# A Replication Study of the TA-Prioritized Algorithm with a Modified Deadlock Avoidance Method

**NICOLAAS PIETER CAWOOD**

*Corresponding author: pieter.cawood@gmail.com*

*Compiled June 4, 2020*

**This paper proposes a modified deadlock avoidance method for the TA-Prioritized algorithm by Liu et al. [1]. Their algorithms were developed for the offline Multi-Agent Pickup and Delivery (MAPD) problems where a team of agents have delivery tasks with known release times (when the tasks are ready for pickup.) Offline MAPD problems exist in settings such as warehouses and factories where the release times of tasks are known in advance and Liu et al. [1] make use of this information to compute a good task sequence for each agent using a travelling salesman problem (TSP) solver. The task sequences are then used to plan agent paths accordingly and the deadlock avoidance method proposed attempts holding the pickup locations (keeping an agent stationed at a vertex) if an agent has reached it before the release time and retrying a 1 timestep delayed path from the agent's initial parking location if any of the agent's tasks' path finding fails.**

## 1. INTRODUCTION

In an MAPD problem setting there exists multiple robots in a shared environment intending to have the lowest makespan and collision-free paths planned over the entire set of tasks distributed amongst agents. Multi-Agent Path Finding (MAPF) problems are known to be NP-hard, however, they might be computed within polynomial time when treated as a max-flow problem when allowing task goals to be swapped [2].

The TA-Hybrid and TA-Prioritized algorithms were designed for an offline setting, where the tasks for an instance and their release times (time that collection is ready) are known before initialising the system. There are also settings of lifelong (online) problems, where tasks are only known to exist after their release times have expired. Online MAPD algorithms such as the one studied by Ma et al. [3] may also be used to solve offline problems, however, Liu et al. [1] mention that they do not utilise all of the information and may compute less effective solutions.

The main objective of this paper is to replace the original "reserving dummy paths" [1] deadlock avoidance method and conclude whether it improves makespans by implementing it for the TA-Prioritized algorithm.

## 2. RELATED WORK

The MAPD problem is related to the Generalized Task Assignment and Path Finding (G-TAPF) problem [4] where each agent may have multiple tasks. Nguyen et al. [4] proposed using answer set programming to perform both the task assignment and path finding, however, their results found that it only scales to 20 agents and the number of tasks has to be less than the number of agents.


**Fig. 1.** Kiva Robotics system. Now owned by Amazon and used in their sortation centers as an autonomous delivery system. [5].

Nunes et al [6] published a taxonomy paper on task allocation problems with temporal constraint and Liu et al. [1] described the most related approach as treating the task assignment problem as a travelling salesman problem (TSP.) The travelling salesman problem is a well known NP-complete problem and pickup and delivery travelling salesman problems receive increasing interest in recent studies [7–9].

The MAPF component has similarly attracted many researchers and there are several different methods studied since it is also NP-hard to solve optimally [10]. There exist Anonymous Multi-Agent Path Finding (AMAPF) methods that allow

solving the path finding in polynomial time using max-flow algorithms such as [11] for problems where agent goals may be swapped. The MAPF problem may not swap agent goals on the other hand and it may be solved with specialised solvers or other combinatorial methods and [12, 13] are suggested surveys on them.

## 3. OVERVIEW

This paper is structured from the following section with the theoretical background of the TA-Prioritized algorithm. The problem is then divided into a two sub-problems: to find good task sequences for each agent using a Travelling Salesman Problem (TSP) solver and then to solve the MAPF for each agent to its allocated tasks' endpoints. Modifications to the original algorithm's deadlock avoidance method are then discussed and the algorithm was then implemented and analyzed using the original instance data sets.

## 4. THEORETICAL BACKGROUND

The MAPD problem is formalised [1] as a set of agents, $A = \{a_1, ..., a_M\}$ with a set of task $T = \{t_1, ..., t_N\}$ and an environment as an undirected graph $G = (V, E)$, with vertices $V$ that contain locations of the environment and edges $E$ that permit travelling between possible vertices. Each agent has an assigned parking location $p_i \in V$ where the agent is initially located and moves to this location only again once it has finished executing all of its tasks. Each task is characterised by a pickup location (start) $s_j \in V$, delivery location (goal) $g_j \in V$ and release time (the time when the pickup is available) $r_j \in V$.

Each agent has to move from its parking location to its first task's pickup location and may start moving to its delivery location after the release time has expired. Agent paths may wait in a vertex (hold it) for a period of timesteps or move to a neighbouring vertex if it does not cause a collision. Once an agent reaches its current task's delivery location, it immediately assigns its next task as its current task and assigns the task's pickup location as its next endpoint. There are two types of collisions that have to be avoided; the first is called vertex collisions, which occur when two agents occupy the same vertex at a given timestep and the second type called edge collisions occur when two agents move to each other's location within the same timestep.

### A. Task assignment

TA-Prioritized first computes an ordered task sequence for each agent. The primary objective of MAPD is to minimize the makespan- which is described as the timestep when all tasks deliviries are complete [14].

Liu et al. [1] first construct a directed weighted graph for all the tasks using their endpoint locations and release times to determine the weights. They then use a TSP solver to compute improved sets of task sequences and distribute them among all of the MAPD instance agents.

#### *A.1. Directed weighted graph*

Liu et al. [1] constructed a directed weighted graph $G' = (V', E')$ with $V' = A \cup T$. $V'$, which contains an integer representing the order of each agent along with integers representing each task of the instance. There are four types of edges in $E'$, which each have an integer weight reflecting a timestep and they are listed below:

1. The first type, $max(dist(p_i, s_j), r_j)$, computes the weight as the distance from $a_i$'s parking location to its first task's pickup location $s_j$. If the agent arrives at the pickup location before the task's release time. The weight is taken as release time instead.
2. The second edge type, $dist(s_i, g_i) + dist(g_i, s_j)$, computes the edge weight from and agent's current assigned task's pickup location $s_i$ to its delivery location $g_i$ and then to its pickup location of its next task $s_j$.
3. The third type, $dist(s_i, g_i)$, computes the edge weight from the agent's last task's pickup location $s_i$ to its respective delivery location $g_i$.
4. And the last type is an edge with zero weight as the agent has finished its tasks and does not add to the makespan any further.

The concept is for $G'$ to be a complete graph that contains Hamiltonian cycles that may be divided into $M$ partitions to allocate a sequence to each agent.

#### *A.2. TSP Solver*

Liu et al. [1] then made use of the extended Lin-Kernighan-Helsgaun TSP solver (LKH-3) [15] to improve the hamiltonian cycles of $G'$ to more efficient sequences of task execution. The graph is loaded and configured as a Pickup-and-delivery problem with time windows, and the solver is found to deliver good task sequences.

### B. Path finding

Once the task sequences have been computed, TA-Prioritized then performs prioritized path planning based on each agent's execution time. Liu et al. [1] solve the Multi-Agent Path Finding (MAPF) problem by planning agent paths one by one in decreasing order of their execution times. The algorithm starts by calculating each agent's execution time by finding paths for its entire sequence. The agent with the highest execution time's path is stored, while the others are reinitialised to be computed again. The algorithm then finds the next agent with the highest execution time while avoiding collision with the path already found for the previous agent. This cycle continues until each agent's path for its entire task sequence has been calculated. This approach aims to minimize the number of constraints that higher execution-time agents have to keep the makespan minimal.

The paths of each agent are found as a concatenation of sub-paths for all of its allocated tasks. The sub paths are computed from the parking location (for the agent's first task only) or delivery locations to the next task's pickup location and then another sub-path from this pickup location to its respective delivery location. This cycle continues until the last delivery is complete and a sub-path is then computed back to the agent's parking location.

The paths may be found by an A* search in the space of location-time pairs $(x, t)$ and TA-Prioritized plans the paths without backtracking as it avoids deadlocks with a method referred to as "reserving dummy paths". The dummy paths may also be found with an A* search from every sub-path's goal location to the agent's parking location. The dummy paths are replaced by the agent's new calculated path to its goal after every iteration and agent paths have to avoid collisions with other agents' paths and their dummy paths. Agents never move along their dummy paths, except for the last one that takes the agent back to its parking location from its last delivery location. Collision free paths are guaranteed to exist for each agent when the MAPD instance is regarded as well-formed and in this case, no path should not transverse parking locations.

**Algorithm 1.** TA-Prioritized

**Input** G, agents (A), tasks
**Output** none

$sequences \leftarrow TaskAssignment(G, agents, tasks)$
$paths \leftarrow \emptyset$
$closed \leftarrow \emptyset$
**while** size of $closed \neq$ size of $A$ **do**
  $a \leftarrow HighestExecutionTime(\forall A \notin closed)$
  **for each** task in sequences[a] **do**
    $path1 \leftarrow PickupSubPath(G, paths, s_j)$
    $path2 \leftarrow DeliverySubPath(G, paths, g_j)$
    $paths[a] \leftarrow paths[a] + (path1 + path2)$
  **end for**
  $closed \leftarrow closed + a$
**end while**

## 5. MODIFICATIONS

### A. Deadlock avoidance modification

The TA-Prioritized deadlock avoidance method- "reserving dummy paths", was replaced with modifying the A* search to attempt holding the pickup location of a task for as long as the release-time of the task has not yet expired. If a higher priority agent's path leads to the same pickup location, the A* search finds a path for the current (lower priority) agent to the nearest safe neighbouring vertex of the pickup location (or to other vertices), and finds the path back to the task's pickup location should it become safe again. The goal is to keep the agent as close as possible to the pickup location, instead of holding the agent's initial location, which might be constraint more when planning the path back to the pickup location.

Deadlocks still occur and the path planning might fail to find any safe path once the agent gets surrounded by more than one higher priority agent, which results in a failed A* search. When this occurs, the agent path is recalculated after holding the initial parking location for 1 timestep.

The original algorithm makes use of the final dummy path to make the agent travel back to its parking location and this modification requires an additional sub-path calculated from the agent's last delivery location to its parking location.

**Algorithm 2.** Modified TA-Prioritized

**Input** G, agents (A), tasks
**Output** none

$sequences \leftarrow TaskAssignment(G, agents, tasks)$
$paths \leftarrow \emptyset$
$closed \leftarrow \emptyset$
**while** size of $closed \neq$ size of $A$ **do**
  **while** a.path = False **do**
    $a \leftarrow HighestExecutionTime(\forall A \notin closed)$
  **while** path1 = False or path2 = False **do**
    **for each** task in sequences[a] **do**
      $path1 \leftarrow PickupSubPath(G, paths, s_j, r_j)$
      $path2 \leftarrow DeliverySubPath(G, paths, g_j)$
      $paths[a] \leftarrow paths[a] + (path1 + path2)$
  **end for**
  $paths[a] \leftarrow paths[a] + ParkingSubPath(G, paths, p_i)$
  $closed \leftarrow closed + a$
**end while**

*Modifications in blue

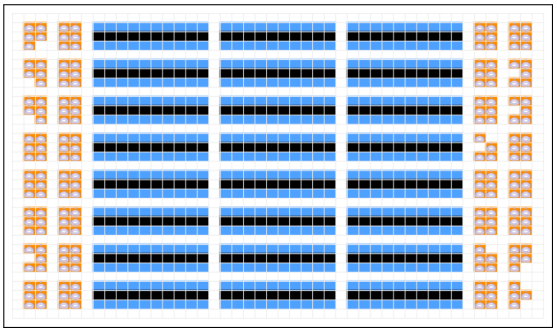

**Fig. 2.** A 33×46 grid with 180 agents and 2000 tasks in simulation.

| | | TA-Hybrid (Original results[1]) | | TA-Prioritized (Original results[1]) | | Modified TA-Prioritized | |
|---|---|---|---|---|---|---|---|
| f | agents | makespan | runtime (secs) | makespan | runtime (secs) | makespan | runtime (mins) |
| 1 | 10 | **1087** | 13 | 1094 | 10 | **1087** | 0.4 |
| | 20 | 612 | 38 | **608** | 21 | 610 | 3.2 |
| | 30 | **528** | 118 | 546 | 35 | 547 | 12.1 |
| | 40 | **525** | 182 | 534 | 44 | 549 | 24.1 |
| | 50 | **525** | 727 | 540 | 58 | 535 | 32.1 |
| 2 | 10 | **1048** | 10 | 1056 | 10 | 1046 | 0.5 |
| | 20 | 561 | 23 | 569 | 20 | **554** | 1.1 |
| | 30 | **385** | 38 | 394 | 29 | 403 | 1.9 |
| | 40 | **323** | 94 | 328 | 39 | 335 | 3.6 |
| | 50 | **300** | 130 | 327 | 44 | 324 | 4.9 |
| 5 | 10 | **1039** | 10 | 1054 | 10 | 1044 | 0.4 |
| | 20 | 549 | 19 | 551 | 19 | **538** | 1.0 |
| | 30 | 377 | 21 | **370** | 29 | 372 | 1.9 |
| | 40 | **285** | 31 | 289 | 41 | 288 | 3.3 |
| | 50 | **241** | 17 | 244 | 48 | 248 | 4.8 |
| 10 | 10 | **1045** | 10 | 1036 | 10 | 1048 | 0.5 |
| | 20 | 541 | 19 | 559 | 19 | **532** | 1.1 |
| | 30 | 373 | 21 | **369** | 19 | 371 | 1.9 |
| | 40 | 285 | 31 | 294 | 40 | **282** | 3.3 |
| | 50 | 241 | 57 | 236 | 50 | **235** | 4.8 |
| 500 | 10 | **1037** | 11 | 1045 | 10 | 1041 | 0.4 |
| | 20 | 539 | 14 | 535 | 19 | **529** | 1.0 |
| | 30 | **362** | 21 | 370 | 29 | 364 | 1.7 |
| | 40 | 280 | 22 | **275** | 39 | 280 | 3.0 |
| | 50 | **231** | 28 | 235 | 50 | 233 | 4.5 |
| average | | **532** | 30 | 538 | 30 | 537 | 4.68 |

**Table 1.** Small warehouse results.

## 6. EXPERIMENTAL RESULTS

The modified TA-Prioritized algorithm was implemented in python 3 and was ran on a PC with an i7 CPU at 2.6 GHz with 16 GB installed RAM. An agent-based modelling framework called Mesa [16] was used to simulate the computed MAPF instances. The source code may be found on GitHub.

The original map, task and TSP sequence instance data were supplied by the authors and used to compute makespans that may be compared to the original results. The runtimes (program execution) captured were considerably larger than the original findings since this implementation was done in Python- a dynamically typed programming language as opposed to the original results found with c++. The deadlock avoidance modification is expected to reduce the runtimes and since the main objective of MAPD problems is to compute the lowest makespans- only the makespans are considered to compare the results.

| | | TA-Hybrid (Original results[1]) | | TA-Prioritized (Original results[1]) | | Modified TA-Prioritized | |
|---|---|---|---|---|---|---|---|
| f | agents | makespan | runtime (secs) | makespan | runtime (secs) | makespan | runtime (mins) |
| 2000 | 60 | **991** | 500 | 1045 | 507 | 1009 | 79.5 |
| | 90 | **699** | 637 | 721 | 789 | 729 | 226.7 |
| | 120 | **556** | 1091 | 578 | 1098 | 563 | 459.7 |
| | 150 | **479** | 1803 | 524 | 1317 | 505 | 593.3 |
| | 180 | **419** | 2457 | 475 | 1683 | 470 | 1162.5 |
| average | | **629** | 1298 | 669 | 1079 | 655 | 504 |

**Table 2.** Large warehouse results.

### A. Small warehouses

The original small warehouse instances were simulated for a total of 500 tasks each. Each instance includes 5 different frequencies of tasks being released and they include frequencies of 1, 2, 5, 10 and 500 tasks per timestep. The results show that the modified TA-Prioritized algorithm's makespan was lower than the other algorithms in three more instances than before and its average makespan was reduced by 1 timestep.

### B. Large warehouses

The original large warehouse instances were simulated for a total of 2000 tasks each. All tasks were released at the first timestep. The modified TA-Prioritized algorithm's results almost consistently improved the makespans for the large warehouse instance and the average makespan was reduced by 14 timesteps.

## 7. CONCLUSION AND FUTURE WORK

A modified deadlock avoidance method was proposed in this paper by replacing the original "reserving dummy path" [1] deadlock avoidance method. The modification was implemented for the TA-Prioritized algorithm and the results show a minor improvement for smaller instance makespans and a more significant improvement for the large instance makespans.

The modification was only implemented for the TA-Prioritized algorithm, and future work may modify other algorithms that use dummy paths as a deadlock avoidance method to provide a richer set of results for analysis.

## REFERENCES


1. M. H. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *AAMAS,* (2019).
2. W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *AAMAS,* (2018).
3. H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," CoRR **abs/1705.10868** (2017).
4. V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh, "Generalized target assignment and path finding using answer set programming," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17,* (2017), pp. 1216–1223.
5. "Kiva systems' robots shuttled merchandise around a gilt groupe distribution center in shepherdsville, ky." Available at https://bostonglobe-prod.cdn.arcpublishing.com/resizer/_xPRAp2x2DM90dvR1xpZzv5hnEo=/1024x0/arc-anglerfish-arc2-prod-bostonglobe.s3.amazonaws.com/public/32LCY3DSHAI6DFFXFPJTA4U56U.jpg (2012).
6. E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," Robotics Auton. Syst. **90**, 55–70 (2017).
7. E. Osaba, F. Diaz, E. Onieva, P. López-García, R. Carballedo, and A. Perallos, "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, 2015), pp. 557–567.
8. K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," Roskilde: Roskilde Univ. (2017).
9. S. Yoon and J. Kim, "Efficient multi-agent task allocation for collaborative route planning with multiple unmanned vehicles," IFAC-PapersOnLine. **50**, 3580 – 3585 (2017). 20th IFAC World Congress.
10. J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *2013 IEEE International Conference on Robotics and Automation,* (IEEE, 2013), pp. 3612–3617.
11. H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," arXiv preprint arXiv:1612.05693 (2016).
12. A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Tenth Annual Symposium on Combinatorial Search,* (2017).
13. H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon, "Overview: Generalizations of multi-agent path finding to real-world scenarios," arXiv preprint arXiv:1702.05515 (2017).
14. K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," CoRR **abs/1901.11282** (2019).
15. K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," (2017).
16. D. Masad and J. Kazil, "Mesa: an agent-based modeling framework," in *14th PYTHON in Science Conference,* (2015), pp. 53–60.