Fools Rush in Where Angels Fear to Tread in Multi-Goal CBS *

Grigorios Mouratidis[1], **Bernhard Nebel**[1], **Sven Koenig**[2]

[1]University of Freiburg
[2]University of Southern California
mouratig@cs.uni-freiburg.de, nebel@uni-freiburg.de, skoenig@usc.edu

Abstract

Research on multi-agent pathfinding (MAPF) has recently shifted towards problem variants that are closer to actual applications. Such variants often include the assignment of multiple goals to agents. To solve them, researchers have extended the Conflict Based Search (CBS) algorithm to multiple goals. This extension might look straightforward at first sight but it is tricky and this has already led to the development of algorithms that despite claiming to be optimal, return suboptimal solutions for some MAPF instances. In this paper, we provide a detailed analysis of the issue to raise awareness among the search community so that this mistake will not be perpetuated. Furthermore, a first evaluation against an optimal implementation is conducted which shows why this issue might have been difficult to spot. In only one of the randomly generated instances, the suboptimal behavior emerged.

Introduction

Multi-agent path finding (MAPF) is a subfield of AI planning, which has been heavily researched during the last few years due to its multiple practical applications. Such applications are, e.g., autonomous vehicles (Okoso et al. 2021), warehouse management (Hönig et al. 2019) and autonomous aircraft towing (Morris et al. 2016).

Solving MAPF (Stern et al. 2019) optimally is an *NP-complete* problem (Yu and LaValle 2013), where a set of m agents are dispersed on the vertices of a graph $G = (V, E)$ and each of them should reach its respective goal position while avoiding collisions with other agents. There are numerous approaches in the literature that solve MAPF either optimally (Felner et al. 2017) or suboptimally (Barer et al. 2014) with respect to an objective function. One of the most common objective functions is the *sum-of-costs*, which is the accumulated number of timesteps that the agents need to reach their goal positions. Alternatively, there is the objective function of *makespan*, which is the minimum timestep that all agents have reached their respective goals.

Lately, the research focus has shifted to variants of the original MAPF problem which could be used in real-life applications, where the assumptions of the original MAPF problem are not sufficient. Some of these variants focused on cases where the agents have to visit multiple goals, which can be either already assigned to specific agents but without being ordered (Surynek 2021) or where the goals can be visited by any agent (or a subgroup of them) (Ren, Rathinam, and Choset 2022). To tackle such problems optimally, *Conflict Based Search (CBS)* (Sharon et al. 2015) which is a state-of-the-art algorithm, was extended to incorporate the search of different possible goal allocations and sequences to agents.

CBS was initially developed to solve MAPF instances where each agent is assigned a single goal. Extending this algorithm to deal with MAPF variants where the agents are assigned multiple ordered or unordered goals has some subtleties that can get easily overlooked and in fact, have been already overlooked twice in the literature. As a result, the algorithms provided in these papers return suboptimal solutions for some MAPF instances, despite claiming optimality.

In the rest of this paper, we give a detailed analysis of the caveats of such CBS extensions by providing some minimal counter-examples for optimality along with some interesting insights about the reason that those extensions fail to find an optimal solution. Therefore, this paper has a broad impact on numerous approaches that extend CBS to tackle different MAPF variants with multiple goals. Subsequently, we prove that the cost difference between the suboptimal and optimal solutions can get arbitrarily large. Finally, we provide a first evaluation that shows that those CBS extensions will return an optimal solution, despite being suboptimal, for many randomly generated MAPF instances. This might have been one of the reasons that this subtle mistake is difficult to spot.

Related Work

The lifelong variant (Ma et al. 2017) of the multi-agent pathfinding problem, which is known as the multi-agent pickup and delivery problem (MAPD), was one of the first variants of MAPF where agents have to visit more than one goal. This problem deals with real-world scenarios in warehouses where there are several pickup and delivery tasks that the agents should carry out in a reasonable time while avoiding collisions. Two new algorithms, namely, TP and TPTS were developed, which can efficiently solve any instance in

*This is a proverb indicating that being in a hurry to act (like the agents who rush to their subgoals in multi-goal MAPF) without enough deliberation, can have unexpected consequences.

a specific subclass of MAPD instances.

Subsequently, a new algorithm called multi-label A* (MLA*) (Grenouilleau, van Hoeve, and Hooker 2019) was introduced to improve the TP algorithm. MLA* augments the state space of the classical A* algorithm and was coupled with TP to compute solutions for lifelong problems in the pickup and delivery domain. The benchmarks showed that the use of MLA* improved substantially the solutions in terms of makespan and service times in comparison to the original TP algorithm.

Another variant of MAPF, called multi-goal multi-agent pathfinding (MG-MAPF) (Surynek 2021), assumes that each agent is assigned a predefined unordered set of goals, which means that the agents should not only plan their paths optimally but also search for the optimal goal ordering. Two algorithms were developed, namely, HCBS and SMT-HCBS. HCBS is an extension of the CBS algorithm and SMT-HCBS is a compilation-based approach.

MSMP (Ren, Rathinam, and Choset 2021) and MCPF (Ren, Rathinam, and Choset 2022) are two generalizations of MAPF where the goals are dispersed on the map but are not allocated to any agent in the beginning. The goal is to find the optimal allocation of goals to agents, where each agent can be assigned multiple goals and then conflict-free joint paths must be computed. Both variants solve the problem by using techniques from the traveling salesman problem (Laporte 1992). MSMP combined it with subdimensional expansion, while MCPF used an extension of conflict-based search.

The aforementioned CBS extensions, namely HCBS and MCPF, are two algorithms that solve a different MAPF variant. However, both of them suffer from the same issue that renders them suboptimal.

Ordered Multi-Goal MAPF

To simplify the analysis, we will first consider a simple variant of the multi-goal MAPF problem where the multiple goals are already ordered. Later on, we will show that the identified problems also arise in more general variants of the problem.

Definition 1 (OMG-MAPF) ***Ordered multi-goal multi-agent pathfinding*** *is a* ***5-tuple*** $(\mathbf{G}, \mathbf{A}, \mathbf{s_0}, \mathbf{g}, \prec)$ *where* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *is an undirected graph,* $\mathbf{A} = (a_1, a_2, \ldots, a_n)$ *is a set of* n *agents,* $\mathbf{s_0} : A \rightarrow V$ *maps agents to their starting positions,* $\mathbf{g} : A \rightarrow 2^V$ *represents the set of goals for each agent and* $\prec$ *is a total order on* $g(a_i)$ *that represents the precedence constraints between the subgoals of the agents.*

In OMG-MAPF, each agent should visit the goals assigned to it in their specified order. To achieve that, the agents perform some actions. These actions are either *wait actions*, where the agent waits at its vertex for one timestep, or *move actions*, where the agent moves to any of its adjacent vertices in the graph. Formally, the actions are defined as a function $\alpha : V \rightarrow V$, such that $\alpha(u) = u'$ means that an agent that is located in vertex u and performs action α, will end up in vertex u'. Furthermore, it is assumed that time is discretized and all actions have *unit costs* and are completed in one *timestep*.

Agents' actions can lead to agents' collisions which are called conflicts. In OMG-MAPF, we account for two types of conflicts, the vertex conflicts and the edge conflicts. A *vertex conflict* occurs when two agents are located at the same vertex at the same time and an *edge conflict* occurs when two agents cross the same edge in opposing directions at the same time.

The following definition formalizes the notion of a single-agent plan (Stern et al. 2019) in OMG-MAPF domains.

Definition 2 (Single-agent plan) *Given an agent* a_i*, its starting position* s_i *and a sequence of actions* $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_k)$*, we define as* $\pi_i[t]$ *the location of agent* a_i *after executing the first* t *actions of* π*, beginning from its starting position* s_0*. Formally:*

$$\pi[t] = \alpha_t(\alpha_{t-1}(\ldots \alpha_1(s_0)))$$

A single-agent plan for agent a_i *in OMG-MAPF is a sequence of actions* π *which lead* a_i *to visit all the goals assigned to it in their specified order, beginning from its starting position. Formally:*

$$\forall (g_1, g_2, \ldots, g_k) \in g(a_i), \text{ where } g_1 \prec g_2 \prec \ldots \prec g_k$$
$$\exists t_1, t_2, \ldots, t_k \text{ so that } \pi_i[t_1] = g_1, \pi_i[t_2] = g_2, \ldots,$$
$$\pi_i[t_k] = g_k \text{ and } t_1 < t_2 < \ldots < t_k$$

Definition 3 (OMG-MAPF solution) *A solution to OMG-MAPF consists of a set of* n *single-agent plans, one for each agent.*

Definition 4 (OMG-MAPF valid solution) *A* valid *solution to OMG-MAPF is a solution where no conflicts occur while each agent executes its single-agent plan.*

Definition 5 (OMG-MAPF optimal solution) *An* optimal *solution to OMG-MAPF is a valid solution that minimizes the cost of an objective function.*

In an OMG-MAPF instance, there are multiple valid solutions. In the rest of this paper, we will demonstrate an algorithm that finds an optimal solution. It should also be stated that the waiting actions at the final destination of an agent still contribute to the sum-of-costs objective function unless the agent does not need to move again from its destination location. Finding an optimal solution to OMG-MAPF is an *NP-hard* problem since MAPF has been proved to be NP-hard and is a special case of OMG-MAPF where there is only one goal per agent.

Ordered Multi-Goal CBS High Level

Often, the go-to approach when solving a new variant of a problem is to alter or extend the scope of the state-of-the-art algorithms developed for the original problem. Thus, to solve OMG-MAPF, we decided to extend CBS to OMG-CBS, which is able to deal with multiple goals per agent.

OMG-CBS is very similar to CBS. It is a *two level* search where the *high level* is identical to that of CBS and creates a binary *constraint tree (CT)*. Each node of the CT contains a set of constraints and a solution along with its cost (sum of costs). On the high level, each time a new conflict between agents a_i and a_j emerges (a_i, a_j, u, t), the parent node is

split into two child nodes which inherit the constraints of their parent. Furthermore, to one of them, the constraint (a_i, u, t) is added to prevent agent a_i from visiting vertex u at timestep t, and to the other one the constraint (a_j, u, t) is added to do the same for agent a_j. Consequently, the low level is called for both nodes, to find a new path for the agent that the new constraint was imposed on.

Potential Low-Level Algorithms of Ordered Multi-Goal CBS

Standard CBS employs A* as the low-level algorithm. However, in the presence of multiple goals, this does not work any longer. In the rest of this paper, we will describe **two different approaches** to implement a low-level algorithm of OMG-CBS. First, we will present the *chaining* approach and present a counter-example exposing its limitations when coupled with OMG-CBS. Subsequently, we will present the *holistic* approach that avoids these issues.

The *chaining* approach is a straightforward extension of A* to a sequence of goals. A shortest path through a sequence of (sub-)goals is constructed by concatenating shortest paths between the sub-goals. However, due to the multi-agent nature of the problem, this approach will not necessarily find optimal solutions. As will be demonstrated in the next section, there are cases where the chaining low-level search renders OMG-CBS (and other possible multi-goal CBS extensions) suboptimal. This chaining approach is apparently used in some extensions of CBS to deal with multiple goals (Surynek 2021; Ren, Rathinam, and Choset 2022), which are more general than OMG-CBS, though. As we will discuss later, both approaches are susceptible to the problems we have identified.

To address this problem, we will present afterwards the holistic approach, which employs MLA* in order to guarantee overall optimality.

Interestingly, others have also made similar observations before us. Li et al. (2021) stated in a footnote in a slightly different context that what we have called the chaining approach does not guarantee optimality. Further, they also pointed out that MLA* can be a solution. However, they neither provided a detailed analysis nor provided a proof that MLA* is sufficient.

Chaining Low-Level Approach

The **chaining low-level** search consists of consecutive augmented space-time A* searches between the ordered goals of the agent while taking into account its constraints. Let's assume for example that the agent's a_i starting position is s_0, its ordered goals are $g(a_i) = (A, B)$ and its constraint set is empty $c_i = \{\}$. Then, the chaining low-level search will call initially A*$(s_0, A, t = 0, c_i)$ to find the optimal path of agent a_i to the first goal, starting from $s_0(a_i)$, then assuming that the agent arrives at A at timestep t_a, it will call subsequently A*(A, B, t_a, c_i) to compute the optimal path to the second subgoal. Finally, the paths will get concatenated and the resulting path will get returned. The pseudocode for the chaining low-level search is shown in Algorithm 1.

Algorithm 1: chaining low-level search

```
Given (G = (V, E), a_i, s_0, g, c_i)
s = s_0
t = 0
path = {}
for each goal in ordered g(a_i) do
    subpath = A*(s, goal, c_i, t)
    if no possible subpath then
        return False
    else
        s = goal
        t = arrival time at goal
        path = concatenate(path, subpath)
    end if
end for
return path
```

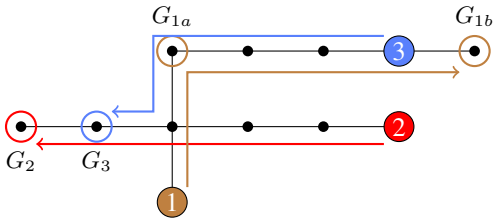

Figure 1: Graph problem instance

Counter-Example

We will prove now, by giving a counter-example, that OMG-CBS with chaining low-level search does not always return the optimal solution with respect to either the sum-of-costs or makespan objective function. In this section, every time we mention OMG-CBS, it should be assumed that its low-level search is the **chaining low level** search.

Let's assume that we have the OMG-MAPF problem instance depicted in Figure 1. In this instance, agent 1 is assigned two goals, namely, G_{1a} and G_{1b} where $G_{1a} \prec G_{1b}$. Whereas, agent 2 and agent 3 are assigned only one goal each which is G_2 and G_3 respectively.

Intuition Behind the Counter-Example

First, let us give an intuitive reason behind the failure of OMG-CBS to return the optimal solution. The plan of agent 1 in the optimal solution for both sum-of-costs and makespan objective functions is to initially wait for 4 timesteps at its starting position while the other agents move to their goals by following their optimal paths. After timestep 4, agent 1 follows its optimal path to visit both of its goals. This way, there are no conflicts between the agents and the **sum of costs** and **makespan** of this optimal solution are equal to **20** and **10** respectively.

The reason that OMG-CBS fails to find the aforementioned optimal solution lies in its greedy nature. The chaining low-level approach will **never** search for possible delays in the first subroute of agent 1 if no conflicts occur there. Hence, in cases of congestion like the one in our counter-

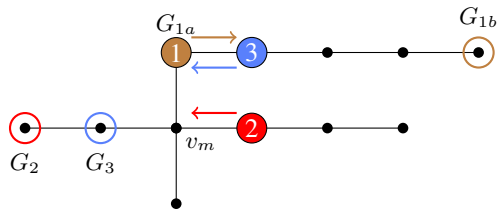

Figure 2: Edge conflict at timestep 3

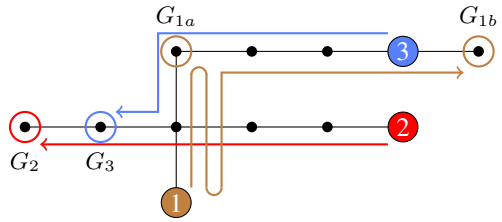

Figure 3: Agents' paths returned by OMG-CBS with chaining low level

example, the chaining low level will always lead agent 1 as fast as possible to G_{1a}, having as a result the return of a suboptimal solution.

Counter-Example Analysis for Sum of Costs

In the beginning, OMG-CBS will create an initial high-level node with the optimal paths of these agents found by the chaining low-level search. The sum of costs of this node will be 16 and the paths of the agents can be seen in Figure 1.

Subsequently, the high-level search will find the edge conflict between agent 1 and agent 3 at timestep 3 which is shown in Figure 2. Since all the possible paths of both agent 1 and agent 3 which lead to their respective goals, include the vertices G_{1a} and v_m, the only way to resolve this conflict will be for agent 1 to retreat to its initial position, to let agent 3 pass through.

The retreat of agent 1 will lead to a vertex conflict with agent 2 at v_m at timestep 3. To resolve this conflict, either of the two agents will have to wait for 1 timestep at its position to let the other one pass.

After 2 more trivial conflict resolutions, OMG-CBS will find its best solution. This occurs when agent 1 waits at timestep 3 at G_{1a} to let agent 2 pass and the sum of costs of this solution is **23** which includes the cost of 20 move actions, 2 wait actions of agent 1 at timesteps 3 and 6 and 1 wait action of agent 3 at timestep 3. The agents' paths for the best solution (without showing their wait actions) are depicted in Figure 3. The cost of this solution is higher than the cost of the optimal solution (SoC = 20) presented in the previous subsection. Therefore, the returned solution of OMG-CBS is suboptimal with respect to the sum-of-costs objective function.

Counter-Example Analysis for Makespan

The main conflicts described in the counter-example above will occur again in case OMG-CBS searches for the optimal solution with respect to the makespan. The previous best solution for the sum of costs is also the best solution with respect to the makespan. The makespan of the solution is equal to the cost of the path of agent 1 which is **12**. This is higher than the optimal solution's makespan of 10. Hence, the returned solution of OMG-CBS is also suboptimal with respect to the objective function of makespan.

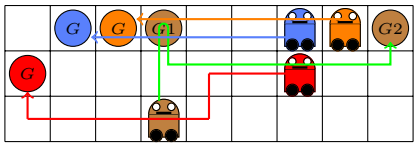

Figure 4: Grid counter-example for SoC

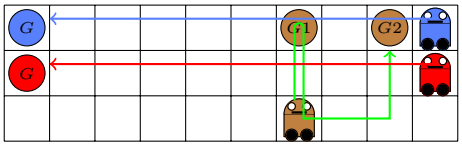

Figure 5: Grid counter-example for makespan

Counter-Examples in Grid Environments

The graph counter-examples provided in the two previous sections cannot be directly translated to grid environments. Therefore, we also provide a grid counter-example for the sum-of-costs objective function in Figure 4 and another one for the objective function of makespan in Figure 5. In both counter-examples, the brown agent is assigned the goals G_1 and G_2 where $G_1 \prec G_2$ and the other agents are assigned only one goal which has their respective color.

In the optimal solution of both counter-examples, the brown agent waits in its initial position for three timesteps while the rest of the agents follow their optimal paths to the goals. At timestep 4, the brown agent starts moving, following his optimal path to G_1 and then to G_2. This way, no conflict occurs and the sum of costs of the first counter-example amounts to **26**, while the makespan of the second counter-example amounts to **9**.

However, OMG-CBS with chaining low-level will not return the optimal solutions, but the suboptimal ones depicted in the figures. In this case, the SoC of the solution in Figure 4 amounts to **27** and the makespan of the solution in Figure 5 is **10** (the red agent performs a single wait action at timestep 3).

Worst-Case Analysis

In the graph counter-example, the sum-of-cost difference between the optimal solution and the one that OMG-CBS with chaining low level returned is only three. So, one question that one may ask is whether there are examples with larger differences. The difference can become arbitrarily large, as will be demonstrated by the family of instances shown in Figure 6. There are x agents and x corresponding goals per row. The brown agent C is assigned the goals (C_a, C_b) with precedence constraint $(C_a \prec C_b)$, and each of the rest of the

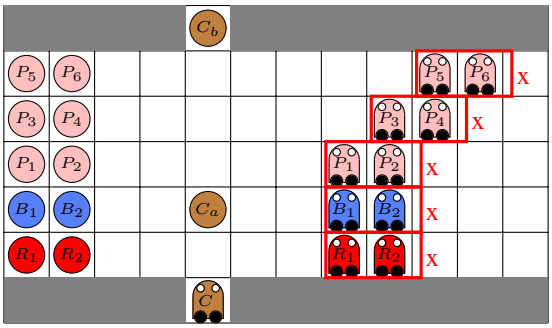

Figure 6: Worst case instance

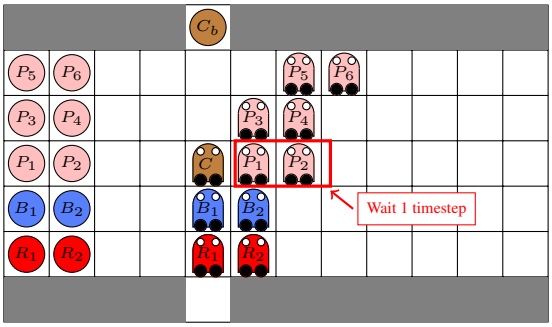

Figure 7: Case 2

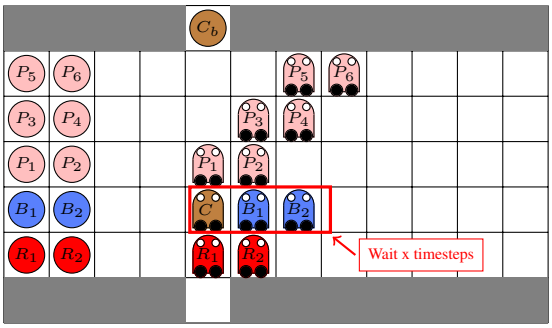

Figure 8: Case 3

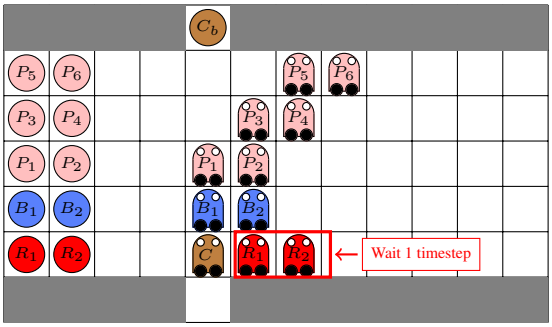

Figure 9: Red agents' waiting actions in case 4

agents is assigned the goal marked with the respective agent name.

If we don't account for any conflicts, every agent will follow its optimal path to the goal and the solution cost will be equal to

$$9x + 8x + 7(3x) + 6.$$

This is just a lower bound to the true solution cost. OMG-CBS will search many different possible joint agents' paths along its way of resolving conflicts. Since we cannot present and analyze all of them, we are going to show how the four most predominant cases scale with respect to the number of agents.

Case 1: Optimal Solution

The optimal solution is the one in which the brown agent waits for (x+2) timesteps in its initial position and then follows its fastest possible path to C_a and C_b. This solution cost will be equal to

$$9x + 8x + 7(3x) + 6 + (x + 2) = 39x + 8.$$

Case 2: Go to C_a and C_b as Fast as Possible

One other interesting solution is when the brown agent moves as fast as possible initially to C_a and then to C_b, while the other agents move towards their goals. As a result, each of the pink agents will have to wait for 1 timestep which will add an extra 3x cost to the lowest bound solution cost:

$$9x + 8x + 7(3x) + 6 + 3x = 41x + 6.$$

In Figure 7 you can see that in timestep 3, the first row of pink agents has to wait for 1 timestep, which adds a cost of x to the lowest bound cost. Each of the other pink agents will have to wait for 1 timestep too, while the brown agent moves towards C_b.

Case 3: Go to C_a and Wait

Another possible solution would be for the brown agent to go to C_a and then wait there till the pink agents pass by. This will cause the brown agent and each of the x blue agents to wait for x timesteps. This becomes more clear in Figure 8. Thus, the solution cost will be

$$9x + 8x + 7(3x) + 6 + (x + x^2) = x^2 + 39x + 6.$$

Case 4: Go to C_a and Retreat

The last case is when the brown agent visits C_a and then retreats to the initial position to let the other agents pass by. In this case, the brown agent needs extra 4 actions in order to move to the initial position and back to C_a later which will add an extra cost of 4 to the SoC. Furthermore, each of the red agents has to wait for 1 timestep as shown in Figure 9 and the brown agent should also wait in the initial position for x-1 timesteps as shown in Figure 10. Thus, the solution cost will be

$$9x + 8x + 7(3x) + 6 + (4 + x + (x - 1)) = 40x + 9.$$

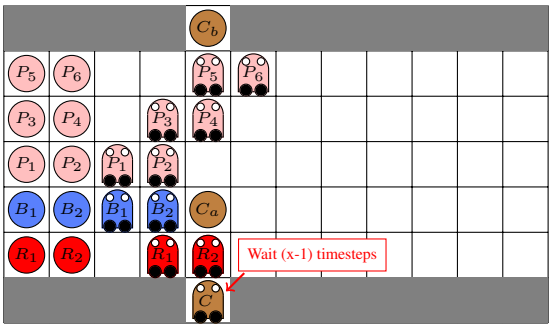

Figure 10: Brown agent's waiting actions in case 4

Solution Cost Difference

OMG-CBS with chaining low level will always return the solution of case 4 when the number of agents per row is greater or equal to 4. This solution obviously scales worse than the optimal solution (case 1). More precisely, the cost difference between these two solutions will increase linearly with the number of agents per row, and it can get arbitrarily large.

Holistic Low-Level Approach

To render OMG-CBS optimal, the low-level search should extend its scope to consider the cost of the whole path of the agents, beginning from their starting position till their final goal position. This **holistic low level** approach can be implemented using an adjusted version of multi-label A* (MLA*) for multiple goals (Zhong et al. 2022). For the rest of this paper, MLA* for multiple goals will be mentioned as MLA* since it is just a straightforward extension of the original algorithm which was originally developed for the multi-agent pickup and delivery problem.

For each low-level node n of MLA* we define the following variables:

- $\mathbf{g_n}$: The agent's path cost.
- $\mathbf{p_n}$: The current position of the agent.
- $\mathbf{l_n}$: The flag that indicates which subgoal the agent approaches e.g. $l_n = 1$ means that the agent approaches the first subgoal.

We also define the set $d = \{d_0, d_1, d_2, \ldots, d_m\}$ along with a total order $\prec$ on d which defines the precedence relations among its elements. d_0 is the agent's starting position and $d_1, d_2, \ldots, d_m$ are the positions of the agent's goals. The f_n value is defined as $f_n = g_n + h_n$ where h_n is the heuristic value of node n and is defined as follows, where $h(x, y)$ is the length of the shortest spatial path between the vertices x and y.

$$h_n = h(p_n, d_{l_n}) + \sum_{i=l_n+1}^{m} h(d_i, d_{i+1}) \qquad (1)$$

It should be stated that h_n could be any other *consistent* heuristic. Moreover, the optimal distance cost between any two vertices could be precomputed and saved in a lookup table to speed up the MLA* search.

Algorithm 2: MLA*

1: $d \leftarrow \{d_0, d_1, \ldots, d_m\}$, where $d_0 \prec d_1 \prec \ldots \prec d_m$
2: $n_0 \leftarrow Node(g_{n_0} = 0, l_{n_0} = 1, f_{n_0} = h_{n_0}, p_{n_0} = d_0)$
3: Insert(n_0, OPEN)
4: **while** OPEN **not** EMPTY **do**
5: $\quad n \leftarrow Node(g_n, l_n, f_n, p_n)$ from OPEN with min f_n
6: $\quad$ **if** $p_n = d_{l_n}$ **and** p_n is **not** d_m **then**
7: $\quad\quad l_n = l_n + 1$
8: $\quad$ **end if**
9: $\quad$ **if** $p_n = d_{l_n}$ **and** p_n is d_m **then**
10: $\quad\quad$ **Return** found path
11: $\quad$ **end if**
12: $\quad$ **Expand** n by adding to OPEN, nodes with adjacent locations that are not blocked by other agents or blocked cells in the environment
13: $\quad$ **Remove** n from OPEN
14: **end while**
15: **Return** False

MLA* is actually an A* search in the augmented space-time-label (u, t, l) state space. The allowed transitions in this state space are the following:

- $(u, t, l) \rightarrow (v, t + 1, l)$, if $(u, v) \in E$ or $u = v$
- $(u, t, l) \rightarrow (v, t + 1, l + 1)$, if $(u, v) \in E$ and $v = d_l$

Furthermore, the state space is reduced by the constraints of the agent in the same way that the state space is reduced by constraints in space-time A*. The pseudocode of MLA* is provided in Algorithm 2.

For an A* search to be optimal, it suffices that the heuristic that guides the search is admissible. Hence, in this case, we just need to prove that the heuristic of MLA* is admissible. The heuristic computes the shortest distance from the current position of the agent till the next subgoal and adds to it the shortest distances between the consecutive subgoals till the final goal. Since this distance measure gives the lower bounds for the needed steps, this heuristic will always underestimate the cost of reaching the goal. Therefore, the heuristic is **admissible**, and MLA* is **optimal**.

Proposition 1 *MLA* is optimal.*

OMG-CBS Optimality

The CBS' optimality proof is based on two definitions and two Lemmas (Sharon et al. 2015). This proof doesn't hold in the case of OMG-CBS with chaining low level because Lemma 1 in the original publication does not hold. To be more precise, the first definition and the first Lemma state:

Definition 6 *For a given node N in a constraint tree, let CV(N) be the set of all solutions that are: (1) consistent with the set of constraints of N and (2) are also valid (i.e, without conflicts).*

Lemma 1 *The cost of a node N in the CT is a lower bound on minCost(CV(N)), where minCost(CV(N)) is the minimum cost over all solutions in CV(N).*

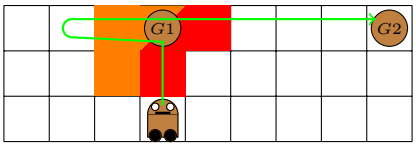

Figure 11: Suboptimal chaining low-level solution

This is not the case when we have a chaining low level. Let's see the output of the chaining low level in Figure 11, where the cells in red color are the vertex constraints at timestep 3 and the cells in orange are the vertex constraints at timestep 4. At the goal vertex G_1, there is a vertex constraint both at timesteps 3 and 4. The chaining low level will return the solution depicted with the green arrow, which has a cost of 11. This happens because the chaining low level will move the agent as fast as possible to G_1 and then, due to congestion, the agent will have to retreat to the left. As a result, the returned solution has a higher cost than the optimal one, which is when the brown agent waits for 3 timesteps in its initial position and then follows its fastest path to the goals (cost = 10). Thus, the cost of the node with these constraints in the CT won't be a lower bound on minCost(CV(N)), which renders the Lemma false.

To prove that OMG-CBS with MLA* at the low level is optimal, we make use of the original proof construction of conflict-based search (Sharon et al. 2015). This proof construction always holds because Lemma 1 is true when MLA* is used as the low level of OMG-CBS. This is the case because MLA* always finds a lower bound on minCost(CV(N)), as stated in Proposition 1.

Experiments

To test the soundness of our theoretical results, but also to check the runtime differences and the frequency and size of the solution cost differences between the chaining and holistic low-level methods, OMG-CBS with chaining and holistic low level was implemented[1] in C++ and a series of experiments were conducted using a quad-core Intel Xeon 2.8G Hz and 128GB RAM.

Initially, we tested the worst-case scenario with 3-6 agents per row, since larger instances could not be tackled by OMG-CBS due to memory constraints. OMG-CBS with a holistic low level always returned the optimal solution whose cost followed the formulae provided in case 1 of the worst-case analysis. OMG-CBS with chaining low level always provided the solution of case 4 (or a slightly different one) whose cost also followed the formulae provided for this case.

Furthermore, a series of experiments were conducted using 6 maps in total. Two empty maps, namely, empty-8-8 and empty-16-16 were taken directly from `movingai.com`. Subsequently, two maps with dimensions 16x16 and random obstacles were created, namely, random-16-16-10 with 10% random obstacles and random-16-16-20 with 20% random obstacles. Finally, two maze maps with dimensions 16x16 were also created. For the small map (empty-8-8), 20 instances were created per number of agents and number of goals, where the number of agents varied between 2 and 10 and the number of goals varied between 2 and 5. For the rest of the maps, the same number of instances per configuration were created, but the number of agents varied from 2 to 13, whereas the number of goals per agent remained between 2 and 5. The starting positions of the agents along with their goal positions were produced randomly and a time limit of 15 minutes was set for each instance.

The results showed that there was only one instance of one of the maze maps, where OMG-CBS with chaining low level found a suboptimal solution and its solution cost was only 1 timestep longer than the optimal one. This was not surprising since the counter-examples signal that the type of conflicts, that are responsible for the suboptimality of the chaining low level, occur mainly in maps with high agent congestion.

Figure 12 shows that the optimal OMG-CBS is mainly up to an order of magnitude slower than its suboptimal counterpart. This was to be expected since MLA* has to generate on average more nodes than the chaining low-level approach for the same single agent problem. This becomes clear in Figure 13 that compares the average low-level nodes expanded per high-level node of every instance for both approaches.

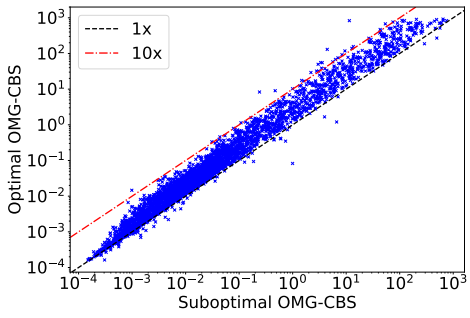

Figure 12: Runtime comparison between OMG-CBS with chaining low level (suboptimal) and OMG-CBS with MLA* as low level (optimal) for each problem instance

MAPF Variants with Unordered or Unassigned Goals

So far, we have only considered CBS with ordered goals (OMG-CBS). However, as mentioned in the beginning, there are also scenarios, when the goals are unordered and one has to find the best goal ordering (Surynek 2021), which has been called MG-MAPF. A CBS extension dealing with this scenario, which can be called MG-CBS, faces the same problems as OMG-CBS. In particular, the counter-example presented in Figure 1 is also a counter-example to the optimality of MG-CBS with a chaining low-level algorithm. The reason is that each solution with a goal order that reverses G_{1a} and G_{1b} is obviously longer than the one we considered. In other words, in this case, it is enough to consider the goal ordering we analyzed. In the specific paper

[1] https://github.com/GrFr/libMultiRobotPlanning

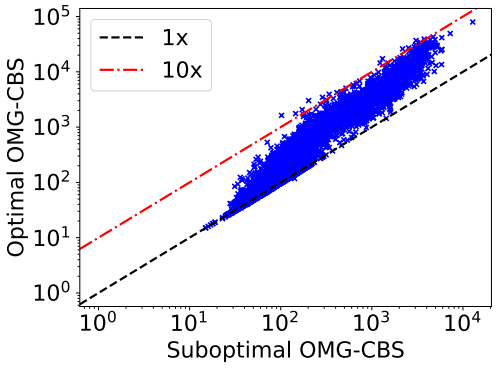

Figure 13: Comparison of the average low-level nodes expanded per high-level node between OMG-CBS with chaining low level (suboptimal) and OMG-CBS with MLA* as low level (optimal) for each problem instance

of MG-MAPF, an algorithm called HCBS was developed to tackle the problem. HCBS should find the best goal ordering (along with the agents' paths) which minimizes the SoC objective function. Though, it is clear from the pseudocode that the chaining approach is used. In lines 32-33, the algorithm calls A* for each goal that is yet unvisited (starting from the current vertex of the agent) and finds the shortest path to it, then the paths get concatenated (line 41). Thus, the agents are guided as fast as possible to the next goal with the aforementioned consequences.

There exist even more general MAPF variants, where the potential goals have to be assigned to the agents, a setting which has been called MCPF (Ren, Rathinam, and Choset 2022). One can extend CBS to deal with this setting by extending the high-level search to a forest search (Hönig et al. 2018) where there is a root node for every possible assignment and sequence of the goals to agents, which can then be solved by OMG-CBS. Since the optimality of this forest search is based on the optimality of each tree search, the chaining approach on the low level is not sufficient for optimality. In the specific MCPF paper, a two-level forest search algorithm called CBSS was developed. In the analysis of CBSS' low level, it is not stated explicitly that a chaining approach is used. However, since the repository of the implementation is provided, we checked it out and found out that the chaining approach is indeed used. Finally, we transformed our counter examples to the problem formulation of the paper (since their approach solves a very similar problem) and their algorithm indeed returned the anticipated suboptimal solutions.

Conclusion & Discussion

We have presented an analysis of possible extensions to the CBS algorithm dealing with multiple goals. We considered two low-level search approaches, namely, the chaining and the holistic one. For the first approach, we presented an example demonstrating non-optimality. In addition, it was shown that the solution cost difference between the two low-level approaches can get arbitrarily large. We then proved that a holistic low-level approach employing MLA* is guaranteed to find an optimal solution to the problem. However, the experimental evaluation showed that OMG-CBS with chaining low level almost always finds an optimal solution. This can be attributed to the rare occurrence of the types of conflicts, which render OMG-CBS with chaining low level suboptimal, in environments with low congestion.

An alternative approach to MLA* which could potentially speed up the low level search of OMG-CBS would be to use an extension of safe interval path planning (SIPP) (Phillips and Likhachev 2011) similar to the one used in the continuous-time CBS with disjoint splitting (CCBS-DS) (Andreychuk et al. 2021). SIPP in CCBS-DS searches all the SIPP nodes that could reach a milestone and then uses all these nodes as root nodes to start the next search to find all nodes that reach the next milestone. A milestone in CCBS-DS is a time interval on a location, therefore SIPP could be extended to tackle multiple ordered goals optimally in an OMG-MAPF setting.

We hope that the insights provided in this paper will help fellow researchers to understand better the possible caveats regarding CBS' extensions for multiple goals, which might lead to further improvements in future algorithms.

Acknowledgements

The first author has been supported by the state graduate funding program of the federal state of Baden-Wuerttemberg.

References

Andreychuk, A.; Yakovlev, K. S.; Boyarski, E.; and Stern, R. 2021. Improving Continuous-time Conflict Based Search. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 11220–11227. AAAI Press.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. AAAI Press.

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 *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA*, 29–37. AAAI Press.

Grenouilleau, F.; van Hoeve, W.; and Hooker, J. N. 2019. A Multi-Label A* Algorithm for Multi-Agent Pathfinding. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, 181–185. AAAI Press.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2018. Conflict-Based Search with Optimal Task

Assignment. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, 757–765. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics Autom. Lett.*, 4(2): 1125–1131.

Laporte, G. 1992. The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. *European Journal of Operational Research*, 59(2): 231–247.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 11272–11281. AAAI Press.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, 837–845. ACM.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016*, volume WS-16-12 of *AAAI Technical Report*. AAAI Press.

Okoso, A.; Okumura, B.; Otaki, K.; and Nishi, T. 2021. Network-Flow-Problem-Based Approach to Multi-Agent Path Finding for Connected Autonomous Vehicles. In *24th IEEE International Intelligent Transportation Systems Conference, ITSC 2021, Indianapolis, IN, USA, September 19-22, 2021*, 1946–1953. IEEE.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, 5628–5635. IEEE.

Ren, Z.; Rathinam, S.; and Choset, H. 2021. MS*: A New Exact Algorithm for Multi-agent Simultaneous Multi-goal Sequencing and Path Finding. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, 11560–11565. IEEE.

Ren, Z.; Rathinam, S.; and Choset, H. 2022. Conflict-Based Steiner Search for Multi-Agent Combinatorial Path Finding. In *Robotics: Science and Systems XVIII, New York City, NY, USA, June 27 - July 1, 2022*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artif. Intell.*, 219: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, 151–159. AAAI Press.

Surynek, P. 2021. Multi-Goal Multi-Agent Path Finding via Decoupled and Integrated Goal Vertex Ordering. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 12409–12417. AAAI Press.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2013, July 14-18, 2013, Bellevue, Washington, USA*, 1443–1449. AAAI Press.

Zhong, X.; Li, J.; Koenig, S.; and Ma, H. 2022. Optimal and Bounded-Suboptimal Multi-Goal Task Assignment and Path Finding. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, 10731–10737. IEEE.