Journal of Artificial Intelligence Research 70 (2021) 923–954 Submitted 09/2020; published 03/2021

Safe Multi-Agent Pathfinding with Time Uncertainty

Tomer Shahar STOMER@POST.BGU.AC.IL
Shashank Shekhar SHEKHAR@POST.BGU.AC.IL
Dor Atzmon DORAT@POST.BGU.AC.IL
Ben-Gurion University of the Negev, Israel

Abdallah Saffidine ABDALLAH.SAFFIDINE@GMAIL.COM
The University of New South Wales, Sydney, Australia

Brendan Juba BJUBA@WUSTL.EDU
Washington University in St. Louis, USA

Roni Stern
Ben-Gurion University of the Negev, Israel STERNRON@POST.BGU.AC.IL
Palo Alto Research Center, USA RSTERN@PARC.COM

Abstract

In many real-world scenarios, the time it takes for a mobile agent, e.g., a robot, to move from one location to another may vary due to exogenous events and be difficult to predict accurately. Planning in such scenarios is challenging, especially in the context of Multi-Agent Pathfinding (MAPF), where the goal is to find paths to multiple agents and temporal coordination is necessary to avoid collisions. In this work, we consider a MAPF problem with this form of time uncertainty, where we are only given upper and lower bounds on the time it takes each agent to move. The objective is to find a *safe* solution, which is a solution that can be executed by all agents and is guaranteed to avoid collisions. We propose two complete and optimal algorithms for finding safe solutions based on well-known MAPF algorithms, namely, A^* with Operator Decomposition ($A^* + OD$) and Conflict-Based Search (CBS). Experimentally, we observe that on several standard MAPF grids the CBS-based algorithm performs better. We also explore the option of *online* replanning in this context, i.e., modifying the agents' plans during execution, to reduce the overall execution cost. We consider two online settings: (a) when an agent can sense the current time and its current location, and (b) when the agents can also communicate seamlessly during execution. For each setting, we propose a replanning algorithm and analyze its behavior theoretically and empirically. Our experimental evaluation confirms that indeed online replanning in both settings can significantly reduce solution cost.

1. Introduction

Multi-Agent Pathfinding (MAPF) is the problem of finding how to move a team of agents over edges in a graph from an initial configuration to a goal configuration. The key constraint is that agents must not occupy the same vertex or traverse the same edge at the same time during plan execution. A *solution* to a MAPF problem is a set of *single-agent plans* for all the agents that do not violate this constraint. MAPF problems arise in real-world applications, such as aircraft towing vehicles (Morris et al., 2016), video game characters (Silver, 2005), office robots (Veloso et al., 2015), and warehouse robots (Wurman et al., 2007). Modern MAPF solvers can optimally solve problems with over a hundred agents (Sharon et al., 2015; Boyarski et al., 2015b; Felner et al., 2018).

However, most of the earlier work assumed the time it takes an agent to traverse an edge is fixed and known a priori. In reality, exogenous events may cause the edge traversal time to be non-deterministic. One approach to address this is to gather statistics over these exogenous events and reason about them (Wagner & Choset, 2017; Atzmon et al., 2020a; Ma et al., 2017). However, gathering sufficiently accurate statistics of exogenous events is often difficult. For example, in many automated warehouses, humans are responsible for packing items into boxes and the robots move the packages. The time it takes a human to pack a box may depend on many factors, such as the number of items in the box, their weight, and the human's fatigue. Creating an accurate statistical model for the packing time is a challenging problem on its own. In this work, we consider weaker knowledge about such exogenous events, in the form of upper and lower bounds.[1] That is, we consider the case where each edge is associated with a time range that bounds the duration it takes an agent to traverse it. This form of uncertainty poses a challenge to current MAPF solvers, especially if one wants to find a *safe* solution. A solution is safe if it is guaranteed that when executing it there will not be any collision.[2] We call the problem of finding a safe solution in this setting, the Multi-Agent Pathfinding with Time Uncertainty (MAPF-TU) problem.

In the first part of this work, we adapt two popular MAPF solvers, namely $\mathrm{A}^* + \mathrm{OD}$ (Standley, 2010) and Conflict-Based Search (CBS) (Sharon et al., 2015), to solve the MAPF-TU problem. We call the adapted algorithms $\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ and $\mathrm{CBS}_{\mathrm{TU}}$. Both $\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ and $\mathrm{CBS}_{\mathrm{TU}}$ are sound and complete, in the sense that any solution they return is safe, and if a safe solution exists they will return it. In addition, they can guarantee various forms of optimality. We implemented $\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ and $\mathrm{CBS}_{\mathrm{TU}}$, and compared them on a set of standard MAPF benchmarks adapted to include time uncertainty. Our results show that $\mathrm{CBS}_{\mathrm{TU}}$ performs significantly better in almost all cases.

$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ and $\mathrm{CBS}_{\mathrm{TU}}$ are both *offline* algorithms, that is, they output a solution before the agents start to move. In the second part of this paper, we explore the option of *online replanning* to improve the solution returned by an offline MAPF-TU solver, i.e., the option to modify the given plan while executing it.

Specifically, we consider two possible capabilities the agents may have that can be used for online replanning: *sensing* and *communication*. Sensing in our context is the ability of an agent to observe, during execution, its current time and location. Communication in our context is the ability to send the sensed information to all the other agents. We demonstrate that online replanning can reduce the overall execution cost even if the agents only sense and cannot communicate. We also demonstrate that the ability to communicate can lead to further reduction of execution costs. Then, we propose two fast online replanning techniques that preserve safety and can reduce the overall execution cost. We implemented both techniques and evaluated the benefit of using them on several benchmark domains. The results show that these online replanning techniques, especially when the agents can communicate, can reduce execution cost by a factor of 1.5 less than the cost of following the original offline solution. Moreover, the time overhead of using them is negligible compared to the time required to compute the original offline solution.

1. We show a way to obtain such bounds from observations in Section 9.
2. Such a solution is known as a *strong* solution in the planning literature (Cimatti et al., 2018).

The paper is structured as follows. In Section 2, we provide details on relevant background and related work. In Section 3, we define formally the MAPF-TU problem. Sections 4 and 5 introduce the $\text{A}^* + \text{OD}_{\text{TU}}$ and CBS_{TU} algorithms for solving MAPF-TU, respectively, and in Section 6 we evaluate them empirically. Then, in Section 7, we analyze the benefit of online replanning in MAPF-TU and propose two fast algorithms for this purpose. In Section 8, we evaluate these online replanning algorithms experimentally. Finally, in Section 9, we discuss and outline several key extensions, such as creating an individual execution policy for each agent and bounding edge traversal time from experience.

2. Background and Related Work

This work is related to two areas in Artificial Intelligence: multi-agent pathfinding (MAPF) and planning under temporal uncertainty. We briefly discuss the relevant literature below.

2.1 Multi-Agent Pathfinding (MAPF)

An instance of a classical MAPF problem (Stern et al., 2019) is defined by a tuple $\langle G, s, t\rangle$ where $G = (\mathcal{V}, \mathcal{E})$ is a graph, $s : [1, \ldots, k] \to \mathcal{V}$ maps an agent to its *initial* location, and $t : [1, \ldots, k] \to \mathcal{V}$ maps an agent to its *goal* location. Time is assumed to be discretized. At the beginning of each time step, each agent is located in one of the graph vertices. Then, the agent can perform a single action: either *wait* in its current location, or *move* to one of the vertices that is adjacent to its current location. An action is a pair (u, v) where for a wait action $u = v$ and for a move action (u, v) is an edge in the graph. A *single-agent plan* for an agent a_i is a sequence of actions that move a_i to its goal, i.e., a sequence of actions $\pi = ((v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n))$ where $v_0 = s(i)$ and $v_n = t(i)$. A *solution* to a MAPF instance is a vector of k single-agent plans. A solution is said to have a *conflict* $\langle a_i, a_j, x, t\rangle$ iff the two agents a_i and a_j are planned to occupy the same vertex or edge x at the same time step t. We assume that edges are undirected, so a conflict also arises if agents swap locations over the same edge. We say that a solution to a MAPF instance is *valid* if it does not have any of these conflicts. That is, we consider only vertex, edge, and swapping conflicts, and do not consider other types of conflicts (Stern et al., 2019).

Traditionally, the cost of a single-agent plan π in MAPF is the number of its constituent actions. In this work, we define the cost of a solution to be the sum of costs of its constituent single-agent plans. This solution cost function is known as the *sum-of-costs* (SOC). A valid solution is *optimal* when there is no other valid solution with a lower cost. A range of algorithms have been proposed in the past decade for finding optimal solutions to such MAPF instances (Felner et al., 2017), such as $\text{A}^* + \text{OD}$ (Standley, 2010), ICTS (Sharon et al., 2013), M* (Wagner & Choset, 2015), CBS (Sharon et al., 2015), BCP (Lam et al., 2019b), as well as several algorithms that find optimal solutions by compiling the problem to Boolean satisfiability (Surynek, 2012; Surynek et al., 2016), Constraint programming (Barták et al., 2017), or Answer set programming (Erdem et al., 2013).

The problem addressed by these optimal solvers has been termed *classical MAPF* (Stern et al., 2019). In classical MAPF, there is no uncertainty in the edge traversal time, and the edge traversal and the wait action have unit duration. These latter assumptions have been relaxed in recent work. Cohen et al. (2019) and Walker et al. (2018, 2020) suggested optimal and suboptimal algorithms for solving MAPF problems with non-unit edge traversal time and non-unit but fixed wait action duration. Andreychuk et al. (2019) proposed an optimal algorithm to further allow wait actions with arbitrary durations. However, none of the above algorithms considers time uncertainty over edge durations.

Various forms of MAPF with time uncertainty have also been studied in the past. Several algorithms were proposed for the MAPF Problem with Delay Probabilities (MAPF-DP) problem, which is a variant of MAPF in which each agent is delayed with some given probability, which is known to the problem solver.

The UM* (Wagner & Choset, 2017) algorithm returns a solution for a MAPF-DP problem in which the probability for every potential collisions to occur is below a given threshold. The pR-CBS (Atzmon et al., 2020a) algorithms returns a solution for a MAPF-DP problem in which the probability for successful execution (i.e., having no collisions) is above a given threshold. Neither algorithms guarantee that the solutions they return are *safe*, in the sense that a collision might still occur while executing these solutions. Approximate Minimization in Expectation (AME) is another algorithm for MAPF-DP, which aims to minimize the expected cost of a solution. However, it only guarantees no collisions occur in the next time step. Thus, it must have online communication to avoid conflicts, and hence is not safe.

kR-CBS (Atzmon et al., 2020b) is an algorithm that creates solutions that can be executed without collisions, but only if each agent is not delayed more than k times. MAPF-POST (Hönig et al., 2016) is an algorithm that gets a valid MAPF solution and reschedules the times in which agents visit their locations to avoid collisions. Thus, MAPF-POST is safe and can be applied to the MAPF-TU setting (which is defined later). However, it does not provide any solution quality guarantees and a lower cost solution or execution may exist, while the algorithms we propose in this work also provide optimality guarantees.

2.2 Planning under Temporal Uncertainty

The type of time uncertainty addressed in our work has been considered in the context of the Simple Temporal Problem under Uncertainty (STPU) (Vidal & Fargier, 1999). Simple Temporal Problem (STP) (Dechter et al., 1991) is the problem of finding a schedule for a set of activities, where each activity has a set of temporal constraints over when it can be scheduled. In STP with Uncertainty (STPU), the duration of an activity may lie within some bounds but cannot be decided upfront, just like in MAPF-TU. However, MAPF-TU is a planning problem while STPU is a scheduling problem: a solution to MAPF-TU is a set of single-agent plans, one per agent, while a solution for STPU is a schedule for a given set of activities.

The MAPF-TU problem is a non-deterministic planning problem, where the non-determinism manifests in the edges traversal durations. Cimatti et al. (2003) identified three types of solutions to non-deterministic planning problems: weak solutions, strong solutions, and strong cyclic solutions. A weak solution is a plan that may achieve the goal, a strong solution is a plan that must achieve the goal under any outcome of the non-determinism, and a strong cyclic solution is a plan that must achieve the goal but may do so via trial-and-error, and that trial-and-error may be infinitely long. In this work we aim to find a *safe* solution, which is a solution that leads all the agents to their respective goals without collisions as long as the edge traversal durations are in their specified time ranges. Thus, a safe solution is a strong solution.

Temporal planning (Coles et al., 2008; Coles & Coles, 2014; Cashmore et al., 2019) is a popular and well-researched sub-area of AI planning in which actions have durations. Cimatti et al. (2018) recently defined the Strong Temporal Planning with Uncertain action Durations (STPUD) problem, which is a temporal planning problem in which action durations are *uncontrollable.* In particular, they propose a planning approach that can handle cases where actions have uncertain durations that are within known bounds. The MAPF-TU problem can be seen as a special case of the STPUD problem, and it may be possible to translate the MAPF-TU problem to the single-agent STPUD problem and use their planner to provide strong solutions to MAPF-TU. However, the algorithms we propose also guarantee optimality, while their strong temporal planning algorithm does not. Moreover that it is evident from the planning literature that factored-approaches are often very efficient for multi-agent planning problems (Brafman & Domshlak, 2008; Pajarinen & Peltonen, 2011; Shekhar et al., 2019). This has especially been observed in MAPF, where all state of the art algorithms directly exploit the decoupled nature of the problem (Sharon et al., 2015, 2013; Lam et al., 2019a; Gange et al., 2019).

3. Safe MAPF with Time Uncertainty

An instance of the *Multi-Agent Pathfinding with Time Uncertainty* problem (MAPF-TU) for k agents is defined by a tuple $\langle G, s, t, w^-, w^+\rangle$, where $G = (\mathcal{V}, \mathcal{E})$, s, and t, are the same as in classical MAPF; and $w^- : \mathcal{E} \rightarrow \mathbb{N}$ and $w^+ : \mathcal{E} \rightarrow \mathbb{N}$ are functions that return the minimal and maximal duration it takes an agent to traverse a given edge. That is, the time it takes an agent to traverse an edge e is a number in the range $[w^-(e), w^+(e)]$. We refer to $w^-(e)$ and $w^+(e)$ as the lower and upper bounds of the edge duration, respectively. Note that if an agent traversed the same edge twice, the durations of these traversals may be different. The only requirement is that these durations are in the specified time range $[w^-(e), w^+(e)]$.

Time is assumed to be discretized, and a solution is a vector of single-agent plans as defined for classical MAPF. A solution is said to have a *potential conflict* if there are two agents that may occupy the same vertex or edge at the same time. In this work, we aim to find a solution for a given MAPF-TU instance has no potential conflicts. We call such a solution a *safe* solution.

3.1 Computing the Potential Presence

To identify a safe solution, we need to formally define what is a potential conflict. To this end, we introduce the notion of *potential presence.* The *potential presence* induced by a single-agent plan π at a vertex v, denoted $\tau(\pi, v)$, is defined as the set of time intervals in which an agent may be at v when following π. The potential presence is similarly defined for an edge.

Example 1. *(Figure 1) Consider a single-agent plan π in which the agent moves from its initial vertex s to its goal vertex g through vertices v_1 and v_2 without waiting. The time interval depicted above each edge is the range of possible edge durations. The* potential

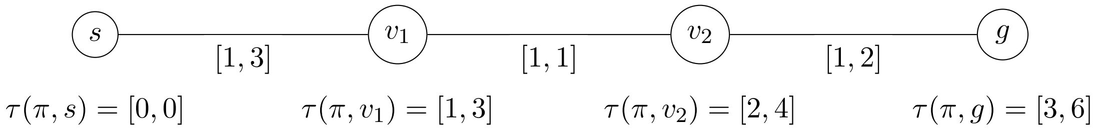

Figure 1: An example of the potential presence of the agent when moving from s to g.

presences *induced by* π *at vertices* s*,* v_1*,* v_2*, and* g *are* $\{[0,0]\}$*,* $\{[1,3]\}$*,* $\{[2,4]\}$*, and* $\{[3,6]\}$*, respectively.*

Computing the potential presence can be done as follows. Let $\pi = \big((v_0,v_1)\dots(v_{n-1},v_n)\big)$ be a single-agent plan for some agent. For every $v_i \neq v_0$, the *potential presence* induced by π is given by

$$\tau(\pi,v) = \bigcup_{\substack{0\le i\le n\\ v_i=v}} \left[\sum_{0\le j<i} w^-(v_j,v_{j+1}), \sum_{0\le j<i} w^+(v_j,v_{j+1})\right] \tag{1}$$

The potential presence induced by π on edge (u,v) is given by

$$\tau(\pi,u,v) = \bigcup_{\substack{0\le i<n\\ v_i=u\\ v_{i+1}=v}} \left[\sum_{0\le j<i} w^-(v_j,v_{j+1}), \sum_{0\le j\le i} w^+(v_j,v_{j+1})\right] \tag{2}$$

In both of these formulas, when $i=0$ there are no values of j that satisfy $0 \le j < i$. In this case, we consider the empty sum to be zero.

3.2 Potential Conflicts

We are now equipped to define the different types of potential conflict that can occur in the MAPF-TU problem. Two distinct single-agent plans have a *potential vertex conflict* at some vertex $v \in \mathcal{V}$ if their potential presences in v intersect. They have a *potential edge conflict* at a pair of vertices (u,v) if their potential presences intersect when traversing the edge in the same direction. They have a *potential swapping conflict* at a pair of vertices (u,v) if their potential presences intersect when traversing the edge in opposite directions. A solution $(\pi_1,\dots,\pi_k)$ is *safe* if it does not contain any potential conflict of these three types. That is, a solution is safe iff it satisfies the following

$$\begin{aligned}\bigcup_{1\le i<j\le k} \Big[&\bigcup_{v\in\mathcal{V}} \tau(\pi_i,v)\cap\tau(\pi_j,v)\\ &\cup \bigcup_{u,v\in\mathcal{V}} \tau(\pi_i,u,v)\cap\tau(\pi_j,u,v)\\ &\cup \bigcup_{u,v\in\mathcal{V}} \tau(\pi_i,u,v)\cap\tau(\pi_j,v,u)\Big] = \emptyset\end{aligned} \tag{3}$$

3.3 Solution Cost

In classical MAPF, it is common to define the cost of a solution as the sum-of-costs (SOC) of its constituent single-agent plans. However, in MAPF-TU the cost of a single-agent plan is ill-defined: the duration required to execute a single-agent plan is not known a-priori due to the uncertainty over edge duration. Therefore, we consider two alternative cost functions: *optimistic* and *pessimistic*. The *optimistic* cost of a single-agent plan is the sum of lower bound (w^-) durations of all its constituent actions. This represents the earliest time in which the agent can reach its goal. The *pessimistic* cost of a single-agent plan is the sum of upper bound (w^+) durations of all its constituent actions. This represents the latest time in which the agent will reach its goal.

Correspondingly, we consider two cost functions for a MAPF-TU solution: optimistic SOC and pessimistic SOC, denoted SOC_{opt} and SOC_{pes}, respectively. Other solution cost functions can also be envisioned, e.g., reducing the average cost of a solution or minimizing the size of the solution cost interval.

4. A* + OD under Time Uncertainty ($\text{A}^* + \text{OD}_{\text{TU}}$)

The first algorithm we propose for finding a safe solution to a MAPF-TU instance, called $\text{A}^* + \text{OD}_{\text{TU}}$, builds on the $\text{A}^* + \text{OD}$ algorithm (Standley, 2010) for classical MAPF. For completeness, we first provide a brief background on $\text{A}^* + \text{OD}$.

4.1 $\text{A}^* + \text{OD}$

As its name suggests, $\text{A}^* + \text{OD}$ consists of running the A^* algorithm and using a technique called the Operator Decomposition (OD) technique (Standley, 2010). Specifically, $\text{A}^* + \text{OD}$ runs the well-known A^* algorithm to search the *k-agent search space*. A state in this search space is a vector of k vertices representing the current location of each agent. An action in this search space is a *joint action* of all agents, i.e., a vector of k actions, one for each agent, which are performed concurrently in a single time step. A joint action is not legal if performing it would create a conflict between the agents. The number of joint actions applicable in a given state is exponential in the number of agents.

OD is a technique for keeping the branching factor of this search-space manageable by constructing joint actions in an incremental manner. Concretely, the agents are arranged in some fixed order. Then, when expanding a state, OD selects a single agent according to this order and generates child states by only considering the actions of the selected agent that are *legal*. An action of the i^{th} agent is legal if it does not conflict with the $i-1$ previously selected actions. Since the agents are selected in a fixed order, a sequence of k actions constitutes a joint action for all agents, and states generated by such a joint action are *standard states* of the k-agent search space. All other states are called *intermediate states*, which represent the locations of some agents at some time step and the locations of the other agents at the next time step. Given an admissible heuristic, $\text{A}^* + \text{OD}$ is a sound, complete, and optimal algorithm for classical MAPF.

4.2 $A^* + OD_{TU}$

$A^* + OD_{TU}$ runs A^* in a search space in which a state is a vector of k tuples

$$\big(\langle a_1, v_1, T_1\rangle, \cdots, \langle a_k, v_k, T_k\rangle\big)$$

where every tuple $\langle a_i, v_i, T_i\rangle$ represents that the potential presence of agent a_i at vertex v_i is T_i. Like OD, when a state is expanded $A^* + OD_{TU}$ chooses a single agent and generates a child state for every *legal action* that agent can perform. However, in $A^* + OD_{TU}$ the agents are not chosen in a round-robin manner and the definition of a legal action is more complex, as we explain below.

4.2.1 Choosing the Next Agent in $A^* + OD_{TU}$

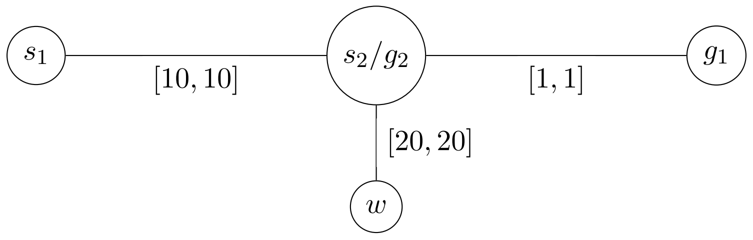

Figure 2: This is an example where a trivial round-robin order of $A^* + OD_{TU}$ returns an illegal solution for a weighted graph, even without uncertainty.

Choosing agents in a round-robin manner as done by $A^* + OD$ may lead to an unsafe solution for MAPF-TU. For example, consider the MAPF-TU instance depicted in Figure 2. Agent a_2 is at its goal location (g_2) which is also its initial location (s_2) and agent a_1's goal is g_1 and its initial location is s_1. Suppose a round-robin order is followed, and a_1 is followed by a_2. The search begins, a_1 moves to s_2 and occupies it at $[10, 10]$. Next, a_2 waits at its goal, since it is already at its goal location. Consequently, the resulting state would be $\big(\langle a_1, s_2, \{[10, 10]\}\rangle, \langle a_2, s_2, \{[1, 1]\}\rangle\big)$. Then, a_1 again moves to g_1 and a_2 waits at g_2, resulting in a goal state, $s_g = \big(\langle a_1, s_2, \{[11, 11]\}\rangle, \langle a_2, s_2, \{[2, 2]\}\rangle\big)$. All moves taken up to this point are valid but the resulting solution is not safe, since a_2 cannot continue waiting there indefinitely.

To avoid this problem, in every iteration of $A^* + OD_{TU}$ the agent with the potential presence having the minimal optimistic bound is selected. [3] For example, in a $A^* + OD_{TU}$ state $\big(\langle a_1, v_1, \{[1, 6]\}\rangle, \langle a_2, v_2, \{[2, 4]\}\rangle\big)$ we will choose to expand this state with agent a_1. If this approach is applied in the example above (Figure 2), a_1 still moves first and a_2 performs the second action, but then a_2 continues waiting at g_2 until $[9, 9]$. Then, waiting in g_2 will raise a conflict, which will eventually lead the search to find a safe solution. Note that the distinction between standard states and intermediate states is not relevant for $A^* + OD_{TU}$ as the same agent may move multiple times and the number of actions has no inherent value. A similar form of OD-style expansion was suggested by Walker (2018) in the E-ICTS algorithm.

3. This is similar to tracking the agent controller that complete its assigned task first in a multi-robot navigation setup, as done by Wagner et al. (2016).

4.3 Identifying Legal Actions

The second significant difference between $A^* + OD$ and $A^* + OD_{TU}$ is what constitutes a legal action for an agent. In $A^* + OD$, it is sufficient to check that the chosen action of agent a_i does not conflict with one of the $i - 1$ agents that were already chosen for an action in this time step. In $A^* + OD_{TU}$, we must check if an action creates a conflict with any of the actions performed by the other agents in any of the predecessors of the current state. This is done by traversing up the parent and parent-of-parent of the current node until either we find a conflict or can guarantee that such a conflict does not exist. This can be guaranteed when reaching a predecessor state in which the maximum time of the potential presences of all other agents is lower than the minimum time of the potential presence of the current agent.

4.4 Optimality

$A^* + OD_{TU}$ runs A^* over the search space defined above, computing for every generated state N its g and h values, and expanding in every iteration a state with the minimal $g + h$ value in the open list. The definition of g and h depends on the objective function we want to optimize, SOC_{opt} or SOC_{pes}. If we aim to optimize SOC_{opt}, then the g value is the sum over the *lower* time-bound in each potential presence. If we aim to optimize SOC_{pes}, then the g value is the sum over the *upper* time bound in each potential presence.

Let C^* be the optimal solution cost for the chosen objective. A heuristic for a state N is admissible if it is a lower bound on $C^* - g(N)$. The admissible heuristic we used is based on the Sum of Individual Costs (SIC) heuristic, which is commonly used in classical MAPF. SIC is the sum of costs of the agents' lowest-cost single-agent plans when ignoring all other agents. For MAPF-TU, the costs in the SIC computation depend on the objective function. When computing SIC for the SOC_{opt} objective, then it is the sum of the *optimistic cost* of the single-agent plans that have the lowest *optimistic cost*. When computing SIC for the SOC_{pes} objective, then it is the sum of the *pessimistic cost* of the single-agent plans that have the lowest *pessimistic costs*.

Theorem 1. $A^* + OD_{TU}$ *is guaranteed to return an optimal safe solution if such a solution exists, such that it is sound, complete, and optimal.*

Proof outline. Soundness follows directly from the fact that a state is generated only if it does not create any potential conflict with the previously chosen actions according to their potential presences. $A^* + OD_{TU}$ is a best-first search, maintaining all non-conflicting nodes it generates in the open list. Thus completeness follows. $A^* + OD_{TU}$ halts when a goal state N_g is expanded. Since we expand in every iteration the state with the minimal $g + h$ and the h of every goal state is zero, then $g(N_g) \leq g(N) + h(N)$ for every state N in the open list. Since h is admissible, we have that $g(N_g)$ is optimal, as required. □

5. CBS under Time Uncertainty (CBS_{TU})

The second algorithm we propose for MAPF-TU is called CBS_{TU}. CBS_{TU} is based on the Conflict-Based Search (CBS) algorithm (Sharon et al., 2015). For completeness, we provide a brief description of CBS.

5.1 Conflict Based Search (CBS)

CBS (Sharon et al., 2015) is a complete and optimal MAPF solver, designed for classical MAPF. CBS finds a solution by iteratively planning for each agent separately, detecting *conflicts* between these single-agent plans, and resolving them by replanning for the individual agents subject to specific *constraints*. A CBS conflict is defined by a tuple $\langle a_i, a_j, x, t\rangle$, representing that there is a conflict between agents a_i and a_j at vertex/edge x at time t. A CBS constraint is defined by a tuple $\langle a_i, x, t\rangle$, representing that agent a_i cannot occupy x at time step t, where x is either a vertex or an edge. To choose which constraints to add to which agent, CBS maintains a Constraint Tree (CT). The CT is a binary tree, in which each node N represents a solution ($N.solution$) and a set of constraints ($N.constraints$). If $N.solution$ is a valid solution then N is referred to as a goal CT node. Therefore, if N is not a goal CT node then $N.solution$ has a conflict. Expanding a non-goal CT node N means choosing a conflict $\langle a_i, a_j, x, t\rangle$, and generating two child CT nodes N_i and N_j. The constraints of N_i and N_j are all the constraints of N, as well as the constraints $\langle a_i, x, t\rangle$ and $\langle a_j, x, t\rangle$, respectively. Then, a search algorithm such as A* is used to generate a single-agent plan for agent a_i that satisfies the constraints in N_i, and to generate a single-agent plan for agent a_j that satisfies the constraints in N_j. The search algorithm used for finding these single-agent plans is referred to as the *CBS low-level search.* These single-agent plans are used in their respective CT nodes instead of the single-agent plans for the constrained agent in N.

CBS searches the CT in a best-first manner according to the cost of the solution in each CT node. The search halts when a goal CT node is chosen for expansion. This best-first search is referred to as the *CBS high-level search.* Many improvements have been suggested to the basic CBS algorithm over the years (Felner et al., 2018; Gange et al., 2019; Lam et al., 2019a; Li et al., 2019).

5.2 From CBS to CBS$_{\text{TU}}$

CBS$_{\text{TU}}$ is an adaptation of CBS for solving MAPF-TU instances. It requires changing how conflicts are defined, how they are resolved, and how the low-level search operates. We describe these changes in detail next.

5.2.1 Conflicts over Potential Presences

There is a CBS$_{\text{TU}}$ conflict in a solution Π iff there is a potential conflict in Π. A CBS$_{\text{TU}}$ constraint is defined by a tuple $\langle a_i, a_j, x, T\rangle$, where a_i and a_j are the conflicting agents, x is the vertex or edge in which the potential conflict exists, and T is the intersection between the potential presences of a_i and a_j on x. That is, if π_i and π_j are the single-agent plans for a_j and a_j then $T = \tau(\pi_i, x) \cap \tau(\pi_j, x)$. Recall that in MAPF-TU there may be edge, vertex, and swapping potential conflicts. Thus, there are respective CBS$_{\text{TU}}$ conflicts that may occur.

5.2.2 Resolving a Conflict

One may be tempted to resolve a CBS$_{\text{TU}}$ conflict $\langle a_i, a_j, v, T\rangle$ by imposing *range constraints*, which are constraints that prevent an agent from occupying any time step in time range T.

Range constraints have been shown to be useful in CBS (Atzmon et al., 2020b; Li et al., 2019). Unfortunately, using these kind of range constraints in our context would lead to a suboptimal or even incomplete algorithm, as shown in the following example.

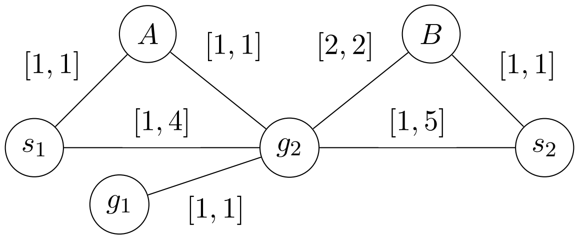

Figure 3: A MAPF-TU instance showing that using naive range constraints in CBS_{TU} leads to suboptimal solutions.

Example 2. *Consider the MAPF-TU instance depicted in Figure 3, and assume the objective is to minimize optimistic SOC. The initial solution* $\Big\{\big((s_1,g_2),(g_2,g_1)\big),\big((s_2,g_2)\big)\Big\}$ *has a vertex conflict* $\langle a_1,a_2,g_2,[1,4]\rangle$*. However, imposing the constraints that agent* a_1 *or agent* a_2 *cannot visit* g_2 *at any time step in* $[1,4]$ *rules outs all optimal solutions. Specifically, in all optimal solutions agent* a_1 *has the single-agent plan* $\big((s_1,A),(A,g_2),(g_2,g_1)\big)$ *and agent* a_2 *has a single-agent plan in which* a_2 *arrives to* g_2 *at time step 4 (either by waiting at* s_2 *for 3 time steps of by moving through* B*).*

In fact, we can even show that sometimes even restricting an agent for two consecutive time steps t and $t+1$, such that t and $t+1$ belong to T, makes the approach suboptimal or even incomplete. Thus, we resolve a vertex conflict $\langle a_i,a_j,v,T\rangle$ by choosing a single time step $t\in T$, and creating two Constraint Tree (CT) nodes N_i and N_j with the added constraints $\langle a_i,v,t\rangle$ and $\langle a_j,v,t\rangle$ respectively. Any time step t in T is a valid choice. The same strategy is used for resolving edge and swapping conflicts.[4] The above approach preserves optimality and completeness, since we know that in any valid solution at most one of the conflicting agents can occupy the conflict location at a given instant.

One may consider imposing range constraints over edges to indirectly resolve a vertex conflict. This can by done by imposing constraints on all the edges leading to or originating from that vertex. However, the resulting range constraints are not more powerful than a single time-step constraint over a single vertex. To see this, consider constraining a vertex v at some time step t. This vertex constraint restricts taking any action that reaches v with a potential presence *that includes* the time step t. Thus, one may view constraining a vertex at a time step as an implicit range constraint over edges that reach that vertex.

5.2.3 Low-Level Search

The low-level search of CBS_{TU} is invoked for a given CT node to find the lowest cost single-agent plan for an agent that satisfies a given set of constraints. We implemented the CBS_{TU} low-level search using the A* algorithm over the following search space. A state in

4. See Appendix B for an example similar to Figure 3, demonstrating why range constraints cannot be trivially used to resolve a swapping conflicts.

this search space is a pair of a vertex and potential presence. The set of actions in a state (v, T) is the set of all single-agent action applicable from v that do not violate the given set of constraints. To optimize the pessimistic SOC, the g value of a state is its pessimistic cost. As an admissible heuristic, we used the shortest single-agent plan to the goal when considering only w^+ for all edges. To optimize the optimistic SOC, one can set g to be the optimistic cost and the same admissible heuristic, except that it considers w^- instead of w^+. In our implementation of the CBS_{TU} low-level search, we break ties between in favor of states that have fewer conflicts with the single-agent plans of the other agents. This tie-breaking technique, often referred to as *using a conflict avoidance table*, is known to be effective in CBS-based algorithms (Sharon et al., 2015) and is indeed effective in CBS_{TU} as well.

5.2.4 The CBS_{TU} Algorithm

Algorithm 1 lists the pseudo code for CBS_{TU}, focusing on the high-level search. The root of the search is a CT node without any constraints on the agents. Initially, OPEN contains only this root node (line 4). Then, in every iteration we extract the best (minimal cost) node N from OPEN (line 6). If *N.solution* does not contain a conflict, we return this solution (lines 8-9). Otherwise, we choose one of the conflicts in that solution (line 10). In our basic implementation of CBS_{TU}, we chose the conflict that manifested earliest in the solution. For each agent a_i in this conflict, we create a new CT node N_i and add an appropriate constraint (lines 11-16). Then, we call the *low-level* solver to find a new single-agent plan that satisfies the old and new constraints (line 18). Finally, we compute the solution cost for N_i and insert the node to OPEN (lines 19-20).

Theorem 2. *CBS_{TU} is sound, complete, and optimal.*

Proof outline. If CBS_{TU} returns a solution, it is associated with a goal CT node, i.e., a CT node without conflicts. Since the conflicts are computed w.r.t. the potential presence, then a solution without a conflict must be a safe solution. Thus, CBS_{TU} is sound. Whenever CBS_{TU} expands a CT node, it means that the CT node is not a solution and it has a conflict. Since the conflict cannot occur in any solution, the solution must exist in one of that node's children. Since the search is done in a best-first manner according to our objective, and it can only increase when adding constraints, then CBS_{TU} is complete and optimal. □

5.3 CBS Enhancements in CBS_{TU}

Many enhancements have been proposed for CBS in recent years, including heuristics for searching the constraint tree (Felner et al., 2018), symmetry detection mechanisms (Li et al., 2019), and conflict prioritization (Boyarski et al., 2015b). Migrating all these enhancements to CBS_{TU} is beyond the scope of this work. Nevertheless, we chose to incorporate in CBS_{TU} two CBS enhancements, Bypassing Conflicts (Boyarski et al., 2015a) and Prioritizing Conflicts (Boyarski et al., 2015b). Both enhancements are known to have a dramatic effect on CBS performance in classical MAPF.

Bypassing Conflicts (BP): BP is a technique for avoiding a conflict by finding a different single-agent plan for one of the conflicting agents that has the same cost as the original plan but bypasses the conflict. The main benefit in BP is that it reduces the number of

Algorithm 1: The CBS_{TU} Algorithm.

```
Input: A MAPF-TU instance with $k$ agents; An optimization criteria, $f$
Output: A set of collision-free single-agent plans
1  $root.constraints = \emptyset$
2  $root.solution \leftarrow$ individual single-agent plans returned by low-level() approach
3  $root.cost \leftarrow SOC(root.solution)$
4  Add root to OPEN
5  while OPEN not empty do
6      $N \leftarrow$ the best node from OPEN        // based on some objective function
7      Validate single-agent plans in $N$ until the first conflict occurs
8      if no conflict found then
9          return $N.solution$
10     $conflict \leftarrow FindConflict(N.solution)$
11     for agent $a_i$ belongs to the conflict do
12         $N_i \leftarrow$ Create a new CT node
13         if vertex-conflict(conflict) then
14             $N_i.constraints \leftarrow N.constraints \cup (a_i, v, t)$
15         else
16             $N_i.constraints \leftarrow N.constraints \cup (a_i, e, t)$
17         $N_i.solution \leftarrow N.solution$
18         Update $N_i.solution$ with a single-agent plan returned by low-level($N_i$) for $a_i$

19         $N_i.cost = SOC(N_i.solution)$
20         Add node $N_i$ to OPEN
21 return No solution found
```

nodes in the constraint tree. Let N be a non-goal CT node, i.e., N contains a conflict $C = \langle a_i, a_j, v, T\rangle$ (or, similarly, a common edge e). Following (Boyarski et al., 2015a), three conditions must be satisfied for single-agent plan ${\pi_i}'$ to be a *valid bypass* for the single-agent plan π_i: (1) ${\pi_i}'$ does not cause the conflict C; (2) both π_i and ${\pi_i}'$ have the same length; and (3) ${\pi_i}'$ is also consistent with $N.constraints$. Let $solution'$ be a vector of all single-agent plans in $N.solution$, except for π_i, which is replaced with ${\pi_i}'$. Bypassing C means setting $N.solution \leftarrow solution'$. We perform a bypassing only when the number of conflicts in $N.solution$ is higher than the number of conflicts in $solution'$. The main difference between BP in CBS and BP in CBS_{TU} is how we count the number of conflicts in a given solution. In CBS_{TU}, for a conflict $C = \langle a_i, a_j, v, T\rangle$, we count one conflict for each time step $t \in T$ such that $\langle a_i, a_j, v, t\rangle$ is a classical MAPF conflict.

Prioritizing Conflicts (PC): PC is a technique for choosing which conflict to resolve in a given CT node. Following (Boyarski et al., 2015b), we divide conflicts into three types: (1) a *Cardinal* conflict, for which a resolution must increase the solution cost in both child CT nodes; (2) a *Semi-cardinal* conflict, for which a resolution must increase the solution cost in one of the two child CT nodes; and (3) a *Non-cardinal* conflict, which is not cardinal or semi-cardinal. It has been shown that resolving cardinal conflicts first, then resolving

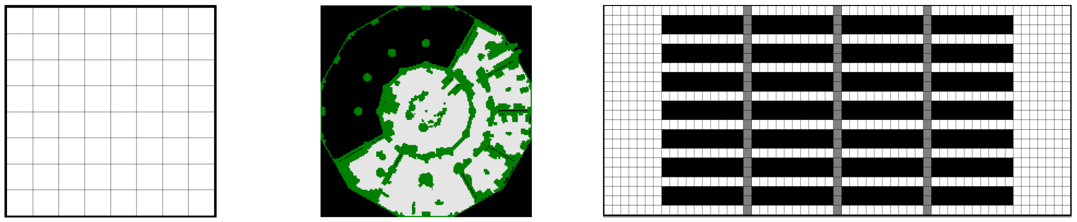

Figure 4: The Open (left), DAO (middle), and Warehouse (right) grids we used in our experiments. For the Warehouse grid, the gray cells mark the locations where uncertainty is introduced.

semi-cardinal conflicts, and, finally, resolving non-cardinal conflicts, reduces the number of CT nodes that need to be expanded. To implement PC in CBS_{TU}, we perform the following modification to Algorithm 1. When a CT node N is chosen such that $N.cost = c$, we find all conflicts in $N.solution$, unlike the basic approach where we choose *the first* conflict encountered. In this process, if a cardinal conflict is encountered, it is immediately chosen for the split action. A possible advantage of this approach is that, since each child N' of node N would have as its cost $N'.cost > c$, if there exists a node N'' in OPEN such that $N''.cost = c$, it can further be developed without expanding N at this point in time. This is safe to do since the cardinal conflict will reappear in all of the nodes in the subtree below N until it is chosen and resolved. However, if the algorithm decides to apply split to N, then it generates all the children for every cardinal conflict in N.

In a preliminary set of experiments, we evaluated the impact of these two CBS improvements on the performance of CBS_{TU}. The results showed that CBS_{TU} with PC performs much better than basic CBS_{TU}. CBS_{TU} with BP also performed better than the basic CBS_{TU}, but adding it on top of PC did not yield significant gains. Therefore, unless stated otherwise hereinafter, we assume that CBS_{TU} is implemented only with PC.

6. Empirical Evaluation

In this section, we evaluate empirically the performance of $\text{A}^* + \text{OD}_{\text{TU}}$ and CBS_{TU} by performing a set of experiments.

6.1 Experimental Setup

All our experiments were performed over the following 4-neighborhood grids:

- **Open.** An 8×8 grid with no obstacles.
- **DAO.** A grid from the `ost003d` map of the game Dragon Age Origins (DAO), made publicly available by Sturtevant (Sturtevant, 2012).
- **Warehouse.** A grid structured like an automated warehouse.

Figure 4 depicts these grids, where black cells and green cells are obstacles.

k	$U=0$		$U=1$		$U=2$		$U=4$	
	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$
3	100%	100%	100%	100%	100%	100%	100%	100%
7	100%	100%	100%	88%	94%	72%	58%	46%
10	100%	87%	80%	43%	54%	13%	12%	0%
15	100%	13%	18%	3%	0%	0%	0%	0%

Table 1: Success rates results for $\mathrm{CBS}_{\mathrm{TU}}$ and $\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ algorithms on the Open grid.

k	$U=0$		$U=1$		$U=2$		$U=4$	
	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$	CBS_{TU}	$\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$
4	100%	100%	52%	17%	44%	23%	22%	13%
7	92%	93%	14%	0%	2%	0%	0%	0%
10	80%	70%	4%	0%	3%	0%	0%	0%
13	70%	27%	0%	0%	0%	0%	0%	0%

Table 2: Success rates results for $\mathrm{CBS}_{\mathrm{TU}}$ and $\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ algorithms on the DAO grid.

To define $w^-(e)$ and $w^+(e)$ for every edge e we introduce a parameter U called the *uncertainty rate*. For a given value of U, we set $w^-(e)$ to be a value chosen uniformly at random from the range $[1, U+1]$, and set $w^+(e)$ to be a value chosen uniformly at random from the range $[w^-(e), U+1]$. Thus, the uncertainty rate parameter U allows control over the amount of uncertainty in the generated experiment, where $U = 0$ means no uncertainty. For each grid, we run experiments with an increasing number of agents, starting from 2 and going up to 20 agents. For each configuration of grid type, U, and number of agents, we created 50 experiments. These 50 experiments differ in the start and goal locations of the agents, which were randomly selected. We set a *timeout* of **five minutes** for each solving each MAPF-TU instance. The *success rate* of an algorithm is the ratio of problems solved before reaching this timeout. Unless stated otherwise, the objective function we optimized for was SOC_{pes}.

The existing MAPF code frameworks that we explored were not easily adaptable to the MAPF-TU setting. So, we implemented all algorithms from scratch, including the two CBS enhancements mentioned in Section 5.3. As such, the performance of our algorithms is not directly comparable to state-of-the-art implementations of algorithms for classical MAPF, even when $U = 0$, and scales less gracefully. Nevertheless, the source code for running all our experiments is publicly available at `https://github.com/Tomer-Shahar/Conformant-CBS`.

6.2 Experimental Results

Table 1 and Table 2 show the success rates of $\mathrm{CBS}_{\mathrm{TU}}$ and $\mathrm{A}^* + \mathrm{OD}_{\mathrm{TU}}$ for different values of U and numbers of agents on the Open and DAO grids, respectively. Rows correspond to different numbers of agents, and columns represent different U values and algorithms. Each inner cell contains success rates.

We observe the following trends. First, increasing U significantly decreases the success rates of the algorithms. For example, in the DAO grid with 7 agents, $\mathrm{CBS}_{\mathrm{TU}}$ solves all prob-

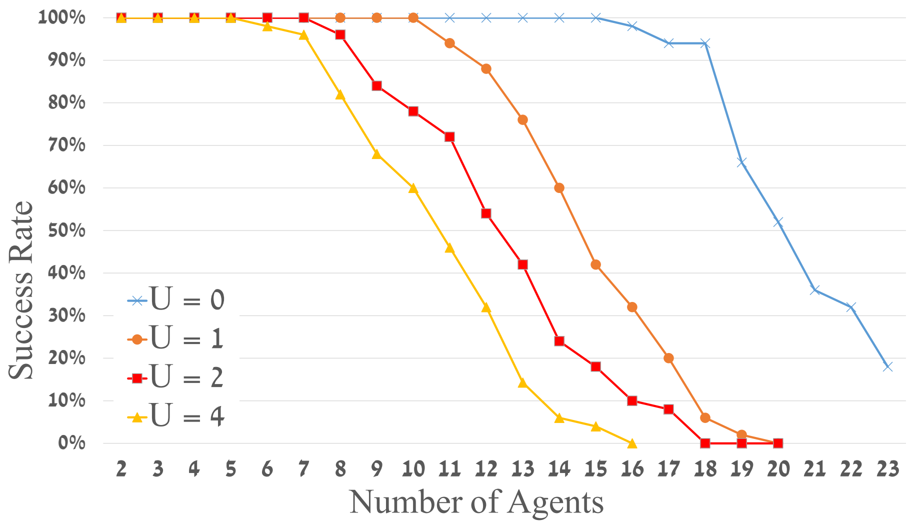

Figure 5: Success rates results for CBS_{TU} on the Open grid.

lem instances with $U = 0$, but solves only 14% for $U = 1$. This trend is expected, as adding U means more conflicts and a longer runtime to resolve new conflicts. Similarly, adding more agents makes problems harder for both algorithms, reducing the overall success rates. Another clear trend is that CBS_{TU} outperforms $\text{A}^* + \text{OD}_{\text{TU}}$ for all the instances except for the case when 7 agents with $U = 0$ on a DAO map. Indeed, CBS is also known to outperform A^* in classical MAPF on similar domains, and thus, the superior performance of its time uncertainty counter-part is expected. As CBS_{TU} generally outperformed $\text{A}^* + \text{OD}_{\text{TU}}$, we show results of only CBS_{TU} hereinafter.

Figure 5 shows the success rate of CBS_{TU} with more agents than in Table 1. As expected adding more agents or increasing the uncertainty rate decreases the success rate. We note that CBS_{TU}'s performance when $U = 0$ is similar to the performance of Improved CBS (ICBS) on a similar grid (Boyarski et al., 2015b).

6.2.1 Sum of Costs Results

To have a deeper understanding of the impact of U on CBS_{TU}'s behavior, Table 3 shows the results of experiments with a wider range of uncertainty rate values (U). These experiments were conducted on the Open grid with 9 agents. The table columns show the SOC_{opt} (the "Opt." column), SOC_{pes} ("Pes."), the difference between them ("Range"), the CBS_{TU} computational runtime ("Time") in seconds, and the success rate ("Success").

As seen in Tables 1 and 2, increasing U in general decreases the success rate. However, this relation between U and success rate is noisy, indicating that the complexity of MAPF-TU is affected by other problem features as well. Similarly, the computational time does not fully correlate with U, but the general trend clearly indicates that higher uncertainty rate makes the problem, in general, harder to solve for CBS_{TU}.

Next, consider the difference between SOC_{opt} and SOC_{pes} (shown in the "Range" column), which we will refer to as the *uncertainty range* of the solution. The results clearly show that the uncertainty range increases significantly with U. For example, the uncertainty

U	*Sum of Cost (SOC)*			*Time*	*Success*
	Opt.	*Pes.*	*Range*		
0	47.7	47.7	0.0	0.04	100%
1	59.5	70.3	10.8	2.59	100%
2	73.6	96.9	23.3	5.59	98%
3	86.4	122.2	35.8	20.2	84%
4	104.0	146.7	42.7	19.4	70%
5	107.3	166.0	58.7	34.3	67%
6	122.5	179.4	56.0	30.1	69%
7	137.3	205.2	67.9	23.9	69%
8	145.9	232.8	86.9	32.6	53%
9	171.5	286.5	115.0	42.0	50%

Table 3: SOC$_{opt}$, SOC$_{pes}$, runtime, and success rate results for Open grid with 9 agents.

k	$U = 0$		$U = 1$		$U = 2$		$U = 4$	
	CU	*PU*	*CU*	*PU*	*CU*	*PU*	*CU*	*PU*
4	100%	100%	96%	100%	92%	98%	86%	96%
7	98%	98%	84%	96%	68%	92%	48%	80%
10	94%	94%	48%	76%	34%	68%	10%	56%
13	82%	82%	18%	42%	6%	28%	0%	18%

Table 4: Success rates results for CBS$_{\text{TU}}$ on the Warehouse grid in the complete uncertainty (CU) and the partial uncertainty (PU) experiment types.

range is 23.3, 42.7, and 86.9 for U=2, 4, and, 8, respectively. More generally, the results show that the uncertainty range increases linearly with U, with an increase of around 11 per U. Note that CBS$_{\text{TU}}$ and A* + OD$_{\text{TU}}$ are both optimal and thus their solutions will have exactly the same SOC$_{pes}$. Their SOC$_{opt}$, however, can differ. While this is not reported these results, we have observed that both algorithms yielded solutions with SOC$_{pes}$ that were almost identical for all the problems, and thus have almost identical uncertainty range.

6.2.2 Warehouse Results

Finally, we present results for the Warehouse grid. Here we performed two types of experiments: *Complete Uncertainty* (CU) and *Partial Uncertainty* (PU). CU experiments are the same as those described above. PU experiments are different in that they introduce uncertainty only on a specific set of edges. These edges are marked in gray in Figure 4 (right). The motivation for PU experiments is to simulate automated warehouse scenarios, where most of the uncertainty is condensed in specific areas such as the pickup depots and packaging areas. Obviously, experiments under PU and CU settings would be the same when $U = 0$.

Table 4 shows the success rate for different number of agents (k) and uncertainty rate (U). The results show that CBS_{TU} performed better with PU than with CU in terms of the success rate. For example, 10 agents and $U = 4$, CBS_{TU} with PU received a success rate of 56%, while CBS_{TU} with CU received only a success rate of 10%. A similar trend was also observed on other grids. This is expected, since PU has strictly less uncertainty than CU.

To summarize, we demonstrated that both CBS_{TU} and $\text{A}^* + \text{OD}_{\text{TU}}$ can be applied to find safe and optimal solutions on real MAPF benchmarks. Increasing the uncertainty rate lowers the success rate and increases the SOC and uncertainty range of the returned solutions. If we compare the CBS_{TU} and $\text{A}^* + \text{OD}_{\text{TU}}$ approaches, from the experiments we conducted, we observe that CBS_{TU} performs significantly better.

7. Safe Replanning

In some scenarios, acting agents can *sense* their environment, i.e., observe during the execution something that was unknown previously, and *replan* based on their updated knowledge (Seuken & Zilberstein, 2008). In this section, we consider how sensing and online replanning can be useful in the context of MAPF-TU. The type of sensing we focus on is where an agent can sense the current time when it arrives at a vertex v. This eliminates the uncertainty embodied by the potential presence of that agent at v. Therefore, replanning at this stage may yield a better solution.

In the rest of this section, we investigate two sensing and replanning settings. In the first, every agent attempts to improve its current single-agent plan individually while ensuring safety of the resulting overall solution. In the second setting, the agents share sensing information with a central controller that may replan for multiple agents at once. We refer to the first setting as SENSE and the second setting as SENSE+COM. In both settings, we assume that the agents are initially given a safe solution to the current MAPF-TU instance, computed offline using a MAPF-TU algorithm such as CBS_{TU} or $\text{A}^* + \text{OD}_{\text{TU}}$.

For each setting, we provide theoretical examples showcasing the potential gains of replanning and propose concrete replanning algorithms. Section 8 shows experimental results that measure the performance of these algorithms on MAPF-TU instances. The results confirm that our algorithms for SENSE and SENSE+COM can result in executing a solution with a lower cost.

7.1 The SENSE Setting

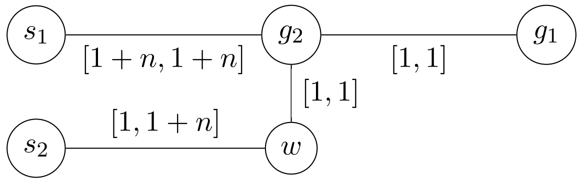

Figure 6: MAPF-TU instance showing the benefits of replanning when *sensing* might occur. Compared to the optimal, purely offline solution, the replanning solution decreases the pessimistic sum-of-costs from $4 + 3n$ to $4 + 2n$, when sensing actually occurs at vertex w.

The following example demonstrates the potential benefit of sensing in the SENSE setting.

Example 3. *Consider the MAPF-TU instance shown in Figure 6, and the solution comprising the single-agent plan $\big((s_1, g_2)(g_2, g_1)\big)$ for a_1 and the single-agent plan $\big((s_2, w) \dots (w, g_2)\big)$ for a_2, where the agent waits in w for n time steps. This solution is safe. Moreover, it is an optimal solution, in terms of both optimistic and pessimistic SOC (which are $4 + 2n$ and $4 + 3n$ respectively). However, if we allow replanning then when agent a_2 reaches vertex w it can decide to reduce the amount of waiting based on how long it actually took to go from s_2 to w. For example, assume that it took n time steps for a_2 to reach w. Following the original safe solution, the agent would wait in w for n additional time steps before moving to g_2, reaching its goal at time $2n + 1$. By sensing that it reached w at time step n, the agent can reduce the planned wait time to a single time step, reaching its goal at time $n+2$. In general, through sensing and replanning optimistic and pessimistic SOC can be reduced down to $4 + 2n$ and $4 + 2n$, respectively.*

7.1.1 Replanning Algorithm

We propose the following replanning algorithm for the SENSE setting. Let a_i be an agent following a single-agent plan π_i that senses it has just reached vertex v and the current time is t. Our replanning algorithm searches for the lowest-cost single-agent plan from v to the goal of a_i, that is guaranteed to avoid conflicts with the other agents' execution of their single-agent plans. To this end, we restrict the new single-agent plan π'_i such that the potential presences it induces are subsets of the potential presences induced by π_i. This means we can only choose to decrease the duration of its planned wait actions. Finding an optimal way to do this is a trivial—each agent simply waits until it reaches the lower time bound of its next non-wait action.

7.1.2 Theoretical Analysis

Let Π be the MAPF-TU solution prior to replanning and let Π' be the MAPF-TU solution after replanning. Since the potential presences according to Π' are either the same or a subset of the potential presences in Π, the solution Π' must be safe. Introducing replanning cannot lead to worse SOC bounds since the search space for each agent during replanning includes its original single-agent plan. Thus, in the worst case the replanning will not change the current single-agent plan and $SOC(\Pi) = SOC(\Pi')$. Since our replanning algorithm only removes wait actions, the best-case reduction in SOC (i.e., $SOC(\Pi) - SOC(\Pi')$) is bounded by the number of *wait* moves in the offline solution.

7.2 The SENSE+COM Setting

In the SENSE+COM setting, agents share the information they sensed. This sharing occurs when an agent reaches a vertex and it is able to sense. Thus, the input to replanning here is a set of agents that arrived at a vertex at the same time and were able to sense. We call this set of agents the *replanning agents.* The following example demonstrates the potential benefit of sensing in the SENSE+COM setting, showing that this setting allows further reduction in SOC over what is possible in the SENSE setting.

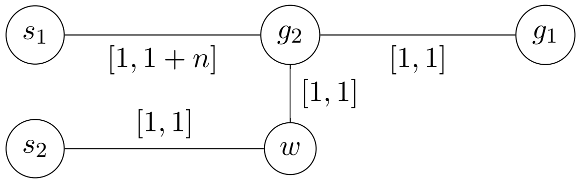

Figure 7: MAPF-TU instance showing the benefits of replanning when sensing and *communication* might occur. Compared to the optimal, purely offline solution, the replanning solution decreases the optimistic SOC from $4+n$ to 4, when sensing occurs by agent a_1 in vertex g_2.

Example 4. *Consider the MAPF-TU instance shown in Figure 7, and the solution comprising the single-agent plan $\big((s_1, g_2)(g_2, g_1)\big)$ for a_1 and the single-agent plan $\big((s_2, w) \dots (w, g_2)\big)$ for a_2, where the agent waits in w for n time steps. This solution is safe and optimal in terms of SOC (the pessimistic and optimistic SOC is $4+n$ and $4+2n$, respectively).*

However, if we allow replanning and communications, and if agent a_1 senses when reaching vertex g_2 and broadcasts its arrival time, then agent a_2 can decide to reduce the amount of waiting based on how long it took going from s_1 to g_2. In the optimal case where agent a_1 reaches g_2 at time 1, the SOC range for the replanning approach is $[4, 4+2n]$.

7.2.1 Replanning Algorithm

We propose to use CBS_{TU} to generate a solution for the replanning agents that takes them from their current locations to their respective goal locations. To maintain safety, we initialize CBS_{TU} with a set of CBS_{TU} constraints that cover every potential presences of every agent that is not a replanning agent. This ensures that the agents with new single-agent plans will never conflict with agents that have not sensed. This is true even when the replanning phase assigns the agents a completely different solution to execute. This is why communication adds a facet of flexibility to replanning.

7.2.2 Theoretical Analysis

Let Π be the MAPF-TU solution prior to replanning, Π_{WOC} be the MAPF-TU solution created after replanning under the SENSE setting, and Π_{WC} be the MAPF-TU solution created after replanning under the SENSE+COM setting. The ability to communicate expands the set of single-agent plans that may be assigned to the replanning agents. Thus,

$$SOC(\Pi) \geq SOC(\Pi_{WOC}) \geq SOC(\Pi_{WC}) \tag{4}$$

for the SOC being optimized (pessimistic or optimistic). Note, that $SOC(\Pi)$ is the SOC of the plan Π, as oppose to the cost of executing Π. For example, it may be that SOC(Π)=10 but executing Π would cost more. Thus, Equation 4 does not mean executing Π will necessarily cost more than executing Π_{WOC}, and that executing Π_{WC} will necessarily cost less than Π_{WOC}.

Time complexity Since this variant uses a centralized planner, the runtime of every replanning step is in the worst-case exponential in the number of replanning agents. However, the overall computation time is typically dominated by the time it takes to compute the initial solution since it always takes into consideration all the agents simultaneously with maximal uncertainty. At a certain time-step, even if all the agents were to sense and re-plan simultaneously sometime during the execution, this would still be a considerably easier MAPF-TU instance to solve compared to the initial, offline solution that was already found. This is because much of the time uncertainty is removed from the solution up to this point leading to fewer potential conflicts over shorter time intervals across the entire solution. Indeed, we observed in our experiments that all subsequent replanning stages were on an average several times faster than the time required to compute the initial solution.

Safety The output of each replanning stage is guaranteed to be safe. This occurs because whenever replanning happens, each agent either sensed (and subsequently replans) or it did not sense (since it is currently traveling or did not sense for any other reason). For all agents that did not sense, their current single-agent plans are treated as constraints for all the agents that will replan. Then, the central planner computes new single-agent plans for the sensing agents, using a *safe* MAPF-TU algorithm. Since the other agents are taken into account through constraints, and a safe MAPF-TU solver is used, the solution found is guaranteed to contain no collisions in the future. Thus, the output for each replanning stage is a safe MAPF-TU solution.

8. Empirical Evaluation: Online Replanning

In this section, we experimentally evaluate the benefit of using our online replanning algorithms for the SENSE and SENSE+COM setting. We performed a range of experiments and report here only a representative subset of them. Specifically, we report here the results only for the Open grid and for the Warehouse grid with partial uncertainty (PU). In each experiment, we first obtained an initial safe solution using CBS_{TU}. Then, we simulated the execution of this solution by sampling uniformly the edges' durations within their ranges. Whenever an agent finishes traversing an edge, it applies the appropriate replanning algorithm. Two types of experiments were performed, one where the objective is to minimize SOC_{opt} and one where the objective is to minimize SOC_{pes}. The main performance metric we consider here is the *Final SOC* which is the SOC (either SOC_{opt} or SOC_{pes}) of the solution the agents ended up executing, following all the performed replanning.

8.1 Results for the Open grid

Table 5 shows the Final SOC results for 8 agents in the Open Grid when optimizing for SOC_{opt} (left) and for SOC_{pes} (right). The *Initial SOC* is the SOC (either SOC_{opt} or SOC_{pes}) of the initial solution. The SENSE and S+C columns represent the SENSE and SENSE+COM settings, respectively. Each line corresponds to a different uncertainty rates ($U = 0, 1, \ldots, 8$). Every data cell is an average over 50 problem instances. Note that increasing U inherently increases the SOC since it increases the edge traversal duration.

The results in Table 5 show that online replanning in this set of experiments is only marginally beneficial. This can be seen when comparing the Final SOC with the Initial SOC—if they are the same then there is no benefit to online replanning. The largest difference between the average Final SOC and Initial SOC is for $U = 7$ in the SENSE

U	*Initial SOC*	*Final SOC*	
		SENSE	*S+C*
0	43.06	43.06	43.06
1	59.32	58.22	58.05
2	72.89	72.61	72.96
3	93.45	90.79	91.10
4	111.09	109.10	107.64
5	115.60	118.67	111.07
6	129.53	137.96	127.33
7	155.57	150.10	148.57
8	166.89	163.47	159.22

U	*Initial SOC*	*Final SOC*	
		SENSE	*S+C*
0	43.06	43.06	43.06
1	58.58	58.16	58.63
2	74.44	75.70	73.56
3	93.76	92.74	91.88
4	116.04	116.41	114.13
5	124.07	125.02	120.04
6	137.63	138.90	134.60
7	163.10	158.98	160.68
8	176.07	173.42	173.81

Table 5: Initial and Final SOC results for 8 agents in the Open grid, for the SENSE and SENSE+COM (S+C) settings. The left side shows results for SOC$_{opt}$ and the right side shows results for SOC$_{pes}$.

Agents	*U=1*	*U=2*	*U=3*	*U=4*
2	11%	9%	12%	17%
3	11%	9%	11%	15%
4	9%	9%	10%	14%
5	8%	8%	9%	13%
6	7%	7%	9%	11%
7	7%	8%	9%	9%
8	6%	7%	9%	10%
9	6%	7%	9%	-
10	5%	7%	9%	-
11	4%	-	-	-
12	4%	-	-	-

Agents	*U=1*	*U=2*	*U=3*	*U=4*
2	0%	0%	0%	0%
3	0%	0%	0%	1%
4	0%	0%	0%	0%
5	0%	0%	1%	1%
6	0%	1%	1%	2%
7	0%	2%	1%	2%
8	0%	1%	2%	2%
9	0%	2%	2%	3%
10	0%	2%	2%	2%
11	1%	1%	1%	2%
12	1%	1%	1%	1%

Table 6: Overall SOC reduction, SENSE+COM setting, Open grid. (left) Results for optimizing SOC$_{opt}$, (right) Results for optimizing SOC$_{pes}$.

setting (150.10 Final SOC vs. 155.57 Initial SOC) and for $U = 8$ in the SENSE+COM setting (159.22 Final SOC vs. 166.89 Initial SOC). In almost all cases, the Final SOC when in the SENSE+COM setting is smaller than the Final SOC in the SENSE setting, which supports our theoretical analysis in the previous section.

Next, we focus on the SENSE+COM setting and consider the impact of changing the number of agents on the ratio between the Final SOC and the Initial SOC. We refer to this ratio as the *reduction in SOC* due to online replanning. Table 6 shows the reduction in SOC for different numbers of agents (the table rows) and values of U (the columns), in the SENSE+COM setting. The left and right sides show results when optimizing for SOC$_{opt}$ and SOC$_{pes}$, respectively. Cells marked with "-" indicate that the success rate is 10% or less, to avoid drawing conclusions from such experiments.[5]

5. Table 5 shows results for experiments with fewer agents, and so in all cases there the success rate was higher than 10%.

U	*Initial SOC*	*Final SOC*	
		SENSE	*S+C*
0	240.00	240.00	240.00
1	265.31	265.11	255.83
2	283.29	282.79	257.88
3	305.43	305.10	260.95
4	316.17	315.39	254.78
5	338.43	337.57	264.29
6	346.46	345.85	264.46
7	368.38	367.62	267.15
8	423.10	421.91	280.60

U	*Initial SOC*	*Final SOC*	
		SENSE	*S+C*
0	240.00	240.00	240.00
1	248.70	251.07	248.50
2	262.55	266.12	262.05
3	264.44	272.18	264.56
4	268.30	268.30	268.47
5	270.45	270.45	270.66
6	272.97	272.97	273.84
7	276.68	276.68	276.54
8	285.80	285.80	284.39

Table 7: Initial and Final SOC results for 8 agents in the Warehouse grid with PU, for the SENSE and SENSE+COM (S+C) settings. The left side shows results for SOC_{opt} and the right side shows results for SOC_{pes}.

The results in this table show several interesting trends. First, we see that the effectiveness of online replanning (i.e,. the reduction in SOC) increases with U. This occurs because online replanning benefits from reducing the uncertainty through sensing (and communication in the SENSE+COM setting). Having more uncertainty (higher U) therefore provides more opportunities for the online replanning to improve on the initial safe solution.

The second trend we observe is that here online replanning was significantly more effective when optimizing for SOC_{opt} then when optimizing for SOC_{pes}. For example, for $U = 2$ and 7 agents, in the SOC_{opt} setting the reduction in SOC is 8% while in the SOC_{pes} the reduction in SOC is only 2% for the same U and number of agents. We conjecture that this difference between the objective functions is because aiming to minimize SOC_{pes} has an indirect effect of decreasing the SOC range of the initial solution. Since the Final SOC is planned to be within the SOC range, a smaller SOC range means limited opportunity for online replanning to improve over the Initial SOC. Thus, the reduction in SOC for SOC_{opt} is significantly larger.

8.2 Warehouse Domain

We performed a similar experiment on the Warehouse grid with 8 agents, where Table 7 follows the same format as Table 5. The results here emphasizes the trends observed for Open grid. First, the reduction in SOC when optimizing for the SOC_{pes} objective is negligible. Similarly, there is no significant reduction in SOC in the SENSE setting, i.e, when agents cannot communicate. However, there is significant reduction SOC in the SENSE+COM setting when optimizing for SOC_{opt}. This reduction increases with U, where for $U = 8$, the Initial SOC was 423.10 while the Final SOC is 280.60. This is expected, as a higher U means more uncertainty, a larger SOC range, and thus more opportunity for sensing to reduce uncertainty and lead to a better solution.

8.3 Experiments Summary

In all experiments, we observed the following trends.

- A larger SOC reduction in the SENSE+COM setting compared to the SENSE setting.
- Very limited SOC reduction in the SENSE setting.
- A larger SOC reduction when optimizing for SOC_{opt} than when optimizing for SOC_{pes}
- A larger SOC reduction when there is a higher uncertainty rate (U)

The first trend corroborates our theoretical analysis, demonstrating that having the ability to share sensing information and coordinate with other agents is beneficial in MAPF-TU. The second trend suggests that the practical benefit in the SENSE setting with our replanning algorithm is limited. The third and fourth trends are related to the "size" of the initial uncertainty. When there is limited uncertainty over the duration of the initial solution, exhibited by a small SOC range, then there is limited opportunity for online replanning to improve the Final SOC by gaining information during execution.

9. Discussion

There can be many variants and extensions to the MAPF-TU problem and the MAPF-TU solvers we proposed. In this section, we outline several key extensions.

9.1 Offline Planning for MAPF-TU with Sensing

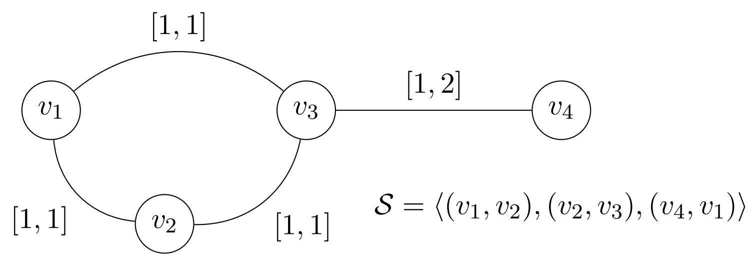
$[1,1]$
$[1,2]$
v_1
v_3
v_4
v_2
$[1,1]$
$[1,1]$
$\mathcal{S} = \langle (v_1, v_2), (v_2, v_3), (v_4, v_1) \rangle$

Figure 8: MAPF-TU with *sensing* instance showing that replanning is not complete. The initial and goal vertices of the three agents are indicated in the bottom right. A policy-based solution consists in Agents 1 and 2 waiting for 2 time steps, then moving to their goal vertices, and Agent 3 moves immediately to v_3, then depending on the sense elapsed time, moves to its goal vertex or waits one time step and then moves.

In Section 7, we proposed an *online* approach to consider agents' ability to sense their current location and time. An alternative approach is to consider the agents' ability to sense their location offline, i.e., when executing the solution. This means the planning algorithm will output an execution policy for each agent instead of a sequence of *moves* and *waits*. A policy may instruct the agent to perform different actions depending on the information it obtains via sensing. Considering sensing opportunities in this way can lead to significant cost reductions, compared to the replanning approach proposed in Section 7. Some problems cannot be solved with the replanning approaches we proposed, but can be solved with a carefully crafted offline policy.

Figure 8 illustrates an example of such an MAPF-TU problem instance. For this problem instance, there exists no safe, offline solution. This implies that we cannot perform online replanning. For safety, our current online approach begins by executing an offline solution and improves upon it. However, creating offline policies for multi-agent problems can be extremely difficult, e.g., in models such as Decentralized-POMDP (Bernstein et al., 2002). Developing efficient offine algorithms for MAPF-TU is left for future work.

9.2 Bounding Edge Traversal Time From Experience

The main assumption in a MAPF-TU problem is that the upper and lower bounds on edges' durations are given as input. While in some cases such information is given as input, another approach is to *learn* these upper and lower bounds from experience. We describe here one way to do so.

Consider a setting where the time it takes to traverse an edge e is independent across executions and drawn from some unknown distribution D_e of edge-traversal durations. For every edge e, we collect $M(e)$ samples $e_1, \dots e_{M(e)}$ from D_e. Each sample is collected from a different execution, since multiple traversals of e may be correlated with each other. Note, however, that we do assume that every traversal of e, taken in isolation, has the same distribution D_e.

Let $L(e) = \min_{i\in\{1,\dots,M(e)\}} e_i$ and $U(e) = \max_{i\in\{1,\dots M(e)\}} e_i$ be the minimum and maximum observed duration it took to traverse e. The probability that the duration of a future traversal of e will be outside the interval $[L(e), U(e)]$ is given by the following theorem.

Theorem 3. *If $M(e) \geq \Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$, then with probability $1-\delta$ over the prior executions, the delay on edge e lies in the interval $[L(e), U(e)]$ with probability $1-\epsilon$.*

In other words, if the number of samples $M(e)$ is at least $\Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$, then with probability higher than $1-\delta$ the L and U of our sample satisfies that the probability that traversing e will take a duration not in $[L(e), U(e)]$ is smaller than ϵ. The proof of Theorem 3 is given in Appendix A.

Now, consider a MAPF-TU instance in which we set the lower and upper bounds edge durations of every edge e to be the observed lower and upper bounds $L(e)$ and $U(e)$, respectively. I.e., setting $w^-(e)$ and $w^+(e)$ to be $L(e)$ and $U(e)$. We call this MAPF-TU instance an *empirical MAPF-TU instance.* We can use Theorem 3 to upper bound on the probability that a solution to an empirical MAPF-TU instance will cause a collision. In other words, we can compute a lower bound to the probability that a solution to an empirical MAPF-TU instance is indeed safe w.r.t. the real world.

To do so, we introduce some additional notation. Let Π be a solution to an empirical MAPF-TU instance and let E be the number of distinct edges in Π. For any $\delta' \in (0,1)$, let $\epsilon(e,\delta')$ be the minimum ϵ for which the bound of Theorem 3 holds for the number of examples we possess for e and δ set to δ'/E. That is,

$$\epsilon(e) = O\left(\frac{1}{M(e)}\log\frac{E}{\delta'}\right) \tag{5}$$

Theorem 4. *Let $\Pi = \{\pi_1, \dots, \pi_k\}$ be a solution to an empirical MAPF-TU problem. Then for $\delta, \epsilon > 0$ with probability $1-\delta$ we obtain lower and upper bounds for every edge such that the probability that Π can be safely executed is at least $1 - \sum_{i=1}^{k}\sum_{e\in\pi_i}\epsilon(e)$.*

Proof. Π is a safe solution to the corresponding empirical MAPF-TU instance. Therefore, if executing Π leads to a collision, then the duration of at least one move action in π exceeded the lower or upper bound of the corresponding edge duration. Taking the union bound over all move actions of all agents, we obtain that the probability of this occurring is at most $\sum_{i=1}^{k}\sum_{e\in\pi_i}\epsilon(e)$. Hence the probability no collisions occur is as claimed. $\square$

The bound $[L(e), U(e)]$ is always valid for the given $\epsilon(e)$ we obtain as a function of δ and $M(e)$. However, if we are seeking a particular safety level, say some target ϵ' for each edge giving an overall bound that the solution is safe with probability $1-\epsilon$, then the bounds thus obtained may be too conservative. Eventually, especially with a large sample size, our examples will include "outlier" events that occur with probability far less than ϵ'. Here is an alternative for large sample sizes that will allow us to approach an optimal interval $[L, U]$ for a given safety bound $1-\epsilon$.

Theorem 5. *For any $\alpha > 1$, $\delta \in (0,1)$, and $\epsilon \in (0,1)$, let $[L(e), U(e)]$ be any interval containing at least $(1-\frac{\epsilon}{\alpha})M(e)$ examples for e where $M(e) \geq \Omega(\frac{\alpha}{\epsilon(\alpha-1)^2}(\log\frac{\alpha}{\epsilon} + \log\frac{1}{\delta}))$. Then, with probability $1-\delta$ over prior executions, the delay for edge e lies in the interval $[L(e), U(e)]$ with probability $1-\epsilon$.*

The proof for Theorem 5 is also given in Appendix 9.2. This theorem provides greater flexibility when constructing a MAPF-TU instance from empirical data, as one can take for every e any interval in $[L(e), U(e)]$ that captures a sufficient number of samples. Then, one can use this more refined bound in Theorem 4 to obtain a more refined bound on the safety of the generate plans. A deeper study of this is a topic of future work.

10. Conclusion and Future Work

In this paper, we studied the Multi-Agent Pathfinding with Time Uncertainty (MAPF-TU) problem, which is a Multi-Agent Pathfinding (MAPF) problem in which there is uncertainty over the time it takes an agent to traverse an edge. Specifically, for every edge we are given a lower and upper bound on the duration it takes to traverse it. We focused on the problem of finding a *safe* solution to a MAPF-TU instance, which is a solution that is guaranteed to avoid a conflict. To this end, we introduce the notion of *potential presence* and *potential conflicts*, where a solution is safe iff it has no potential conflicts. Then, we propose two algorithms, called $A^* + OD_{TU}$ and CBS_{TU}, that find safe and optimal solutions to a MAPF-TU instance. We implemented these algorithms and compared them experimentally on a variety of settings and maps. The results show that on our benchmark set of instances CBS_{TU} significantly outperforms $A^* + OD_{TU}$ in terms of success rate.

Then, we considered two online replanning settings, SENSE and SENSE+COM, where the agents have sensing or communicating capabilities while executing a solution. We demonstrate the potential benefit of online replanning in both settings, and propose online replanning algorithms that can improve the executed solution cost. We analyze these replanning algorithms theoretically, showing that using them is always advantageous. However, experimentally, we observed that signifcant benefit for replanning only appeared in the SENSE+COM when optimizing for the optimistic SOC.

Finally, we discussed possible extensions to MAPF-TU. One such extension is to considering offline the possibility that the agents will sense new information and replan online.

Another extension suggests a statistical analysis that allows extracting from observed data the lower and upper bound edge traversal times. Yes another extension is where there lower and upper bound also on wait actions, e.g., for cases where an agent is not allowed to stay too long in some locations.

There are many directions for future work. One can explore other objective functions to the MAPF-TU problem, such as minimizing the expected cost (assuming the traversal time distribution is known). Another direction is to adapt additional MAPF solvers to solve MAPF-TU, e.g., ICTS (Sharon et al., 2013), EPEA* (Goldenberg et al., 2014), or M* (Wagner & Choset, 2015). Finally, one may explore how to extend MAPF-TU to cases in which time is not discretized and the environment is continuous (Andreychuk et al., 2019), or when the bounds over the edge traversal durations change over time.

Acknowledgments

This research is partially funded by NSF award IIS-1908287 to Brendan Juba; and BSF grant #2018684 and ISF grant # 210/17 to Roni Stern.

Appendix A. Sample Complexity Proofs

In this appendix, we provide formal proofs for the main Theorems in Section 9.2.

A.1 Proof for Theorem 3

This confidence bound is an easy consequence of basic statistical learning theory. We recall the number of examples needed to fit a class of Boolean functions is determined by the *VC-dimension* (Vapnik & Chervonenkis, 1971) of that class:

Definition 1. *A set of points of a domain X is* shattered *by a class $\mathcal{C}$ of Boolean functions on X if for every Boolean labeling of the set, there is some $c \in \mathcal{C}$ that gives each point in the set the desired label. $\mathcal{C}$ is then said to have* VC-dimension *d if the size of the largest set shattered by $\mathcal{C}$ contains d points.*

The exact asymptotic dependence of the confidence δ and accuracy ϵ obtainable for classes of a given VC-dimension and a given number of examples was recently determined by Hanneke (2016). We state this in slightly simplified form for our purposes[6]:

Theorem 6 (cf. (Hanneke, 2016))**.** *Let $\mathcal{C}$ be a class of Boolean functions of VC-dimension d. Then, for every $\delta, \epsilon \in (0, 1)$, with probability $1 - \delta$ every $c \in \mathcal{C}$ that is true on $\Omega(\frac{1}{\epsilon}(d + \log \frac{1}{\delta})$ examples independently drawn from a distribution D on X satisfies $\Pr_{x \in D}[c(x) = 1] \geq 1 - \epsilon$.*

This bound is asymptotically optimal, but the constants hidden by the Ω are large. Other forms of this bound that feature an additional $\log \frac{1}{\epsilon}$ factor may give a better quantitative guarantee for the concrete values of ϵ that occur in practice.

Theorem 3 is now an immediate consequence:

6. We obtain this statement from the usual form by fixing the "labels" to be all 1, since we are looking for a set that contains all of the examples seen so far.

Proof. Recall that the set of intervals has VC-dimension 2: it is easy to verify that no set of three points on the real line can be shattered (either two are identical or we can label the max/min points 1 and the middle point 0). It then follows from Hanneke's bound (Theorem 6) that with probability $1-\delta$, any interval that contains $\Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$ examples drawn independently from some common distribution D will contain further points drawn from D with probability $1-\epsilon$. In particular, $[L, U]$ is such an interval. $\square$

A.2 Proof for Theorem 5

To prove this Theorem, we will need the relative-error "agnostic"/"non-realizable" version of the VC-dimension bound (Li et al., 2001), again stated in simplified form for our purposes[7]:

Theorem 7 (cf. (Li et al., 2001))**.** *Let $\mathcal{C}$ be a class of Boolean functions of VC-dimension d, let $\epsilon, \delta \in (0,1)$, and let $\alpha > 1$. With probability $1-\delta$ every $c \in \mathcal{C}$ that is true at least a $(1-\epsilon)$ fraction of $m \geq \Omega(\frac{1}{\epsilon(\alpha-1)^2}(d\log\frac{1}{\epsilon}+\log\frac{1}{\delta}))$ examples satisfies $\Pr[c(x)=1] \geq 1-\alpha\epsilon$.*

Theorem 5 is now immediate:

Proof. Again, the class of intervals has VC-dimension 2. Thus, by Theorem 7 with ϵ set to ϵ/α, we see that if $M(e) \geq \Omega(\frac{\alpha}{\epsilon(\alpha-1)^2}(\log\frac{\alpha}{\epsilon}+\log\frac{1}{\delta}))$, every interval $[L, U]$ that is true of $(1-\frac{\epsilon}{\alpha})M(e)$ examples indeed satisfies $\Pr[e \in [L, U]] \geq 1-\alpha\frac{\epsilon}{\alpha} = 1-\epsilon$. $\square$

Appendix B. Unsound Range Constraints for Edge Conflicts

The following example demonstrate that using a naive way to set range constraints to resolve a swapping conflict may lead to finding a suboptimal solution.

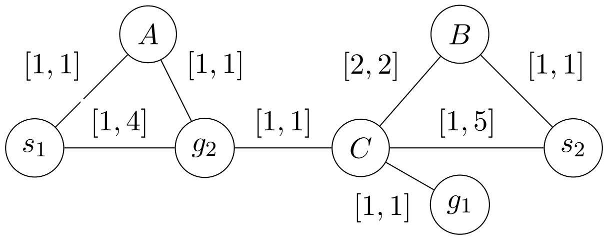

Figure 9: A MAPF-TU instance showing that using naive range constraints for edge-based conflicts in CBS_{TU} leads to suboptimal solutions.

Example 5. *Consider the MAPF-TU instance depicted in Figure 9, and assume the objective is to minimize optimistic SOC. The initial solution*

$$\Big\{\big((s_1, g_2), (g_2, C), (C, g_1)\big), \big((s_2, C), (C, g_2)\big)\Big\}$$

has a swapping conflict $\langle a_1, a_2, (g_2, C), [2, 5]\rangle$. However, imposing the constraints that agent a_1 or agent a_2 cannot traverse the edge (g_2, C) at any time step in $[2, 5]$ rules outs all

7. In Li et al's statement, we actually put $\nu = 2\epsilon$ and replace α by $\frac{\alpha-1}{4+(\alpha-1)} \geq \frac{\alpha-1}{4}$, so that if r is the true error and $s < \epsilon$ is the empirical error, the obtained bound on $d_\nu(r, s) = \frac{|r-s|}{r+s+\nu}$ of $\frac{\alpha-1}{4+(\alpha-1)}$ implies $r < \alpha\epsilon$.

optimal solutions. Specifically, in all optimal solutions agent a_1 has the single-agent plan $\big((s_1, A), (A, g_2), (g_2, C)(C, g_1)\big)$ and agent a_2 has a single-agent plan in which a_2 arrives to C at time step 4 and moves along (g_2, C) immediately after.

References

Andreychuk, A., Yakovlev, K. S., Atzmon, D., & Stern, R. (2019). Multi-agent pathfinding with continuous time. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 39–45.

Atzmon, D., Stern, R., Felner, A., Sturtevant, N. R., & Koenig, S. (2020a). Probabilistic robust multi-agent path finding. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 29–37.

Atzmon, D., Stern, R., Felner, A., Wagner, G., Barták, R., & Zhou, N.-F. (2020b). Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, *67*, 549–579.

Barták, R., Zhou, N.-F., Stern, R., Boyarski, E., & Surynek, P. (2017). Modeling and solving the multi-agent pathfinding problem in picat. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 959–966.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, *27*(4), 819–840.

Boyarski, E., Felner, A., Sharon, G., & Stern, R. (2015a). Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 47–51.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., & Shimony, S. E. (2015b). ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 740–746.

Brafman, R. I., & Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pp. 28–35.

Cashmore, M., Coles, A., Cserna, B., Karpas, E., Magazzeni, D., & Ruml, W. (2019). Replanning for situated robots. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 665–673.

Cimatti, A., Do, M., Micheli, A., Roveri, M., & Smith, D. E. (2018). Strong temporal planning with uncontrollable durations. *Artificial Intelligence*, *256*, 1–34.

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, *147*(1), 35–84.

Cohen, L., Uras, T., Kumar, T. S., & Koenig, S. (2019). Optimal and bounded-suboptimal multi-agent motion planning. In *Symposium on Combinatorial Search (SoCS)*.

Coles, A., & Coles, A. (2014). PDDL+ planning with events and linear processes. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 74–82.

Coles, A., Fox, M., Long, D., & Smith, A. (2008). Planning with problems requiring temporal coordination. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 892–897.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*(1-3), 61–95.

Erdem, E., Kisa, D. G., Oztok, U., & Schüller, P. (2013). A general formal framework for pathfinding problems with multiple agents. In *AAAI Conference on Artificial Intelligence*.

Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T. S., & Koenig, S. (2018). Adding heuristics to conflict-based search for multi-agent path finding. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 83–87.

Felner, A., Stern, R., Shimony, S. E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N. R., Wagner, G., & Surynek, P. (2017). Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *the International Symposium on Combinatorial Search (SoCS)*, pp. 29–37.

Gange, G., Harabor, D., & Stuckey, P. J. (2019). Lazy CBS: implicit conflict-based search using lazy clause generation. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 155–162.

Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N., Holte, R. C., & Schaeffer, J. (2014). Enhanced partial expansion a*. *Journal of Artificial Intelligence Research*, *50*, 141–187.

Hanneke, S. (2016). The optimal sample complexity of pac learning. *The Journal of Machine Learning Research*, *17*(1), 1319–1333.

Hönig, W., Kumar, T. S., Cohen, L., Ma, H., Xu, H., Ayanian, N., & Koenig, S. (2016). Multi-agent path finding with kinematic constraints. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 477–485.

Lam, E., Bodic, P. L., Harabor, D., & Stuckey, P. J. (2019a). Branch-and-cut-and-price for multi-agent pathfinding. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1289–1296.

Lam, E., Bodic, P. L., Harabor, D. D., & Stuckey, P. J. (2019b). Branch-and-cut-and-price for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1289–1296.

Li, J., Harabor, D., Stuckey, P. J., Ma, H., & Koenig, S. (2019). Symmetry-breaking constraints for grid-based multi-agent path finding. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 6087–6095.

Li, Y., Long, P. M., & Srinivasan, A. (2001). Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, *62*(3), 516–527.

Ma, H., Kumar, S., & Koenig, S. (2017). Multi-agent path finding with delay probabilities. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3605–3612.

Morris, R., Pasareanu, C. S., Luckow, K. S., Malik, W., Ma, H., Kumar, S. T. K., & Koenig, S. (2016). Planning, scheduling and monitoring for airport surface operations. In *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop*.

Pajarinen, J., & Peltonen, J. (2011). Efficient planning for factored infinite-horizon dec-pomdps. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 325–331.

Seuken, S., & Zilberstein, S. (2008). Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, *17*(2), 190–250.

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

Sharon, G., Stern, R., Goldenberg, M., & Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, *195*, 470–495.

Shekhar, S., Brafman, R. I., & Shani, G. (2019). A factored approach to deterministic contingent multi-agent planning. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, pp. 419–427.

Silver, D. (2005). Cooperative pathfinding. In *the First Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 117–122.

Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems.. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 28–29.

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., & Boyarski, E. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *the International Symposium on Combinatorial Search (SoCS)*, pp. 151–159.

Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(2), 144–148.

Surynek, P., Felner, A., Stern, R., & Boyarski, E. (2016). Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*.

Surynek, P. (2012). Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*, pp. 564–576.

Vapnik, V., & Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, *16*(2), 264–280.

Veloso, M. M., Biswas, J., Coltin, B., & Rosenthal, S. (2015). Cobots: Robust symbiotic autonomous mobile service robots. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4423–4429.

Vidal, T., & Fargier, H. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, *11*(1), 23–45.

Wagner, G., & Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, *219*, 1–24.

Wagner, G., & Choset, H. (2017). Path planning for multiple agents under uncertainty. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 577–585.

Wagner, G., Choset, H., & Siravuru, A. (2016). Multirobot sequential composition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2081–2088.

Walker, T. T., Sturtevant, N. R., & Felner, A. (2018). Extended increasing cost tree search for non-unit cost domains.. In *IJCAI*, pp. 534–540.

Walker, T. T., Sturtevant, N. R., & Felner, A. (2020). Generalized and sub-optimal bipartite constraints for conflict-based search. In *AAAI Conference on Artificial Intelligence*, Vol. 34, pp. 7277–7284.

Wurman, P. R., D'Andrea, R., & Mountz, M. (2007). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1752–1760.