POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Planning Paths for Multi-Agent Pickup and Delivery with Velocity Control

Laurea Magistrale in Automation and Control Engineering - Ingegneria dell'Automazione

Author: Filippo Cipollone

Advisor: Prof. Francesco Amigoni

Co-advisor: Dr. Benedetta Flammini

Academic year: 2024-2025

1. Introduction

Multi-Agent Pickup and Delivery (MAPD) [5] addresses the problem of a set of agents engaged in planning collision-free paths in a shared environment, while handling a dynamic set of incoming tasks that require them to navigate from a pickup location to a delivery location. MAPD, that is a life-long version of Multi-Agent Path Finding (MAPF) [10], faces the task assignment problem, in addition to the path planning that must be managed accordingly. Despite the enormous range of applications, solving the combination of these two sub-problems may result extremely challenging. Nevertheless, the literature provides a variety of sound and efficient solutions for a large span of real-life contexts. The kinematics and dynamics of the multi-agent system, however, are still a poorly addressed matter, both in MAPF and MAPD contexts. In particular, the majority of MAPD solutions assumes that agents perform actions at a predefined constant speed [4]. From a practical perspective, this represents a considerable drawback in relation to the stability of the agent physical system. Allowing a direct control on agents' velocity during the planning phase supports the solving of this limitation, besides characterizing the planner with flexibility and a wider range of solutions. This work provides a renewed formulation of MAPD in relation to the kinematics and dynamics of the agent, so as to properly connect a velocity selection logic to the problem. Along with this, we propose an extension of a well-known MAPF planner, namely Safe Interval Path Planning (SIPP), that integrates the velocity control in the path planning phase, which is eventually embedded into a task assignment and agents coordination method, that is Token Passing (TP). Experimental results show an overall good performance with respect to tasks' completion times and the relevance of this new approach in particular environmental circumstances.

2. State of the Art

This section provides the formulation of both MAPF and its life-long extension MAPD, alongside a brief overview of some main solvers for both categories.

2.1. MAPF

A classical MAPF problem [10] is defined by means of an undirected graph $G = (V, E)$ and a set of k agents, each of whom is assigned a source vertex $s_i \in V$ and a target vertex $g_i \in V$. Moving in a discrete context, agents perform actions that are defined as functions $a : V \to V$, usually distinguished between *wait* and *move*. At every time step each agent occupies one of the graph vertices and can perform an action. A single-agent plan π_i for an agent i is a finite sequence of actions $(a_1, a_2, ..., a_n)$ that, executed from s_i, leads to g_i. We denote by $\pi_i[x]$ the location of agent i after executing x actions in π_i. A *valid solution* to a MAPF problem is a set of k single-agent collision-free plans. Among the main kind of collisions, two of them are presented:

- **Vertex conflict**: it occurs between two paths π_i and π_j when agents i and j plan to occupy the same vertex at the same time step, i.e., $\exists x \mid \pi_i[x] = \pi_j[x]$.
- **Edge conflict**: it occurs between two paths π_i and π_j when agents i and j plan to traverse the same edge at the same time step, i.e., $\exists x \mid \pi_i[x] = \pi_j[x+1] \ \wedge \ \pi_j[x] = \pi_i[x+1]$.

The two common metrics used to assess the quality of a MAPF solution are:

- **Makespan**: the number of time steps required for all agents to perform their collision-free plans.
- **Sum of Costs**: the sum of time steps for every agent to reach their target.

Among the numerous MAPF solvers, it is worth naming Conflict-Based Search (CBS) [7], an algorithm that develops through a conflict tree (CT) built on conflicts between agents, and Increasing Cost Tree Search (ICTS) [8], which instead relies on a multi-value decision diagram to perform the search. Another important algorithm, that is similar for some aspects to our problem, is Cooperative A* [9]: this planner uses space-time A* to compute collision-free paths, one agent at a time, considering the others that have already planned.

2.2. MAPD

Our work focuses on Multi-Agent Pickup and Delivery [5], an extension of MAPF, where agents have to execute a dynamic set of tasks, each one consisting in moving from a *pickup* location to a *delivery* location, while avoiding collisions. An instance consists of a set of k agents and an undirected graph $G = (V, E)$, in addition to a set $\mathcal{T}$ of dynamic tasks, each of them characterized by a pickup location $s_i \in V$ and a delivery location $g_i \in V$. Such tasks are not known a priori, but are spawned in an online fashion. A *free* agent, i.e., an agent not engaged in executing any task, is assigned one from $\mathcal{T}$ and becomes *occupied*.
In order to solve a MAPD problem, agents have to plan collision-free paths while also addressing the task assignment issue. Given the *service time*, which is a common evaluation metric for MAPD, defined as the average number of time steps required for a task to be accomplished since its appearance in $\mathcal{T}$, a MAPD problem is *solved* iff the resulting service time (or, equivalently, the resulting makespan) of all tasks is bounded [3].

2.2.1 Well-Formedness

A MAPD problem is solvable as long as the instance is *well-formed*. Let us consider an *endpoint* as a location where agents do not constitute an obstacle of any kind [5], and, given V_{ep} be the set of all endpoints and V_{tsk} the set of task endpoints, i.e., those related to the pickup and delivery locations, one denotes V_{ep}/V_{tsk} the set of non-task endpoints, where agents can idle without blocking the others. A MAPD instance is well-formed iff the following hold:

- the number of tasks is finite.
- there are no fewer non-task endpoints than the number of agents.
- for any two endpoints, there exists a path between them that traverses no other endpoint.

Algorithms for solving the MAPD problem can be divided into two main categories, as proposed in the work [5]. *Centralized* solvers, like CENTRAL, plan for multiple agents at once, using global information, in opposition to *decentralized* algorithms, like Token Passing (TP) and its evolution Token Passing with Task Swaps (TPTS), which instead rely on one-at-a-time distributed decisions. Although centralized methods deliver better results in terms of makespan and service time, they are computationally heavy.

2.3. Token Passing

With Token Passing [5], agents make use of a shared block of memory, called *token*, upon which they base their planning. This element contains all the current paths of all agents, as well as the dynamic set $\mathcal{T}$.
After the initialization of the token, where all the stored paths correspond to the initial location of the agents, the system sends the token to all the free agents requesting it. In the meanwhile, at each time step, all new tasks are incorporated in the task set $\mathcal{T}$. The agent with the token can assign itself a task whose pickup vertex is closest to its current location, as long as no other agent has already planned to end at the pickup or the delivery location of such task. The agent thus computes a collision-free path according to the assigned task, returns the token to the system and eventually moves a step forward, according to the stored plan. Whether a solution cannot be found, the agent either stays at its current location or moves to a non-task endpoint.

2.4. Introducing Velocity Control

The main feature of the thesis consists of the ability of agents to plan their paths along with a velocity selection strategy. Previous work has already addressed velocity control in multi-agent systems. In particular, the *Velocity Obstacle Paradigm* [1] has set the start to velocity control in continuous trajectory planning, allowing a reactive run-time approach in response to dynamic obstacles. What we propose is a different approach that performs the velocity selection in parallel to the path planning. MAPF and MAPD still have not addressed kinematics in the way we intend: solutions that take into account the dynamics of the system have been developed, such as MAPF-POST [2], in which velocity control is performed after the planning, or TP-SIPPwRT [4], where agents still have no option to choose their travel speed along the planned paths. This feature is the main contribution of our work.

3. Problem Formulation

This section provides a new formulation of the MAPD problem aligned with the purpose of our work. A physical model for the robot-load system is presented, and a new definition of task is given, in relation to the dynamics of the system.

3.1. MAPD with Velocities

We address the problem of MAPD with velocities, a MAPD problem where the path planning considers the agents' speed, by means of a set of default velocities, that each agent can engage at every time step. An instance shares the same characteristics as in classical MAPD, with the addition of this set of speeds, as well as a new value that characterizes the task, namely a_{max} which supports the velocity selection.
A set of k agents moves in a grid-like workspace, composed of squared cells of size $L \times L$, where L is also considered as the distance covered by an agent when moving between two adjacent cells. The environment presents both static obstacles, such as walls or shelves, and dynamic obstacles, i.e., the agents themselves, and it is assumed to be *well-formed*.
A set of default speeds is at the disposal of agents, by means of which they shape the planning according to their current environmental circumstances. Such speeds are assumed to stay constant during the whole time step, implying that an infinitely fast acceleration (or deceleration) occurs as soon as the agent makes the choice. In order to conform to the discrete domain in which the algorithm is developed, velocities assume fractional values of the distance L. Therefore, a typical set of available velocities is defined as $V_{def} = \{L/2,\ L/3,\ L/4, ...\}$.
An essential difference with respect to classical MAPD is the absence of the *wait* action: in view of this, at the occurrence of a moving obstacle, agents can either change velocity, overcoming such obstacle faster or slower, or, if no change is feasible, they directly plan an alternative path around it.

3.2. Task Characteristics and Robot-Load Model

In MAPD, agents reach pickup locations to collect loads and bring them to the delivery locations. Besides the usual pickup-delivery pairs, tasks are characterized by a_{max}, a new parameter that supports the velocity selection strategy. This value, which is the maximum acceleration that the agent can withstand along the task when changing velocity, sets a limit that guarantees the stability of the robot-load system. This acceleration is fictitious: as stated in Section 3.1, changes in speed are simultaneous.

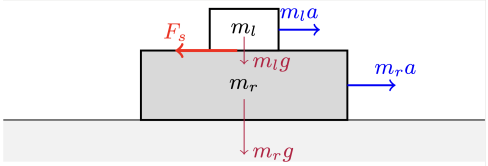

Figure 1: Robot-Load system model.

Figure 1 represents an abstract physical model of a robot carrying a load in a MAPD context. A wheeled mobile robot of mass m_r, equipped with a stable holding device, is assigned the task to carry a load m_l from a pickup to a delivery location. Assuming that both bodies undergo the same acceleration, and that friction only occurs between them, the following calculations lead to the value of a_{max}.

$$m_r a + m_l a - F_s = 0 \quad (1)$$

$$m_r a + m_l a - \mu_s m_l g = 0 \quad (2)$$

$$a_{max} = \frac{m_l}{m_r + m_l} \mu_s g \quad (3)$$

Classical mechanical equilibrium (Equation 1) leads to the computation of a_{max}, values μ_s and g (Equation 2) are the friction coefficient and the gravity acceleration, respectively, and a_{max} (Equation 3) is the eventually computed result. Apart from kinematics, the rest of our problem follows the same pattern as in MAPD (refer to Section 2.2).

4. Proposed Solution

The following section presents our solution, namely Safe Interval Path Planning with Velocity Control (SIPPwVC), eventually integrated in Token Passing to solve the overall MAPD problem. The planner is an extension of Safe Interval Path Planning (SIPP) [6], a MAPF graph-based search strategy.

4.1. Safe Interval Path Planning

Safe Interval Path Planning (SIPP) is a search method that addresses navigation in dynamic environments without explicitly searching the full space-time grid. For each spatial *configuration*, e.g., a cell location and possibly an orientation, it computes sets of *safe intervals*, defined as contiguous periods of time during which no collision occurs in that configuration. The planning is performed with an A* search over states identified by the configuration and the related safe interval, while the earliest feasible arrival time within that interval is stored as an associated value. When expanding a state, SIPP considers feasible motions to neighbor configurations and determines whether the agent can wait (if needed) and then move, so as to arrive inside one of the neighbor's safe intervals without collisions along the motion.

Before going further, two essential characteristics demand consideration. Firstly, the earliest arrival time calculated during the search is treated as a dependent value stored with the state, but it does not identify it: two nodes with the same configuration and safe interval are considered as the same state by the planner, even if they are reached at different times, and SIPP keeps only the earliest reachable arrival time for that configuration-safe interval pair.

Secondly, SIPP considers *"wait and move"* actions, implying that, for each safe interval in a new configuration, the agent waits the minimum amount of time possible, so that when the move is executed, it arrives in the new safe interval as early as possible.

4.2. Safe Interval Path Planning with Velocity Control

We now introduce the novelty of our solution, by extending SIPP to velocity control. The observations made in the previous section are revisited in order to obtain a coherent model for our purpose. The arrival time, previously extraneous to the state identification, now has a crucial role, since it is strictly related to the speed selection logic. Indeed, an agent can arrive within the same safe interval of a neighbor configuration, traveling at two different speeds when possible: two different arrival times are computed, dependent on such speeds, and thus two new nodes are generated.

Moreover, the *wait* action is not considered: the possibility of virtually accelerating and decelerating makes this feature obsolete and removes the risk of damaging the mechanical stability of the robot-load system.

Safe intervals and velocities are features that belong to the cell and the agent, respectively, therefore the search is performed along with two parallel states: at time step t, the *cell state* is defined as $s = (v,\ SI_v,\ si(t),\ t)$, where v is the spatial configuration, SI_v is the set of safe inter-

vals for that configuration and $si(t) \in SI_v$ is the safe interval that contains time step t, whereas the *agent state* $a = (v,\ vel,\ t)$ is characterized by the velocity vel by which the agent arrives at configuration v at time step t.
As well as in SIPP, our solution exploits A* for the search, following the same steps as presented in the work [6]: the expansion occurs by means of a new function, that we call *extendedGS*, which generates successor nodes while considering the velocity control logic. The pseudo-code for this function is shown in Algorithm 1.

Algorithm 1 extendedGS

1: // current states are $s = (c,\ SI_c,\ si(t_{curr}),\ t_{curr})$ and $a = (c,\ v_{curr},\ t_{curr})$
2: $successors = \emptyset$
3: $V_{admissible} = \emptyset$
4: **for** each vel in V_{def} **do**
5: a_{mean} = acceleration needed to change the current speed to vel
6: **if** $a_{mean} < a_{max}$ **then**
7: $V_{admissible} \leftarrow vel$
8: **else**
9: **continue**
10: **for** n in $neighbors$ **do**
11: **for** v in $V_{admissible}$ **do**
12: $t_m = L/v$
13: $t_{start} = t_{curr} + t_m$
14: $t_{end} = endTime(si(t_{curr})) + t_m$
15: **for** each safe interval i in SI_n **do**
16: **if** $t_{start} > endTime(i)$ **or** $t_{end} < startTime(i)$ **then**
17: **continue**
18: **if** $(t_{start} - 1) \geq startTime(si(t_{curr}))$ **and** $(t_{start} - 1) \leq endTime(si(t_{curr}))$ **then**
19: $t = t_{start}$
20: **else**
21: **continue**
22: $s' \leftarrow$ new *cell* state with i and t
23: $a' \leftarrow$ new *agent* state with v and t
24: **return** $successors$

Before going through the selection mechanism, the function computes the set of admissible velocities $V_{admissible}$ upon which the selection is based, taking into account the task parameter a_{max} and the fictitious acceleration a_{mean} that the agent would perform to engage a given velocity in V_{def}. After that, for each of the admissible velocities, the function calculates t_m, which is the number of time steps required to reach a neighbor configuration with that candidate velocity v, and a new time interval $[t_{start}, t_{end}]$ is computed as well. If such new interval fits in a safe interval of the neighbor cell and the agent leaves its current configuration in time, while moving at the candidate velocity, then no conflict will occur. Finally the new successor states s' and a' are generated, with updated stored values.
Another subtle difference with respect to the original SIPP is the formulation of the *f-value* (Equation 4) that leads the search.

$$f(s) = g(s) + \frac{h(s)}{v_{max}} \tag{4}$$

In our case $f(s)$ is a pure time function: $g(s)$ is intended as the number of time steps required to reach state s, whereas the second term is the ratio between $h(s)$, i.e., the *Manhattan distance* from the current configuration to the goal configuration, and v_{max}, which is the highest velocity in V_{def} and gives the best possible estimate to reach the goal state.

4.3. TP-SIPPwVC

The final step of the solution is the integration of SIPPwVC in a MAPD algorithm. In particular, the path planning solver is embedded into Token Passing. TP can replace the original space-time A* with any other one-shot single-agent path planning algorithm, as long as it provides collision-free paths in relation to the ones stored in the token. As such, our planner SIPPwVC, which is guaranteed to provide a collision-free solution, opportunely fits into the TP algorithm.

5. Experiments

5.1. Experimental Setting

The algorithm is tested by running experiments in modular environments of increasing size, with units composed of 4-connected 20×20 cell grids, as illustrated in Figure 2. The compound environments, of total sizes 40×40, 60×40, and 60×60, simulate real-life workspaces, where agents are located along the lateral walls and static obstacles, e.g., shelves, are depicted as black cells. Pickup locations are situated in front

of each side of the shelves, and delivery locations are positioned between the agents rows.

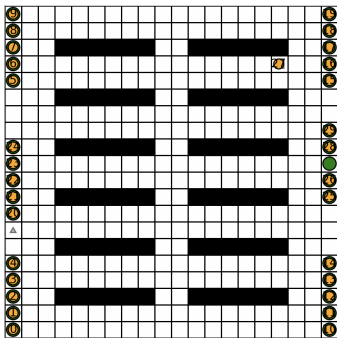

Figure 2: Map module of a *warehouse* with 30 agents.

The online feature of the MAPD task assignment is elaborated through a Poisson distribution of the tasks with parameter $\lambda = 1.2$ while the sets of fractional velocities for performance comparison are $V_{def} = \{L/2\}$, $V_{def} = \{L/6, L/2\}$, and $V_{def} = \{L/10, L/6, L/2\}$, selected so as to observe the impact of additional speeds to each preexistent set.
Experiments follow the two lines of organization illustrated in Table 1: we test three environments with a fixed number of task, allocated to varying sets of agents, and vice versa. All combinations provide results for all three sets of V_{def} over 20 different runs, each one with different pickup-delivery pairs and task characteristics.

	M/T		**Agents**			
Fixed Tasks	40x40	120	10	20	30	40
	60x40	160	10	20	30	40
	60x60	200	30	40	50	60
	M/A		**Tasks**			
Fixed Agents	40x40	20	60	100	140	180
	60x40	40	80	120	160	200
	60x60	60	120	160	200	240

Table 1: Table of experimental settings.

The metrics are the *makespan*, *service time*, (Section 2.2), and the *execution time*, i.e., the average number of time steps required to complete a task since its allocation; we also measure the *frequency in change of speed*, which is the average number of times that an agent changes velocity along the whole path. In this summary, we considered sufficient to provide illustrative

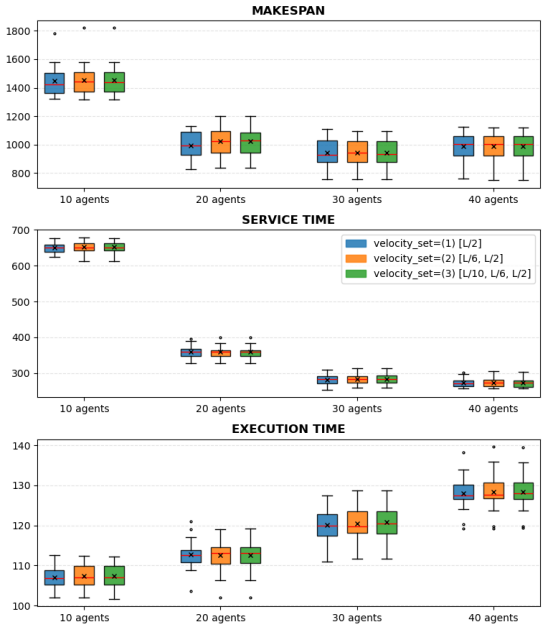

Figure 3: Map 40×40 with 120 fixed tasks and varying sets of agents.

results for all configurations in Table 1, only in relation to the map of size 40×40, since the other settings delivers similar trends.

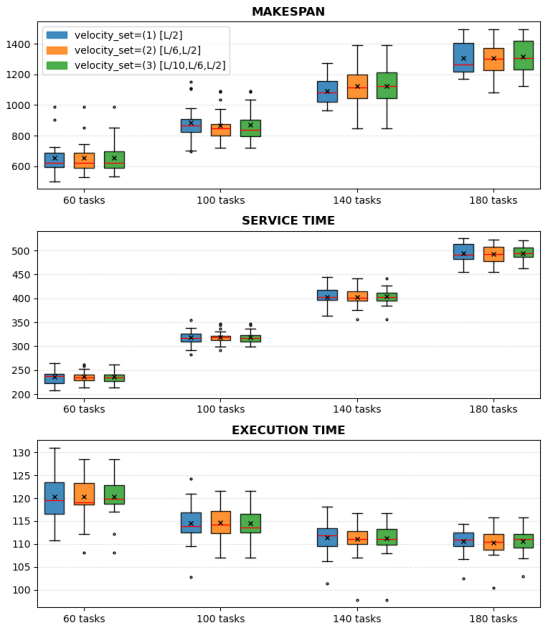

Figure 4: Map 40×40 with 20 fixed agents and varying sets of tasks.

5.2. Makespan, Service Time and Execution Time

Results confirm what is expected in relation to the tested environments and their settings.

As for the category of fixed tasks with varying agents (Figure 3), we observe a decreasing trend of the average makespan and service time (these two metrics are indeed strictly related): as an increasing number of agents is involved in the instance, the fixed amount of tasks is accordingly allocated, resulting in shorter values of makespan and service time. The average execution time increases as a result of a more crowded environment.
On the other hand, if the number of agents is fixed while we increase the number of tasks (Figure 4), we get a specular situation: the average makespan and service time increases since a growing number of tasks is assigned to the same set of agents, while the average execution time follows a decreasing trend.

5.3. Frequency of Speed Changes

The number of occurrences of speed changes for each task is counted from the start to the delivery location, including the pickup location. The average value is calculated over all the assigned tasks in the instance. Such values are eventually gathered, and we analyze the distribution in relation to all instances.

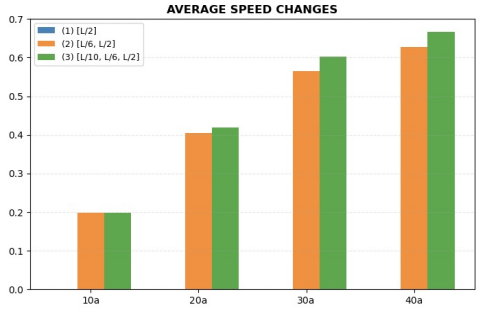

Figure 5: Map 40×40 with 120 fixed tasks and varying sets of agents.

Figure 5 and Figure 6 prove the utility of the velocity selection mechanism. In the first case (Figure 5), given a fixed number of tasks, as we increase the number of agents, we observe that the averages increase in value, implying that the selection mechanism is consistently employed as the environment becomes more crowded. On the other side (Figure 6), if we maintain a fixed number of agents while varying the number of tasks, we observe that the averages stay almost steady; therefore the velocity control logic is proved to be exploited by the majority of agents throughout a single instance.

Figure 6: Map 40×40 with 20 fixed agents and varying sets of tasks.

6. Conclusion and Future Work

6.1. Contribution

This thesis presents the work behind a brand new approach to MAPD. The introduction of a velocity control strategy in support of the path planning provides a flexible algorithm that delivers positive global results and that highlights the benefits of allowing a multi-agent system to manage the travel speed of the agents along with the planning. In-depth analysis of experimental results of the instances also shows the practicality of SIPPwVC in relatively crowded environments, as much as it proves adaptability and scalability in large environments. Moreover, the possibility to handle the kinematics addresses the safety issue of the robot-load system, that can be eventually properly managed in relation to its mechanical stability.

6.2. Future developments

The proposed formulation of the problem serves as a starting model for further development, in line with increasingly complex scenarios. Improvements in relation to the spatial configuration of the agents, e.g. adding a rotational variable or a third coordinate, as well as to the reintroduction of a properly adapted definition of the *wait* action can be addressed, along with the adaptability to a diverse range of environments, such as maze-like workspaces or randomly crowded open spaces.

References

[1] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[2] Wolfgang Hönig, TK Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 477–485, 2016.

[3] Hang Ma. *Target assignment and path planning for navigation tasks with teams of agents*. PhD thesis, University of Southern California, 2020.

[4] Hang Ma, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7651–7658, 2019.

[5] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, pages 837–845, 2017.

[6] Mike Phillips and Maxim Likhachev. SIPP: Safe interval path planning for dynamic environments. In *Proceedings of the International Conference on Robotics and Automation*, pages 5628–5635, 2011.

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

[8] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 2013.

[9] David Silver. Cooperative pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122, 2005.

[10] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, pages 151–158, 2019.