Streaming Multi-agent Pathfinding

Mingkai Tang[1] , **Lu Gan**[1] and **Kaichen Zhang**[2]
[1]Hong Kong University of Science and Technology
[2]Hong Kong University of Science and Technology (Guangzhou)
{mtangag, lganaa, kzhangbi}@connect.ust.hk

Abstract

The task of the multi-agent pathfinding (MAPF) problem is to navigate a team of agents from their start point to the goal points. However, this setup is unsuitable in the assembly line scenario, which is periodic with a long working hour. To address this issue, the study formalizes the streaming MAPF (S-MAPF) problem, which assumes that the agents in the same agent stream have a periodic start time and share the same action sequence. The proposed solution, Agent Stream Conflict-Based Search (AS-CBS), is designed to tackle this problem by incorporating a cyclic vertex/edge constraint to handle conflicts. Additionally, this work explores the potential usage of the disjoint splitting strategy within ASCBS. Experimental results indicate that ASCBS surpasses traditional MAPF solvers in terms of runtime for scenarios with prolonged working hours.

1 Introduction

The assembly line is a manufacturing process where workpieces are moved between workstations for semi-assembly until final assembly. It is commonly applied in industries such as cars, airplanes, and consumer electronics [Boysen *et al.*, 2022]. Workpieces can be transferred using a conveyor, which can only move them along fixed paths, making adjustments difficult. An alternative method involves using a team of robots to transport workpieces. The main challenge is navigating robots between workstations without collisions. This challenge is known as the multi-agent pathfinding (MAPF) problem and has been extensively studied. The MAPF problem is applied in various real-world scenarios, including warehouse management [Li *et al.*, 2021; Xu *et al.*, 2022; Zhang *et al.*, 2024], traffic control [Ho *et al.*, 2019; Li *et al.*, 2023], and pipe design [Belov *et al.*, 2020].

The assembly line operates with a production rhythm. Each workstation periodically receives a workpiece and produces a higher assembly workpiece. The time interval for the period is defined as the cycle time. In the view of the robot team, they need to transport the workpiece between workstations every cycle time. The traditional MAPF problem can be adapted for planning in a periodic scenario with finite working hours, which describes the agent by the start point, the goal point, and the start time. However, in assembly line factories with long working hours, the number of agents grows rapidly, implying that the running time for computing the optimal solution increases exponentially due to the NP-hardness of the MAPF problem [Banfi *et al.*, 2017]. Additionally, in environments where humans and robots share space, the predictability of robot motions is crucial, which cannot be achieved by the traditional MAPF problem setup.

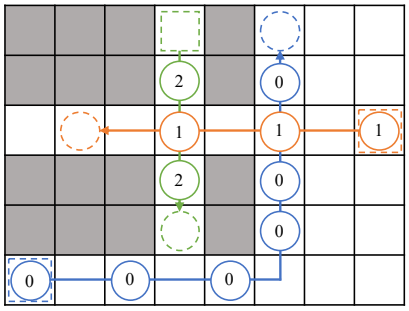

Figure 1: A snapshot of the S-MAPF problem where the cycle time is 2. The snapshot is taken at a time step of $2 \times k$ where k is a sufficiently large integer. White cells are feasible, while grey cells are infeasible. The dashed square marks the start point of the agent stream, and the dashed circle indicates the goal point. The solid circle represents an agent in the stream, with the number denoting the stream ID. Arrows depict the cells that the path crosses. The initial start times for agent streams 0, 1, and 2 are 0, 0, and 1, respectively, with action sequences 'RRRRRUUWUUU', 'LLLLLL', and 'DDDD'. 'U' stands for up, 'D' for down, 'L' for left, 'R' for right, and 'W' for wait.

Hence, we suggest planning with *agent streams* to solve the problems of long working hours and predictability. An agent stream is an agent group containing an infinite number of agents, where agents in an agent stream depart from the start vertex every cycle time and share the same action sequence. At both the start and goal points, there are designated private parking zones for robot loading and unloading. This setup ensures that agents disappear after reaching their goals without disrupting others. It is also adopted in the online MAPF problem [Švancara *et al.*, 2019]. Formally describing an agent stream, let s be the initial start time of the agent stream and c be the cycle time. Within the agent stream, there exists an

agent departing at time step $k \cdot c + s$, where $k \in \mathbb{N}$. We term the task of finding a collision-free path for each agent stream from its start point to its goal point as *streaming multi-agent pathfinding* (S-MAPF). Figure 1 shows an example of the S-MAPF problem. The S-MAPF solution offers endless working hours availability and predictable agent movement when collaborating with humans due to the consistent action sequence. To solve the S-MAPF problem, the Agent Stream Conflict-Based Search (ASCBS) algorithm is proposed, ensuring both optimality and completeness. To resolve the conflict between different agent streams in ASCBS, we introduce the cyclical vertex constraint and the cyclical edge constraint. Additionally, the disjoint splitting strategy [Li *et al.*, 2019b] is tailored to suit the S-MAPF problem setting. The main contributions of this paper are as follows.

- We propose the formalization of the S-MAPF problem, which can be adopted in the scenario with unlimited working hours.
- We present a two-level approach, ASCBS, to optimally solve the S-MAPF problem. The cyclical vertex constraint and cyclical edge constraint are proposed to resolve agent stream conflicts. We also explore the potential use of disjoint splitting within ASCBS.
- We conduct a comprehensive experimental evaluation to compare the computational efficiency of different ASCBS variants. Additionally, an experiment is done that demonstrates ASCBS's high computational efficiency over traditional MAPF solvers in scenarios with long working hours.
- We introduce several extensions of the S-MAPF problem that relax certain assumptions.

2 Related Work

Some MAPF problem variants are introduced to address real-world applications. For example, the MAPF problem with delay probabilities [Ma *et al.*, 2017] accounts for situations where agents may experience delays in their paths. The MAPF problem for large agents [Li *et al.*, 2019c] assumes that the agent may occupy more than one vertex. In this study, we introduce the S-MAPF problem tailored to the assembly line scenario, where agents can form streams to continuously transport workpieces. To our knowledge, there are two closely related works for a similar scenario. The Precedence Constrained Multi-Agent Task Assignment and Path-Finding Problem [Brown *et al.*, 2020] is also applied in assembly, but it focuses on planning for a single final product and emphasizes precedence constraints for operations. The Periodic Multi-Agent Path Planning Problem [Kasaura *et al.*, 2023] assumes agents' periodic appearances but is defined in continuous space. It is solved as a continuous optimization problem, limiting its applicability to small-scale instances.

In the field of MAPF, conflict-based search [Sharon *et al.*, 2015] is a well-studied optimal solver, with techniques proposed to accelerate the search process, such as Prioritize Conflicts [Boyarski *et al.*, 2015], Disjoint Splitting [Li *et al.*, 2019b], and Mutex Reasoning [Zhang *et al.*, 2022]. This study introduces a variant of conflict-based search to solve the S-MAPF problem, incorporating some of these acceleration techniques into the algorithm.

3 Problem Definition

The S-MAPF problem instance is denoted by the tuple $\langle G, c, AS \rangle$. In this case, $G = (V, E)$ is an undirected graph with unit edge length, and c denotes the cycle time shared by all agent streams. The set $AS = \{as_0, as_1, ..., as_{n-1}\}$ represents the agent streams, where n is the number of agent streams. Each as_i can be described as $\{v_i^s, v_i^g, t_i^s\}$ where v_i^s and v_i^g are the start and goal vertices, respectively. The initial start time of as_i is denoted by $t_i^s \in [0, c-1]$. At each time step $k \cdot c + t_i^s$ for all $k \in \mathbb{N}$, an agent of as_i will appear at v_i^s. After that, at each time step, the agent can take an action to move to the neighboring vertex or wait at the current vertex. Agents in the same agent stream are assumed to share the same action sequence, which implies that they also have the same path. In addition, the agent is assumed to disappear after reaching its goal vertex.

The objective of the S-MAPF problem is to find a collision-free path for each agent stream from its start vertex to its goal vertex. The solution can be denoted as $P = \{p_0, p_1, ..., p_{n-1}\}$, and the path of as_i is $p_i = [p_i^0, p_i^1, ..., p_i^{l_i-1}]$, with l_i as the path length. We name p_i^j the j-th *step* of agent stream as_i. For an agent whose start time is $k \cdot c + t_i^s$ for a specific k, this agent will be at the vertex $p_i^{t-(k \cdot c + t_i^s)}$ at the time step t when $k \cdot c + t_i^s \leq t < k \cdot c + t_i^s + l_i$. A solution is collision-free if and only if there are no *cyclic vertex conflicts* and *cyclic edge conflicts*. Formally, the cyclic vertex conflict is denoted as $\langle as_i, as_j, q_i, q_j, v \rangle$. It occurs if and only if there exist two nonnegative integers k_i, k_j such that the following equations are satisfied:

$$p_i^{q_i} = v = p_j^{q_j} \tag{1}$$

$$(k_i \cdot c + t_i^s) + q_i = (k_j \cdot c + t_j^s) + q_j \tag{2}$$

$$(i-j)^2 + (k_i - k_j)^2 \neq 0 \tag{3}$$

It means that the agent of as_i with start time $k_i \cdot c + t_i^s$ is located at the same vertex v at the time step $(k_i \cdot c + t_i^s) + q_i$ as the agent of as_j with start time $k_j \cdot c + t_j^s$. In addition, when as_i and as_j are the same agent stream ($j - i = 0$), k_i equal to k_j won't cause a conflict because these two agents are the same agent. We use $\langle as_i, as_j, q_i, q_j, v_x, v_y \rangle$ to denote the cyclic edge conflict. It occurs if and only if there exist two non-negative integers k_i, k_j such that the following equations are satisfied:

$$p_i^{q_i} = v_x = p_j^{q_{j+1}} \wedge p_i^{q_{i+1}} = v_y = p_j^{q_j} \tag{4}$$

$$(k_i \cdot c + t_i^s) + q_i = (k_j \cdot c + t_j^s) + q_j \tag{5}$$

$$(i-j)^2 + (k_i - k_j)^2 \neq 0 \tag{6}$$

It implies that the agent of as_i with start time $k_i \cdot c + t_i^s$ crosses the same edge (v_x, v_y) at the time step $(k_i \cdot c + t_i^s) + q_i$ with the agent of as_j with start time $k_j \cdot c + t_j^s$. Similar to the cyclic vertex conflict, when as_i and as_j are the same agent stream, k_i equal to k_j won't cause a conflict. Notably, Equations 2 and 5 are the same and can be further derived:

$$t_i^s + q_i \equiv t_j^s + q_j \ (mod\ c) \tag{7}$$

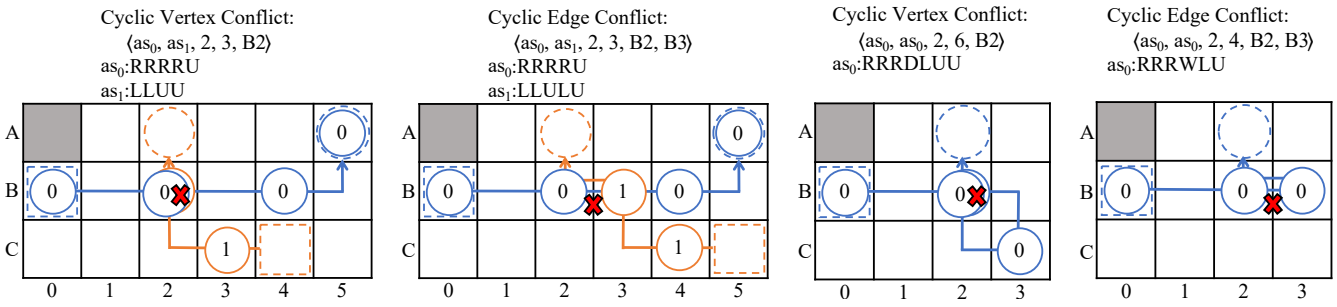

Figure 2: Examples of the conflicts where the cycle time is 2. The time step for the snapshot is $2 \times k$, where k is a sufficiently large integer. The initial start times of agent streams 0 and 1 are 0 and 1. The conflict and the action sequences are shown at the top of the figure.

It should be noted that as_i and as_j in the cyclic vertex conflict $\langle as_i, as_j, q_i, q_j, v\rangle$ and cyclic edge conflict $\langle as_i, as_j, q_i, q_j, v_x, v_y\rangle$ are not necessarily different, because the agents of the same agent stream might have vertex and edge conflicts. Figure 2 presents several examples of the conflicts.

Referencing the MAPF problem, we adopt the sum-of-cost (SOC) for the objective function. Let $C(P)$ denote the SOC of the solution P. It can be computed by the equation:

$$C(P) = \sum_{i=0}^{n-1}(l_i - 1) \tag{8}$$

An optimal solution is the one that is collision-free and has the lowest SOC compared to other collision-free solutions.

Theorem 1. *The S-MAPF problem is NP-hard to solve optimally.*

Theorem 1 can be proven by reduction, and the details are provided in the Supplementary Material.

4 Agent Stream Conflict-Based Search

We present a two-level algorithm, ASCBS, designed to solve the S-MAPF problem. The high-level solver creates a constraint tree (CT) to resolve conflicts from different or the same agent streams. The low-level solver uses the A* algorithm [Hart *et al.*, 1968] to plan the shortest path for an individual agent stream.

4.1 High-Level Solver

The high-level solver uses CT to avoid conflicts. Each node in the CT stores the constraint set, the path of each agent stream, and the corresponding SOC.

The pseudocode of the high-level solver is presented in Algorithm 1. In lines 1 $\sim$ 4, the low-level solver is invoked at the root node to calculate the shortest path for each agent stream without considering the conflict. During each iteration, the algorithm obtains the CT node with the minimum cost (Line 9) and detects and selects the best conflict (Line 11). The algorithm detects vertex conflicts by Equations 1, 7, 3 and edge conflicts by Equations 4, 7, 6. The selection of the best conflicts is based on Prioritize Conflicts [Boyarski *et al.*, 2015]. The prioritization process for cyclic vertex conflicts and cyclic edge conflicts is similar to that for the vertex conflict and the edge conflict. A multi-value decision diagram (MDD) is constructed for each agent stream to evaluate conflict priority. By examining the number of occurrences of conflicts at a layer with a width of 1 in the MDD of two associated agent streams, a conflict with a larger number is given higher priority. In Lines 12$\sim$14, if no conflicts are found, the optimal solution is reached, and the algorithm terminates. In the case of a conflict that involves two distinct agent streams, two cyclic vertex constraints or cyclic edge constraints are generated to resolve the conflict. Alternatively, if the conflict is from the same agent stream, two vertex constraints or edge constraints are established (Line 15). Further details will be explained in the following paragraph. In Lines 16 $\sim$ 24, two

Algorithm 1 $ASCBS_{high}$

Input: agent streams AS, graph G, cycle time c

1: $R \leftarrow$ new node
2: $R.cons \leftarrow \emptyset$
3: **for** each agent stream as in AS **do**
4: $\quad R.paths[as] \leftarrow ASCBS_{low}(as, R.cons, c)$
5: **end for**
6: $R.cost \leftarrow$ calculate the SOC of $R.paths$.
7: $OPEN \leftarrow \{R\}$
8: **while** $OPEN \neq \emptyset$ **do**
9: $\quad N \leftarrow$ minimum cost node from $OPEN$.
10: $\quad OPEN \leftarrow OPEN \backslash \{N\}$
11: $\quad F \leftarrow$ the best collision in N
12: $\quad$ **if** F is $None$ **then**
13: $\quad\quad$ **return** $N.paths$
14: $\quad$ **end if**
15: $\quad Cons \leftarrow$ build cyclic vertex / edge constraints or vertex / edge constraints from F
16: $\quad$ **for** constraint $con = \overline{(as, v, q_r, q_e)}$ / $\overline{(as, v_i, v_j, q_r, q_e)}$/ $\overline{(as, v, q)}$ / $\overline{(as, v_i, v_j, q)}$ in $Cons$ **do**
17: $\quad\quad D \leftarrow$ new node
18: $\quad\quad D.cons \leftarrow N.cons \cup con$
19: $\quad\quad D.paths[as] \leftarrow ASCBS_{low}(as, D.cons, c)$
20: $\quad\quad$ **if** $D.paths[as]$ is not $NULL$ **then**
21: $\quad\quad\quad D.cost \leftarrow$ calculate the SOC of $D.paths$
22: $\quad\quad\quad OPEN \leftarrow OPEN \cup \{D\}$
23: $\quad\quad$ **end if**
24: $\quad$ **end for**
25: **end while**
26: **return** $NULL$

child CT nodes are generated, each with one of the two additional constraints. The low-level solver is then utilized to determine the path for the newly constrained agent stream (Line 19). After constructing the child nodes, the algorithm proceeds to the next iteration.

We propose the cyclic vertex constraint and the cyclic edge constraint to resolve the conflict between different agent streams. The cyclic vertex constraint is denoted as $\overline{(as, v, q_r, q_e)}$, indicating that the agent stream as cannot occupy the vertex v at the q-th step if q satisfies the equations:

$$q \equiv q_r \ (mod\ c) \tag{9}$$

$$q \neq q_e \tag{10}$$

The cyclic edge constraint is denoted as $\overline{(as, v, u, q_r, q_e)}$, meaning that the agent stream as cannot go through the edge from v to u with the q-th step if q satisfies the equations:

$$q \equiv q_r \ (mod\ c) \tag{11}$$

$$q \neq q_e \tag{12}$$

Notably, q_e in a cyclic vertex/edge constraint might be $\emptyset$, and in this case, Equations 10 and 12 are always satisfied.

Here we consider using the constraint to resolve the conflict between different agent streams. Recall that the cyclic vertex conflict $\langle as_i, as_j, q_i, q_j, v\rangle$ is caused by the q_i-th step of the path of as_i and the q_j-th step of the path of as_j. When replanning as_i's path, it is not sufficient to just avoid as_i occupying at vertex v at q_j-th step to resolve the conflict at vertex v. This is because the conflict might also occur at v in the q-th step of as_i's path where $q \equiv q_i \ (mod\ c)$, according to Equation 7. We should add constraints on all the q-th steps that satisfy $q \equiv q_i \ (mod\ c)$. It is symmetric for as_j. Specifically, the two child CT nodes are generated respectively as follows to resolve the cyclic vertex conflict $\langle as_i, as_j, q_i, q_j, v\rangle$ where $as_i \neq as_j$:

- Add the cyclic vertex constraint $\overline{(as_i, v, q_i, \emptyset)}$ to the constraint set and replan the path of as_i.
- Add the cyclic vertex constraint $\overline{(as_j, v, q_j, \emptyset)}$ to the constraint set and replan the path of as_j.

Similarly, to resolve the cyclic edge conflict $\langle as_i, as_j, q_i, q_j, v_i, v_j\rangle$ where $as_i \neq as_j$, the two child CT nodes are operated respectively as follows:

- Add the cyclic edge constraint $\overline{(as_i, v_i, v_j, q_i, \emptyset)}$ to the constraint set and replan the path of as_i.
- Add the cyclic edge constraint $\overline{(as_j, v_j, v_i, q_j, \emptyset)}$ to the constraint set and replan the path of as_j.

However, when a conflict arises within the same agent stream, we cannot adopt a similar strategy to resolve the conflict using cyclic vertex/edge constraints. This is because the strategy cannot preserve the completeness of the algorithm, as discussed in the Technical Appendix. Instead, we resolve the conflict through the vertex constraint and the edge constraint. The notation $\overline{(as, v, q)}$ represents the vertex constraint, indicating that the agent stream as cannot be located at the vertex v at the q-th step. Similarly, the notation $\overline{(as, v, u, q)}$ denotes the edge constraint, which means that the agent stream as cannot go through the edge from the vertex v to the vertex u in the q-th step. To resolve the cyclic vertex conflict $\langle as_i, as_j, q_i, q_j, v\rangle$ where $as_i = as_j$, two child CT nodes are generated:

- Add the vertex constraint $\overline{(as_i, v, q_i)}$ to the constraint set and replan the path of as_i.
- Add the vertex constraint $\overline{(as_i, v, q_j)}$ to the constraint set and replan the path of as_i.

The two child CT nodes are generated to resolve the cyclic edge conflict $\langle as_i, as_j, q_i, q_j, v_i, v_j\rangle$ where $as_i = as_j$.

- Add the edge constraint $\overline{(as_i, v_i, v_j, q_i)}$ to the constraint set and replan the path of as_i.
- Add the edge constraint $\overline{(as_i, v_j, v_i, q_j)}$ to the constraint set and replan the path of as_i.

4.2 Low-Level Solver

The low-level solver of ASCBS uses an A* algorithm to compute the optimal path of an agent stream under constraints. It is similar to the low-level solver of CBS, where the search progressively explores nodes until reaching the goal vertex. A key distinction is that the ASCBS algorithm's low-level search additionally considers cyclic vertex/edge constraints instead of only vertex/edge constraints. In addition, the Conflict Avoidance Table (CAT) [Sharon *et al.*, 2015] is employed to help break ties by favoring nodes with fewer conflicts with other agent streams when their f values are equal. Specifically, the nodes use a designated variable to mark the number of collisions along the path from the start vertex to the current vertex. The conflicts with other agent streams at the current step are checked by Equations 1, 4, 7. Notably, conflicts within the same agent stream are not counted because it needs to construct the entire path from the start vertex to the current vertex, significantly increasing the runtime.

There is an alternative implementation in that the low-level solver uses the Iterative-Deepening-A* (IDA*) algorithm [Korf, 1985] to generate the path. The IDA* algorithm utilizes a depth-first search approach, which allows it to maintain the entire path to the current search state. This capability enables the pruning of states when a cyclic vertex/edge conflict arises from the same agent stream during the search. Consequently, the IDA* algorithm can ensure that no collision exists from the same agent stream in the result path. As a result, the high-level solver only needs to resolve the conflict between agents of different agent streams when adopting the IDA* algorithm as the low-level solver. In contrast, the A* algorithm is unable to prune states with a cyclic vertex/edge conflict during the search because it does not maintain the entire path at every state. Such conflicts can only be resolved by the high-level solver.

4.3 Disjoint Splitting

Referencing the improvement technique of the CBS algorithm, we consider adopting the disjoint splitting strategy [Li *et al.*, 2019b] into the ASCBS algorithm. This strategy involves applying asymmetric constraints to two child CT nodes to divide the problem into two separate subproblems, which helps prevent redundant searches.

In this context, (as, v, q) represents a positive vertex constraint, indicating that the agent stream as must be placed at the vertex v in the q-th step. Similarly, (as, v, u, q) denotes a positive edge constraint, indicating that the agent stream as must pass through the edge from vertex v to vertex u at the q-th step. The positive vertex/edge constraint can impact other agent streams' paths as well as other steps of the current agent stream's path. For example, when (as_i, v, q_i) is added into the constraint set, it restricts another agent stream as_j from being at the vertex v in the q_j-th step, where $t_i^s + q_i \equiv t_j^s + q_j \ (mod\ c)$. Furthermore, the path of as_i cannot be at vertex v at $q_i{}'$-th step where $q^i \equiv q_i{}' \ (mod\ c)$ and $q^i \neq q_i{}'$.

Specifically, the two child CT nodes are generated to resolve the cyclic vertex conflict $\langle as_i, as_j, q_i, q_j, v\rangle$. Let k represent the agent stream ID for the positive constraint. Let k' denote the ID of the newly constrained agent stream in the other child CT node. Formally, If $as_i \neq as_j$, then $k' = \{i, j\} \backslash k$; otherwise, $k' = k$.

- Add the positive vertex constraint (as_k, v, q_k), the cyclic vertex constraints $\overline{(as_k, v, q_k, q_k)}$ and $\{\overline{(as_o, v, t_k^s + q_k - t_o^s, \emptyset)} : o \neq k\}$ to the constraint set and replan the path of as_k'.
- Add the vertex constraint $\overline{(as_k, v, q_k)}$ to the constraint set and replan the path of as_k.

To resolve the cyclic edge conflict $\langle as_i, as_j, q_i, q_j, v_i, v_j\rangle$, two child CT nodes are generated.

- Add the positive edge constraint (as_k, v_i, v_j, q_k), the cyclic edge constraints $\overline{(as_k, v_j, v_i, q_k, q_k)}$ and $\{\overline{(as_o, v_j, v_i, t_k^s + q_k - t_o^s, \emptyset)} : o \neq k\}$ to the constraint set and replan the path of as_k'.
- Add edge constraint $\overline{(as_k, v_i, v_j, q_k)}$ to the constraint set and replan the path of as_k.

For the selection of k, we randomly choose k from $\{i, j\}$ if $as_i \neq as_j$. Otherwise, $k \leftarrow \arg\min_{o\in\{i,j\}}(q_o)$, where the motivation is that the later step of the path depends on the earlier step of the same path, and it is unreasonable to keep still the agent stream in the later step while altering it in the earlier step.

4.4 Theoretical Analysis

Based on the implementation of the ASCBS, the following theorem holds:

Theorem 2. *The ASCBS algorithm that employs A* as the low-level solver is an optimal and complete solution for the S-MAPF problem.*

The proof of Theorem 2 is provided in the Supplementary Material.

5 Experiment

This section presents experiments conducted to evaluate the computational efficiency and solution quality of the ASCBS on three grid-like graphs selected from the MAPF Benchmark Set [Stern *et al.*, 2019], with varying scaling and features. The three chosen graphs are 'empty-8-8' (size: 8 × 8), 'random-64-64-10' (size: 64 × 64), and 'Paris_1_256' (size: 256 × 256). Each graph includes 25 scenario files that specify the start and goal points of the agents. For every scenario, 4 instances are generated with a randomly assigned initial start time for each agent stream within the range of $[0, c-1]$ for the setting of circle time c and agent number n, resulting in a total of 100 instances for each setting. All experiments are performed on a Ubuntu 20.04 computer equipped with an Intel Core i7-8700 CPU running at 3.2 GHz with 32GB of main memory. The code is publicly available at https://github.com/tangmingkai/S-MAPF.

The following algorithms are used in the experiments:

- **ASCBS-A-ND**: ASCBS with A* as the low-level solver and without the disjoint splitting strategy.
- **ASCBS-A-D**: ASCBS with A* as the low-level solver and adopting the disjoint splitting strategy.
- **ASCBS-IDA-ND**: ASCBS with IDA* as the low-level solver and without the disjoint splitting strategy.
- **ASCBS-IDA-D**: ASCBS with IDA* as the low-level solver and adopting the disjoint splitting strategy.
- **CBS**: The conflict-based search with WDG guidance [Li *et al.*, 2019a] and mutex reasoning [Zhang *et al.*, 2022], with the setting that agents disappear upon reaching their goals. It is the solver for the traditional MAPF problem, but ignore the assumption that agents in the same agent stream must share the same action sequence.

5.1 Various Number of Agent Streams

We assessed the computational efficiency of the different implementations of ASCBS on the instances with varying numbers of agent streams, where the circle time is set at 3. The average running time and the success rate are used as evaluation metrics. An instance is considered unsuccessful if the running time exceeds 60 seconds, in which case the running time is set at 60 seconds directly.

The results are presented in Figure 3. The performance of ASCBS-A-ND and ASCBS-A-D surpasses that of ASCBS-IDA-ND and ASCBS-IDA-D. This difference in performance arises because the IDA* algorithm repeatedly executes the search from the start vertex of an agent stream whenever the bound of the f-value increases. The overhead associated with using the IDA* algorithm as the low-level solver outweighs its advantage, which is that the high-level solver does not need to resolve conflicts from the same agent stream. This finding suggests that resolving conflicts from the same agent streams at the high-level solver is more efficient than resolving them at the low-level solver. In the empty-8-8 and random-64-64-10, ASCBS-A-D outperforms ASCBS-A-ND, with the gap more pronounced in smaller maps. In contrast, on the larger map (Paris_1_256), ASCBS-A-ND outperforms ASCBS-A-D. We believe that the reason is as follows. On the one hand, the strategy that imposes a larger additional constraint set on child CT nodes has a higher efficiency because it is more likely to elevate the lower bound of the optimal SOC of the CT node. A cyclic vertex/edge constraint on the agent stream with path length l and cycle time c can be extracted to approximately $\lfloor \frac{l}{c} \rfloor$ vertex/edge constraint. In

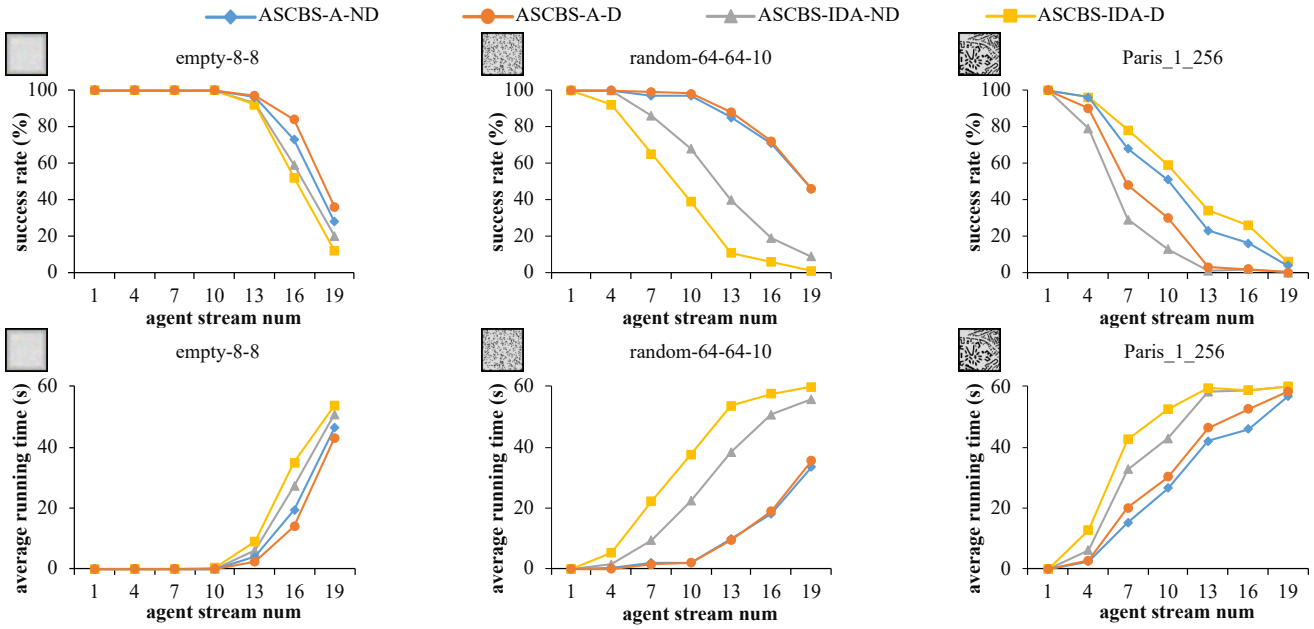

Figure 3: Success rate and average running time for different implementations of ASCBS on the instances with cycle time 3.

the solver with the disjoint splitting strategy, one of the child CT nodes has an additional constraint set that only contains a vertex/edge constraint, which is smaller than that in the solver without disjoint splitting. This makes the non-disjoint splitting strategy more beneficial than the disjoint splitting strategy in terms of computational efficiency. Consequently, the advantage caused by non-disjoint splitting diminishes in a smaller size map due to the shorter path length l, resulting in a smaller value $\lfloor \frac{l}{c} \rfloor$. On the other hand, the disjoint splitting prevents the same conflict recurrence in the subtree rooted by two child CT nodes. In this view, the disjoint splitting strategy is more beneficial than the non-disjoint splitting strategy. Considering the above two factors, when the map size is small, the advantage caused by the disjoint splitting from the solver with A* search is larger than the disadvantage. Nonetheless, ASCBS-IDA-D is outperformed by ASCBS-IDA-ND, as the computing time of IDA* constitutes a significant portion of the total computing time and is heavily influenced by constraints. Therefore, the advantage of the disjoint splitting strategy in the solver with IDA* cannot cover the disadvantage.

5.2 Comparing ASCBS with CBS

We evaluate the computational efficiency and quality of the solution of ASCBS-A-ND compared to CBS. Since CBS is unable to generate solutions for unlimited working hours, we introduce an integer variable called the time horizon. This variable restricts CBS to considering only the agents whose start time is less than or equal to the time horizon. Our evaluation includes instances with 10 agent streams and circle time 3. We measure computational efficiency using average running time. We named the instance feasible by the algorithm if its running time is within 60 seconds and can produce a feasible solution. To evaluate solution quality, we calculate the average relative error of the SOC among instances feasible by both algorithms. Notably, the traditional MAPF problem has less constraint than the S-MAPF problem, resulting in a no larger SOC in CBS than in ASCBS.

Figure 4 displays the average running time of ASCBS and CBS, the number of instances feasible by both algorithms and the average relative error of ASCBS compared to CBS. It is noteworthy that the solution of ASCBS can be applied across all time horizons, leading to consistent average running times. As the time horizon increases, CBS's running time scales due to the larger number of agents considered. In contrast, the relative error of ASCBS compared to CBS is very small and decreases as the time horizon grows. This trend suggests that when the working hours are extended, the difference in solution quality between ASCBS and CBS diminishes, supporting the assumption that ASCBS is more suitable to apply in the scenario with long working hours compared to CBS.

6 Extensions

The formalization of the S-MAPF problem is based on several strong assumptions. In this section, we introduce extensions of the S-MAPF problem to relax these assumptions. It is evident that the solver for the basic S-MAPF problem can be adapted to solve the extended problem. We denote the basic S-MAPF problem as $P1$.

6.1 Stay in the Environment

The S-MAPF problem assumes that agents in the agent stream appear at the starting vertex at specific time steps and disappear upon reaching the goal vertex. This assumption is suitable for the intersection scenarios and the environments that have some private parking zones at both the start and goal vertices for robot loading and unloading [Švancara *et*

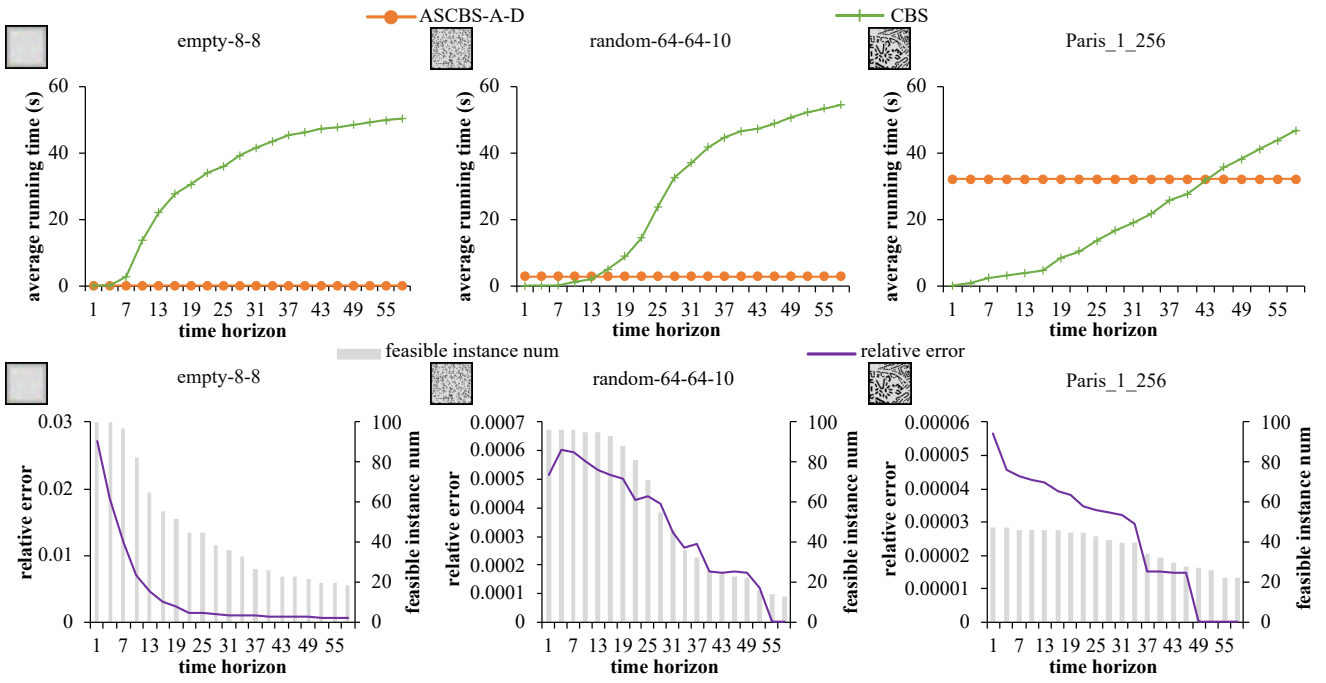

Figure 4: Average running time of ASCBS and CBS, the number of feasible instances by both two algorithms, and average relative error of ASCBS compared to CBS on the instance with 10 agent streams and cycle time 3.

al., 2019]. An extension of the S-MAPF problem (denoted as $P2$) relaxes this assumption, by requiring all agents to remain in the environment throughout the entire working process.

Several methods can be employed to solve the extended problem. One method is to generate a return path for each agent stream, dividing the $P2$ into two stages. In the first stage, a path is generated for each agent stream from its start vertex to its goal vertex, similar to $P1$. The second stage focuses on generating a path for each agent stream from the goal vertex back to the start vertex, ensuring that this path avoids conflicts with the paths established in the first stage. Additionally, the path length in the second stage must adhere to a specific constraint to maintain the cyclic pattern. Let l_1^i and l_2^i represent the length of the path generated in the first and second stages for agent stream i, receptively. These path lengths must satisfy the condition $\forall i, l_1^i + l_2^i \equiv 0 \ (mod\ c)$. In the implementation, the ASCBS algorithm can enhance the goal-reaching conditions in the low-level solver to ensure that this condition is met.

An alternative method for the $P2$ is to jointly generate the paths in both stages, ensuring that the path lengths are multiples of c. This can be achieved by employing a multi-label A* algorithm [Grenouilleau *et al.*, 2019] as the low-level solver within the ASCBS algorithm.

A more advanced method for the $P2$ is to allow for the concentration of multiple agent streams. In this method, an agent that has reached the goal vertex of an agent stream can subsequently move to the start vertex of another agent stream. Future research will focus on this category of methods.

6.2 Non-Uniform Cycle Time

In the context of $P1$, we assume that all agent streams operate on the same cycle time. This assumption restricts the applicability of the S-MAPF problem to scenarios that require non-uniform cycle times. In the case of the problem with non-uniform cycle time agent streams (denoted as $P3$), conflicts also display in a periodic pattern, similar to that observed in $P1$. Consequently, we can extend the definition of cyclic conflict and constraints of the ASCBS algorithm to accommodate non-uniform cycle times. Further details are available in the Supplementary Material.

7 Conclusion

This work formalized the S-MAPF problem, which assumes that agents in the same agent stream have a periodic start time and use the same action sequence. To solve the S-MAPF problem, an optimal and complete algorithm, ASCBS, was introduced, which includes cyclic vertex/edge constraints. The potential of the disjoint splitting strategy in the ASCBS algorithm was also investigated. Experimental comparisons were conducted to evaluate the performance of different implementations of ASCBS, indicating that resolving conflicts within the same agent stream in the high-level solver is more effective than in the low-level solver, and disjoint splitting is advantageous when the map size is small. Moreover, experiments demonstrated that when the working hour is long, ASCBS outperforms CBS in terms of runtime while maintaining a slightly lower or equal solution quality. Finally, we presented several extensions of the S-MAPF problem. The solver developed for the basic S-MAPF problem can be adapted to solve these extended problems.

References

[Banfi *et al.*, 2017] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2d grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.

[Belov *et al.*, 2020] Gleb Belov, Wenbo Du, Maria Garcia De La Banda, Daniel Harabor, Sven Koenig, and Xinrui Wei. From multi-agent pathfinding to 3d pipe routing. In *Proceedings of the International Symposium on Combinatorial Search*, volume 11, pages 11–19, 2020.

[Boyarski *et al.*, 2015] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, Oded Betzalel, David Tolpin, and Eyal Shimony. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, pages 223–225, 2015.

[Boysen *et al.*, 2022] Nils Boysen, Philipp Schulze, and Armin Scholl. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3):797–814, 2022.

[Brown *et al.*, 2020] Kyle Brown, Oriana Peltzer, Martin A Sehr, Mac Schwager, and Mykel J Kochenderfer. Optimal sequential task assignment and path finding for multi-agent robotic assembly planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 441–447. IEEE, 2020.

[Grenouilleau *et al.*, 2019] Florian Grenouilleau, Willem-Jan Van Hoeve, and John N Hooker. A multi-label a* algorithm for multi-agent pathfinding. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, pages 181–185, 2019.

[Hart *et al.*, 1968] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Ho *et al.*, 2019] Florence Ho, Ana Salta, Ruben Geraldes, Artur Goncalves, Marc Cavazza, and Helmut Prendinger. Multi-agent path finding for uav traffic management. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 131–139, 2019.

[Kasaura *et al.*, 2023] Kazumi Kasaura, Ryo Yonetani, and Mai Nishimura. Periodic multi-agent path planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6183–6191, 2023.

[Korf, 1985] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[Li *et al.*, 2019a] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, volume 2019, pages 442–449, 2019.

[Li *et al.*, 2019b] Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. Disjoint splitting for multi-agent path finding with conflict-based search. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, pages 279–283, 2019.

[Li *et al.*, 2019c] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019.

[Li *et al.*, 2021] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11272–11281, 2021.

[Li *et al.*, 2023] Jiaoyang Li, Eugene Lin, Hai L Vu, Sven Koenig, et al. Intersection coordination with priority-based search for autonomous vehicles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11578–11585, 2023.

[Ma *et al.*, 2017] Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

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

[Stern *et al.*, 2019] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019.

[Švancara *et al.*, 2019] Jiří Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. Online multi-agent pathfinding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7732–7739, 2019.

[Xu *et al.*, 2022] Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma. Multi-goal multi-agent pickup and delivery. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9964–9971. IEEE, 2022.

[Zhang *et al.*, 2022] Han Zhang, Jiaoyang Li, Pavel Surynek, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with mutex propagation. *Artificial Intelligence*, 311:103766, 2022.

[Zhang *et al.*, 2024] Yulun Zhang, He Jiang, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. Guidance graph optimization for lifelong multi-agent path finding. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 311–320, 2024.