A Multi-Label A* Algorithm for Multi-Agent Pathfinding[*]

Florian Grenouilleau,[1] **Willem-Jan van Hoeve,** [2] **J. N. Hooker** [2]
[1] Polytechnique Montréal
2900 Boulevard Édouard-Montpetit, Montréal, Québec H3T1J4
[2] Tepper School of Business, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
florian.grenouilleau@cirrelt.net, vanhoeve@andrew.cmu.edu, jh38@andrew.cmu.edu

Abstract

Given a set of agents, the multi-agent pathfinding problem consists in determining, for each agent, a path from its start location to its assigned goal while avoiding collisions with other agents. Recent work has studied variants of the problem in which agents are assigned a sequence of goals (tasks) that become available over time, such as the online multi-agent pickup and delivery (MAPD) problem. In this paper, we propose a multi-label A* algorithm (MLA*) for this problem. It extends the classic A* algorithm by allowing the computation of paths with multiple ordered goals (such as a pickup and delivery). Moreover, we develop a new h-value-based centralized heuristic for the MAPD. Computational experiments show that our proposed MLA* obtains substantial improvements in terms of makespan and service time as compared to existing methods, while being more computationally efficient. On instances with a thousand tasks and hundreds of agents, our method reduces the average service time by 43% compared to the state of the art, with considerably less computational effort.

KEYWORDS Multi-agent pathfinding, pickup and delivery, multi-label A* algorithm

INTRODUCTION

Multi-agent path finding (MAPF) is a classical problem in planning and has been studied extensively (Ma et al. 2017a; Felner et al. 2017). Given a set of agents, MAPF consists in finding, for each agent, a path from its start location to its goal while avoiding collisions between agents. Collisions occur when two agents simultaneously occupy the same location or simultaneously cross the same edge in opposite directions. This problem has applications in contexts as diverse as warehouse logistics (Wurman, D'Andrea, and Mountz 2008), video games (Silver 2005), and even search and rescue (Kitano et al. 1999).

Several variants of this problem have recently been developed by introducing such elements as deadlines for reaching goals (Ma et al. 2018), kinematic constraints on the agent's motion (Hönig et al. 2016), or task assignments over a set of anonymous agents (Ma and Koenig 2016). A particular variant is the extension of single goals to more complex tasks that consist of multiple goals. For example, Nguyen et al. (2017) study the MAPF with multiple-goal tasks and deadlines and develop an exact resolution method based on answer set programming. This method allows a complete formulation of the problem but quickly meets scalability issues when the number of agents and/or tasks increases.

Ma et al. (2017b) consider a specific multi-goal MAPF problem in which each task consists of a pickup and a delivery goal. They refer it to as the *multi-agent pickup and delivery* (MAPD) problem. Tasks are released at different time steps and must be assigned to agents dynamically. An agent performs a task by moving to the pickup point and from there to the delivery point. An agent may also perform several tasks sequentially. This resembles a warehouse logistics scenario in which robots move inventory pods between storage and handling positions in real time; see Figure 1 for an example.

To solve the MAPD problem, Ma et al. (2017b) propose a decoupled heuristic algorithm based on token passing (with or without task swapping), as well as a centralized algorithm that jointly assigns available agents to tasks and finds paths with conflict-based search (Sharon et al. 2015). Since the centralized algorithm does not scale to larger instances, it is primarily used as a baseline comparison. Some parallels could be done between this centralized algorithm and the cooperative auction literature (Koenig et al. 2006; Koenig, Keskinocak, and Tovey 2010)

For smaller instances of up to 50 agents, the faster version of the decoupled heuristic method (the one without task swapping) takes up to about 5 ms per time step. When compared to the centralized algorithm on instances with 50 agents, it provides solutions that are about 15% worse with respect to average makespan and 70% worse with respect to average service time. Furthermore, while the decoupled method without task swapping can handle up to 200 agents for larger instances (taking 500 ms per time step), it needs about 6,000 ms per time step for 500 agents.

Contributions. The purpose of this paper is to develop improved methods for the MAPD problem. First, we present a

[*]Partially supported by ONR Grant No. N00014-18-1-2129.

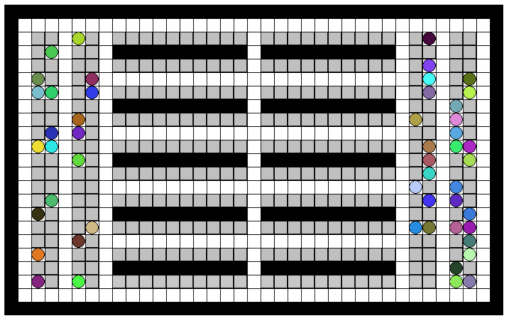


Figure 1: Representation of a warehouse-like grid with 50 agents from (Ma et al. 2017b). Black cells are blocked, gray cells correspond to possible pickup and delivery locations. Colored circles correspond to the agents' initial positions.

*multi-label A** (MLA*) algorithm that computes the shortest path for an agent with respect to an ordered list of goals. Second, we present a new centralized heuristic for assigning available agents to tasks, based on the agents' h-values. Third, we apply our MLA* and heuristic assignment algorithm to the MAPD problem and show that we can improve both the solution quality and the computation time relative to the state of the art. For example, for the larger instances with 500 agents, our approach reduces makespan as much as 30% and average service time as much as 43%, while using at most 74 ms per time step (a reduction of 97%).

THE MULTI-AGENT PICKUP AND DELIVERY PROBLEM

The MAPD problem is defined as follows (Ma et al. 2017b). We are given a set $\{a_1, a_2, \ldots, a_m\}$ of agents that can move on a graph $G = (V, E)$, where V is the set of possible positions and E is the set of feasible connections between locations. At each time step, new tasks are added to the task set T, and idle agents are assigned tasks from T when possible according to some heuristic.

At a given time step t, each agent a occupies a position $p_a(t) \in V$. Between two consecutive time steps, an agent can stay at its current position or move along an incident edge in E. Throughout the time horizon (which dynamically expands in order to fit with agent's needs), collisions between agents must be avoided. A collision occurs when $p_a(t) = p_{a'}(t)$, or $p_a(t) = p_{a'}(t+1)$ and $p_{a'}(t) = p_a(t+1)$, for some pair of distinct agents a, a'. The objective is to execute the tasks as soon as possible, as measured by the service time (average number of time steps to complete tasks after they are added to T) or the makespan (difference between first release time and latest completion time).

In Ma et al.'s decoupled token-passing algorithm without task swapping, called TP, idle agents take turns selecting a task from T (if possible) and finding a path that performs the task. The selected task is one with minimum h-value, defined as the Manhattan distance between the agent's current location and the pickup location. If no task assignment is feasible (e.g., T is empty, or the pickup or delivery location of every task is blocked by another agent), the agent checks whether its current location is the pickup or delivery location assigned to another agent and, if so, moves out of the way. If a task is assigned, the agent computes a path from its current location to the pickup location using an A* algorithm, and then a path from the pickup location to the delivery location, again via A*.[1] The paths it finds must not collide with paths in the *token* inherited from the previous agent, where the token contains the paths already determined for other agents. The token is updated to contain the new paths and passed to the next idle agent.

The authors also introduce an extension of TP that allows for task swapping between agents, called TPTS. It can find improved solutions but requires more computation time.

MULTI-LABEL A* ALGORITHM

To motivate our multi-label A* algorithm, recall that TP and TPTS employ two sequential A* calls, one to find a path from the agent's location to the pickup location, and another to find a path from the pickup to the delivery location. This sequential approach over-constrains the problem at the pickup node, in the following sense. When the A* algorithm computes agent a's path from its current location toward the pickup node, it assumes that a will rest at the pickup location until the end of the time horizon. This can unnecessarily constrain a's options in two ways:

- Case 1. The pickup node is another agent's scheduled delivery node. The A* algorithm will conclude that a cannot reach the node because this would block the other agent's future arrival, when actually a could pass through the pickup node and proceed toward its delivery node before the other agent arrives.
- Case 2. Another agent is scheduled to pass through the pickup node at some point in the near future. The A* algorithm will delay a's arrival at the node until the other agent has passed through it, when actually a could pass through the node and proceed to its delivery node before the other agent passes through the node.

To avoid these situations, we propose a multi-label A* algorithm (MLA*) which combines the search for a path to the pickup location with that of the delivery location (and more generally any other goals that follow sequentially). We can therefore create a path to the pickup location even if another agent uses this location later. This modification allows us to make more (and therefore better) assignments (Case 1) and to find shorter paths (Case 2).

Formally, we use the classic A* algorithm structure (Hart, Nilsson, and Raphael 1968). For each search node n of the MLA* algorithm, we define g_n, p_n and ℓ_n, which respectively denote the node's g-value (number of time steps elapsed), position, and label. The label indicates the current state of the node; if $\ell_n = 1$ the agent is seeking a path to the pickup location, while if $\ell_n = 2$ the agent is seeking a path to the delivery location. We also define π_1 and π_2 to be the pickup and delivery locations, respectively. We denote

[1]The two-step sequential application of A* is not explicit in the text, but is based on the code provided by (Ma et al. 2017b).

```
Create the initial search node $n_0$ with $g_{n_0} = 0$, $\ell_{n_0} = 0$,
  $f_{n_0} = h_{n_0}$, and $p_{n_0} =$ the agent's current location;
Add $n_0$ to the queue $Q$;
while $Q$ is not empty do
    Get the node $n$ in $Q$ with the smallest $f_n$;
    if $\ell_n = 1$ and $g_n > t_{\max}$ then
        continue (reject node $n$, go to top of while loop)
    if $\ell_n = 1$ and $p_n = \pi_1$ then
        Create node $n'$ with $p_{n'} = p_n$, $g_{n'} = g_n$ and
          $\ell_{n'} = 2$, and add $n'$ to $Q$;
    if $\ell_n = 2$ and $p_n = \pi_2$ then
        Return the found path;
    Expand node $n$ by adding to $Q$ nodes with adjacent
      locations that are not blocked by other agents;
end
Return false (no feasible path) ;
```

Algorithm 1: Outline of the multi-label A* algorithm

by $t_{\max}$ the latest time step at which the agent can reach the pickup location, with $t_{\max} = \infty$ when no other agent is scheduled to terminate at the pickup location. Finally, we define the f-value $f_n = g_n + h_n$, where the h-value h_n of the search node n is defined

$$h_n = \begin{cases} h(p_n, \pi_1) + h(\pi_1, \pi_2) & \text{if } \ell_n = 1 \\ h(p_n, \pi_2) & \text{if } \ell_n = 2 \end{cases}$$

and where $h(\cdot, \cdot)$ is the distance between two points.

Algorithm 1 presents the outline of the proposed MLA*. The method first creates the initial search node corresponding to the current location of the agent and assigns it label 1. This initial node is added to the queue Q. While Q is nonempty, the search node n with the smallest f-value is selected. Three cases are then tested. If n has label 1 and g_n is greater than the latest pickup time ($t_{\max}$), node n is rejected, and another node is selected. If n corresponds to arrival at the pickup location ($\ell_n = 1$ and $p_n = \pi_1$), a search node with the same values and label 2 is created and added to Q. If n corresponds to arrival at the delivery node, the search is stopped and the algorithm returns the found path. If none of those cases obtain, node n is expanded by adding successor nodes to Q.

H-VALUE-BASED HEURISTIC

We now present our centralized h-value-based heuristic (HBH) for assigning tasks to agents over the entire time horizon. The heuristic is summarized in Algorithm 2.

At each time step, HBH repeatedly solves an assignment and path-finding problem until all the tasks are assigned. The algorithm first retrieves the list of currently available agents (agents with no assigned task, resting at a location) and the list of open tasks (tasks released and not yet assigned). A list of agent-task pairs is created and sorted in increasing order of h-value. HBH then scans the list and tries each agent-task pair (a, τ) using MLA*. If the assignment is feasible, HBH assigns a to τ and updates a's path to the one found by MLA*. If some agents remain available, HBH checks their current locations and move them toward the closest free endpoint if necessary. Endpoints are locations at which agents are authorized to rest indefinitely, including pickup and delivery points.

```
Set the time step $t$ = 0;
while not all the tasks have been assigned do
    Get the list $A_t$ of agents available at time step $t$;
    Get the list $T_t$ of released and nonassigned tasks at time
      step $t$;
    Create the list $L_t$ of agent-task pairs $(a, \tau)$ with $a \in A_t$
      and $\tau \in T_t$, and sort the pairs in nondecreasing order of
      h-value;
    forall pairs $(a, \tau)$ in $L_t$ do
        if agent $a \in A_t$, task $\tau \in T_t$, and MLA* sccessfully
          returns a path for $(a, \tau)$ then
            Assign task $\tau$ to $a$;
            Update $a$'s path to the path found by MLA*;
            Remove $a$ from $A_t$ and $\tau$ from $T_t$;
    end
    forall remaining agents $a$ in $A_t$ do
        if agent $a$ currently occupies the pickup or delivery
          location for some task in $T_t$ then
            Move the $a$ to the closest free endpoint;
    end
    $t = t + 1$;
end
Return the solution found;
```

Algorithm 2: Outline of the h-value-based heuristic

Table 1: Experimental Design

Method	Type	Description
TP	decoupled	code from Ma et al., run on our machine
TP+MLA*	decoupled	TP that uses MLA* instead of two-step A*
HBH+MLA*	centralized	uses h-value heuristic but MLA* instead of CBS
Central	centralized	reported times as in Ma et al.

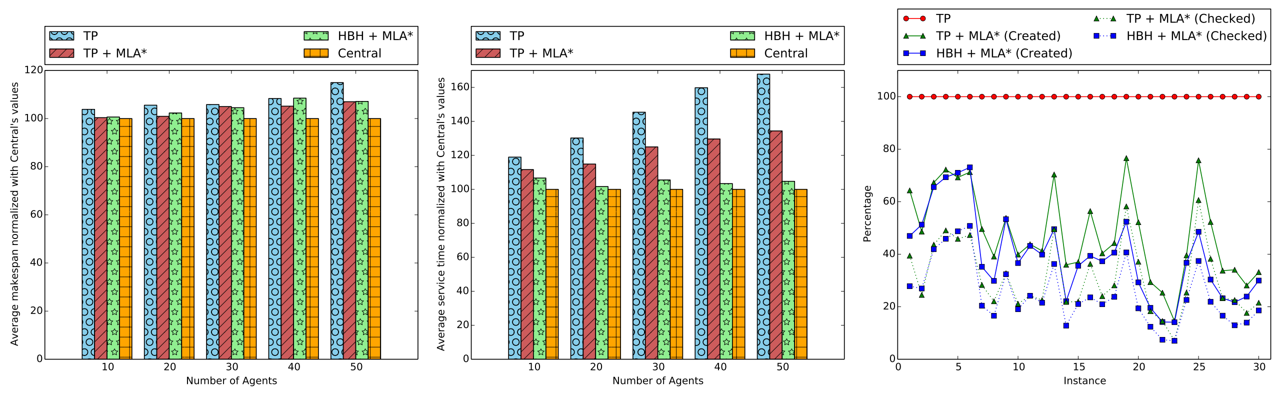
COMPUTATIONAL EXPERIMENTS

The goal of our experiments is twofold: assess the efficiency of the MLA* algorithm and analyze the impact of the new h-value-based heuristic. These methods are compared to the TP and Central methods of Ma et al. (2017b). Specifically, we compare the methods as indicated in Table 1.

We performed tests on the warehouse instances of Ma et al. (2017b). The first set contains 30 instances with 500 tasks that are released over time with a frequency of 0.2 to 10 per time step, and 10 to 50 agents. The second set has 5 instances containing 1000 tasks with a frequency of 50 per time step, and 100 to 500 agents.

The TP, MLA* and HBH methods were implemented in C++ and run on a 2.7 GHz Intel Core i5 Macbook, with 16 Gb RAM and using one core. The results for the Central method are those presented in Ma et al. (2017b).

Figure 2 and Table 2(a) show the results for the small instances. Figures 2(a) and 2(b) present, respectively, the average makespan and service time normalized to the Central



(a) Average normalized makespan (b) Average normalized service time (c) Normalized number of A* states

Figure 2: Performance of the TP, TP+MLA*, and HBH+MLA* algorithms on small instances as compared to the Centralized algorithm, in terms of average normalized makespan (a) and service time (b). Figure (c) shows the number of created and checked states for each method (normalized with respect to TP).

Table 2: Performance on small and large instances. The percentages represent the relative improvement of HBH+MLA* to TP.

Nb Agents	TP	TP+MLA*	HBH+MLA*	Central
10	0.12	0.12	0.10	123
20	0.53	0.31	0.22	370
30	1.19	0.56	0.41	728
40	3.08	0.86	0.59	1284
50	3.73	1.22	0.84	2085

(a) Computation time on small instances, ms/time step.

	TP			HBH + MLA*					
Instance	MS	ST	CT	MS	Gap MS	ST	Gap ST	CT	**Gap CT**
50_100_1000	990	463	90	800	**–19%**	363	**–22%**	2.83	**–96.9%**
50_200_1000	745	330	503	606	**–19%**	208	**–37%**	9.04	**–98.2%**
50_300_1000	754	302	1458	507	**–33%**	157	**–48%**	18.9	**–98.7%**
50_400_1000	753	289	2261	491	**–35%**	136	**–53%**	47.5	**–97.9%**
50_500_1000	946	290	3290	501	**–47%**	125	**–57%**	74.0	**–97.8%**
Av. Gap					**–30.5%**		**–43%**		**–97.9%**

(b) Performance on large instances, where MS = makespan, ST = service time, CT = computation time in ms/step.

values. Figure 2(c) compares the total number of created and checked search nodes during the MLA* calls to those during the A* calls in the TP heuristic. Finally, Table 2(a) presents the computation times for all the scenarios, in milliseconds per time step.

We first observe that, when coupled with TP, MLA* allows substantial improvement in terms of makespan but more importantly in terms of service time. MLA* also reduces the computation time. These improvements are mainly due to the ability of MLA* to reduce by 50% the number of created nodes and by 75-80% the number of checked modes, compared to the TP heuristic (Fig. 2(c)). Second, we observe that in term of makespan, HBH+MLA* behaves similarly to TP+MLA*, but the new heuristic substantially improves service time for all the instances. It also obtains solutions similar on quality to those of Central but with computation times three orders of magnitude smaller (0.84 ms versus 2085 ms for 50 agents).

Computational experiments on large instances are presented in Table 2(b). We observe that HBH+MLA* reduces makespan by 30% and service time by 43% but more importantly, reduces computation time by 98% compared with TP. These results show the efficiency of both MLA* and HBH even on large instances. No comparison with the Central method is done on those instances due to its scalability issues.

CONCLUSIONS

We studied the multi-agent pickup and delivery (MAPD) problem, which consists in dynamically assigning a set of pickup and delivery tasks to a set of agents while determining the agents' paths without collisions. To solve the problem, we first presented an innovative multi-label A* algorithm (MLA*) that extends the classic A* algorithm by computing a path through a set of ordered goal locations. This method determines paths with a broader vision of the problem and avoids the unnecessary constraints that result from sequential A* calls. Secondly, we proposed a new h-value-based heuristic (HBH) which iteratively assigns tasks based on their h-values.

Computational experiments show that the proposed MLA* algorithm brings substantial improvements in terms of makespan, service time and computation time. Those improvements are enhanced when MLA* is combined with HBH, which results in computation times that are three orders of magnitude less than the state-of-the-art baseline. On large instances, HBH reduces average makespan by 30%, service time by 43%, and computation time by 98%.

References

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *SOCS*, 29–37.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Hönig, W.; Kumar, T. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *ICAPS*, 477–485.

Kitano, H.; Tadokoro, S.; Noda, I.; Matsubara, H.; Takahashi, T.; Shinjou, A.; and Shimada, S. 1999. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, 739–743. IEEE.

Koenig, S.; Tovey, C.; Lagoudakis, M.; Markakis, V.; Kempe, D.; Keskinocak, P.; Kleywegt, A.; Meyerson, A.; and Jain, S. 2006. The power of sequential single-item auctions for agent coordination. In *Proceedings of the national conference on artificial intelligence*, volume 21, 1625. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Koenig, S.; Keskinocak, P.; and Tovey, C. 2010. Progress on agent coordination with cooperative auctions. In *Twenty-fourth aaai conference on artificial intelligence*.

Ma, H., and Koenig, S. 2016. Optimal target assignment and path finding for teams of agents. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 1144–1152. International Foundation for Autonomous Agents and Multiagent Systems.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2017a. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515*.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017b. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 837–845. International Foundation for Autonomous Agents and Multiagent Systems.

Ma, H.; Wagner, G.; Felner, A.; Li, J.; Kumar, T.; and Koenig, S. 2018. Multi-agent path finding with deadlines. *arXiv preprint arXiv:1806.04216*.

Nguyen, V.; Obermeier, P.; Son, T. C.; Schaub, T.; and Yeoh, W. 2017. Generalized target assignment and path finding using answer set programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017), Melbourne, Australia*, 1216–1223.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Silver, D. 2005. Cooperative pathfinding. *AIIDE* 1:117–122.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine* 29(1):9.