Multiagent Pickup and Delivery for Capacitated Agents

Evren Çilden(✉) and Faruk Polat

Middle East Technical University, 06800 Ankara, Turkey
{evren.cilden,polat}@ceng.metu.edu.tr

Abstract. In Multi-Agent Pickup and Delivery (MAPD), multiple robots continuously receive tasks to pick up packages and deliver them to predefined destinations in an automated warehouse. If the capacity of agents is increased, agents can pick up more than one item on their way, which will presumably reduce the time required to accomplish all deliveries–that is, makespan. In this paper, we propose two algorithms for MAPD with Capacities (MAPDC) that are *complete* and scalable: Token Passing with Multiple Task Assignments (TPMT) and Token Passing with Multiple Capacity (TPMC). Both of the methods are based on the Token Passing (TP) algorithm, one of the *suboptimal* and complete solutions by Ma et al. [6]. The performance of the algorithms is analyzed in terms of makespan, service time, and throughput. TPMC turns out to be more effective than TPMT at utilizing capacitated agents.

Keywords: Multiagent Pickup and Delivery (MAPD) · MAPD with Capacities (MAPDC) · Warehouse automation

1 Introduction

A current trend in logistics is automated warehouses, where robots operate to store and retrieve objects at fulfillment/distribution centers (Fig. 1). A problem originating from this real-world domain is Multi-Agent Pickup and Delivery (MAPD) [6]. The aim is to perform some delivery tasks with a fixed number of homogeneous and autonomous agents, where a new task can enter the system at any time. When an agent is assigned to a task, it moves to the pickup location, takes the item, and carries it to the delivery point, while planning non-conflicting paths with other agents.

As inventory pods allocate most of the space at automated warehouses, robots navigate through narrow paths and occasionally need to avoid collisions with other agents. Hence, MAPD is closely related to the classical NP-hard problem of multi-agent path-finding (MAPF), where each agent located in a grid environment is assigned a destination position (the robots operate in an environment by following bar-code stickers on the floor provided for discretization of the navigation space, which we can model as a grid-environment). The solver computes the location of agents at each time step so that all agents reach

F. Dignum et al. (Eds.): PAAMS 2022, LNAI 13616, pp. 76–87, 2022.
https://doi.org/10.1007/978-3-031-18192-4_7

their destinations without colliding with obstacles or other agents. MAPD is the lifelong version of MAPF [6] where new tasks can enter the system at any time. Agents are assigned new targets, which better suits the domain of automated warehouses. Two interdependent concerns of MAPD, task assignment and collision-free path-finding, make the problem even more challenging than MAPF. There is a search for online and efficient solvers for MAPD that can plan for hundreds of agents.

(a) Amazon warehouse robot [9]

(b) HAI Pick System [1]

Fig. 1. Robots operating at fully automated warehouse environments.

In real life, there are cases where agents are capable of carrying multiple items (Fig. 1(b)). We can generalize MAPD for capacitated agents so that they can pick up or deliver items on their way to reduce the average time for the completion of a series of tasks. Multiple capacity was first considered by Chen et al. [1], exploring a coupled solution of task assignment and path-finding called Regret Based Marginal Cost Assignment (RMCA). Although they obtained a valuable decrease in makespan by employing capacitation to RMCA, they noted the completeness of the algorithm needs improvement [1]. Later, Tajelipirbazari et al. [10] named the capacitated variant as MAPDC and devised a solution within the declarative framework of Answer Set Programming (ASP). The algorithm guarantees optimality for makespan but fails to scale to real-world situations. In this study, we provide two complete, scalable yet suboptimal solutions to MAPDC: TPMT and TPMC.

2 Related Work

Most of the existing solutions to MAPD attack the problem sequentially by decoupling assignment and path-finding. They first assign tasks to agents to determine the pickup and put-down points they will travel to. Then they apply MAPF techniques to find a non-colliding path sequence for each agent. A classical problem that is most relevant to the task assignment aspect of MAPD is Vehicle Routing within the problem class of Multi-robot Task Allocation (MRTA) [4,7]. Under predefined constraints, it focuses on making optimal route assignments, assuming that agents do not collide. Multiagent Path Finding (MAPF) is also a well-studied problem that aims at planning a path for each agent while

avoiding collisions with other agents [2]. The optimality criterion is minimizing the sum of paths of all agents. A foundational algorithm used in developing solutions to MAPF problems is A* [3], a complete and optimal algorithm that performs a best-first search on a spanning tree. CBS [8] is an optimal algorithm that keeps track of a high-level search tree of constraints by examining collisions between agents, which is shown to examine fewer states compared to A*.

After its first introduction in 2017 by Ma et al. [6], many solutions to MAPD was proposed. In decentralized solutions, agents individually calculate their paths and then fix collisions. Otherwise a central solver computes paths of all agents. Offline solvers require a priori knowledge of the task list to compute the whole path. Online algorithms provide a solution composed of computation and movement phases.

Token-Passing (TP) [6] is a decentralized online solution based on the idea of tokens. A global token stores the set of unexecuted tasks and paths of agents. As new tasks are added, unoccupied agents select the task with the closest pickup location and take over the token, in turn, to calculate their paths using the A* algorithm. Agents hold task endpoints for deadlock avoidance, which guarantees completeness for well-formed instances [6]. Token-Passing with Task Swaps (TPTS) [6] enhances task assignment of TP by allowing task transfer between agents while the previously assigned agent is on its way for pickup.

CENTRAL [6] is an online algorithm, where a centralized solver assigns tasks by the Hungarian method and calculates paths by CBS. TA-Prioritized [5] is another centralized algorithm that computes a task sequence for each agent, then plans paths by prioritized planning. Regret Based Marginal Cost Assignment (RMCA) [1] integrates task assignment and path planning to optimize the total task delivery time of agents. Different from other approaches, RMCA considers actual delivery costs instead of lower-bound estimates in path planning. RMCA is based on prioritized planning with A* and does not guarantee completeness. Chen et al. executed RMCA with capacitated agents and reported a huge enhancement in makespan [1]. Tajelipirbazari et al. [10] devised a centralized offline solution based on ASP to MAPDC.

3 Problem Description

We define MAPDC formally as:

- a set of m agents $A = \{a_1, a_2, ..., a_m\}$, where each agent has a capacity of carrying at most n items.
- an undirected connected graph $G = (V, E)$, where V is the set of the locations and E is the set of edges connecting the locations.
- a set of tasks $\mathcal{T}$, which are waiting to be executed by an agent (solver's task list). Each task $\tau_j \in \mathcal{T}$ is a tuple (s_j, g_j) of pickup location $s_j \in V$ and delivery location $g_j \in V$.

In this context, MAPD is a special case of MAPDC, where $n = 1$. In MAPDC, each agent has a set of assigned and ongoing tasks T with a size bound n, since

each agent is capable of carrying at most n items. An agent is *under-capacity* when $0 \leq |T| < n$, so that the solver can assign a new task $\tau_j \in \mathcal{T}$ to the agent (i.e. $T \cup \{\tau_j\}$). If the agent's path to the location s_j is planned, the task τ_j is removed from the solver's task list $\mathcal{T}$ to avoid re-assignments. The execution of the task ends when the agent navigates to the location g_j of the task τ_j, and the task is removed from the agent's set of tasks, T.

While executing a task, the agent a_i either waits at its current position or moves to an adjacent node in a single time step ($l_i(t+1)$ is either $l_i(t)$ or $l_i(t+1) \neq l_i(t)$ where $(l_i(t), l_i(t+1)) \in E$). Agents cannot simultaneously be at the same location ($\forall a_i, a_j$ where $a_i \neq a_j$ and $\forall t : l_i(t) \neq l_j(t)$). Two agents cannot simultaneously move in opposite directions along the same edge ($\forall a_i, a_j$ where $a_i \neq a_j$, and $\forall t : l_i(t) \neq l_j(t+1)$ or $l_j(t) \neq l_i(t+1)$).

A valid solution for MAPDC completes the set of tasks in a finite number of timesteps and returns non-colliding paths of agents. For a MAPD problem to be solvable at finite steps, Ma et al. introduced the notion of *well-formedness* and stated conditions for a well-formed MAPD instance [6]. Since MAPDC is an extension of the task-assignment aspect of MAPD, the underlying constraints about path-finding are also applicable. Thus, we can infer that the constraints defined for well-formedness hold for the capacitated variation of the problem as well.

4 Method

TP algorithm, a complete and scalable solution to MAPD, is a promising candidate for devising such a solution to MAPDC. We examined possible enhancements to the TP algorithm and proposed two methods. The first algorithm is Token Passing with Multiple Task Assignments (TPMT). In this method, the agent holding the token can plan a path for up to n tasks, where n is its capacity. In the second method named Token Passing with Multiple Capacity (TPMC), every under-capacity agent holding the token can choose a task before completing its tasks to pick up items on the way.

4.1 TPMT

The pseudo-code for TPMT is available at Algorithm 1. Firstly, agents are initialized to stay at their current locations ($[loc(a_i)]$) (Line 1). New tasks introduced at the current timestep are added to the end of the task set $\mathcal{T}$ (line 3). Solver passes *token* to the next free agent that has reached the end of its path. The agent takes the control of execution after Line 4. The agent selects tasks until its capacity is full ($|T_i| = n$) or there are no available tasks among the task set such that no path of other agents ends in the pickup or delivery location of the task (lines 5 to 12). The selection criterion is finding the tasks with the start location that has the minimum h-value to the current location of the agent (line 9). For efficiency, h-values, that is, path costs from all locations to all endpoints (i.e., possible pickup and delivery locations) are pre-calculated to be used by

Algorithm 1. Token Passing with Multiple Task Assignments (TPMT)

Initialize *token* with path $[loc(a_i)]$ for each agent a_i
while *true* **do**
  Add all new tasks, if any, to the task set $\mathcal{T}$
  while agent a_i exists that requests *token* **do**
    $\mathcal{T}' \leftarrow \{\tau_j \in \mathcal{T} \mid$ no other path in *token* ends in s_j or $g_j\}$
    if $\mathcal{T}' \neq \emptyset$ **then**
      $T_i \leftarrow \emptyset$ ▷ empty task set of a_i
      while $\mathcal{T}' \neq \emptyset$ and $|T_i| \neq n$ **do** ▷ n: capacity of a_i
        $T_i \cup \{\arg\min_{\tau_j \in \mathcal{T}'} \mathrm{h}(loc(a_i), s_j)\}$
        Remove τ_j from $\mathcal{T}'$
      end while
      $P \leftarrow$ GetOrderedPoints(T_i)
      for each $p \in P$ **do**
        Update a_i's path in *token* with Path1($a_i, p, token$)
      end for
      for each $\tau \in T_i$ **do**
        Remove τ from $\mathcal{T}$
      end for
    else if no task $\tau_k \in \mathcal{T}$ exists with $g_k = loc(a_i)$ **then**
      Update a_i's path in *token* with path $[loc(a_i)]$
    else
      Update a_i's path in *token* with Path2(a_i, *token*)
    end if
  end while
end while
function GetOrderedPoints(T_i)
  Add s_1 to P ▷ P is the list of ordered points
  Add g_1 to V ▷ V is the list of points to visit
  for each $j \neq 1$ and $\tau_j \in T_i$ **do**
    Add s_j to V
    Add (s_j, g_j) to C ▷ C is the list of visit constraints
  end for
  while V is not empty **do**
    $l \leftarrow \mathrm{last}(P)$ ▷ l is the last visited point
    $v_k \leftarrow \arg\min_{v_k \in V} \mathrm{h}(l, v_k)$
    Add v_k to P
    if there exists (v_k, g_k) in C **then**
      Add g_k to V
      Remove (v_k, g_k) from C
    end if
    Remove v_k from V
  end while
  return P
end function

task allocation and A* searches. Agent a_i finds an ordered list P of endpoints for all s_j and g_j of $\tau_j \in T_i$ (Line 13). For each point $p \in P$, the agent calculates its path from $[loc(a_i)]$ to p by A* (Path1), and updates the *token* (lines 13–15). After calculating its path for all points in P, the agent removes all tasks in T_i from $\mathcal{T}$ (lines 16–18). If T_i is empty and there is no task that the agent can execute, it checks whether its location is the delivery location of another task. If so, to avoid deadlocks, updates its path to Path2 that is the path to move to an unoccupied endpoint. Otherwise, the agent plans the trivial path to stay at its final destination, $[loc(a_i)]$.

The function GetOrderedPoints(T_i) calculates the visiting order for start and goal points of tasks in T_i. The problem is a special case of the TSP on the directed graph $G_i = (V_i, E_i)$ where V_i contains the start and delivery locations of tasks in T_i. Since there is a path between any two endpoints for every well-formed MAPD problem, the graph is fully connected except for the directed edges from g_j to s_j for every task $\tau_j \in T_i$ (because agents visit the delivery location only after picking up an item). The function keeps track of the visiting constraints and the list of nodes to visit and applies the nearest neighborhood heuristic to choose the point with the minimum h-value to the last visit position. A goal location is not considered as a candidate for the next visit until its pickup location is in the ordered list of points.

4.2 TPMC

TPMT selects the first n tasks that have the nearest pickup locations to the agent at the time of decision. As the agent advances in execution, previously selected tasks can be distant to its updated position, which may have a huge negative impact on the overall makespan. TPMC method aims at making better task assignments in an online fashion by allowing under-capacity agents to choose the next task after visiting a point.

Algorithm 2 presents the outline of the TPMC method. Like TPMT, it is a decoupled algorithm based on TP [6] with the same initialization and agent selection logic. Different from TPMT, the agent keeps a list of points to visit (i.e. V_i). After taking the token, the agent counts the number of items waiting for delivery (i.e. finds the number of delivery points in V_i). If it is under-capacity, the agent decides that it can pick up another item, and if any, chooses an available task τ_j having the minimum h-value for the location pair $(loc(a_i), s_j)$ (Lines 5–8). The constraint that no other path in *token* ends in s_j or g_j is also valid for the task assignment of TPMC. The agent updates the state of the task τ_j as taken and appends s_j to its V_i (Lines 9–10). If V_i is not empty, the agent finds the point p in V_i that has the minimum h-value to its current location and updates its path in the *token* with the path calculated by A* (Lines 20–21). After path-finding, p is removed from V_i (Line 22). If p is a start point of a task τ_k, a_i adds the delivery location g_k to end of V_i (Lines 23–24). If it is a delivery point, τ_k is removed from the task list of *token* (Line 26). If V_i is empty, and if the agent is in the delivery location of a task, it calls function Path2 and updates its path to move to an unoccupied endpoint to avoid deadlocks (Line

17). Otherwise, the agent plans the trivial path to stay in its current location (Line 15).

Algorithm 2. Token Passing with Multiple Capacity (TPMC)

```
1: Initialize token with path [loc(a_i)] for each agent a_i
2: while true do
3:     Add all new tasks, if any, to the task set 𝒯
4:     while agent a_i exists that requests token do
5:         if the number of goal points in V_i < n then   ▷ n is the capacity and V_i is to-Visit list of a_i
6:             𝒯' ← {τ_j ∈ 𝒯 | τ_j is not taken and no other path in token ends in s_j or g_j}
7:             if 𝒯' ≠ ∅ then
8:                 τ_j ← arg min_{τ_j ∈ 𝒯'} h(loc(a_i), s_j)
9:                 Add s_j to V_i
10:                Mark τ_j as taken
11:            end if
12:        end if
13:        if V_i is empty then
14:            if no task τ_k ∈ 𝒯 exists with g_k = loc(a_i) then
15:                Update a_i's path in token with path [loc(a_i)]
16:            else
17:                Update a_i's path in token with Path2(a_i, token)
18:            end if
19:        else
20:            p ← arg min_{v_k ∈ V_i} h(loc(a_i), v_k) and no other path in token ends with v_k
21:            Update a_i's path in token with Path1(a_i, p, token)
22:            Remove p from V_i
23:            if p is the start point of τ_k then
24:                Add g_k to V_i
25:            else
26:                Remove τ_k from 𝒯
27:            end if
28:        end if
29:    end while
30: end while
```

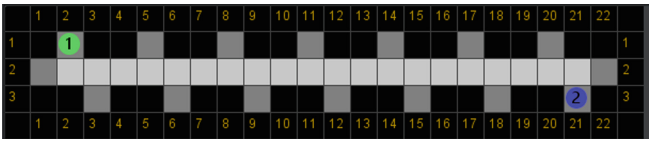


Fig. 2. Narrow domain for case studies.

Table 1. Task list for case studies

Task number	Pickup → Delivery
1	(5, 1) → (20, 1)
2	(8, 1) → (17, 1)
3	(11, 1) → (14, 1)
4	(18, 3) → (3, 3)
5	(15, 3) → (6, 3)
6	(12, 3) → (9, 3)

5 Evaluation

5.1 Case Studies

Two case studies were performed on the narrow domain (Fig. 2), where the black and gray cells are obstacles and endpoints respectively. Colored circles represent the initial locations of agents and their ids. The capacity of agents is set to 5 for both cases. The first case study examines whether the algorithms generate a valid solution for a single agent, therefore the second agent is removed from the environment. Table 1 is the list of tasks introduced at the first timestep of execution. The second case study is a simple multi-agent setting to see whether the paths planned are collision-free. The top half of the task list is introduced at the first timestep, and the rest at the following timestep. At each run, we measured the timestep at which the task list is empty and all deliveries are complete (i.e. makespan) and the average number of timesteps to complete tasks (i.e. service time).

Both TPMT and TPMC provide valid solutions for the Narrow Domain with single and double agent cases. As seen in Table 2, figures for makespan and service time of TPMT and TPMC appear to be less when compared with the single capacity algorithm TP. In the first case, the TPMT algorithm plans for the first five tasks in a row. The agent has to visit the distant pickup location of the last task separately. Instead, the TPMC algorithm assigns tasks with the closest start location whenever the agent is under-capacity, resulting in a better makespan. TPMT algorithm can make poor assignments when the start and the delivery of tasks are distant. Conversely, TPMT performs much better than TPMC in the second case study. The makespan for TPMC is the same as TP, indicating an insufficient utilization of multiple capacities. In the TPMT run, the first agent takes over the first three tasks, and the second agent takes the rest. In TPMC execution, tasks are assigned in an interleaved fashion. The first agent is under-capacity most of the time, hence greedily takes on new tasks including the fourth one. The second agent finishes earlier than first one and starts waiting. It would be much better if tasks were more evenly distributed among agents.

Table 2. Results for case studies I and II

	Case study I		Case study II	
Algorithm	Makespan	Service time	Makespan	Service time
TP	98	64	60	36
TPMT	84	44	33	25
TPMC	60	41	60	33

Table 3. Measures for TP algorithm

Agents	Tasks	Freq.	Makespan	Service time	Runtime
2	100	1	1161	508.9	0.19
5	100	1	531	196.7	2.39
10	500	1	1198	311.8	5.67
50	100	5	153	60.8	722.20
50	500	5	395	124.6	675.43

5.2 Experimental Setup

We performed an experimental evaluation to compare the efficiency and effectiveness of the two methods on the simulated warehouse environment [6], which is used as a benchmark in studies on MAPD. The sample environment consists of a 21×35 4-neighbor grid with narrow pathways. We performed experiments with various numbers of agents, capacities, and task frequencies on lists of 100 and 500 tasks. Task files were generated by randomly selecting pickup and delivery locations among the endpoints defined in the environment map file. All experiments are implemented in C++ programming language and run on a 1.80 GHz Intel Core i7-8565U laptop with 24 GB RAM.

5.3 Results

Tables 3 and 4 summarize the results of experiments. Increasing the number of agents has a huge positive impact on the makespan of all algorithms. Task frequency (number of tasks released at each timestep) seems to have a minor effect on makespan and service time for experimented cases. Even though there is a slight positive effect of capacity expansion for TPMT at the test with two agents, performance degrades with capacitation for more agents. On the other hand, for five agents, there is about 30% improvement on the makespan for TPMC. The greedy task assignment approach of TPMT is the reason for worse efficiency. Agents try to get more tasks to fill their capacity, and an agent closer to the pickup point may not have the chance to take over the task. In TPMC, an agent can visit the start point of a task τ and finish many other tasks before visiting the delivery point of τ. This may result in longer service times for some tasks. Compared to TP, the average service times of TPMC are still better for

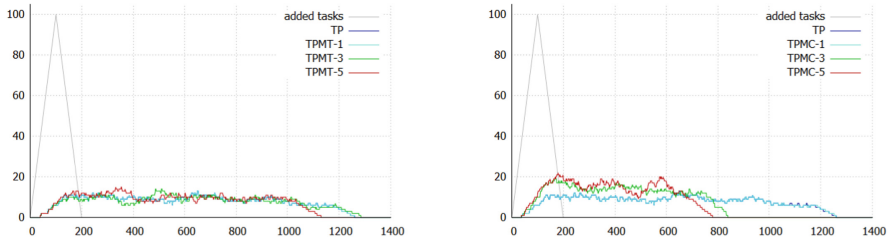
capacitated agents. As seen in Table 4, there is a trend of decrease in makespan as capacity of agents increase for TPMC. In general, TPMC better utilizes the capacity of agents than TPMT. Additionally, runtime values (in milliseconds per timestep) of TPMC are strongly preferable to TPMT. Therefore, TPMC suits better to a lifelong and real-time operation setting.

Table 4. Summary of experimental results (f.: frequency, Cap.: Capacity, S.Time: Service Time)

				TPMT			TPMC		
Agents	Task	f	Cap	Makespan	S.Time	Runtime	Makespan	S.Time	Runtime
2	100	1	1	1161	508.88	0.30	1161	509.12	0.16
			3	1186	519.23	0.93	740	319.5	0.19
			5	1031	458.19	1.16	680	285.15	0.20
5	100	1	1	531	196.7	1.66	500	190.05	0.70
			3	536	220.33	4.26	379	136.9	0.82
			5	746	357.45	33.58	341	126.25	0.92
10	500	1	1	1198	311.78	4.95	1192	308.39	2.05
			3	1603	474.37	63.40	841	168.97	2.10
			5	2414	815.61	165.66	771	134.30	1.93
50	100	5	1	153	60.82	131.77	126	47.7	19.59
			3	307	144.81	486.79	140	51.97	14.64
			5	679	312.36	494.36	126	51.96	17.16
50	500	5	1	395	124.59	97.58	353	102.62	36.64
			3	1056	479.25	1138.78	295	81.77	33.49
			5	2321	1235.74	1593.524	284	87.66	33.46

As in Ma et al. [6], we analyzed the throughput of the algorithms. Throughput is measured as the number of executed tasks within a 100-timestep window as a function of timestep t. Figure 3 visualizes the number of tasks added and executed for different types of algorithms (the number to the right of the name of the algorithm indicates the capacity). We preferred to present the throughput data for two agents and 100 tasks as the number of tasks per agent is higher, which may better reflect the long-term behavior of the system. It is apparent from the figure that TPMC is much better at utilizing capacitated agents.

Table 5 presents a comparison of TPMC algorithm with RMCA-r [1], the only scalable MAPD algorithm for capacitated agents to our knowledge. We performed experiments with 500 tasks of frequency 10 on small simulated warehouse instances of 20 and 50 agents with capacities 1, 3, and 5 respectively. Both makespan and total travel delay (total time elapsed between release and completion of all tasks) are smaller in RMCA than TPMC. This is an expected result as task assignment search is informed by actual costs in RMCA, whereas TPMC



(a) TPMT (b) TPMC

Fig. 3. Results for 2 agents, 100 tasks and task frequency 1 (Timestep vs. number of tasks).

uses lower-bound estimates. RMCA-r also incorporates meta-heuristic improvement strategies, which are reported to improve solutions substantially [1]. In spite of its better performance, RMCA does not guarantee to solve all well-formed instances of MAPD and needs an improvement on completeness.

Table 5. TPMC vs. RMCA-r (Cap.: Capacity, TTD: Total Travel Delay)

		RMCA-r		TPMC	
Cap.	Agents	Makespan	TTD	Makespan	TTD
1	20	624	106290	702	136912
	50	280	38050	371	64344
3	20	322	52771	524	95060
	50	153	15851	276	46912
5	20	247	36790	423	81507
	50	129	11464	270	43091

6 Conclusion

Despite the substantial amount of related work, MAPD for capacitated agents seems an under-explored area. We believe that MAPDC solvers will have potential uses in future automated warehouse systems, as well as in automated manufacturing and sorting systems. This study introduced TPMT and TPMC methods that expanded the Token Passing algorithm [6] to make use of capacitated agents. Agents executing TPMT request the token less often. Once they obtain it, they take over as many tasks as possible and plan a visiting route to complete the tasks. In TPMC, agents plan the smallest portion of their overall route each time they take the token. Whenever under-capacity, they take over the task with the nearest pick up location. Even though the results for TPMT are encouraging

for the narrow domain, evaluations on the small simulated warehouse environment suggest TPMC as a better choice for all performance measures taken into consideration. Although RMCA-r performs better than TPMC with capacitated agents, its prioritized planner needs improvement on completeness. TPMC guarantees completeness for well-formed instances of MAPD since it employs the same deadlock prevention mechanisms as TP (the proof is similar to the one given for Theorem 3 stated in [6]).

As the incorporation of capacitated agents remarkably improves the throughput of the system, MAPDC is a promising step toward highly scalable automated warehouse systems. Being a complete, efficient, and decentralized solution to MAPDC, TPMC is an encouraging method for such systems. Directions for future work include improving the solution quality of TPMC and exploring better task dispatching strategies for capacitated agents.

Acknowledgment. This work is partially supported by the Scientific and Technological Research Council of Turkey under Grant No 120E504 (Incremental Multi-Agent Path Finding).

References

1. Chen, Z., Alonso-Mora, J., Bai, X., Harabor, D.D., Stuckey, P.J.: Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. IEEE Robot. Autom. Lett. **6**(3), 5816–5823 (2021)
2. Felner, A., et al.: Search-based optimal solvers for the multi-agent pathfinding problem: summary and challenges. In: SOCS (2017)
3. Knight, K.: Are many reactive agents better than a few deliberative ones? In: Bajcsy, R. (ed.) Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 – September 3, 1993, pp. 432–437. Morgan Kaufmann (1993). http://ijcai.org/Proceedings/93-1/Papers/061.pdf
4. Korsah, G.A., Stentz, A., Dias, M.B.: A comprehensive taxonomy for multi-robot task allocation. Int. J. Robot. Res. **32**(12), 1495–1512 (2013)
5. Liu, M., Ma, H., Li, J., Koenig, S.: Task and path planning for multi-agent pickup and delivery, p. 9 (2019)
6. Ma, H., Li, J., Kumar, T.K.S., Koenig, S.: Lifelong multi-agent path finding for online pickup and delivery tasks. arXiv:1705.10868 [cs] (2017). http://arxiv.org/abs/1705.10868, arXiv: 1705.10868
7. Nunes, E., Manner, M.D., Mitiche, H., Gini, M.L.: A taxonomy for task allocation problems with temporal and ordering constraints. Robot. Auton. Syst. (2017). https://doi.org/10.1016/j.robot.2016.10.008
8. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
9. Statt, N.: Amazon says fully automated shipping warehouses are at least a decade away (2019). https://www.theverge.com/2019/5/1/18526092/amazon-warehouse-robotics-automation-ai-10-years-away
10. Tajelipirbazari, N., et al.: Multi-agent pick and delivery with capacities: action planning vs path finding. In: Cheney, J., Perri, S. (eds.) PADL 2022. LNCS, vol. 13165, pp. 24–41. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94479-7_3