min-cost max-flow algorithm is to minimize the makespan, taking into account the release times of the current tasks of the agents as well as the time steps when agents have to start executing them to minimize the makespan according to the

estimated execution times of their task sequences once the first tasks in the task sequences have started to get executed (as provided by the task-assignment part). Third, TA-Hybrid uses an A* search to plan for each agent in the subgroup, one by one, a dummy path from its assigned pickup location to its parking location (Line 12). Each dummy path has to hold the parking location and avoid collisions with the sub-paths of all other agents (and their dummy paths if they have been planned already).

Whenever Procedure PLANPATHSTOPICKUP plans sub-paths for all agents in Group 2 (and their dummy paths), it is guaranteed to find collision-free ones since they exist. All agents can move along their sub-paths (and their dummy paths) to their parking locations. Then, all agents in a subgroup of Group 2, one by one, can move to their assigned pickup locations and stay there for as many time steps as needed. Then, all agents in the subgroup, one by one, can move to their parking locations and stay there for as many time steps as needed. Such collision-free paths exist since the MAPD instance is well-formed and the assigned pickup locations are pairwise different, which implies that paths can avoid all parking and pickup locations.

5.4 Min-Cost Max-Flow for AMAPF Instances

We now provide details on Procedure MINCOSTMAXFLOW. Let $A' \subseteq A$ be the agents in a subgroup of Group 2, $T' \subseteq T$ be the first tasks in their task sequences, and t_0 be the current time step. Let L be a lower bound on the makespan of the AMAPF instance, initially the maximum of the estimated execution times of the task sequences of all agents in A' (as provided by the task-assignment part). Given makespan bound L, TA-Hybrid computes the deadline L_j of each task $t_j \in T'$ as L minus the estimated execution time of the task sequence that task t_j belongs to once task t_j starts to get executed. Since the estimated execution time is actually a lower bound on the execution time, a makespan of L can only be achieved if each task $t_j \in T'$ starts to get executed at or before its deadline L_j. TA-Hybrid constructs a time-extended directed flow network to check whether this appears possible. It uses a min-cost max-flow algorithm to plan a feasible integer flow of $|A'|$ units, the maximum possible, with first priority and to minimize the sum-of-costs of the flow with second priority. If the resulting flow is not of $|A'|$ units, then TA-Hybrid increments the makespan bound, and the procedure repeats. Otherwise, TA-Hybrid transforms the flow to sub-paths for all agents in A' from their (current) locations at time step t_0 to the pickup locations of all tasks in T'. If these sub-paths have edge collisions with each other, then TA-Hybrid re-assembles the paths of the involved agents until all edge collisions have been resolved. Afterwards, if an agent is assigned the current pickup location of a different agent, TA-Hybrid replaces its current task sequence with the task sequence of this different agent. We now provide details on how TA-Hybrid constructs the flow network and transforms the resulting flow to sub-paths for all agents in A'. Figure 3 shows an example.

TA-Hybrid follows [11, 34] to construct the flow network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ from time step t_0 to time step $\mathcal{L} = \max(L_j)$. All edges have unit capacity. Every location $v \in V$ at time step t_0 is represented by a vertex $v_{t_0}^{out} \in \mathcal{V}$. Every location $v \in V$ at every time step $t \in [t_0 + 1, \mathcal{L}]$ is represented by two vertices $v_t^{in} \in \mathcal{V}$ and $v_t^{out} \in \mathcal{V}$, which are connected with a zero-cost edge $(v_t^{in}, v_t^{out}) \in \mathcal{E}$ to prevent

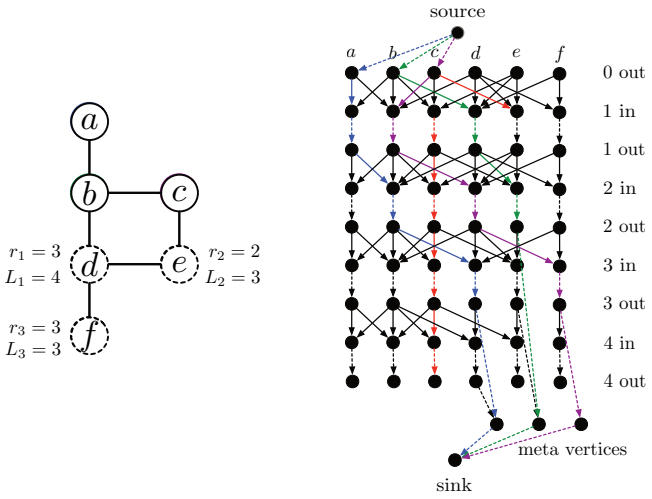


Figure 3: Example of constructing the flow network (right) for a 3-agent AMAPF instance (left). The current locations of the three agents are a, b, and c, and the pickup locations of the three tasks are d, e, and f. The release times and deadlines for all tasks are listed next to their pickup locations. All edges have unit capacities. All solid edges have unit costs, while all dotted edges have zero costs. A different agent has sub-path (e, c, c, c, c), which results in all red edges being removed from the flow network. The blue, green, and purple lines show the feasible integer min-cost max-flow $\{(a, a, b, d), (b, d, e), (c, b, d, f)\}$, whose sum-of-costs is 8.

vertex collisions. (We explain below how to prevent edge collisions.) A move action from location u to location v (corresponding to an edge $(u, v) \in E$) at time step t is represented by a unit-cost edge $(u_t^{out}, v_{t+1}^{in}) \in \mathcal{E}$, and a wait action in location v at time step t is represented by a unit-cost edge $(v_t^{out}, v_{t+1}^{in}) \in \mathcal{E}$. Constraints on the movement of the agents in A' result in the removal of some of these edges. First, the agents have to hold the pickup locations s_j of tasks $t_j \in T'$. To achieve this, TA-Hybrid allows only one agent to visit the pickup location s_j of each such task t_j after its release time r_j. The edges $((s_j)_t^{out}, u_{t+1}^{in})$ for all edges $(s_j, u) \in E$ and time steps $t = r_j, \ldots, \mathcal{L} - 1$ are therefore removed. Second, the agents have to avoid collisions with the sub-paths of agents not in A' (and their dummy paths). For each such path $P = (p_{t_0}, p_{t_0+1}, \ldots)$, the edges $((p_t)_t^{in}, (p_t)_t^{out})$ for all time steps $t = t_0, \ldots, \mathcal{L}$ are removed to avoid vertex collisions, and the edges $((p_{t+1})_t^{out}, (p_t)_{t+1}^{in})$ for all time steps $t = t_0, \ldots, \mathcal{L} - 1$ are removed to avoid edge collisions.

The idea behind the design of the flow network is to plan a feasible integer flow from the source vertex to the sink vertex that corresponds to the sub-paths of all agents in A'. Thus, the source vertex is connected with zero-cost edges to the vertices that correspond to the locations of the agents in A' at time step t_0. The start of the execution of each task $t_j \in T'$ in its pickup location s_j is represented by a "meta" vertex in $\mathcal{V}$. The execution of the task has to start between its release time s_j and its deadline L_j but the pickup location s_j can only be held after the last time step r_j' when the sub-path of any agent not in A' passes through this location. Thus, only the vertices $(s_j)_t^{out}$ for all time steps $t = \max(r_j, r_j' + 1), \ldots, L_j$ are connected with zero-cost edges to the meta vertex of the task,

which, in turn, is connected with a zero-cost edge to the sink vertex. If $\max(r_j, r'_j + 1) > L_j$, then a makespan of L is unachievable. In this case, TA-Hybrid increments the makespan bound, and the procedure repeats.

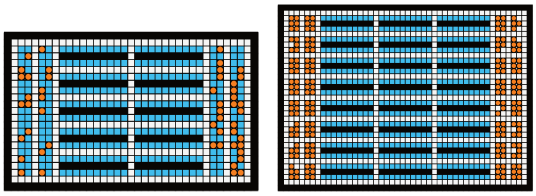
If the min-cost max-flow algorithm finds a feasible integer flow of $|A'|$ units, TA-Hybrid transforms it to $|A'|$ sub-paths. Each unit flow from the source vertex to vertex $v_{t_0}^{out}$ is transformed to the start of the sub-path of the agent in A' whose location at time step t_0 is v. Each unit flow from vertex u_t^{out} to vertex v_{t+1}^{in} is transformed to an action that moves the agent that is at location u at time step t to location v. Finally, each unit flow from vertex $(s_j)_t^{out}$ to a meta vertex is transformed to the end of the sub-path of the agent that is at pickup location s_j of task t_j at time step t, with the understanding that the sub-path holds the pickup location. The resulting sub-paths can have edge collisions with each other since we opted to keep the flow network small and thus have not used the gadget from [11, 34] to avoid them. Instead, TA-Hybrid resolves each edge collision by re-assembling the paths of the involved agents. It lets both agents wait for one time step and then follow the path of the other agent, which is possible for AMAPF instances since the agents can swap their pickup locations. This change does not increase the resulting makespan or sum-of-costs.

5.5 Improvements

We use several improvements to TA-Hybrid to decrease the resulting makespan: First, avoiding collisions with the dummy paths of other agents when solving MAPF or AMAPF instances is necessary for completeness but can result in an unnecessarily large makespan. Thus, TA-Hybrid first solves each MAPF or AMAPF instance without avoiding collisions with the dummy paths and, in case of resulting collisions with them, then tries to re-plan the affected dummy paths. If that fails, then it keeps the original dummy paths and re-solves the MAPF or AMAPF instance with avoiding collisions with the original dummy paths. Second, holding the pickup locations when computing the min-cost max-flows for AMAPF instances is also necessary for completeness but can result in an unnecessarily large makespan. Thus, TA-Hybrid first computes each min-cost max-flow without holding the pickup locations. If it cannot plan dummy paths for the agents in the subgroup afterward, then it re-solves the min-cost max-flow with holding the pickup locations and re-plans the dummy paths for the agents in the subgroup. Finally, not letting agents execute tasks once their task sequences are empty can also result in an unnecessarily large makespan. Thus, whenever the task sequence of an agent becomes empty, TA-Hybrid removes the last task from the task sequence with the largest estimated execution time and assigns it to the agent.

6 EXPERIMENTAL RESULTS

In this section, we compare TA-Prioritized and TA-Hybrid to CENTRAL and three other strawman MAPD algorithms. Table 1 summarizes these MAPD algorithms. Online MAPD algorithms consider only tasks whose release times are not in the future, while offline MAPD algorithms consider all tasks. For example, GREEDY_1 and CENTRAL are very similar, except that GREEDY_1 is an offline MAPD algorithm that performs MAPF-based path planning for the group of new task agents and combines task assignment and AMAPF-based path planning for the group of free agents to take the actual travel times of agents into account, while CENTRAL is an online MAPD algorithm that first uses the Hungarian algorithm with estimates of the travel times for task planning and then performs the slower MAPF-based path planning for the group of new task agents and the group of free agents separately. We implemented all algorithms in C++ and ran them on a 2.50 GHz Intel Core i7 PC with 16GB RAM to perform experiments on offline MAPD instances in two simulated warehouses.



(a) 21×35 grid with 50 agents (b) 33×46 grid with 180 agents

Figure 4: Two 4-neighbor grids that represent simulated warehouses. Black cells are blocked, blue cells are task endpoints (pickup locations and/or delivery locations of tasks), and orange circles are non-task endpoints (both start and parking locations of agents).

The **small warehouse** is shown in Figure 4(a). We generated one sequence of 500 pickup and delivery tasks by randomly choosing their pickup and delivery locations from all task endpoints. We used five different task frequencies (characterized by the number of tasks that are released at each time step): 1, 2, 5, 10, and 500 (equivalent to releasing all tasks in the beginning). For each task frequency, we used five different numbers of agents: 10, 20, 30, 40, and 50. We ran the TSP solver for 1,000 seconds for the task-assignment part of the TA algorithms and used the best Hamiltonian cycle found. We used a runtime limit of 5 minutes for each invocation of path planning and report a timeout if the runtime limit was reached. Table 2 reports the makespans and (total) runtimes of the path-planning part (in seconds) for the TA algorithms, and the (total) runtimes (in seconds) for all other MAPD algorithms. The numbers in the last row are averages over instances solved by the algorithm. The smallest makespan and runtime for each scenario are highlighted in bold. The runtimes of the TA algorithms have to be computed as the sums of the runtimes of the task-assignment part (that is, 1,000 seconds), which is not shown in the table, and the path-planning part, which is shown in the table. The table also reports the estimated makespans as provided by the task-assignment part of the TA algorithms.

The **large warehouse** is shown in Figure 4(b). We repeated the previous experiment with the following changes: We generated one sequence of 2,000 pickup and delivery tasks. We released all tasks in the beginning. We used five different numbers of agents: 60, 90, 120, 150, and 180, resulting in larger agent densities than in the small warehouse. We ran the TSP solver for 6,000 seconds. Table 3 reports the results, except that CENTRAL and TA-ICBS are omitted since they timed out.

We first evaluate our innovations: With regard to **task assignment**, the TA algorithms, that use our task-assignment method that assigns a sequence of tasks to each agent instead of only the next task, typically produce solutions with smaller makespans than

Table 1: MAPD Algorithms.

	Offline?	Task Assignment	Path Planning (at each time step except for TA-Prioritized, that plans paths only once)	Deadlock Avoidance
CENTRAL	online	next task only (Hungarian algorithm) before path planning	MAPF-based (CBS) for two agent groups (1 for all free and 1 for all new task agents)	holding task endpoints
GREEDY$_1$	offline	next task only accomplished during/by AMAPF-based path planning	mix of MAPF-based (ICBS) for 1 agent group (for all new task agents) and AMAPF-based (min-cost max-flow algorithm) for 1 agent group (for all free agents)	holding task endpoints
GREEDY$_2$	offline	next task only accomplished during/by AMAPF-based path planning	mix of MAPF-based (ICBS) for 1 agent group (for all new task agents) and AMAPF-based (min-cost max-flow algorithm) for 1 agent group (for all free agents)	reserving dummy paths
TA-ICBS	offline	task sequence (TSP solver) before path planning	MAPF-based (ICBS) for 1 agent group (for all agents)	none necessary
TA-Prioritized	offline	task sequence (TSP solver) before path planning	MAPF-based (prioritized planning) for M agent groups (for 1 agent each)	reserving dummy paths
TA-Hybrid	offline	task sequence (TSP solver) before path planning but task sequences can be changed by AMAPF-based path planning	mix of MAPF-based (ICBS) for 1 agent group (for all new task agents) and AMAPF-based (min-cost max-flow algorithm) for several agent groups (that form a partition of all free agents)	reserving dummy paths

Table 2: Results in the small warehouse. "f" stands for the task frequency, "mkspn" for the makespan, and "time" for the runtime.

		CENTRAL		GREEDY$_1$		GREEDY$_2$		TA-ICBS		TA-Prioritized		TA-Hybrid		TSP
f	agents	mkspn	time	mkspn	time	mkspn	time	mkspn	time	mkspn	time	mkspn	time	mkspn
1	10	1155	**51**	1132	95	1129	199	**1079**	33	1094	10	1087	13	1062
	20	661	122	690	252	648	**102**	**603**	25	608	21	612	38	590
	30	553	180	552	582	533	**178**	timeout		546	35	**528**	118	525
	40	555	482	699	1347	**525**	**336**	timeout		534	44	**525**	182	525
	50	553	945	584	2513	526	**537**	timeout		540	58	**525**	727	525
2	10	1129	59	1119	**31**	1090	180	**1044**	14	1056	10	1048	10	1028
	20	613	190	618	**58**	594	86	**552**	152	569	20	561	23	534
	30	449	214	455	97	419	**44**	timeout		394	29	**385**	38	369
	40	366	346	401	147	347	**63**	timeout		328	39	**323**	94	308
	50	335	423	369	232	303	**88**	timeout		327	44	**300**	130	300
5	10	1117	110	1120	**17**	1090	180	**1039**	15	1054	10	**1039**	10	1020
	20	603	281	602	**23**	585	80	**539**	20	551	19	549	19	519
	30	424	257	436	**30**	422	40	timeout		**370**	29	377	21	345
	40	332	345	354	**40**	320	49	timeout		289	41	**285**	31	268
	50	313	506	317	**51**	284	52	timeout		244	48	**241**	57	225
10	10	1130	64	1114	**12**	1080	174	**1034**	14	1036	10	1045	10	1017
	20	589	131	594	**15**	592	75	**525**	253	559	19	541	17	512
	30	422	221	424	**20**	402	42	**361**	453	369	19	373	21	343
	40	344	356	338	**25**	336	42	timeout		294	40	**279**	29	261
	50	301	1056	298	**29**	272	42	timeout		**236**	50	238	41	213
500	10	1101	50	1094	**10**	1088	174	**1033**	14	1045	10	1037	11	1016
	20	580	124	591	**13**	580	63	timeout		535	19	539	14	508
	30	421	720	429	**17**	408	28	timeout		370	29	**362**	21	338
	40	357	976	337	**20**	314	35	timeout		**275**	39	280	22	254
	50	timeout		283	**21**	266	283	timeout		235	50	**231**	28	211
average		600	342	598	228	566	**127**			538	30	**532**	69	513

Table 3: Results in the large warehouse.

	GREEDY$_1$		GREEDY$_2$		TA-Prioritized		TA-Hybrid		TSP
agents	mkspn	time	mkspn	time	mkspn	time	mkspn	time	mkspn
60	1044	**173**	1007	875	1045	507	**991**	500	847
90	803	**323**	746	450	721	789	**699**	637	576
120	650	**372**	598	476	578	1098	**556**	1091	455
150	764	1126	505	**612**	524	1317	**479**	1803	409
180	777	5053	452	**608**	475	1683	**419**	2457	368
average	808	1409	662	**604**	669	1079	**629**	1298	531

the other MAPD algorithms with their greedier task-assignment methods. With regard to **path planning**, both TA-Prioritized and TA-Hybrid, that use our faster path-planning methods, scale better than CENTRAL and TA-ICBS with their slower path-planning methods. With regard to **deadlock avoidance**, GREEDY$_2$, that uses "reserving dummy paths" as our deadlock-avoidance method, typically produces solutions with smaller makespans than GREEDY$_1$ with its existing "holding task endpoints" deadlock-avoidance method. Both TA-Prioritized and TA-Hybrid combine all three innovations but use different path-planning methods.

We now evaluate all MAPD algorithms with respect to makespan and runtime: With regard to **makespan**, the TA algorithms produce solutions with smaller makespans than the other MAPD algorithms due to their better task-assignment method. TA-ICBS appears to the best, followed by TA-Hybrid in the large warehouse. With regard to **runtime**, the GREEDY algorithms, that use faster path-planning methods, tend to run faster than CENTRAL with its slower path-planning method. CENTRAL does not scale to the large warehouse. The TA algorithms incur additional runtime over the other MAPD algorithms for assigning a sequence of tasks to each agent instead of only the next task, which requires solving a special TSP. However, larger runtimes are justified for offline MAPD problems compared to online ones, and faster TSP solvers that apply to our special TSP can be used to reduce the runtimes. TA-Prioritized and TA-Hybrid, that use faster path-planning methods, scale better than CENTRAL with its slower path-planning method and even better than TA-ICBS with its even slower path-planning method. TA-Prioritized and TA-Hybrid produce solution of slightly larger makespans than TA-ICBS (even though all of them use the same task-assignment method) since their faster path-planning methods result in slightly longer paths.

7 CONCLUSION

In this paper, we studied the Multi-Agent Pickup-and-Delivery (MAPD) problem and introduced two offline MAPD algorithms, TA-Prioritized and TA-Hybrid, that improve on the existing online MAPD algorithm CENTRAL for the offline MAPD problem. With respect to task assignment, CENTRAL always greedily assigns only the next task to an agent. Our MAPD algorithms, on the other hand, compute one task sequence per agent by solving a special traveling salesman problem. With respect to path planning, CENTRAL uses slow MAPF-based path planning. Our MAPD algorithms, on the other hand, use faster path planning to scale better and, in case of TA-Hybrid, can even improve the task sequences. With respect to deadlock avoidance, CENTRAL uses "holding task endpoints". Our MAPD algorithms, on the other hand, use "reserving dummy paths". Overall, they produce solutions of smaller makespans and scale better than CENTRAL in simulated warehouses with hundreds of robots and thousands of tasks. In the future, we may extend them to more real-world scenarios. For example, they apply with minor adaptations to the case where agents start at different time steps. We may also consider agents with different velocities and tasks with different deadlines. Finally, we may use their ideas to improve MAPD algorithms for the online MAPD problem.

REFERENCES

[1] T. Bektas. 2006. The Multiple Traveling Salesman Problem: an Overview of Formulations and Solution Procedures. *Omega* 34, 3 (2006), 209–219.

[2] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. E. Shimony. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *IJCAI*. 740–746.

[3] R. W. Calvo and A. Colorni. 2007. An Effective and Fast Heuristic for the Dial-a-Ride Problem. *4OR* 5, 1 (2007), 61–73.

[4] A. Das, S. Gollapudi, A. Kim, D. Panigrahi, and C. Swamy. 2018. Minimizing Latency in Online Ride and Delivery Services. In *WWW*. 379–388.

[5] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller. 2013. A General Formal Framework for Pathfinding Problems with Multiple Agents. In *AAAI*. 290–296.

[6] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. R. Sturtevant, G. Wagner, and P. Surynek. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *SoCS*. 29–37.

[7] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research* 50 (2014), 141–187.

[8] K. Helsgaun. 2017. *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems*. Technical Report. Roskilde University.

[9] W. Hönig, T. K. S. Kumar, H. Ma, N. Ayanian, and S. Koenig. 2016. Formation Change for Robot Groups in Occluded Environments. In *IROS*. 4836–4842.

[10] R. Luna and K. E. Bekris. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *IJCAI*. 294–300.

[11] H. Ma and S. Koenig. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *AAMAS*. 1144–1152.

[12] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. K. S. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon. 2016. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

[13] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *AAMAS*. 837–845.

[14] H. Ma, C. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI*. 3166–3173.

[15] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig. 2017. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *AIIDE*. 270–272.

[16] P. MacAlpine, E. Price, and P. Stone. 2014. SCRAM: Scalable Collision-Avoiding Role Assignment with Minimal-Makespan for Formational Positioning. In *AAMAS*. 1463–1464.

[17] R. Morris, C. Pasareanu, K. Luckow, W. Malik, H. Ma, S. Kumar, and S. Koenig. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*.

[18] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh. 2017. Generalized Target Assignment and Path Finding using Answer Set Programming. In *IJCAI*. 1216–1223.

[19] E. Nunes, M. Manner, H. Mitiche, and M. Gini. 2017. A Taxonomy for Task Allocation Problems with Temporal and Ordering Constraints. *Robotics and Autonomous Systems* 90 (2017), 55–70.

[20] E. Osaba, F. Diaz, E. Onieva, P. López-García, R. Carballedo, and A. Perallos. 2015. A Parallel Meta-Heuristic for Solving a Multiple Asymmetric Traveling Salesman Problem with Simulateneous Pickup and Delivery Modeling Demand Responsive Transport Problems. In *International Conference on Hybrid Artificial Intelligence Systems*. Springer, 557–567.

[21] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[22] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. 2013. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 195 (2013), 470–495.

[23] T. S. Standley. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*. 173–178.

[24] A. Stentz, M B. Dias, R. M. Zlot, and N. Kalra. 2004. Market-Based Approaches for Coordination of Multi-Robot Teams at Different Granularities of Interaction. In *International Conference on Robotics and Remote Systems for Hazardous Environments*.

[25] N. R. Sturtevant and M. Buro. 2006. Improving Collaborative Pathfinding Using Map Abstraction. In *AIIDE*. 80–85.

[26] P. Surynek. 2015. Reduced Time-Expansion Graphs and Goal Decomposition for Solving Cooperative Path Finding Sub-Optimally. In *IJCAI*. 1916–1922.

[27] J. P. Van den Berg and M. H. Overmars. 2005. Prioritized Motion Planning for Multiple Robots. In *IROS*. 430–435.

[28] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *IJCAI*. 4423–4429.

[29] G. Wagner and H. Choset. 2015. Subdimensional Expansion for Multirobot Path Planning. *Artificial Intelligence* 219 (2015), 1–24.

[30] K. Wang and A. Botea. 2011. MAPP: A Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research* 42 (2011), 55–90.

[31] P. R. Wurman, R. D'Andrea, and M. Mountz. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* 29, 1 (2008), 9–20.

[32] S. Yoon and J. Kim. 2017. Efficient Multi-Agent Task Allocation for Collaborative Route Planning with Multiple Unmanned Vehicles. *IFAC-PapersOnLine* 50, 1 (2017), 3580–3585.

[33] J. Yu and S. M. LaValle. 2013. Multi-Agent Path Planning and Network Flow. In *WAFR*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus (Eds.). Vol. 86. Springer, 157–173.

[34] J. Yu and S. M. LaValle. 2013. Planning Optimal Paths for Multiple Robots on Graphs. In *ICRA*. 3612–3617.

[35] J. Yu and S. M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*. 1444–1449.