Conflict Based Search under Uncertainty and Uncontrollability

Authors
ABC Institute
xyz@ijcai19.org

Abstract

In many real-world scenarios, the time it takes a mobile agent, *e.g.*, a robot, to move from one location to another may vary due to exogenous events and cannot be accurately predicted upfront. This poses a significant challenge especially when planning paths for multiple agents, since temporal coordination is necessary in order to avoid collisions. The problem becomes much severe to tackle when an agent is not sure about the *exact* initial and goal locations of other agents during the plan execution. Each agent maintains a pair of belief states, captures uncertainty over initial and goal states, for each of the remaining agents. In this work, we address this challenge of path planning for multiple agents when there is uncertainty over action duration, initial state and goal state. We propose two algorithms for solving this MAPF problem which are based on the Conflict-Based Search and $\mathrm{A}^* + \mathrm{OD}$ algorithms. They are offline algorithms that strive for a *strong* solution that consists of one plan for each agent that is guaranteed to reach their goal safely under given uncertainty. We prove the soundness, completeness, and optimality *w.r.t.* to various optimality measures, of the proposed algorithms, and demonstrate its applicability on standard multi-agent pathfinding domains.

1 Introduction

In a Multi-Agent Path Finding (MAPF) problem, paths are found for multiple agents where each agent has different start and goal positions, such that the agents do not collide while executing their obtained paths in a distributed manner (without communication). MAPF problems arise for 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], among several other scenarios.

Many recently proposed MAPF solvers scale to large MAPF instances [Sharon *et al.*, 2012; Felner *et al.*, 2018]. They are based on some unrealistic assumptions, for example: 1. agents are always *certain* about their starting positions; 2. a *finite* time to travel a path between two locations; 3. their goal locations are definite too. In a more realistic setting, during the distributed plan execution phase, an cannot always be sure about other agents' initial locations, effects of actions they perform – known as non-deterministic action effect (for ease of exposition we consider only time uncertainty), and their goal locations. There is a well researched sub-area of AI Planning known as conformant planning [Hoffmann and Brafman, 2006; Palacios and Geffner, 2009], which is closely related to this. Conformant planning is the task of generating plans given uncertainty about the initial state and action effects, and without any sensing capabilities during plan execution. The plan should be successful regardless of which particular initial world we start from.

Sometimes, a *time* required to traverse an edge (a path) can be uncertain too. Moreover that the plan executor cannot control it at the same time too. A robot sometimes takes more (less) time to travel from one place to other, depending on the external conditions such as slippery floor or path blockage. Further, an agent might be certain only about its own goal position(s) but uncertain about other agents' goal positions [Bolander *et al.*, 2018]. Basically, the agent is certain about a set of possible goal locations for every other agent. Given all these uncertainties the high level goal still remains the same which is that all agent should reach their goal positions from their starting positions without getting into any collision.

Therefore, if proper uncertainty is not taken care of during centralized planning phase, an agent may follow a completely different time frame during the execution phase. One approach to solve such problems is to do *online planning* for each agent, under certain assumptions about their initial positions and traversal time. This would often lead to a runtime-intensive replanning or plan-execution failures [Ma *et al.*, 2017]. Further, we might have scenarios where agents would not have enough time to replan online [Cimatti *et al.*, 2018]. In the scenarios where online planning is not suitable, each agent having a solution that must be robust enough with respect to the execution time uncertainty. A robust solution is also known as *strong* solution [Cimatti *et al.*, 2018].

In this paper we consider that, for each edge of a given graph, there is an *oracle* that gives us its traversal time frame, $[T_{min}, T_{max}]$. It indicates that the time (an integer value) required in the execution phase to traverse this edge lies in the time frame including both the end values: T_{min} and T_{max}.

However, the *finish time* cannot be determined by the plan executor itself, and that can only be determined by the nature at run time.

As per our extended MAPF settings, suppose that there is a given graph, $G = (V, E)$, and a set of k-agents labeled $a_1 ... a_k$. Being uncertain about the starting location, each agent i maintains its initial belief state $b_0^{a_i}$ such that $b_0^{a_i} \subseteq V$ and a definite goal $g_{a_i} \in V$. For the simplicity, we consider that agents' moves make only deterministic changes that are fully observable. Also, we say that an agent would take an uncertain amount of time (a discrete value that is a whole number in $[T_{min}, T_{max}]$) to traverse an edge $e \in E$ in either direction. On each clock tick an agent either *waits* – means it remains there at the current location, or *moves* towards a neighboring location, say via traversing an edge e. In the latter case, it either occupies e or the neighboring location in the next time step. We consider an edge or a vertex of a graph as a resource that can be provided to at most one agent at a time. Formally, for graph G, at most one agent can occupy a vertex $v \in V$ or an edge $e \in E$ at a given time t.

Our task is to return a set of conformant-paths Π_{a_i} for each agent a_i, considering each location $l \in b_0^{a_i}$ as a possible initial location, consequently, $|\Pi_{a_i}| = |b_0^{a_i}|$. Note that for any two agents, the sets of their possible initial locations are disjoint. For each plan $\pi_{a_i}^l \in \Pi_{a_i}$ there exists no conflicts between $\pi_{a_i}^l$ and each valid plan $\pi \in \Pi_{a_j}$ for all the other agents a_j. The task is to minimize the overall cumulative cost function too, in the process. We denote a cost function as C, which is equal to $\sum_{a_i \in \{a_1 ... a_k\}} \sum_{\pi \in \Pi_{a_i}} LTB(\pi)$. LTB$(\pi)$ stands for Lower Time Bound – the minimum time an agent takes to reach the goal location following its current plan π, which is discussed formally later in the paper. On the other hand, even if agents have known initial states and only uncertainty lies in traversal time, our overall task remains the same. The only difference is that now a_i finds a single path as $|b_0^{a_i}| = 1$.

For an effective plan execution we assume that when an agent finalizes its conformant-path, an oracle provides it with the knowledge of its true initial state. However, the time uncertainty on the edges remains the same during the execution too. **TODO: Can we say this? We take one possible state from b_0^i, for each agent, find plans. Now we know their real initial states. The best case is that they were their true initial states. The worst case to find a completely new set of plans for the true initial states.**

We propose two new algorithms called Conflict Based Search under Time Uncertainty (CBSTU) and Conformant Conflict Based Search under Time Uncertainty (CCBSTU). They are solely based on the Conflict Based Search (CBS) algorithm and are continuums of coupled and decoupled approaches. Basically, the CCBSTU algorithm extends the CBSTU algorithm further. Both the algorithms guarantee that their solutions are respectively *optimal* and *conformant-optimal*. The algorithms CBSTU and CCBSTU are both a-two-level algorithms, in which, at the *top-level* a search is performed in a Constraint Tree (CT). Nodes of CT include constraints on time and location for a single agent. A location in the graph $G = (V, E)$, could be an edge $e \in E$ or a vertex $v \in V$. At *bottom-level* a search is performed at each node of the constraint tree. For each agent, it generates a new path using CBSTU approach and a conformant-path using CCBSTU approach such that they satisfy the constraints imposed on the agent by the high-level CT node.

The rest of the paper is structured as follows. We begin with describing the CBS algorithm, its background, and other related work. We then discuss a general approach to find and resolve a conflict, which is used in both CBSTU and CCBSTU algorithms. This is followed by the main CBSTU and CCBSTU algorithms. Later, we study the algorithms theoretically followed by their empirical evaluations. We conclude with a summary and future work.

2 Background and Related Work

TODO:

2.1 Conflict Based Search

Conflict Based Search (CBS) [Sharon *et al.*, 2012]...

2.2 MAPF under Time Uncertainty

How would CBS look like if the MAPF problem is considered under temporal Uncertainty...

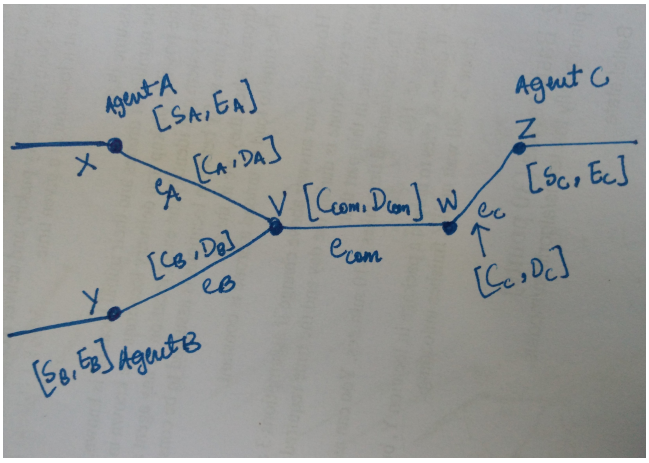

Figure 1: This graph captures three agents A, B and C under the discussed MAPF. The description of our approach is based on this diagram. (**TODO:** Draw this graph properly, though the characters and notations used would remain the same.)

3 To Find and Resolve a Conflict

We demonstrate our example scenario in Figure 1. There are three agents A, B and C, and they occupy vertices X, Y and Z, respectively. $[S_A, E_A]$ indicates the start and end time agent A could possibly occupy vertex X, similarly for agents B and C for the vertices Y and Z, respectively.

Current actions or moves of two agents are *conflicting* if, in the next clock tick, they could possibly occupy the same vertex or edge together which is restricted. Our approach of finding and resolving such conflicts, is an adaptation of the approach used in CBS algorithm.

Vertex Conflict

Consider a case when two agents could possibly occupy a vertex simultaneously by applying a conflicting pair of moves. In Figure 1, agent A could reach V at $T_A = [S_A + C_A, E_A + D_A]$, and agent B could reach V at $T_B = [S_B + C_B, E_B + D_B]$. Their individual moves cause a *vertex conflict* if $T_A \cap T_B \neq \phi$. Assume that, $[t_1, t_2]$ is the intersection of T_A and T_B, which is a common time frame in which A and B occupy vertex V, simultaneously. A vertex conflict is represented as: (A, T_A, B, T_B, V) that reads as agents A and B reach V at T_A and T_B respectively such that $T_A \cap T_B \neq \phi$.

CBS resolves such conflicts by constraining one of the agents such that it does not occupy vertex V at a time instance. Following that an easy solution for our case is to just restrict one of A and B to occupy V in $[t_1, t_2]$. If we deal with time intervals, this approach makes our algorithms incomplete. We could possibly miss an only possible solution due to such strong restrictions. For example: suppose **TODO:** Following this example, we can guarantee that sometimes even restricting an agent for two continuous clock ticks, t and $t+1$, such that t and $t+1$ belong to $[t_1, t_2]$, makes the approach incomplete.

Given a vertex conflict, we know that in any valid solution at most one of A and B can occupy V in $[t_1, t_2]$. We *resolve* it by constraining only one agent at a time, such that it cannot occupy V at an arbitrary $t \in [t_1, t_2]$. Therefore, at least one of the constraints, which are $occupy(A, V, t)$ and $occupy(B, V, t)$ *s.t.* $t \in [t_1, t_2]$, must be added to the set of constraints. A constraint, $occupy(A, V, t)$, specifies that agent A is not allowed to occupy vertex V at time t. In other words, A is not allowed to take a move such that it can reach V at $[t_l, t_m]$ and $t \in [t_l, t_m]$. Note that we choose t in an arbitrary manner. We examine both possibilities to guarantee the solution optimality, which means that current CT node splits into two two children nodes. Both children inherit the constraints from the parent node. Moreover, the left child receives a new constraint $occupy(A, V, t)$ and the right child receives $occupy(B, V, t)$.

Edge Conflict

Uncertainty and uncontrollability help arise two types of *edge conflicts* given our MAPF settings. In Figure 1, when A and B traverse the edge e_{com} from V to W in which one agent might *run-over* the other or possibly occupy e_{com} simultaneously. The other scenario is when the agents have *head-on* collision while they traverse this edge in the opposite directions at the same time.

First Case. Assume that there is no vertex conflict at V such that A occupies V at $[S_A + C_A, E_A + D_A]$ while B occupies V at $[S_B + C_B, E_B + D_B]$ (Figure 1). They traverse the common edge e_{com} from V to W, and reach W respectively at $[S_A + C_A + C_{com}, E_A + D_A + D_{com}]$ and $[S_B + C_B + C_{com}, E_B + D_B + D_{com}]$. This type of edge conflict is only possible if $D_{com} \geq 2$ otherwise no need to look for such conflicts. We assume that $D_{com} \geq C_{com} \geq 1$ and $D_{com} \geq 2$. Here, agent A could possibly occupy e_{com} in the interval $T_A = [S_A + C_A + 1, E_A + D_A + D_{com} - 1]$. Agent A can be on the edge from time $S_A + C_A + 1$ up to $E_A + D_A + D_{com} - 1$. Considering all uncertainty and uncontrollability it is bound to reach W by at most $E_A + D_A + D_{com}$. Similarly for agent B, where it could possibly occupy the edge at $T_B = [S_B + C_B + 1, E_B + D_B + D_{com} - 1]$. Following the previous approach we compute $[t_1, t_2] = T_A \cap T_B$. If the intersection is non-empty, there is an edge conflict, which means, A and B possibly occupy e_{com} together. The conflict is represented as: $(A, B, e_{com}, [S_A + C_A, E_A + D_A], [S_B + C_B, E_B + D_B], [C_{com}, D_{com}])$. Note that if an agent starts moving through e_{com} at t_k then at $t_k + 1$ it can occupy the edge or reach the other end of the edge depending on the time uncertainty. It is also possible that it can keep e_{com} engaged until $t_k + D_{com} - 1$.

Based on how a vertex conflict is resolved in our MAPF framework, to *resolve* this type of an edge conflict, we must restrict one of the agents to occupy e_{com} at $[t_1, t_2]$. Simply restricting agents at $[t_1, t_2]$ would make the algorithm incomplete. For example: **TODO:** . Therefore, one way to resolve this is by restricting A with a constraint: $occupy(A, e_{com}, t)$ where $t \in [t_1, t_2]$. The constraint specifies that A is not allowed to occupy edge e_{com} at t. Similarly, B can be restricted using the constraint: $occupy(B, e_{com}, t)$ s.t. $t \in [t_1, t_2]$. To guarantee solution optimality we examine both these possibilities.

Second Case. A bit complex case compared to the the first one as some subtle issues might arise, is defined as *head-on* collision – when two agents traversing an edge in opposite directions simultaneously. A simple case is, which is also easy to detect, when two agents traverse an edge in opposite directions and they possibly occupy the edge together at $[t_1, t_2]$, computed in a similar manner. For example, in Figure 1, agent A reaches V at $[S_A + C_A, E_A + D_A]$ and another agent C reaches W at $[S_C + C_C, E_C + D_C]$, and they traverse in the opposite directions. Agent A could occupy e_{com} at $T_A = [S_A + C_A + 1, E_A + D_A + D_{com} - 1]$ while agent C could occupy it at $T_C = [S_C + C_C + 1, E_C + D_C + D_{com} - 1]$. There is an edge conflict if $[t_1, t_2] = T_A \cap T_C \neq \phi$. Moreover, this conflict is also possible when $T_A \cap T_C = \phi$ under some conditions. For example, when A reaches V at $[3, 3]$ and C reaches W at $[3, 3]$, and $[C_{com}, D_{com}] = [1, 1]$. Following Figure 1, this case is only possible when $S_A + C_A = E_A + D_A$, $S_C + C_C = E_C + D_C$ and $C_{com} = D_{com} = 1$. In any other situation than this, assuming that $D_{com} \geq C_{com} \geq 1$, there is a proper edge conflict where agents can occupy the edge for some time interval. The conflict is represented in a similar way, $(A, C, e_{com}, [S_A + C_A, E_A + D_A], [S_C + C_C, E_C + D_C], [C_{com}, D_{com}])$. We do not need to mark the direction of movement of an agent as time intervals in the representation suffice.

We resolve this conflict in the same way we resolve a run-over conflict. Generates two new children with two new constraints $occupy(A, e_{com}, t)$ and $occupy(C, e_{com}, t)$ respectively for agents A and C. They inherit all the parent's constraints. We can resolve a collision when they are likely to occupy an edge simultaneously, but this approach cannot capture a case when $T_A \cap T_C = \phi$. To resolve this special type of edge conflict, again two children are generated and are explored to guarantee solution optimality. Left child restricts agent A by $occupy(A, e_{com}, (S_A + C_A + C_{com}/2))$, and in the right C is restricted by $occupy(C, e_{com}, (S_C +$

$C_C + C_{com}/2)$). Note that, $(S_A + C_A + C_{com}/2) = (S_C + C_C + C_{com}/2)$. The children directly inherit all the parent's constraints. We are not over constrained with this approach of resolving a head-on conflict and hence do not miss out on any possible solution, consequently, preserving the completeness.

The above discussed approaches to resolve various kind of conflicts, guarantee that we do not loose the completeness. We always return a valid solution if there exists one. However, we do suspect that there would be some optimistic way to choose $t \in [t_1, t_2]$ instead of an arbitrary, or rather two or more contiguous t's after some preprocessing of the given problem graph. This probably would not improve the worst case theoretical guarantees discussed later in the paper, but could improve things practically.

4 CBS under Time Uncertainty

The state-space of Multi-Agent Path Finding (MAPF) is exponential in the number of agents. In the single-agent case, search-space grows only linearly in the graph size. CBS solves a MAPF problem by decomposing it into a large number of single-agent path finding problems.

In the basic CBS algorithm, an agent takes exactly one unit of time to reach a state connected to their current state (Sharon et el., 2012). We consider that to traverse an edge, e.g., moving from location A to location B, an agent takes a non-deterministic time $t \in [T_{min}, T_{max}]$, where t is decided by the nature at run time. We adapt and improve the technicalities of the basic CBS algorithm to handle this novel setting.

4.1 CBSTU Algorithm

We describe the main CBSTU algorithm that uses the above discussed approaches to find out and resolve a *conflict* present in a node N of a Constraint Tree (CT).

Definition of CBSTU: We will use a term *path* only in the context of single-agent and *solution* to denote a set of k paths for a given set of k agents. Basically, one path for each agent in the solution. A path for an agent would be a sequence of vertices, in which each vertex associates to $[t_{min}, t_{max}]$ that represents the time the agent could occupy this vertex following their path, e.g., for an agent starting from I and the goal G, a path π would look like: $I[0,0] \rightarrow C[3,6] \rightarrow E[4,7] \rightarrow T[5,9] \rightarrow G[6,10]$.

Agents are associated with *constraints* while they pursue their goals. A *consistent* path for an agent a_i is the path that satisfies all its constraints. Similarly, a *consistent solution* is a solution that is made-up of consistent paths for all the agents. A solution is *valid* if all its k paths have *zero* conflict. A consistent solution can be *invalid*, even if it consists of consistent paths based on agents' current constraints, if there exists a pair of conflicting paths. The important aspect of CBSTU is to grow a set of constraints for each agent, and find a path that is consistent with their individual constraints. A pair of conflicting paths makes a solution invalid, and the conflict is *marked* and gets resolved. The conflicting agents must generate new paths by satisfying all the constraints added after the previous conflict was resolved. CBSTU works at two levels - top level and bottom level. At the top-level, conflicts are found and constraints are generated and added to the constraint list. At the bottom-level, each agent generates a new plan respecting their individual constraints. This is done in a *decoupled* manner. Next, we discuss each part of the whole process in detail.

Top-Level: Search the Constraint Tree

The algorithm searches a *constraint tree* (CT) which is a binary tree. Each node N of this tree contains the following. **A set of constraints** ($N.\mathbf{constraints}$), that is empty for the root node of CT. Like the CBS algorithm, each child inherits the set of constraints of their parent. The approach adds exactly one new constraint in this set, exactly one for exactly one agent. **A solution** ($N.\mathbf{solution}$), that is a set of k consistent paths – one for each agent, found by the bottom-level approach, discussed next. **The total cost** ($N.\mathbf{cost}$), that is $\sum_{a_i \in \{a_1 \ldots a_k\}} \mathit{LTB}(\pi_i)$. *LTB* and *UTB* of a path represent respectively the lowest time and the highest time the agent takes to reach the goal. In the previous example, π ($I \rightarrow C \rightarrow E \rightarrow T \rightarrow G$) has $\mathit{UTB}(\pi) = 10$ and $\mathit{LTB}(\pi) = 6$. For an edge they capture the upper and lower time bound, an agent could occupy the edge during traversal starting from their initial location. For π, $\mathit{LTB}(E - T)$ and $\mathit{UTB}(E - T)$ are 5 and 8, respectively. Similarly, for a vertex in a given path, they show the upper and lower time bounds an agent can occupy a vertex. $\mathit{LTB}(E)$ and $\mathit{UTB}(E)$ are 4 and 7, respectively.

Note that for an edge e in the graph G, we consider that the Lowest Traversal Time, $\mathit{LTT}(e) \geq 1$. The Highest Traversal Time, $\mathit{HTT}(e)$, is bounded by some value, say LARGE, s.t., $\mathit{HTT}(e) \geq \mathit{LTT}(e)$.

Processing a Node of CT: In this phase of the algorithm, the bottom-level approach is invoked. This approach generates a shortest path for each agent independently, such that all their individual constraints are satisfied in the process, i.e., each generated path is consistent *w.r.t.* the constraints. Once this is done, the path of one agent gets validated with the paths of every other agents. If there is no conflict found, the node N is declared a *goal* node. N contains agents individual consistent paths, therefore $N.\mathbf{solution}$ is returned. However, during the path validation, suppose two agents a_i and a_j have their paths conflicting, the approach to find such a pair of paths is discussed earlier section. It finds a new conflict that could be either associated to a vertex conflict or an edge conflict (a head-on or run-over case). The validation process halts just after the very first conflict is found, and marks N a *non-goal* node.

Resolving a Conflict: Given a new non-goal node N whose *solution* contains an conflict *conflict* that either arises due to an edge conflict: $(A, B, e_{com}, [S_A + C_A, E_A + D_A], [S_B + C_B, E_B + D_B], [C_{com}, D_{com}])$ or a vertex conflict: (A, T_A, B, T_B, V). As per our assumptions, at most one of A and B is allowed to occupy a vertex or an edge at a given time. Therefore, for each conflict, N produces two new children that are examined independently to maintain optimality. One child adds a new constraint for agent A following our conflict resolution step. Similarly, it is done for agent B, in the other child of N. In the new children, the bottom-level search should be invoked only for the agent that receives the

new constraint. Of course, the paths for other agents remain the same and the same can be inherited from N without any modifications.

To make our approach and explanation simple, we consider only a pair of two agents given their individual paths, to find conflict if there is any.

Bottom-Level: Find a New Path

This approach is invoked to find a new path for an agent a_i given all its constraints. The agent is restricted by the given constraints that need to be satisfied in the new path. The approach finds a shortest path respecting these constraints. For a_i, the new path is obtained independently in a decoupled manner, i.e., the presence of other agents in the graph is ignored. Any single-agent path finding algorithm can be adapted and used. An adaptation is required since each vertex and edge during search carry a notion of explicit times and intervals, as we have discussed earlier. We employed A* with an admissible heuristic in the two spatial (XY) dimensions.

We briefly explain the required modifications in the A* search algorithm. We choose an *eager*-evaluation approach. For agent a_i, suppose that currently search is at node p (selected from OPEN), s.t., $p : [t^p_{min}, t^p_{max}]$, and its expansion (via applying moves) generates two legal children nodes x and y. If a move does not violate the constraints imposed on a_i, it is a legal move. For example: the agent moves to the vertex x from p, is valid if it does not violate its constraints on edge $p - x$ and vertex x. Then, $g(x) = \mathit{LTB}(x)$ and $h(x)$ computes a movement to the goal in the XY space (like the Manhattan Distance for Grid World). Later, $f(x)$ and $f(y)$ will be computed and the new nodes will be added to OPEN.

5 Conformant CBS under Time Uncertainty

We discuss an algorithm that handles a scenario when agents are not certain about their initial locations which is obvious in the real world [Hoffmann and Brafman, 2006]. They maintain initial belief states in that case. The approach in this section builds upon CBSTU algorithm discussed in the last section. We define a new MAPF setting in which possibility of multiple possible initial states called an initial belief state and non-deterministic edge traversal time, are combined. Pseudo code to handle this new setting is shown in Algorithm 1 that implicitly captures the required pseudo code for the CBSTU algorithm discussed in the previous section.

We discuss an approach to find and resolve a conflict when non-deterministic finish time is considered along with an agent maintains an initial belief state and not a defined initial state. The techniques used in the CBSTU algorithm to find a conflict for a given pair of paths and to resolve a conflict, remain the same in CCBSTU. We have changed the *definition* and the way CCBSTU *processes* a node of the constraint tree (CT).

5.1 Conformant CBSTU Algorithm

We discuss the main CCBSTU algorithm. The pseudo code for the top-level approach is shown in Algorithm 1. The time intervals have been referred from Figure 1.

Definition of CCBSTU: We use the term *conf-path* (conformant-path) only in the context of single-agent and *conf-solution* (conformant-solution) to denote a set of k conf-paths for a given set of k agents. A conf-path for an agent a_i is a set of paths Π_{a_i}. This set contains a single regular path (a path described in CBSTU) for each possible initial location $l \in b_0^{a_i}$ and a definite goal location. The rest of the details remain the same as in CBSTU algorithm. Agents are associated with *constraints* while they pursue their goals. A conf-path for an agent a_i is *consistent*, if each path $\pi_{a_i} \in \Pi_{a_i}$ is consistent with all a_i's constraints. Similarly, a *consistent* conf-solution is a solution that is made-up of such consistent conf-paths for individual agents. A solution is a *valid* solution when all its k conf-paths have no conflicts. The solution becomes invalid if there exists a path $\pi^l_{a_i} \in \Pi_{a_i}$, for an agent a_i, such that $\pi^l_{a_i}$ is inconsistent with any path $\pi \in \Pi_{a_j}$, where $a_i \neq a_j$. A consistent conf-solution can be *invalid* even if it consists of *consistent* conf-paths but there exists a *conflicting* pair of paths (π_{a_i},π_{a_j}).

The CCBSTU algorithm grows a set of constraints for each agent as search in CT progresses, and finds a conf-path that is

Algorithm 1: The top-level CCBSTU approach.

Data: An *mapf* instance; An optimization function f
Result: *conf-solution* // a conformant solution

$root.constraints = \phi$;
$root.solution \leftarrow$ individual *conf-paths* returned by *bottom-level*() approach ;
$root.cost = \sum_{a_i \in a_1,\ldots,a_k} \sum_{\pi_i \in \Pi_{a_i}} f(\pi_i)$;
ADD *root* to OPEN ;
while *OPEN not empty* **do**
  $N \leftarrow$ the best node from OPEN ;
  Validate *conf-paths* in N until the first conflict occurs ;
  if *no conflict found* **then**
    return N.*conf-solution* ;
  end
  conflict $\leftarrow$ First conflict ;
  // either vertex or edge
  for *agent* $a_i \in$ *conflict* **do**
    $A \leftarrow$ *Create* a new CT node ;
    if *vertex-conflict*(*conflict*) **then**
      $A.constraints \leftarrow N.constraints +$ (A, T_A, B, T_B, V) ;
    end
    else
      $A.constraints \leftarrow N.constraints +$ $(A, B, e_{com}, [S_A + C_A, E_A + D_A], [S_B + C_B, E_B + D_B], [C_{com}, D_{com}])$;
      // irrespective of the edge conflict types
    end
    $A.solution \leftarrow N.solution$;
    Update $A.solution$ with a *conf-path* returned by *bottom-level*() for a_i;
    $A.cost = \sum_{a_i \in a_1,\ldots,a_k} \sum_{\pi_i \in \Pi_{a_i}} f(\pi_i)$;
    ADD node A to OPEN ;
  end
end

consistent with their individual constraints. For a given node of CT, a pair of conflicting paths makes a conf-solution inconsistent and the node of the CT is *invalid*, and the conflict is marked which needs to be resolved. This node is declared as a *non-goal* node. The agent that receives a new constraint in the processing phase, must generate a new conf-path satisfying all their constraints. The CCBSTU algorithm works at two levels as well. At the top-level, conflicts are found and constraints are generated and added. At the bottom-level, each agent generates a conf-path satisfying their constraints. Next, we discuss each part of this process in detail.

Top-Level: Search the Constraint Tree

The algorithm searches a *constraint tree* (CT) that is a binary tree. Each node N of this tree contains the following things. **A set of constraints** ($N.\mathbf{constraints}$), that is empty for the root node of CT. Like CBSTU algorithm, each child inherits all its parent's constraints, further it adds exactly one new constraint for exactly one agent. **A conf-solution** ($N.\mathbf{conf\text{-}solution}$), that is a set of k consistent conf-paths – one conf-path for each agent, found by the bottom-level approach. **The total cost** ($N.\mathbf{cost}$), that is $\sum_{a_i \in \{a_1 \dots a_k\}} \sum_{\pi \in \Pi_{a_i}} \mathit{LTB}(\pi)$. They are shown in the Algorithm 1 (in Line 1, 2, and 3).

Processing a node: The *bottom-level* approach is invoked in this phase which generates a shortest conf-path for each agent in a decoupled manner, such that all their individual constraints are satisfied in the process, i.e., each generated conf-path is consistent. Then, each agent's conf-path gets validated with every other agents' conf-paths. Unlike the CBS and CBSTU algorithms, this validation is a bit non-trivial. In this process, each $\pi_i \in \Pi_{a_i}$ is validated against each $\pi_j \in \Pi_{a_j}$ for all j s.t. $j \neq i$. But the validation technique of a pair of paths remains the same as it was for CBSTU. If there is no conflict found, the node N is declared a *goal* node. Later, N**.conf-solution** is returned. However, during the path validation, suppose that for two agents a_i and a_j and their paths $\pi_i \in \Pi_{a_i}$ and $\pi_j \in \Pi_{a_j}$, (π_i, π_j) forms a conflicting pair, a new conflict is found in that case. The conflict could be either a vertex conflict or an edge conflict. We halt the validation process just after the very first conflict is found, and mark N a *non-goal* node.

Resolving a conflict: We use different representations for edge and vertex conflicts in the CBSTU and CCBSTU algorithms. That means CCBSTU finds and resolves a conflict in exactly same way as CBSTU does. We have already discussed an approach to find and resolve a conflict. Given a new non-goal node N of CT – for which N.**solution** contains a conflict that either arises due to an edge conflict: $(A, B, e_{com}, [S_A + C_A, E_A + D_A], [S_B + C_B, E_B + D_B], [C_{com}, D_{com}])$ or a vertex conflict: (A, T_A, B, T_B, V). The agents A and B are referred from Figure 1. Following our earlier discussion, CCBSTU generates two new children (in Line 13, Algorithm 1). In the first child a new constraint is added for agent a_i, and vice-versa for a_j in the second child. At each new child, *bottom-level* search is invoked only for the agent that gets the new constraint. The conf-paths for other agents remain the same, and can be inherited directly from parent node, N in CT (in Line 21, Algorithm 1).

Bottom-Level: Find a New Conformant Path

This is invoked to find a new conf-plan, only for the agent a_i that received new constraints. It must find a new conf-path (if there exists one) such that it satisfies all the a_i's constraints. The new conf-path is obtained independently, in a decoupled manner. An important consideration is that each agent maintains an initial belief state $b_0^{a_i}$ - a set of possible initial locations, such that each *pair* of their belief states is *disjoint*. For each location $l \in b_0^{a_i}$ and a definite goal location, any single-agent path finding algorithm can be adapted and used. For this location l a solution, path_l, is found in exactly the same way as it is done in the CBSTU algorithm. It stores each path_l in Π_{a_i}, and returns Π_{a_i} in the end.

6 TODO: Theoretical Analysis

In this section, we prove that CBSTU and CCBSTU will return respectively optimal and conformant-optimal solutions if they exist. However, we start with the proof for the CBSTU algorithm and give several supporting claims first. In the later part of this section, we provide the proof of CCBSTU which will be a simple extension to the proof of CBSTU. Some definitions are directly taken from [Sharon *et al.*, 2012] modified so that they are valid for both the proposed algorithms.

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

If N is not marked as goal node, the $N.\mathbf{solution}$ ($N.\mathbf{conf\text{-}solution}$) will not be in $CV(N)$ as it is not valid.

Definition 2. *For any solution $p \in CV(N)$ for CBSTU (conf-solution for CCBSTU), we say that node N permits the solution (conf-solution) p.*

The cost of a solution (conf-solution) in $CV(N)$ is the sum of the costs of individual agents. Let $minCost(CV(N))$ be the minimum cost over all solutions (conf-solutions) in $CV(N)$, which is *minimized* over all the solutions (conf-solutions), such that for a solution the cost is $\sum_{a_i \in a_1, \dots, a_k} \mathit{LTB}(\pi_{a_i})$ (for a conf-solution the cost is $\sum_{a_i \in a_1, \dots, a_k} \sum_{\pi_i \in \Pi_{a_i}} \mathit{LTB}(\pi_i)$).

Lemma 1. *The cost of a node N in the CT is a lower bound on $minCost(CV(N))$.*

Proof. In both CBSTU and CCBSTU algorithms, $N.\mathbf{cost}$ is the optimal cost for a set of paths (conf-paths for CCBSTU) that satisfies all the constraints for the agents at N. The set of paths (conf-paths) is consistent but does not necessarily to be a valid solution (conf-solution). Therefore, $N.\mathbf{cost}$ is the lower bound on the cost of any set of paths (conf-paths) that make a valid solution (valid conf-solution) for N. This is because no agent in any solution (conf-solution) can achieve their goal sooner than this. □

Theorem 1. *The CBSTU algorithm always returns an optimal solution.*

Proof. □

Theorem 2. *A solution obtained from CBSTU is always a strong solution.*

Proof. □

Definition 3. *When a path called conformant-path....*

Definition 4. *When a solution called conformant-optimal....*

Theorem 3. *The CCBSTU algorithm always returns a conformant-optimal solution.*

7 TODO: Experimental Results

7.1 Environment

7.2 Different MAPF Settings

7.3 Empirical Evaluation

8 Discussion

We need to address thoroughly the issue of uncertainty in the initial and goal states... (Bolander et al., 2018).

9 Summary and Future Work

This paper proposes two new algorithms called CBSTU and CCBSTU that are based on the basic CBS algorithm to solve the MAPF problem under some novel settings like when path traversal time is indefinite and uncontrollable, and the agents are uncertain about their initial locations. We provide thorough theoretical analyses for both the proposed algorithms, and showed scalability of the approaches empirically on new MAPF problems.

In future we intend to extend this work to handle scenarios when an edge has disjunctive traversal time uncertainties with uncontrollability, e.g., the edge traversal time belongs to $[T_{min_1}, T_{max_1}] \cup [T_{min_2}, T_{max_2}]$. Following some recent work, we also intend to introduce the notion of *deadline* for agents in the proposed MAPF settings. In that case, an optimal approach would find a solution that maximizes the number of agents reaching their goal locations without colliding with each other, with a minimum cumulative cost such that agents satisfy their individual deadlines.

References

[Abelson *et al.*, 1985] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Massachusetts, 1985.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Georg Gottlob, and Sergio Flesca. Visual information extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Databases*, pages 119–128, Rome, Italy, September 2001. Morgan Kaufmann.

[Bolander *et al.*, 2018] Thomas Bolander, Thorsten Engesser, Robert Mattmüller, and Bernhard Nebel. Better eager than lazy? how agent types impact the successfulness of implicit coordination. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018.*, pages 445–453, 2018.

[Brachman and Schmolze, 1985] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April–June 1985.

[Cimatti *et al.*, 2018] Alessandro Cimatti, Minh Do, Andrea Micheli, Marco Roveri, and David E. Smith. Strong temporal planning with uncontrollable durations. *Artif. Intell.*, 256:1–34, 2018.

[Felner *et al.*, 2018] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, pages 83–87, 2018.

[Gottlob *et al.*, 2002] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002.

[Gottlob, 1992] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[Hoffmann and Brafman, 2006] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.*, 170(6-7):507–541, 2006.

[Levesque, 1984a] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23(2):155–212, July 1984.

[Levesque, 1984b] Hector J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 198–202, Austin, Texas, August 1984. American Association for Artificial Intelligence.

[Ma *et al.*, 2017] Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3605–3612, 2017.

[Morris *et al.*, 2016] Robert Morris, Corina S. Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016.*, 2016.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[Palacios and Geffner, 2009] Héctor Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res.*, 35:623–675, 2009.

[Sharon *et al.*, 2012] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.

[Silver, 2005] David Silver. Cooperative pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pages 117–122, 2005.

[Veloso *et al.*, 2015] Manuela M. Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, page 4423, 2015.

[Wurman *et al.*, 2007] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1752–1760, 2007.