Time-Independent Planning for Multiple Moving Agents

Keisuke Okumura\textsuperscript{1}, Yasumasa Tamura\textsuperscript{1} and Xavier Défago\textsuperscript{1}
\textsuperscript{1}Tokyo Institute of Technology
okumura.k@coord.c.titech.ac.jp, \{tamura, defago\}@c.titech.ac.jp,

Abstract

Typical Multi-agent Path Finding (MAPF) solvers assume that agents move synchronously, thus neglecting the reality gap in timing assumptions, e.g., delays caused by an imperfect execution of asynchronous moves. So far, two policies enforce a robust execution of MAPF plans taken as input, namely, either by forcing agents to synchronize, or by executing plans while preserving temporal dependencies. This paper proposes a third approach, called time-independent planning, which is both online and distributed. We represent reality as a transition system that changes configurations according to atomic actions of agents, and use it to generate a time-independent schedule. Empirical results in a simulated environment with stochastic delays of agents’ moves support the validity of our proposal.

1 Introduction

Multi-agent systems with physically moving agents are becoming gradually more common, e.g., automated warehouses [Wurman et al., 2008], traffic control [Dresner and Stone, 2008], or self-driving cars. In such systems, it is critical for agents to move smoothly without colliding. This is embodies by the problem of Multi-agent Path Finding (MAPF) [Stern, 2019]. Planning techniques for MAPF have been extensively studied in the recent decade.

The output of such planning is bound to be executed in real-world situations with agents (robots). Typical MAPF is defined in discrete time. Agents are assumed to do two kinds of atomic actions synchronously, namely, either move to a neighboring location or stay at their current location. However, perfect executions for the planning are difficult to ensure since timing assumptions are inherently uncertain in reality, due to the difficulty of: 1) accurately predicting the temporal behavior of many aspects of the system, e.g., kinematics, 2) anticipating external events such as faults and interference, and 3) ensuring a globally consistent and accurate notion of time in the face of clock shift and clock drift. Even worse, the potential of unexpected interference increases with the number of agents, hence the need to prepare for imperfect executions regarding the timing assumptions.

Two intuitive policies tackle imperfect executions of MAPF plans taken as input. The first, and conservative idea is to forcibly synchronize agents’ moves, globally or locally. Most decentralized approaches to MAPF take this approach [Wiktor et al., 2014; Kim et al., 2015; Okumura et al., 2019]. As studied by Ma et al. [Ma et al., 2017a], this policy negatively affects the entire performance of the system with unexpected delays and lacks flexibility. The second policy makes agents preserve temporal dependencies of the planning [Hönig et al., 2016; Ma et al., 2017a; Hönig et al., 2019]. Two types of temporal dependencies exist: 1) internal events within one agent and, 2) order relation of visiting one node. This policy is sound but still vulnerable to delays. Consider an extreme example where one agent moves very slowly, or crashes. Due to the second type of dependencies, the locations where the agent will be are restricted by the use of the other agents. Thus, the asynchrony of the movements is sensitive to the whole system.

We therefore propose a third approach, called time-independent planning, that aims at online and distributed execution. This paper focuses on the time-independence of agents’ moves. We represent the whole system as a transition system that changes configurations according to atomic actions of agents, namely, 1) request the next locations (requesting), 2) move (extended), and, 3) release the past locations, or stay (contracted). In this time-independent model, a scheduler, modeling non-deterministic behavior of the external environment, emulates possible sequences of atomic actions. The challenge is to design algorithms tolerant to all possible sequences.

The main contributions of this paper are: 1) the formalization of the time-independent model and Causal-PIBT, a proposed time-independent planning with guaranteed reachability, i.e., all agents are ensured to reach their destinations within finite time. Causal-PIBT, a proof-of-concept for the approach, extends a recently-developed decoupled approach that solves MAPF iteratively, Priority Inheritance with Backtracking (PIBT) [Okumura et al., 2019]. We also present how an MAPF plan enhances Causal-PIBT. 2) experimental results demonstrating the validity and robustness of the proposal through the simulation with stochastic delays of agents moves, using the MAPF-DP (with Delay Probabilities) [Ma et al., 2017a] setting.

The paper is structured as follows. Section 2 formalizes...
MAPF and introduces related work. Section 3 presents the time-independent model. Section 4 proposes examples of time-independent planning. Section 5 presents empirical results of the proposals using MAPF-DP. Section 6 concludes the paper and outlines future directions.

2 Preliminary

This section first defines MAPF. Then, we explain the MAPF variant emulating asynchrony of movements, called MAPF-DP, which we later use in experiments. We also explain two policies that execute MAPF plans and PIBT, the original form of Causal-PIBT.

2.1 MAPF

MAPF consists of a set of agents, \( A = \{a_1, \ldots, a_n\} \), and an environment given as a graph \( G = (V, E) \). Time is assumed discrete. Let \( \pi_i[t] \in V \) denote the location of an agent \( a_i \) at timestep \( t \in \mathbb{N} \). An agent \( a_i \) has its initial location \( \pi_i[0] \) and destination \( g_i \in V \). At each timestep \( t \), \( a_i \) can move to an adjacent node, or, can stay at its current location, i.e., \( \pi_i[t+1] \in \text{Neigh}(\pi_i[t]) \cup \{\pi_i[t]\} \), where \( \text{Neigh}(v) \) is the set of nodes neighbor to \( v \in V \). Agents must avoid two types of conflicts [Stern et al., 2019]: 1) vertex conflict: \( \pi_i[t] \neq \pi_j[t] \), and, 2) swap conflict: \( \pi_i[t] \neq \pi_j[t+1] \lor \pi_i[t+1] \neq \pi_j[t] \). A solution of MAPF is a set of paths \( \pi = (\pi_1, \ldots, \pi_n) \), where \( \pi_i = (\pi_i[0], \pi_i[1], \ldots, \pi_i[T]) \) such that \( \pi_i[T] = g_i \). Note that \( |\pi_i| = |\pi_j| \) in \( \pi \). Two kinds of objective functions are commonly used to evaluate MAPF solutions: 1) sum of cost (SOC), where cost is the earliest timestep \( T_1 \) such that \( \pi_i[T_1] = g_i, \ldots, \pi_i[T] = g_i, T_1 \leq T \), 2) makespan, i.e., \( T \).

This paper focuses on the SOC.

2.2 MAPF-DP (with Delay Probabilities)

MAPF-DP [Ma et al., 2017a] emulates imperfect executions of MAPF plans by introducing the failure probabilities of agents’ moves. Time is still discrete. At each timestep, \( a_i \) can do two kinds of actions like MAPF: however, moves to adjacent nodes fail with probability \( p_i \), in which case \( a_i \) remains at the current location. The definition of conflicts is more restrictive than with normal MAPF: 1) vertex conflict as defined in MAPF, and, 2) following conflict: \( \pi_i[t+1] \neq \pi_j[t] \). The rationale is that, without the later restriction, two agents might be in the same node due to one failing to move. Note that following conflict contains swap conflict.

Execution Policies

Ma et al. [Ma et al., 2017a] studied two robust execution policies using MAPF plans for the MAPF-DP setting. The first one, called Fully Synchronized Policies (FSPs), synchronizes the movements of agents globally, i.e., \( a_i \) waits to move to \( \pi_i[t+1] (\neq \pi_i[t]) \) until all move actions of \( \pi_i[t'], t' \leq t \) are completed. The second approach, called Minimal Communication Policies (MCPs), executes a plan while maintaining its temporal dependencies. There are two kinds of dependencies: 1) internal events, i.e., the corresponding action of \( \pi_i[t] \) is executed prior to that of \( \pi_i[t+1] \), and, 2) node-related events, i.e., if \( \pi_i[t] = \pi_j[t'] \) and \( t < t' \), the event of \( \pi_i[t] \) is executed prior to that of \( \pi_j[t'] \). As long as an MAPF plan is valid, both policies make agents reach their destinations without conflicts, in spite of delay probabilities.

2.3 Priority Inheritance with Backtracking (PIBT)

PIBT [Okumura et al., 2019] gives fundamental conflict-free moves of agents to solve MAPF iteratively (without delays). Agents are provided with unique priorities in every timestep. Then, they sequentially determine their next locations in decreasing order of priorities while avoiding to use nodes that have requested from agents with higher priority. Priority inheritance, originally considered in resource scheduling problems [Sha et al., 1990], is introduced to deal effectively with priority inversion in path adjustment. When a low-priority agent \( X \) impedes the movement of a higher-priority agent \( Y \), agent \( X \) temporarily inherits the higher-priority of agent \( Y \). Priority inheritance can be applied iteratively, and, is accompanied by a backtracking protocol to prevent agents from being stuck. Backtracking has two outcomes: valid or invalid. Invalid occurs when an agent inheriting the priority is stuck, forcing the higher-priority agent to replan its path. Fig. 1 shows an example of PIBT in one timestep. By this, the agent with highest priority can move to an arbitrary neighbor node if the graph satisfies the adequate property, e.g., biconnected. Subsequently, such an agent moves to its goal along the shortest path to its goal.

One more key component is dynamic priorities, where the priority of an agent increments gradually until it drops upon reaching its goal. By combining these techniques, PIBT ensures the reachability, all agents are ensured to reach their own destinations within finite timesteps. Note that unlike...
complete solvers for MAPF, PIBT does not ensure that all agents are on their goals simultaneously; nevertheless, reachability plays a crucial role in situations where destinations are given continuously such as lifelong MAPF [Ma et al., 2017b].

3 Time-Independent Model

This section presents the time-independent model. The model and the terms used here are partly inspired by the model for distributed algorithms with synchronous message passing [Tel, 2000] and the Amoebot model [Derakhshan-deh et al., 2014], an abstract computational model for programmable matter. Our model is different from those because we assume that agents are physically embodied hence they have a risk of collisions, further, they have destinations.

Components The system consists of a set of agents $\mathcal{A} = \{a_1, \ldots, a_n\}$ and a graph $G = (V, E)$. We assume that each agent knows $G$ a priori to plan their respective paths.

Configuration and State The whole system is represented as a transition system according to atomic actions of agents. Each agent $a_i$ is itself a transition system with its own state, denoted as $\sigma_i$, consisting of its internal variables, its current location, destination, and priority, etc. Since a configuration $\gamma$ of the whole system at a given time is composed of the states of all agents at that time, $\gamma$ is defined as $\gamma = (\sigma_1, \ldots, \sigma_n)$. The change of states of agents causes the change of configuration of the system, e.g., $\gamma = (\ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots) \rightarrow \gamma' = (\ldots, \sigma_{i-1}, \sigma'_i, \sigma_{i+1}, \ldots)$.

Mode We use two terms to represent agents’ locations: $\text{tail}_i \in V$ and $\text{head}_i \in V \cup \{\bot\}$, where $\bot$ is void. They are associated with a mode $\text{mode}_i$ which can be: contracted, requesting and extended.

- $\text{contracted}$: $a_i$ stays at one node $\text{tail}_i$, and $\text{head}_i = \bot$.
- $\text{requesting}$: $a_i$ attempts to move to $\text{head}_i \neq \bot$, being at $\text{tail}_i$.
- $\text{extended}$: $a_i$ is moving from $\text{tail}_i$ to $\text{head}_i$.

Initially, all agents are $\text{contracted}$ and are given distinct initial locations. $\text{head}_i (\neq \bot)$ is always adjacent to $\text{tail}_i$.

Conflict A conflict between two agents $a_i$ and $a_j$ is when $\text{body}_i \cap \text{body}_j \neq \emptyset$. $\text{body}_i \subseteq V$, denoting $a_i$’s physical body, always contains $\text{tail}_i$. Additionally, when $\text{mode}_i$ is $\text{extended}$, $\text{body}_i$ also contains $\text{head}_i$. Thus, $1 \leq |\text{body}_i| \leq 2$.

Transition Movements of agents occur by changing modes. These transitions are accompanied by changing $\text{tail}$ and $\text{head}$ described as follows.

- $a_i$ in $\text{contracted}$ can become $\text{requesting}$ by setting $\text{head}_i$ to $u \in \text{Neigh}(\text{tail}_i)$, to move to a neighbor node.
- $a_i$ in $\text{requesting}$ can go back to $\text{contracted}$ by changing its $\text{head}_i$ to $\bot$.
- $a_i$ in $\text{requesting}$ can become $\text{extended}$ when $\neg \text{occupied}(\text{head}_i)$, where $\text{occupied}(v)$ holds true when $\exists a_j, v \in \text{body}_j$.
- $a_i$ in $\text{extended}$ can become $\text{contracted}$, implying that the movement is finished. This accompanies $\text{tail}_i \leftarrow \text{head}_i$, then $\text{head}_i \leftarrow \bot$.

Other modes transitions are disallowed, e.g., an agent in $\text{contracted}$ cannot become $\text{extended}$ directly.

Conflict-freedom and Deadlock In the above transition rules, the model implicitly prohibits vertex and following conflicts of MAPF. Rather, the model is prone to deadlocks; A set of agents $\{a_k, \ldots, a_l\}$ are in a deadlock when all of them are in $\text{requesting}$ and are in a cycle $\text{head}_k = \text{tail}_{k+1}, \ldots, \text{head}_i = \text{tail}_k$.

Activation and Scheduler Agents spontaneously do any atomic actions according to its state (e.g., transit their modes, change their local variables) without any synchronization. With a global clock – agents neither know nor detect – these actions are temporally ordered. Therefore, even if two faraway agents $a_i$ and $a_j$ seem simultaneously do any actions, we regard as; $(\ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots) \rightarrow (\ldots, \sigma'_{i-1}, \sigma'_i, \sigma'_{i+1}, \ldots)$. In this scheme, it is convenient to introduce a scheduler; an abstract artefact used to model the non-deterministic behavior of the external environment, such as delays of moves. The scheduler conducts repeated activations, meaning that, it picks one agent arbitrarily and lets the agent perform an atomic action. The agent is said to be activated. Note that the scheduler is in no way a component per se, let alone a centralized one. We assume that the scheduler is fair, i.e., if the scheduler runs infinitely, all agents are activated infinitely-often. The activated agent $a_i$, based on its state, might change its variables including $\text{mode}_i$, as stated before. Further, it might change variables of interacted agents, as explained next.

Interaction An activation may affect not only variables of the activated agent, but also the variables of other nearby agents indirectly. For instance, if two $\text{requesting}$ agents have the same $\text{head}$, one might win and become $\text{extended}$ whereas the other loses and becomes $\text{contracted}$, atomically. This type of activation is called an interaction. Interactions include activations such that the activated agents change their variables referring to variables of other agents. We say that agents involved in the interaction except $a_i$ are interacted agents. Given an activated agent $a_i$ and an interacted agent $a_j$, the corresponding transition of the system is $(\ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots) \rightarrow$
\[ (\ldots, \sigma_i', \ldots, \sigma_j, \ldots, \sigma_k, \ldots, \sigma_l, \ldots) \]. Except for interactions, the configuration is changed by the state change of a single agent. We assume that interactions are performed by communication between agents, but the detailed implementation is beyond this paper.

**Termination** Assume that an agent \(a_i\) has its own destination \(g_i \in V\). \(a_i\) reaches \(g_i\) if it becomes contracted and \(\text{tail}_i = g_i\). The scheduler stops activations when, for all agent \(a_i\), \(\text{mode}_i = \text{contracted} \land \text{tail}_i = g_i\). Then, the system is regarded as terminated.

**Example** Fig. 2 shows an example of one execution of the time-independent model. Two agents \(a_1, a_2\) try to go to \(v_3\) and \(v_4\), respectively. According to the scheduler activation, agents transit their states.

**Remarks** We here defined the model targeting MAPF, however, by changing the termination and destination assignments, wide situations can be addressed with this transition system, e.g., iterative MAPF [Okumura et al., 2019].

### 4 Algorithm

This section presents examples of time-independent planning (GREEDY, Causal-PIBT), running on the model defined in the previous section. We also present how to enhance Causal-PIBT by MAPF plans.

#### 4.1 Greedy Approach

The first example GREEDY performs fundamental actions; it can be a template for other time-independent planning. We simply describe its implementation for \(a_i\) as follows.

- when \text{contracted}: Choose the nearest node to \(g_i\) from \(\text{Neigh}(\text{tail}_i)\) as new \(\text{head}_i\), then become \text{requesting}.
- when \text{requesting}: Become \text{extended} when the \(\text{head}_i\) is unoccupied, otherwise, do nothing.
- when \text{extended}: Become \text{contracted}.

Obviously, GREEDY causes deadlocks without any recovery, e.g., when two adjacent agents try to swap their locations, they stop eternally. The time-independent planning without deadlock-free or deadlock-recovery properties are impractical, motivating the next algorithm.

#### 4.2 Causal-PIBT

This algorithm extends both PIBT and GREEDY. Although PIBT relies on the synchronous moves, Causal-PIBT only relies on causal dependencies.

**Concept** There are two key observations to make PIBT time-independent.

First, the path adjustment phase of PIBT in one timestep can be seen as the construction of a depth-first search tree to find an empty node adjacent to the tree. This virtual tree consists of agents, not nodes. The root of the tree is the first agent starting the chain of priority inheritance, i.e., locally highest priority agent, e.g., \(a_7\) in Fig. 1. When an agent \(a_j\) inherits a priority from another agent \(a_i\), \(a_j\) becomes a \text{child} of \(a_i\), \(a_i\) becomes its \text{parent}. Once an empty node adjacent to the tree is found, all agents on the path from the root to the node can move toward one step toward the node. The invalid backtracking corresponds to backtracking in a depth-first search. The valid backtracking is in charge of the notification of the search termination.

Second, PIBT ensures reachability by combining the local movement of the agent with highest priority and dynamic priorities to support fairness.

From the above observations, we get two intuitions to de-

---

**Algorithm 1** Causal-PIBT

```
parent_i \in \mathcal{A}: \text{initially} a_i
children_i \subset \mathcal{A}: \text{initially} \emptyset
\text{pori}_i: \text{original priority of} \ a_i
\text{ptmp}_i: \text{temporal priority of} \ a_i, \ \text{initially} \ \text{pori}_i
\text{C}_i \subset \mathcal{V}: \text{candidate nodes, initially} \ \text{Neigh}(\text{tail}_i) \cup \{\text{tail}_i\}
\text{S}_i \subset \mathcal{V}: \text{searched nodes, initially} \emptyset

1: \text{when} \ mode_i = \text{contracted}
2: \quad \text{if} \ C_i = \emptyset \land \text{parent}_i = a_i \ \text{then}
3: \quad \quad \text{RELEASECHILDREN, RESET}
4: \quad \text{end if}
5: \text{end if}

6: \text{if} \ C_i = \emptyset \text{then}
7: \quad \text{let} \ a_j \ \text{be parent}_i
8: \quad \text{if} \ \text{head}_j = \text{tail}_i \ \text{then}
9: \quad \quad \ S_j \leftarrow S_j \cup \mathcal{S}_i, \ C_j \leftarrow C_j \setminus S_j \quad \text{▷ with} \ a_j
10: \quad \text{head}_j \leftarrow \bot, \ \text{mode}_j \leftarrow \text{contracted}
11: \quad \text{end if}
12: \quad \text{return}
13: \text{end if}
14: \quad u \leftarrow \text{the nearest node to} \ g_i \ \text{in} \ C_i
15: \quad \text{if} \ u = \text{tail}_i \ \text{then}
16: \quad \quad \text{RELEASECHILDREN, RESET}
17: \quad \text{return}
18: \text{end if}
19: \quad C_i \leftarrow C_i \setminus \{u\}, \ S_i \leftarrow S_i \cup \{u, \text{tail}_i\}
20: \quad \text{head}_i \leftarrow u, \ \text{mode}_i \leftarrow \text{requesting}
21: \text{end when}

22: \text{when} \ mode_i = \text{requesting}
23: \quad \text{PRIORITYINHERITANCE}
24: \quad \text{if} \ \text{parent}_i \neq a_i \land \text{head}_i \in \text{S}_\text{parent}_i \ \text{then}
25: \quad \quad \text{head}_i \leftarrow \bot, \ \text{mode}_i \leftarrow \text{contracted} \quad \text{▷ parent}_i
26: \quad \text{return}
27: \text{end if}
28: \quad \text{if} \ \text{occupied}(\text{head}_i) \ \text{then} \quad \text{return} \quad \text{end if}
29: \quad A' \leftarrow \{a_j \mid a_j \in A, \ \text{mode}_j = \text{requesting}, \ \text{head}_j = \text{head}_i\}
30: \quad \ a^* \leftarrow \text{arg max}_{a_j \in A'} \ \text{ptmp}_j
31: \quad \text{for} \ a_j \in A' \setminus \{a^*\} \ \text{do}
32: \quad \quad \text{head}_j \leftarrow \bot, \ \text{mode}_j \leftarrow \text{contracted} \quad \text{▷ agents} \ in \ A'
33: \quad \text{end for}
34: \quad \text{if} \ a^* = a_i \ \text{then}
35: \quad \quad \text{children}_\text{parent}_i \leftarrow \text{children}_\text{parent}_i \setminus \{a_i\} \quad \text{▷ parent}_i
36: \quad \quad \text{parent}_i \leftarrow a_i
37: \quad \quad \text{RELEASECHILDREN}
38: \quad \quad \text{mode}_i \leftarrow \text{extended}
39: \quad \text{end if}
40: \text{end when}

41: \text{when} \ mode_i = \text{extended}
42: \quad \text{tail}_i \leftarrow \text{head}_i, \ \text{head}_i \leftarrow \bot, \ \text{mode}_i \leftarrow \text{contracted}
43: \quad \text{update pori}_i, \ \text{RESET}
44: \text{end when}
```
sign a time-independent algorithm with reachability: 1) build a depth-first search tree such that its root is the agent with highest priority, through the mechanism of priority 2) drop priorities of agents that arrive at their goals.

**Description** We now show the pseudocode in Algorithm 1, 2. In Algo. 1, procedures with activation are denoted for each mode. The procedures in contracted consist of:

- relaying search termination [Line 2–4]
- priority inheritance [Line 5]
- backtracking, i.e., invalid case in PIBT [Line 6–13]
- prioritized planning [Line 14–20].

The procedures in requesting consist of:

- priority inheritance [Line 23]
- deadlock detection [Line 24–27]
- winner determination between agents requesting the same node [Line 28–33]
- preparation for moves [Line 34–39]

The procedures in extended are just back to contracted. Algo. 2 shows subprocedures used in Algo. 1. Interactions and the corresponding interacted agents are explicitly denoted in the pseudocode as comments. We now go on details.

Following the two intuitions obtained form PIBT, we design Causal-PIBT to build depth-first search trees rooted at the agents with locally highest priorities. Every tree has its own priority given by its root agent. When a tree with higher priority comes in contact with a lower-priority tree, the latter is decomposed and is partly merged into the former. This operation is accompanied by priority inheritance, which occurs both in contracted or requesting modes [Line 5, 23 in Algo. 1]. In the reverse case, the tree with lower priority just waits for the release of the area. Implicit backtracking occurs when a child has no candidate node to move due to requests from agents with higher priorities [Line 8–11 in Algo. 1].

There are three kinds of additional local variables of $a_i$: 1) parent$_i$, and children$_i$ to maintain trees. $a_i$ is seen as a root when parent$_i = a_i$. Note that the algorithm updates these variants so that $a_i = \text{parent}_i \not= a_i \in \text{children}_i$. 2) $C_i$ and $S_i$ for searching unoccupied neighbor nodes. $a_i$ in contracted selects a next target node from $C_i$. $S_i$ represents already searched nodes by a tree to which $a_i$ belongs. $S_i$ is propagated with priority inheritance [Line 10 in Algo. 2], or, backproped from its children [Line 9 in Algo. 1]. $C_i$ is updated to be disjoint from $S_i$ [Line 9, 19 in Algo. 1, Line 10, in Algo. 2]. 3) pori$_i$ and pttmp$_i$ represent priorities. They are components of the total order set. pori$_i$ is updated such that be lower than priorities of agents who have not yet reached their goals [Line 43 in Algo. 1]. This is realized by a similar prioritization scheme of PIBT. We assume that pori$_i$ is unique between agents in any configuration. pttmp$_i$ is basically equal to pori$_i$, however, it is changed by priority inheritance [Line 20 in Algo. 2]. Note that pttmp$_i \geq$ pori$_i$ in any time, and only pttmp$_i$ is used for interaction.

We also use two subprocedures, PRIORITYINHERITANCE and RELEASECHILDREN. The former first determines whether priority inheritance should occur [Line 2–5 in Algo. 2], then, updates the structure of trees and inherits both the priority and the searched nodes from the new parent [Line 6–10 in Algo. 2]. The latter just cuts off the relationship with the children of $a_i$.

**Properties** In the time-independent model, deadlocks are critical. Causal-PIBT copes with this danger as follows.

**Proposition 1** (deadlock-recovery). Causal-PIBT ensures no deadlock situation can last forever.

**Proof.** Assume that there is a deadlock. When an agent $a_i$ becomes requesting from contracted, it updates $S_i$ such that $S_i$ includes head$_i$ and tail$_i$ [Line 19 in Algo. 1]. After finite activations, all agents involved in the deadlock must have the same priority pttmp due to priority inheritance. Every priority inheritance is accompanied by the propagation of $S_i$ [Line 10 in Algo. 2]. When an agent in requesting detects its head in $S_{\text{parent}}, a_i$ changes back to contracted [Line 24–27 in Algo. 1]. These implies the statement.

Relying on the above proposition, Causal-PIBT has the reachability.

**Theorem 1** (reachability). Causal-PIBT ensures that all agents reach their destinations in finite number of activations in biconnected graphs if $|A| < |V|$.

**Proof sketch.** Let $a_i$ be the agent with highest pori. If $a_i$ is contracted, this agent does not inherit any priority. Thus, $a_i$ can be requesting so that head$_i$ is any neighbor node of tail$_i$. If $a_i$ is requesting, $a_i$ eventually moves to head$_i$ due to the construction of the depth-first search tree in a biconnected graph. These imply that $a_i$ can move to an arbitrary neighbor node in finite activations. By this, $a_i$ moves along the shortest path to its goal. * Due to the prioritization scheme, an agent that has not reached its goal eventually gets the highest pori, then it starts moving along its shortest path to its goal. This satisfies the statement.

---

**Algorithm 2 Procedures of Causal-PIBT**

1: procedure PRIORITYINHERITANCE
2: $A' \leftarrow \{a_i \mid a_i \in A, \text{mode} = \text{requesting}, \text{tail}_i = \text{head}_i\}$
3: if $A' = \emptyset$ then return end if
4: $a_k \leftarrow \arg \max_{a_i \in A'} \text{pttmp}_i$
5: if $\text{pttmp}_k \leq \text{pttmp}_i$ then return end if
6: RELEASECHILDREN
7: $\text{children}_{\text{parent}} \leftarrow \text{children}_{\text{parent}} \setminus \{a_i\} \triangleright \text{parent}_i$
8: $\text{parent}_i \leftarrow a_k$, $\text{children}_i \leftarrow \text{children}_k \cup \{a_i\} \triangleright a_k$
9: $\text{pttmp}_i \leftarrow \text{pttmp}_k$
10: $S_i \leftarrow S_k \cup \{\text{head}_i\}$, $C_i \leftarrow \text{Neigh}(\text{tail}_i) \setminus \{S_i\}$
11: end procedure

12: procedure RELEASECHILDREN
13: for $a_j \in \text{children}_i$ do
14: $\text{parent}_j \leftarrow a_j$
15: end for
16: $\text{children}_i \leftarrow \emptyset$
17: end procedure

18: procedure RESET
19: $S_i \leftarrow \emptyset$, $C_i \leftarrow \text{Neigh}(\text{tail}_i) \cup \{\text{tail}_i\}$
20: $\text{pttmp}_i \leftarrow \text{pori}_i$
21: end procedure
The intuition is to make agents follow original plans when-\textit{is} still an online and distributed approach during execution. Importantly, even with MAPF plans, \textit{time-independent planning}, i.e., planning paths anticipating only a single step ahead. Although \textsc{Greedy} and Causal-PIBT are for online situations, we tested the time-independent planning, i.e., even though delays of agents cannot be predicted, the trajectories of agents are kept relatively efficient, and, 2) verifying the usefulness to use MAPF plans as hints during executions. We used the MAPF-DP problem since it can emulate imperfect executions, expecting to show the advantage of the time-independent planning. To adapt the time-independent model to MAPF-DP, we designed that the scheduler repeats the following two phases: 1) Each agent $a_i$ in \textit{extended} is activated with probability $1 - p_i$. As a result, $a_i$ successfully moves to $head_i$, with probability $1 - p_i$ and becomes \textit{contracted}. 2) The scheduler activates agents in \textit{contracted} and \textit{requesting} until the configuration becomes stable, i.e., all agents in \textit{contracted} and \textit{requesting} do not change their states unless any agent in \textit{extended} is activated. The rationale is that, the time required by executing atomic actions except for agents’ moves is much smaller than that of the moves. We regard a pair of these two phases as one timestep. The evaluation is based on the sum of cost (SOC) metric. The simulator was developed in C++, and all experiments were run on a laptop with Intel Core i5 1.6GHz CPU and 16GB RAM. In all settings, we tried 100 repetitions.

\textbf{Small Benchmarks} First, we tested the time-independent planning in two small benchmarks shown in Fig. 3. The delay probabilities $p_i$ were chosen uniformly at random from $[0, \bar{p}]$, where $\bar{p}$ is the upper bound of $p_i$. The higher $\bar{p}$ means that agents delay often. We here manipulated $\bar{p}$. Note that $\bar{p} = 0$ corresponds to perfect executions without any delays. We run \textsc{Greedy} and Causal-PIBT in the first testbed. In the second testbed, Causal-PIBT and the enhanced one by an MAPF plan (Causal-PIBT+), see Section 4.3) were performed. Those three were regarded to fail after 10000 activations, implying occurring deadlocks or livelocks. FSPs (Fully Synchronized Policies) and MCPs (Minimam Communication Policies) were also tested as comparisons. To execute FSPs, MCPs and Causal-PIBT+, valid MAPF-DP plans such that minimize SOC, which assumes perfect execution, were computed by an adapted version of Conflict-based Search (CBS) [Sharon et al., 2015], prior to performing MAPF-DP. The results are shown in Fig. 3. Although \textsc{Greedy} failed in most cases due to deadlocks, Causal-PIBT+(+) succeeded in all trials, partly contributed by the deadlock-recovery and reachability properties. In general, FSPs results in bad compared to MCPs. When $\bar{p}$ is small, MCPs was efficient, on the other hand, when $\bar{p}$ increases, the time-independent planning had an advantage. Especially, Causal-PIBT+ worked nicely for all $\bar{p}$.

\textbf{Random Grid} Next, we tested Causal-PIBT and Causal-PIBT+ using one scenario from MAPF benchmarks [Stern et al., 2019], shown in Fig. 4. We manipulated two factors: 1)

- Figure 3: The results in small benchmarks.
changing \( \bar{p} \) while fixing the number of agents (35), and, 2) changing the number of agents while fixing \( \bar{p} \) (0.5). When the number of agents increases, the probability that someone delays also increases. We set sufficient upper bounds of activations. FSPs and MCPs were also tested. In this time, an adapted version of Enhanced CBS (ECBS) [Barer et al., 2014] was used to obtain valid MAPF-DP plans, where the suboptimality was 1.1.

Fig. 4 shows the results. The proposals succeeded in all cases even thought the given graphs are not biconnected. The results are almost similar to previous experiments, i.e., the time-independent planning outputs robust executions maintaining the good SOC in the severe environment as for timing assumptions.

**Large Fields** Finally, we tested proposals using the large fields taken from the MAPF benchmarks, shown in Fig. 5. We respectively picked one scenario for each filed, then tested while changing the number of agents. \( \bar{p} \) is fixed to 0.1. Usual ECBS (suboptimality: 1.1) was used for Causal-PIBT+.

The results, shown in Fig. 5, demonstrated the usefulness of introducing MAPF plans to Causal-PIBT. We observed the huge SOC in den312d due to the existence of one bottleneck. Such a structure critically affects executions with delays.

6 Conclusion

This paper studied the online and distributed planning for multiple moving agents without timing-assumptions. We abstracted reality as a transition system, then proposed the time-independent planning, including Causal-PIBT with the reachability. Through simulations in MAPF-DP, we demonstrated the robustness of the time-independent planning and usefulness to use MAPF plans as hints.

Future directions include the following: 1) Develop time-independent planning other than for PIBT. E.g., it seems to be sound to adapt Push and Swap [Luna and Bekris, 2011] to the time-independent model. 2) Address communication between agents explicitly. This paper neglects delays caused by communication and treats interactions as a black box. The next big challenge is there.

References

[Barer et al., 2014] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In Seventh Annual Symposium on Combinatorial Search, 2014.

[Derakhshandeh et al., 2014] Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot–a new model for programmable matter. In Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures, pages 220–222. ACM, 2014.

[Dresner and Stone, 2008] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. Journal of artificial intelligence research, 31:591–656, 2008.

[Hönig et al., 2016] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In Twenty-Sixth International Conference on Automated Planning and Scheduling, 2016.

[Hönig et al., 2019] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W Durham, and Nora Ayanian. Persistent and robust execution of mapf schedules in warehouses.
[Kim et al., 2015] Kangjin Kim, Joe Campbell, William Duong, Yu Zhang, and Georgios Fainekos. Discof+: Asynchronous discof with flexible decoupling for cooperative pathfinding in distributed systems. In 2015 IEEE International Conference on Automation Science and Engineering (CASE), pages 369–376. IEEE, 2015.

[Luna and Bekris, 2011] Ryan Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In IJCAI, pages 294–300, 2011.

[Ma et al., 2017a] Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[Ma et al., 2017b] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In 16th Conf. on Autonomous Agents and MultiAgent Systems, pages 837–845, 2017.

[Okumura et al., 2019] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pages 535–542, 7 2019.

[Sha et al., 1990] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on computers, 39(9):1175–1185, 1990.

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

[Stern et al., 2019] Roni Stern, Nathan Sturtevant, Ariel Feller, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, pages 151–159, 2019.

[Stern, 2019] Roni Stern. Multi-agent path finding–an overview. In Artificial Intelligence, pages 96–115. Springer, 2019.

[Tel, 2000] Gerard Tel. Introduction to distributed algorithms. Cambridge university press, 2000.

[Wiktor et al., 2014] Adam Wiktor, Dexter Scobee, Sean Messenger, and Christopher Clark. Decentralized and complete multi-robot motion planning in confined spaces. In Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), pages 1168–1175, 2014.

[Wurman et al., 2008] Peter R Wurman, Raffaello D’Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine, 29(1):9, 2008.