Artificial Intelligence 330 (2024) 104087

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

Temporal segmentation in multi agent path finding with applications to explainability ☆

Shaull Almagor [a], Justin Kottinger [b,*], Morteza Lahijanian [b]

[a] Computer Science Department, Technion Israel Institute of Technology, CS Taub Building, Haifa, 3200003, Israel
[b] Ann and H.J. Smead Aerospace Engineering Sciences Department, University of Colorado Boulder, 3775 Discovery Drive, Boulder, 80303, CO, USA

ARTICLE INFO

Keywords:
Multi-agent systems
Path planning
Explainability
MAPF
Path finding
Motion planning
Explainable AI

ABSTRACT

Multi-Agent Path Finding (MAPF) is the problem of planning paths for agents to reach their targets from their start locations, such that the agents do not collide while executing the plan. In many settings, the plan (or a digest thereof) is conveyed to a supervising entity, e.g., for confirmation before execution, for a report, etc. In such cases, we wish to convey that the plan is collision-free with minimal amount of information. To this end, we propose an *explanation scheme* for MAPF. The scheme decomposes a plan into segments such that within each segment, the agents' paths are disjoint. We can then convey the plan whilst convincing that it is collision-free, using a small number of frames (dubbed an *explanation*). We can also measure the simplicity of a plan by the number of segments required for the decomposition. We study the complexity of algorithmic problems that arise by the explanation scheme and the tradeoff between the length (makespan) of a plan and its minimal decomposition. We also introduce two centralized (i.e. runs on a single CPU with full knowledge of the multi-agent system) algorithms for planning with explanations. One is based on a coupled search algorithm similar to A*, and the other is a decoupled method based on Conflict-Based Search (CBS). We refer to the latter as *Explanation-Guided CBS* (XG-CBS), which uses a low-level search for individual agents and maintains a high-level conflict tree to guide the low-level search to avoid collisions as well as increasing the number of segments. We propose four approaches to the low-level search of XG-CBS by modifying A* for explanations and analyze their effects on the completeness of XG-CBS. Finally, we highlight important aspects of the proposed explanation scheme in various MAPF problems and empirically evaluate the performance of the proposed planning algorithms in a series of benchmark problems.

1. Introduction

1.1. Multi agent path finding (MAPF)

MAPF is a fundamental problem in AI, in which the goal is to plan paths for several agents to reach their targets, such that paths can be taken simultaneously, without the agents colliding. Applications of MAPF are ubiquitous in any area where several moving

☆ Portions of this paper were presented in Almagor and Lahijanian [1] and Kottinger et al. [2].

* Corresponding author.
E-mail addresses: shaull@technion.ac.il (S. Almagor), justin.kottinger@colorado.edu (J. Kottinger), morteza.lahijanian@colorado.edu (M. Lahijanian).

https://doi.org/10.1016/j.artint.2024.104087
Received 27 June 2022; Received in revised form 23 January 2024; Accepted 30 January 2024

Available online 7 February 2024
0004-3702/Published by Elsevier B.V.

agents are involved, such as air-traffic control, unmanned aerial vehicles, warehouse robots, autonomous cars, etc. Unfortunately, most variants of MAPF are intractable. However, the importance of this problem has generated a significant body of work over the past decade [3–9], dealing with various aspects of the problem and suggesting increasingly scalable solutions. In particular, a well-performing algorithm for MAPF is *Conflict-Based Search* (CBS) [10], which is a decoupled approach,[1] and has extensions with various heuristics that further improve its performance [11–14].

1.2. Minimal digest of MAPF plans

In many applications the plan (or a digest thereof) need to be conveyed to some entity, either before or after execution. Such applications give rise to various requirements on what type of digest is sent, as we now demonstrate.

1. In safety-critical and heavily-regulated applications (e.g., air-traffic control, hazardous-materials warehouses), planning is not fully automatic, since automated planning has to be trusted in order to act upon, and in order to accept legal and moral accountability for. In such partially-automated setting, a plan is only suggested to a human supervisor, who may act upon it. It is then desirable that the digest is easily understandable to a human.
2. In space-exploration missions, mission-control on earth would want to document and know what plans are being executed. However, sending information in space is costly [15], so we would like to minimize the amount of information sent.
3. In settings where agents often repeat the same tasks, one can reuse existing plans as "building blocks" to construct an on-going plan. Then, to gain understanding of the overall behavior of the system, one needs a digest of the plan to examine and analyze. This can allow e.g., optimizing the space-usage of the environment, by utilizing places that are not visited by agents.

In the applications above, it is not crucial for the recipient of the digest to understand the precise plan. Rather, we would like to abstract away some of the information in the plan in favor of e.g., clarity, amount of data sent, or simplicity of representation.

Our study assumes that the digest of the plan should be sufficient to ensure that the agents indeed do not collide. Indeed, dropping this assumption allows us to represent the plan by simply overlaying the paths of the agents (see Fig. 1). Then, however, it is not possible to determine whether the plan is correct, nor to reason about the behavior of the agents along it.

Guided by this fundamental assumption, we ask what is the "minimal information" that needs to be retained from a plan so that we can ensure it is non-colliding, as well as to facilitate reasoning about it. To this end, we suggest a novel approach by means of *disjoint decompositions*. Intuitively, we decompose the plan into time segments such that, within each time segment, the paths taken by the agents are disjoint. This allows us to discard the precise timing of each agent, and only retain the interval corresponding to the segment. In Fig. 1, we demonstrate such a decomposition of a plan into three parts. Crucially, since the paths are spatially disjoint in each segment, we can ensure the correctness of the plan.

Remark 1. An additional benefit of this approach, in the setting of human supervisors, is that displaying each segment as a picture allows a human to easily see that the agents do not collide, by simply checking that the lines do not intersect. Indeed, the identification of line intersections is done very early in the cognitive process (namely in the primary visual cortex [16,17]). We demonstrate this further in video.[2]

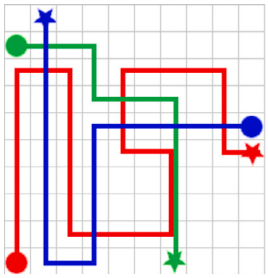
Since our suggested digest helps the recipient to validate that the plan is correct, we henceforth dub it an *explanation*. As we show in Section 2, our approach naturally gives rise to a precise definition of minimality. Moreover, given a plan we show how to compute the minimal explanation for it. The bulk of our work considers how to compute a plan for which this minimal explanation comprises the fewest segments, among all plans.

1.3. What is an explanation?

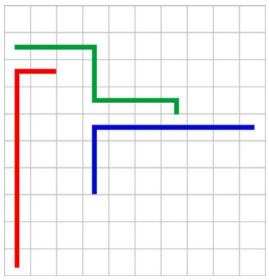
The notion of "explanation" is not well defined and is hugely dependent on the problem at hand and on the entity to whom the explanation is given. For example, in [18], visualization is used to explain the result of certain Machine Learning algorithms, that often come up with complicated classifiers. In this case, the explanation is given to a human, and therefore visualization is helpful. In contrast, in [19], o-minimal sets are given as an explanation for the non-termination of linear loops. These explanations are useful, since it is decidable to check their correctness, whereas the general non-termination problem is not known to be decidable. However, actually checking the correctness is not necessarily tractable, so the recipient in this case is not a human, but rather a computer with unlimited time. This work is closer to [18] in that our explanations are motivated by (and indeed facilitate) visualization. Technically, however, this visualization is a by-product, and our main concern is the combinatorial structure of the explanations, and specifically in minimizing their size. Therefore, in our view, the recipient can be thought of as a communication channel, and we aim to reduce the size of the data we send (while maintaining certain information about the plan).

[1] CBS consists of a centralized component with full knowledge of the system (the conflict tree) but avoids the need to *couple* the multi-agent system into a single meta-agent.

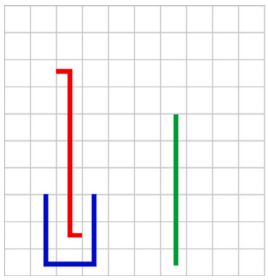
[2] https://www.youtube.com/watch?v=Sh_SslN_X54.



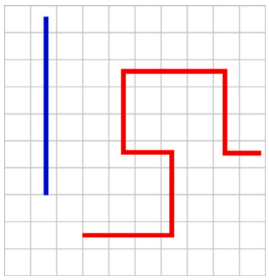
(a) Full Plan



(b) Part 1: time 1–9



(c) Part 2: time 10–16



(d) Part 3: time 17–23

Fig. 1. A plan for three agents in a 10×10 grid. The circles and stars mark the initial and goal locations for the agents, respectively. Figure (a) shows the full plan, and Figures (b), (c), and (d) show a disjoint decomposition.

In order to make the notion of explanation slightly more concrete, we view it in the following way. Consider a decision problem P. An *explanation scheme* for P is a mechanism that outputs, for a given input I, some information called an explanation, or outputs that no explanation is found. Then, we can reason about three properties of the explanation scheme:

1. Soundness: if an explanation exists, then I is a yes-instance (i.e. $I \in P$).
2. Completeness: if I is a yes-instance, then an explanation exists.
3. Simplicity: an explanation is easy to find (if it exists) and to verify.

Note that the simplicity requirement is not formal; it is rather context-dependent. If the scheme is both sound and complete, then the explanations are in fact *proofs*, and if in addition they are simple, then the problem P itself would be simple to solve a priori. Thus, these properties hold simultaneously only in very naive problems. Challenges arise when different combinations of the above properties are considered with their *quantitative* tuning for explanations.

For example, in this work, our explanation scheme is sound but has a quantitative tradeoff between simplicity and completeness. That is, on one hand, the fewer plan segments (pictures) we output, the simpler the explanation is, but on the other hand, we can explain fewer plans using just a few segments, so the scheme is less complete. If we allow an arbitrary number of segments, we can explain any plan (so the scheme is complete), but the explanations are no longer simple.

1.4. Related work

In recent years, there has been significant effort towards providing explanations for problems in AI and in machine learning. Such works are often grouped under the *Explainable Artificial Intelligence (XAI)* title [20,21]. The goal is to provide some meaningful interpretation for the results of algorithms, such that users of the algorithms (although not necessarily human users, as discussed above) are able to gain insight into the result.

Concrete examples of explanations take on various forms. In [22], explanations are given by analyzing possible alternative plans, where the space of alternative plans is determined by certain "interesting properties" given by the user. Then, explanations of plans are contrasive, in that when a user asks e.g., "why does plan π start with A", then answer is given in terms of properties that are violated if A is not taken. When asking, "why is path A optimal, rather than path B which I expected," contrastive explanations are also used in [23] to explain why path A is optimal. More recently, [24] incorporated their contrastive explanation scheme into a single-agent motion planning algorithm. The frameworks explanation scheme is twofold. In the case where a plan was found, the algorithm could explain why it returned solution A, rather than solution B, which the user expected. The algorithm could also explain why it failed for a particular instance.

In [25], explanations for plans are given by a minimal set of differences between the proposed plan and a plan suggested by the user. This type of explanation is often more detailed, as it can be given in terms of comparison with a given alternative. A more unified approach is taken in [26], where several types of explanation queries are allowed. There, the user may change the plan

in certain ways, and the planner should explain why the original plan is better, or re-plan. In the MAPF domain, [27] produced explanations centered around answer-set programming to explain multi-modal MAPF instances.

The study in [28] provides a preliminary taxonomy and a set of important considerations for the design of explainable planners, based on the analysis of a comprehensive user study of motion planning experts. The study explicitly points out the observation that for explanations to be useful they often have to be visualization-centered, further motivating our setting.

1.5. Contributions

There are two main contributions of this work. First, we introduce an explanation scheme for MAPF plans whereby we minimize the amount of data extracted from the plan (in a well-defined sense), while still allowing its validation. We then use this explanation scheme to introduce a new problem, Explainable MAPF. Secondly, we extend two MAPF solvers, namely A* CBS, to solve Explainable MAPF queries. We empirically show that our CBS-based solution can solve the more difficult Explainable MAPF problem with comparable runtimes to CBS solving the classical MAPF problem. Other contributions of this manuscript include:

- we provide a formalization of our MAPF explanation scheme in two- and three-dimensional environments,
- we show that finding an explanation to an existing plan can be done efficiently, whereas generating MAPF plans that produce simple explanations is **NP-Complete**,
- we provide two algorithms capable of generating MAPF plans with short explanations.
- we perform an empirical study comparing our solutions against classical CBS.

1.6. Paper organization

In Section 2, we introduce our explanation scheme for MAPF, namely *vertex-disjoint decompositions*, and state the relevant algorithmic problems. In Section 3, we show that finding optimal explanations for existing plans can be done efficiently, whereas planning for MAPF problems with simple explanations is **NP-Complete**. In addition, we study the tradeoff between time-optimal plans and plans with simple explanations. In Section 4, we introduce two main algorithms to solve the explainable MAPF problem. One is a coupled method that uses A*, whereas the other is a decoupled method that extends CBS for MAPF to Explainable MAPF. The latter requires a low-level search, for which we present three methods by modifying A* for explanations. We analyze the effects of these methods on the completeness of the overall algorithm.

In Section 5 we study MAPF in the three-dimensional setting. There, we wish to still output two-dimensional data. Therefore, we consider projections of the plans onto a plane. However, this may make the paths appear intersecting. We thus adapt the definition to include a *viewpoint*, and study explanations under this modification. We further show how the three-dimensional setting allows for a new perspective on two-dimensional explanations, by means of projecting along spatial dimensions, rather than on the time axis. In this sense, the three-dimensional setting offers an opportunity to further reduce the data in the explanation, by projecting away both time and a spatial dimension.

In Section 6, we present experimental results on various MAPF problems. In particular, we highlight important aspects of the proposed explanation scheme. We also evaluate the proposed algorithms in a series of benchmark problems. The results show that our CBS-based algorithm for the Explainable MAPF is able to scale as well as CBS for the original MAPF problem without the explanations. Finally, we conclude with future research directions in Section 7.

2. Problem formulation

We start by giving the basic definitions used throughout the paper, followed by the formulation of the problems at hand. Note that the following definitions are in the discrete setting, where the combinatorial nature of the problem is emphasized.

Consider a directed graph $G = \langle V, E \rangle$. A *path* in G is a sequence $\pi = v_1 \dots v_m$ such that for all $1 \leq i < m$, $v_i \in V$ and either $(v_i, v_{i+1}) \in E$ or $v_i = v_{i+1}$ (intuitively, agents are allowed to wait in place). Note that we assume that if there exists an edge $(v_i, v_{i+1}) \in E$, then it is unique and that all agents move at the same velocity, traveling at most one edge at every time step. Let $\pi_1 = v_1 \dots v_{m_1}$ and $\pi_2 = u_1 \dots u_{m_2}$ be paths in G. We say that π_1 and π_2 are *non-colliding* if the following hold:

1. $v_i \neq u_i$ for all $1 \leq i \leq \min\{m_1, m_2\}$ (i.e., no vertex collisions),[3]
2. $(v_i, v_{i+1}) \neq (u_{i+1}, u_i)$ for all $1 \leq i < \min\{m_1, m_2\}$ (i.e., no vertex swapping).

We say that π_1 and π_2 are *vertex disjoint* if

$$\{v_0, \dots, v_{m_1}\} \cap \{u_0, \dots, u_{m_2}\} = \emptyset.$$

We lift these definitions to a set of paths by requiring that they hold pairwise.

[3] Since we only consider indices up to $\min\{m_1, m_2\}$, then, intuitively, once the goal is reached, the agent "disappears". However, one can change this and adapt the results mutatis-mutandis.

Let $P = \{\pi_1, \ldots, \pi_k\}$ be a set of non-colliding paths of length at most T. A *vertex-disjoint decomposition* of P is an ordered list ℓ of natural numbers $1 = t_0 < t_1 < \ldots < t_r = T + 1$ such that for every $0 < i \leq r$, the paths $\{\pi_j[t_{i-1}, t_i - 1]\}_{j=1}^{k}$ are vertex-disjoint (where $\pi_j[a, b]$ refers to the infix of π_j between indices a and b, and we cut-off the path if the indices are out of bound). We refer to r as the *index* of ℓ. The minimal decomposition index of a vertex-disjoint decomposition of P is referred to as the *index* of P.

Remark 2. Consider a set of non-colliding paths P of length T, where the length of P is the maximal length of the paths in P. A trivial vertex-disjoint decomposition of P is the list $1, 2, \ldots, T + 1$. Thus, a vertex-disjoint decomposition always exists, and the index of P is at most T.

We are interested in the classical (labeled) *Multi Agent Path Finding (MAPF)* problem where there exists k agents on a graph G where every agent is assigned a *single* and *unique* start and goal region *a priori*. Formally, the classical MAPF problem is as follows.

Problem 1 *(Labeled MAPF).* Given a graph $G = \langle V, E \rangle$ and ordered lists $s_1, \ldots, s_k$ and $g_1, \ldots, g_k$, where $s_i, g_i \in V$ for all $1 \leq i \leq k$, find a plan $P = \{\pi_1, \ldots, \pi_k\}$ such that for all $i \in 1, \ldots, k$, π_i is a path for agent i beginning at s_i and finishing at g_i.

Note that this is a search problem. The decision version of the problem asks whether there exists such a plan, and the threshold version asks whether there exists a plan of length at most T. In addition, the optimization version asks to find the minimal-length plan.

The introduction of explanations gives rise to two problems. The first problem is to compute a minimal decomposition of a given plan, which is formally stated below.

Problem 2 *(Minimum Index Decomposition).* Given a graph $G = \langle V, E \rangle$ and a set of non-colliding paths (i.e., a plan) P, compute a vertex-disjoint decomposition of P with minimal index.

Recall that by Remark 2, a vertex-disjoint decomposition always exists, so Problem 2 is well defined.

While a solution for Problem 2 may give us the "best" explanation for a given plan, there may exist other (possibly "worse") plans that offer better explanations. Thus, we consider the more involved problem of planning with explanations in mind. The formal statement of this problem is as follows.

Problem 3 *(Explainable MAPF).* Given a graph $G = \langle V, E \rangle$, lists $s_1, \ldots, s_k$ and $g_1, \ldots, g_k$ of source and goal vertices, find a plan P for the agents with a minimal vertex-disjoint decomposition index.

Remark 3 *(Edge Conflicts).* A common variant of MAPF also forbids *edge conflicts*, whereby agents traverse an edge in opposite directions. While a-priori a vertex disjoint plan (which therefore admits an explanation) might have edge conflicts, we can easily avoid outputting explanations for such plans by adding a vertex in the "middle" of each edge, so that edge conflicts induce vertex conflicts. This requires only a doubling of the length of the plans, and hence retains the complexity of the decision problems.

3. Explanations of MAPF

In this section, we study vertex-disjoint decompositions. We start by addressing Problem 2.

3.1. Computing minimal vertex-disjoint decompositions

Our first result is that computing a minimal-disjoint decomposition can be done efficiently.

Theorem 1. *Problem 2 can be solved in polynomial time.*

Proof. Let $P = \{\pi_1, \ldots, \pi_k\}$ be a given plan, and let T denote its length. Our algorithm for computing a minimal vertex-disjoint decomposition (see Algorithm 1 for pseudo-code) proceeds in iterations, as follows. Set $t_0 = 1$. Then, at iteration i, the algorithm computes the maximal number $t_i \leq T + 1$ such that $\{\pi_j[t_{i-1}, t_i]\}_{j=1}^{k}$ is a set of vertex-disjoint paths (Line 7 of Algorithm 1). That is, we greedily keep following the paths until there is an intersection, and then we start a new segment in the decomposition (Lines 11-13 of Algorithm 1).

Clearly, this algorithm can be implemented in polynomial time since, in every iteration, we only need to maintain a set of disjoint vertices, and we need to make only a single pass on P.

We turn to prove the correctness of this algorithm. Let $1 = t_0 < \ldots < t_r = T + 1$ be the decomposition found by our algorithm. Consider some r' and a decomposition $1 = t'_0 < \ldots < t'_{r'} = T + 1$ of P. We show that $r' \geq r$.

To this end, we claim that, for every $0 \leq i \leq r'$, it holds that $t'_i \leq t_i$. That is, the "endpoints" of the decomposition found by our algorithm are at least as high as those of the primed decomposition. We prove this by inductions over i. For $i = 0$ this holds by definition since $t_0 = t'_0 = 1$. Next, consider $1 \leq i + 1 \leq r'$; then, by the induction hypothesis, $t'_i \leq t_i$. Now, by the definition of

Algorithm 1: Minimal Disjoint Decomposition.

```
Input: Plan P = {π_1, ..., π_k}
Output: Minimal disjoint decomposition
T ←Length of P;
lastSegStart ← 1;
segments ← [ ];
for index from 1 to T do
    newSeg ← False;
    for j from 1 to k do
        if π_j[index] ∈ ⋃_{r∈{1,...,k}\{j}} π_r[lastSegStart,index−1] then
            newSeg ← True;
            break;
        end
    end
    if newSeg then
        segments.Append(index-1);
        lastSegStart ← index;
    end
end
segments.Append(T);
return segments;
```

our algorithm, t_{i+1} is the maximum such that $\{\pi_j[t_i, t_{i+1}]\}_{j=1}^{k}$ is vertex disjoint. In particular, it must hold that $t'_{i+1} \leq t_{i+1}$, since $\{\pi_j[t'_i, t'_{i+1}]\}_{j=1}^{k}$ is vertex disjoint as well, and $t'_i \leq t_i$.

The inductive argument implies that $r' \geq r$. Therefore, our decomposition is indeed minimal. □

The tractability of Problem 2 implies that explaining the results of MAPF can be readily incorporated into existing MAPF solvers.

3.2. Planning for explainability

In this section, we study the more involved Problem 3, where we wish to find a plan that admits a small decomposition. In order to state complexity results, we address the decision version of Problem 3, namely:

Problem 4. Given a graph $G = \langle V, E \rangle$, lists $s_1, \ldots, s_k$ and $g_1, \ldots, g_k$, and a parameter $r \in \mathbb{N}$, decide whether there exists a plan P for the agents with index at most r.

Observe that when $r = 1$, Problem 4 amounts to deciding whether there are vertex-disjoint paths for the agents, which is **NP-Hard** in several settings. Specifically, by [29,30], we have the following:

Lemma 1. *Problem 4 is **NP-Hard**, even under each of the following restrictions:*

- *G is undirected and planar and $r = 1$ [29].*
- *G is directed and $k = 2$ (i.e., there are only two agents) and $r = 1$ [30].*

Remark 4. Lemma 1 suggests that deterministic algorithms for Problem 3 (and Problem 4) will, intuitively, have an exponential[4] running time with respect to the size of the environment, and not only with respect to the number of agents. As we discuss in Sections 4 and 6, this is indeed a major difficulty in coupled algorithms.

To provide a matching upper bound, we have the following:

Lemma 2. *Problem 4 is in **NP**.*

Proof. In [31], it is shown[5] that the shortest plan has length $B = O(n^3)$, where $n = |V|$ and B is efficiently computable, i.e., the length of the shortest path is bounded above by $O(n^3)$. Thus, if $r > B$, then deciding whether there is a plan with index at most r is equivalent to deciding whether there exists a plan of length at most r at all. Indeed, by splitting the plan into r segments of a single timestep, we obtain a decomposition. Since the latter problem is clearly in **NP**, this case is handled.

Next, we consider the case where $r < B$. A priori, it could be the case that, within each decomposition segment, the paths are very long (i.e., super-polynomial), rendering the plan very long. We show that this is not the case.

[4] more precisely - super polynomial, assuming $\mathbf{P} \neq \mathbf{NP}$.

[5] The model in [31] allows the agents to stay in place, as we do. See Section 7 for the model where staying in place is not allowed.

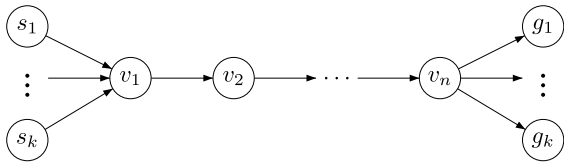


Fig. 2. Setting for Example 1.

Consider a segment in a decomposition of a plan. Since the agents are allowed to stay in place, we can assume that in all paths within the segment, the only cycles are caused by staying in place. Indeed, replacing larger cycles by staying in place will keep the paths disjoint.

Next, observe that agents gain nothing by staying in place within a segment unless this is done so that other agents can finish traversing their (disjoint) path. Thus, staying in place for all agents can be delayed to the last vertex of the segment. Finally, we can omit any transition where all agents stay in place.

We conclude that the length of each segment is at most $O(n)$, since it comprises simple paths with possible waiting at the end of the segment, so that all agents finish their segments. Thus, the overall length of a minimal plan is $O(n \cdot r) = O(n^4)$.

Since verifying that a decomposition is vertex-disjoint can be done in polynomial time in the length of the plan, and we have shown that if there is a plan, there is also one of polynomial length, we conclude that the problem is in **NP**, □

By combining Lemmas 1 and 2, we obtain the following theorem.

Theorem 2. *Problem 4 is **NP-Complete**.*

Lemma 1 shows that the Explainable MAPF problem (Problem 3) is more difficult than MAPF problem, in that it remains **NP-Hard** even under restrictions where MAPF becomes tractable. In Section 4, we introduce algorithms that can solve Problem 3 with comparable runtimes to solving the MAPF problem (Problem 1).

3.3. Explainability versus length

We now turn to study the tradeoff between plans that admit a small decomposition index and plans with small lengths. Traditionally, when solving the MAPF problem (Problem 1), one looks for plans that minimize the maximal time it takes an agent to reach its goal (i.e., length of the plan). As we now show, there is a potential tradeoff of the length and the minimal decomposition index of a plan.

Example 1. Consider the setting depicted in Fig. 2. The lists of source and goal nodes are $s_1, \ldots, s_k$ and $g_1, \ldots, g_k$, respectively. Clearly, there is a (minimal-length) plan for all the agents to reach their goal within $O(n+k)$ time steps by traversing path $v_1, \ldots, v_n$ "back to back". However, the decomposition index of such a plan is $O(n+k)$, since the disjoint parts are of size 1 (i.e., at every step the paths intersect). However, if each agent waits for the previous one to reach its goal, the decomposition index is $O(k)$, but the length is $O(n \cdot k)$.

Remark 5. One recent approach of tackling the MAPF problem is using *highways* [32], which are, roughly speaking, "preferable paths" that were successfully taken by other agents. Example 1 points to an intriguing aspect of disjoint decompositions: if agents want to follow the same path, they can only do so in separate segments of the decomposition. Thus, highways are intuitively bad for our explanation scheme. In Section 7, we discuss possible approaches to combine highways and disjoint decompositions.

4. Algorithms for explainable MAPF

In this section, we introduce two centralized algorithms to solve the Explainable MAPF problem (Problem 3). The first method is coupled algorithm that uses a simple adaptation of A* to perform a search by keeping the history of the paths to be able to keep track of disjoint-segments. This approach is computationally expensive as discussed in Section 4.1. The second method (Section 4.2) mitigates this hurdle by performing a decoupled search. We extend CBS, which is designed for MAPF, to solve the explainable version of MAPF.

4.1. Coupled approach: A^*

One of the traditional approaches for solving MAPF (Problem 1) is search-based, using A^* with various heuristics and optimizations [4,33,34]. The difficulty in search-based methods lies in the state space's size, which is $|V|^k$, where k is the number of agents. Indeed, we must keep track of all the agents simultaneously to avoid a collision. Notably, the state space grows exponentially in the number of agents, but only polynomially in the graph's size.

Unfortunately, Problem 3 gives rise to another blow-up. Recall that within each segment of the decomposition, the paths taken by the agents are disjoint. Thus, to perform a search-based algorithm, each state must include, apart from the location of each agent i, the history of the states visited by Agent i in the current segment. Thus, each state encodes the current state of each agent i and a set $S_i \subseteq V$. Thus, the overall size of the state space is $O(|V|^k \cdot 2^{|V| \cdot k})$, which is exponential both in the number of agents and the size of the graph (cf. Remark 4). The size can be slightly improved by noticing that the S_i are disjoint, and thus the subsets can be represented by a function $f : V \to \{1, \ldots, k, \perp\}$, so that $f(v)$ is the agent that had visited v in the current segment (where $f(v) = \perp$ means that v has not been visited yet). In this view, the state space is of size $O\left(|V|^k \cdot (k+1)^{|V|}\right)$, which is still exponential in both k and $|V|$.

The analysis above implies that even for 2 agents, applying A^* to Problem 3 is challenging. Below, we introduce a decoupled approach that is more scalable.

4.2. Decoupled approach: explanation-guided CBS

Our second solution to Problem 3 stems from CBS [10], a decoupled MAPF algorithm. Here, we first review this algorithm and then present our extensions to it to obtain Explanation-Guided CBS (XG-CBS).

4.2.1. CBS for MAPF

CBS is a two-level search on the space of possible plans, consisting of a high-level conflict-tree search and a low-level graph search. At the high-level, CBS keeps track of a *constraint-tree*, in which each node represents a suggested plan, which might have collisions, referred to as *conflicts*. Initially, a root node is obtained by using a low-level graph search algorithm, typically A^* with a shortest-path heuristic, to find a path for each agent from start to goal, ignoring the other agents (hence the decoupled nature of the method).

At each iteration, CBS picks an unexplored node from the tree, based on some heuristic. Then, the conflicts (namely collisions) in the plan corresponding to that node are identified. CBS attempts to resolve the conflicts by creating child nodes based on the conflicts, as follows: if Agents i and j collide at time t in vertex v, then two children are created for the node, one with the constraint that Agent i cannot be in vertex v at time t, and the other dually for Agent j. Then, in each child node, a low-level search is used to replan a path for the newly-constrained agent, given the set of constraints obtained thus far along the branch of the constraint tree. This process repeats until either a non-colliding plan is found, or no new nodes are created in the constraint tree, at which point CBS returns that there is no solution.

Example 2. We (partially) demonstrate CBS in Fig. 3. In this example, the root node has a colliding plan and hence a constraint is placed on the yellow vertex at time 2, so two children are created with new plans for each of the two colliding agents.

CBS performs well for standard MAPF queries. However, it is ill-suited for solving Problem 3 due to its lack of regard for vertex-disjoint decomposition of the proposed solutions. More precisely, CBS is guided toward short plans, whereas minimizing the plan index typically incurs a tradeoff with plan length (as illustrated in Example 1). Below, we build upon CBS to plan for explainability, in order to address Problem 3 more efficiently than using the A*-based method in Section 4.1.

4.2.2. CBS for explainable MAPF

We modify CBS both at the high-level (constraint tree) and low-level (graph search), to obtain a new algorithm dubbed *Explanation-Guided CBS (XG-CBS)*. To this end, we first introduce *segmentation conflicts* to the constraint tree. These are conflicts that occur when the plan is non-colliding, but whose index is greater than the bound r. These conflicts are resolved by placing appropriate constraints, as we detail below. We elaborate on the low-level search in Section 4.2.3.

4.2.2.1. Segmentation conflicts Recall that in CBS, whenever a plan has collisions, constraints are placed on the colliding agents to force one of them away from the collision point. We keep these constraints in XG-CBS, and introduce additional constraints to handle segmentation. In order to define the new constraints, we recall how vertex-disjoint decompositions are computed.

Consider a plan $P = \{\pi_1, \ldots, \pi_n\}$. Algorithm 1 in Section 3.1 shows that a minimal decomposition of P can be found greedily by lengthening the current interval as long as the paths are disjoint, and starting a new segment once an intersection occurs. More precisely, we set $t_0 = t_1 = 1$ and check $\{\pi_1[t_0, t_1], \ldots, \pi_n[t_0, t_1]\}$ for disjointedness. If it is disjoint, then t_1 is incremented by one. The process continues until the segment is not disjoint, at which point we add $t_1 - 1$ as a segmentation point, set $t_0 = t_1$, and start the process again. This continues until the entire plan is segmented.

We use this greedy characterization to define segmentation constraints as follows. For a non-colliding plan $P = \{\pi_1, \ldots, \pi_n\}$ of length K, let $1 = t_0 < t_1 < \ldots < t_r = K + 1$ be a vertex-disjoint decomposition found by Algorithm 1. It follows that for every $1 \leq \ell \leq r$, we cannot extend the disjoint segment $[t_{\ell-1}, t_\ell - 1]$ to time t_ℓ. That is, there exist agents $i \neq j$ with $\pi_i[t_\ell, t_\ell] \in \pi_j[t_{\ell-1}, t_\ell]$, where $\pi_i[t_\ell, t_\ell]$ is a single vertex. With each such pair of agents i, j, we associate the vertex $v = \pi_i[t_\ell, t_\ell]$ and the times $T_i = t_\ell$ and T_j to be a time such that $\pi_j[T_j, T_j] = v$. Intuitively, T_i and T_j are the times when Agents i and j, respectively, visit v in the segment $[t_{\ell-1}, t_\ell]$. Then, for a node with plan P in the constraint tree of XG-CBS, we add two children with the following constraints: one child prevents Agent i from visiting v at time T_i, and the other prevents Agent j from visiting v at time T_j. Note that, for a node with multiple segmentation conflicts, several such pairs of child nodes are added, one pair per conflict.

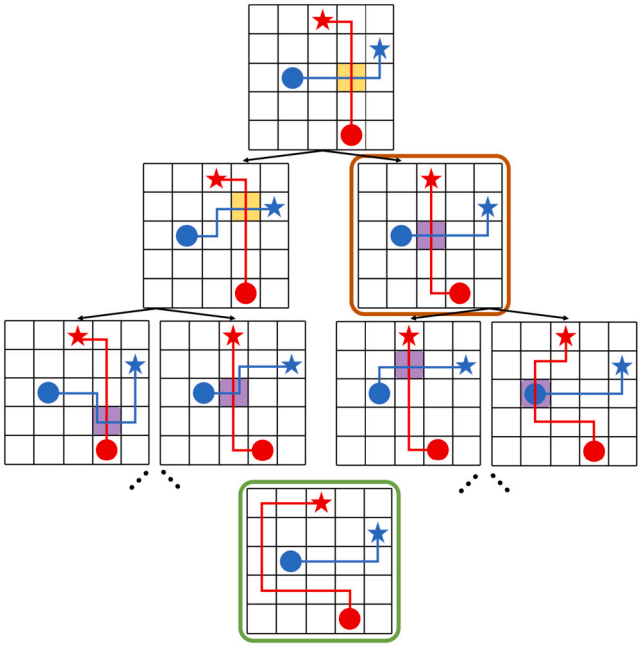


Fig. 3. Illustration of XG-CBS with A^* as the low-level planner. Yellow and purple colors indicate collision and segmentation conflicts, respectively.

Example 3. In Fig. 3, we depict segmentation conflicts as purple squares. For example, in the orange node of the tree, the plan requires two segments, due to the path intersection in the purple node, visited by the blue agent at time 1 and by the red agent at time 3. The two children of this node prevent each of these visits, and replan for the corresponding agent.

4.2.2.2. XG-CBS We are now ready to describe the operation of XG-CBS, with the caveat that we do not explicitly state the implementation of the low-level graph search algorithm. We leave this detail to Section 4.2.3 and only assume that the low-level search algorithm is sound and complete, e.g., A^*.

Algorithm 2: XG-CBS$(G, (s_i)_{i=1}^n, (g_i)_{i=1}^n, r, B)$.

```
1  R.C, Q, P_r ← ∅;
2  for every agent do
3      P_r.add( graphSearch(G, s_i, g_i, R.C, P_r, B) ) ;
4  if P_r.size() < n then
5      return no solution
6  R.plan, R.index, R.cost ← P_r ;
7  Q.add(R);
8  while Q not empty do
9      N ← Q.highestPriority() ;
10     Q.pop(N); c ← conflictCheck(N.plan, r) ;
11     if c is empty then
12         return N.plan
13     for every agent a_i ∈ c do
14         K.C ← N.C ∪ (a_i, v, T_i);
15         P_{-i} ← K.plan \ π_i ;
16         π_i ← graphSearch(G, s_i, g_i, K.C, P_{-i}, B);
17         if π_i exists then
18             P_new ← P_{-i} ∪ π_i;
19             K.plan, K.index, K.cost ← P_new;
20             Q.add(K);
```

The pseudocode of the XG-CBS planner is shown in Algorithm 2. Given an Explainable MAPF instance consisting of a graph G, a list of source $(s_i)_{i=1}^n = (s_1, \ldots, s_n)$, a list of goal vertices $(g_i)_{i=1}^n = (g_1, \ldots, g_n)$, an index bound r, and a path length bound B (see Remark 6 below), XG-CBS proceeds as follows (see Algorithm 2). First, a root node R is initialized with an empty set of constraints C (Line 1 of Algorithm 2). Then, the low-level planner is called to find a path for each agent (Lines 2-3 of Algorithm 2). If the graph search fails to generate a path for an agent, then XG-CBS is also unable to generate a root plan P_r and therefore must return no solution (Line 5

of Algorithm 2). If, however, a full plan is found, then it is saved in R along with all other important information and added to the priority queue Q (Lines 6-7 of Algorithm 2). While the queue is not empty, XG-CBS selects the highest priority node N, removes it from the queue, and evaluates its plan $N.plan$ for a conflict $(a_i, a_j, v_i, v_j, T_i, T_j)$ between Agent i at (v_i, T_i) and Agent j at (v_j, T_j) (Lines 9-10 of Algorithm 2). Note that segmentation conflicts are included in this definition by letting $v_i = v_j$ and $T_i \neq T_j$.

If no conflicts exist for a given nodes plan, then it is returned as the solution (Line 12 of Algorithm 2). Otherwise, for every agent in the conflict, a new node K is added with a new constraint (a_i, v_i, T_i), and the newly constrained agent is re-planned for using low level graph search (Lines 13-16 of Algorithm 2). If successful, the new plan and all its information is added to K before it is added Q, where it will eventually be evaluated for conflicts (Lines 17- 20 of Algorithm 2).

Remark 6. During the low-level planning, an upper bound B is set on the length of the path, and serves to bound the constraint tree, so that XG-CBS necessarily terminates. Note that in order to retain completeness, we must initialize XG-CBS with a large enough bound. Such a bound originates from the proof of membership in **NP** of Problem 4 discussed in Lemma 2. We elaborate on this in Theorem 3.

A crucial aspect of XG-CBS is the cost function on the tree nodes. Recall that a common cost function for the high-level CBS is the combined length of all the paths (a.k.a. sum-of-costs). This approach, however, tends to conflict with optimizing for explainability. Thus, XG-CBS utilizes the index of the plan to define a cost function. Specifically, the primary cost of a proposed plan is the index of the plan, with a small tweak. Recall that plans in the constraint tree may contain collisions, in which case the index is undefined. We circumvent this by viewing collisions as an end of a segment. Then, the combined length of paths is only used as a tie-breaker. This cost function enables XG-CBS to prioritize plans with a lower number of segments.

Example 4. Fig. 3 demonstrates a run of XG-CBS where the low-level planner is standard A^*, and the index bound $r = 1$.

We conclude this section by showing that XG-CBS is complete.

Theorem 3 *(XG-CBS Completeness). Given an instance of Explainable MAPF, we can efficiently compute a bound B such that XG-CBS when run with bound B terminates, and if the instance is solvable then XG-CBS terminates with a valid solution.*

Proof. The proof is a small variation on the completeness proof of standard CBS.

Consider a solvable instance of Explainable MAPF, namely $G = \langle V, E \rangle$, lists $s_1, \ldots, s_n$ and $g_1, \ldots, g_n$, and a bound r. Since this instance is solvable, there exists a plan $P = \{\pi_1, \ldots, \pi_n\}$ with index at most r. Moreover, as in the proof of Lemma 2 we know that such a plan exists whose length is at most B where B is computable in polynomial time from the size of the instance [31].

We obtain from the plan P a maximal set $C_{\max}$ of constraints for all the agents, by adding, for each agent i, every constraint the prevents agent i from being at vertex v at time t, for every v, t such that $\pi_i[t, t] \neq v$. Intuitively, the only paths allowed under $C_{\max}$ prescribe P exactly.

We claim that as long as no solution is found by XG-CBS, there exists a branch in the constraint tree whose set of constraints is a subset of $C_{\max}$, and that this path has an unexplored node. This is easily proved by induction on the constraint tree: the root node does not have any constraints, and $\emptyset \subseteq C_{\max}$. If the root is not a solution, then it has children obtained by conflicts. This completes the base case. For the induction step, consider the aforementioned branch, and consider the unexplored leaf node. If the plan represented in the leaf is not a solution, then it has children obtained by new constraints. We claim that at least one of these children adds a constraint from $C_{\max}$. Indeed, P does not have any conflicts, and so any conflict must produce at least two constraints, one of which is not in $C_{\max}$ (otherwise, $C_{\max}$ allows the conflict in the leaf, which is a contradiction). So we are done.

Therefore, unless XG-CBS terminates with a solution earlier, at least one branch will be expanded toward P. To complete the argument, we observe that the constraint tree of XG-CBS is bounded, since the lengths of the plans are bounded by B. It follows that every branch will eventually be explored. In particular, P will be reached.

Finally, if the instance is not solvable, then eventually every possible constraint is placed, and the constraint tree is no longer updated, thus terminating the search. □

4.2.3. Graph search explainable MAPF

The low-level search has a twofold impact on the behavior of CBS. First, it determines the concrete paths obtained after placing constraints. Second, since it is run for every node, it has a significant impact on the runtime. In this section, we study four low-level search algorithms for XG-CBS. We start with an overview of our approaches.

In classical CBS, the goal is to find the shortest plan, making A^* (with Hamming distance heuristic) a reasonable choice. For XG-CBS, however, the typical behavior of A^* is ill-fitting. Intuitively, this is because A^* tends to make very local changes in plans. Then, a segmentation conflict, which occurs on an intersection of paths, is likely resolved in a way that still intersects the same path in a nearby location or time. To illustrate this, consider the orange node in Fig. 3, and observe that the segmentation conflict for the red agent is resolved by going through the blue agent's origin, creating another segmentation conflict. Hence, many segmentation conflicts are typically required to be able to reduce the index of the plan. Despite this, A^* is very fast, and thus allows a rapid exploration of the constraint tree. Thus, A^* can be seen as one extreme, where speed is preferred over explanation-oriented paths.

At the other extreme, in order to orient XG-CBS toward a minimal-index plan, we propose a low-level search called *Explanation-Guided A^* (XG-A^*)* that uses A^* with a novel, segmentation-based heuristic. Intuitively, XG-A^* guides the search by minimizing the number of segments, as opposed to minizing the length. As we discuss in Section 4.2.3.1, XG-A^* is highly guided towards minimal explanations but is slow due to keeping track of the path *history*. Our next approach is to get the best of both worlds, by combining XG-A^* and A^* in a weighted manner. We elaborate on this in Section 4.2.3.2.

The three approaches above maintain the completeness of XG-CBS. Our final low-level planner, discussed in Section 4.2.3.3, sacrifices completeness in favor of circumventing the need to keep track of the path's history in XG-A^*, thus obtaining a fast, explanation-oriented search (Section 4.2.3.3).

4.2.3.1. XG-A^* – explanation guided A^* Recall that in CBS, the low-level search A^* ignores the existing explanation of other agents when replanning for a certain agent. Thus, standard A^* takes as input the graph $G = \langle V, E\rangle$, start and goal vertices $s, g \in V$ for an agent, and the set of constraints C in the current node. In contrast, XG-A^* accounts for existing segments, and hence, also receives as input the set of paths of the other agents, denoted by P_{-1}, and a bound B on the maximal allowed path length for the agent. We remark that the bound B is only used to terminate the search if the plan becomes too long. This assures progress so that completeness is retained (cf., Theorem 3).

For brevity, in the following, we assume XG-A^* plans for Agent 1, and the paths for the other agents are $P_{-1} = \{\pi_2, \ldots, \pi_n\}$. Intuitively, XG-$A^*$ searches for a path for Agent 1 from s to g (that does not violate the constraints in C), while maintaining that the index of the decomposition of P_{-1} combined with the planned path so far remains minimal. We demonstrate this before giving the precise details.

Example 5. Consider the root node of Fig. 3 with the colliding paths of the two agents. Once the collision constraint is identified, two children are generated with the respective constraints. Now consider XG-A^* planning for the red agent given the path of the blue agent. XG-A^* initially attempts to keep the index at 1, i.e., to keep the paths of the agents disjoint. To this end, XG-A^* arrives at the plan in the green node (bottom of Fig. 3) before even suggesting the plan in the orange node, which the standard A^* does. Indeed, the orange node has index 2, and therefore is not explored until all index 1 plans are exhausted. This example demonstrates how XG-A^* directs XG-CBS toward a minimal-index plan.

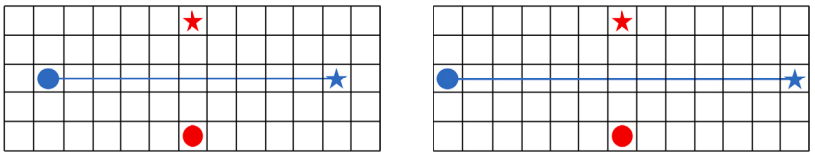
We now turn to the details of XG-A^* as shown in Algorithm 3. The search space of XG-A^* consists of nodes of the form (v, t, H, i) where $v \in V$ is the vertex, $t \in \mathbb{N}$ is the timestamp, H is a sequence of vertices, representing the history of the path from the last segmentation time, and i represents the plan index up to time t. XG-A^* performs a search on the graph G from the start node $(s, 0, \emptyset, 1)$ guided toward any node corresponding to the goal vertex g as long as $i \leq \bar{r}$, where $\bar{r}$ is the index of P_{-1}. The central element is the heuristic guiding the search. A node (v, t, H, i) is assigned two values: the current index i, which is the primary heuristic value, and the shortest-path metric from v to the goal g in the graph G itself, which is used as a tie-breaker. In order to expand a node, a neighbor of v is selected on the graph, and t is increased by 1. At this point the new vertex and time are checked against the constraints C, and if they are not constrained, H and i are computed as per the greedy approach described in Section 4.2.2 (Lines 7-9 of Algorithm 3). Thus, XG-A^* starts by exploring all 1-segment plans, and only once these are exhausted, moves on to 2-segments, etc.

Algorithm 3: XG-$A^*(G, s_i, g_i, C, P_{-i}, B)$.

```
1  Q ← {s_i};
2  while Q not empty do
3      c ← Q.highestPriority();
4      if c.loc = g_i then
5          return π_i ← c.Path()
6      else
7          Q.pop(c); N ← expand(c, G, C, B) ;
8          for every n ∈ N do
9              n.index ← Segment(n.Path(), P_{-i}) ;
10             if n.index ≤ n.parent.index then
11                 n.gScore ← n.parent.gScore+1;
12                 Q.add(n);
13             else
14                 if n.parent.gScore+1 ≤n.gScore then
15                     n.gScore ← n.parent.gScore+1;
16                     Q.add(n);
```

We now make two important observations regarding the behavior of XG-A^*.

Remark 7. Observe that the index of a node depends not only on the plan for Agent 1, but also on the decomposition of the plan P_{-1}. Therefore, if P_{-1} alone causes segmentation, XG-A^* also increases i in the current node. This causes XG-A^* to "synchronize"



(a) $r = 1$ plan exists (b) $r = 1$ plan does not exist.

Fig. 4. XG-A^* drawback of Remark 8.

segmentations. That is, if a path intersection in P_{-1} induces a new segment, the XG-A^* attempts to make Agent 1 intersect another path at that exact time, in order to avoid creating a new segment, which ultimately leads to a lower index.

Remark 8. Since the primary heuristic is not guided toward the goal g, XG-A^* spends a lot of time covering. For instance, it exhausts all index-1 plans before incrementing the index, even if it is impossible to reach g in 1 segment. This is demonstrated in Fig. 4, where XG-A^* is used to compute a path for the red agent given the existing path of the blue agent. In Fig. 4(a), an index-1 plan exists, and XG-A^* finds it relatively quickly, as it is guided toward the goal within the space of index-1 plans (by going around the blue agent). In Fig. 4(b), it is clear that no index-1 plan exists. However, XG-A^* first has to exhaust all index 1 plans, before attempting index 2 (the shortest path). This severe drawback means XG-A^* is slow with respect to the size of the graph G, rather than the number of agents. As we mention in Section 3, this difficulty is inherent to Explainable MAPF.

Since XG-A^* eventually exhausts the space of possible plans, ordered by index, and since this space is bounded using the bound B, we obtain the following.

Theorem 4 *(XG-$\mathbf{A}^*$ Completeness). Given a set of paths $\{\pi_2, \ldots, \pi_n\}$, source and goal vertices s_1 and g_1, respectively, a set of constraints C, and a bound B, if there exists a path from s_i to g_i of length at most B that does not violate the constraints in C, then XG-A^* will terminate with such a path π_i that minimizes the index of $\{\pi_1, \ldots, \pi_n\}$.*

As demonstrated in Remark 8, XG-A^* may spend a lot of time exhausting the plans of a certain index before making any progress. As we now show, we can alleviate some of the computational cost, using simple observations.

Eliminating Cycles: Assume that the graph G allows agents to stay in place, i.e., has self-loops. This is typical in, e.g., warehouse robots. Now, consider a plan $P = \{\pi_1, \ldots, \pi_n\}$ such that in its vertex-disjoint decomposition, Agent i makes a cycle that is contained entirely within a certain segment. Since paths within a segment are disjoint, we can eliminate this cycle and replace it with Agent i waiting in place for the duration of the cycle. Moreover, we can shift this waiting by a wait in the initial vertex of the segment.

Thus, we observe that any plan can be put in a "normal form" where within each segment, no agent makes a cycle, and staying in place is allowed only on the initial vertex. We use this to speed up XG-A^* by limiting the search space to comply with this condition: it is easy to check whether an agent has a cycle within a segment (except for looping in the initial vertex), as H, the history of the segment, is part of the information of each node.

Shortest Path after Segment Bound: Recall that XG-A^* exhausts all the plans with index i before moving to index $i + 1$. We propose a speed up, whereby when the index reaches the bound $\bar{r}$ (index of P_{-1}), the remaining search is performed using standard A^*, i.e., searching for the shortest path to the target, rather than exhausting the remaining plans. Technically, this modification retains the completeness of the algorithm, and hence Theorem 4 is still valid. Intuitively, this offers a speed up since if we already reach an index beyond the given bound, it is unlikely that planning for the current agent helps to reduce segmentation. Therefore, we terminate the search as quickly as possible and allow for further exploration of the conflict tree. We find that empirically, this heuristic speeds up the algorithm, especially for small r.

4.2.3.2. WXG-A^* – weighted explanation guided A^* As demonstrated in Remark 8, XG-A^* spends a lot of time exhausting the plans of a certain index before making any progress towards the goal. This occurs because the cost function of XG-A^* is the plan index and uses path length only as a tie-breaker. Conversely, standard A^* uses path length as the cost function and becomes efficient with a heuristic (estimate of path length to goal), completely ignoring the plan index. These algorithms are two extremities of explanation-guided graph search. To get the best of both worlds, we design a general algorithm called *Weighted* XG-A^* (WXG-A^*) that combines the two search methods. The premise behind WXG-A^* is to simultaneously inherit the index-minimization property of XG-A^* and the efficient search property of A^*.

Let f_x and f_a denote the cost functions of XG-A^* and A^*, respectively. We define the cost function of WXG-A^* to be a linear combination of f_x and f_a, i.e., for node q,

$$f_w(q) = w f_x(q) + (1 - w) f_a(q),$$

where $w \in (0, 1)$. The function $f_w(q)$ encourages *both* index minimization and efficient graph search. The amount that f_w tends toward either type of graph-search depends on weight w. As $w \to 1$, f_w biases more towards minimal-index paths, and hence, the search becomes exhaustive (slower). Conversely, as $w \to 0$, the search tends more towards shortest path length (hence faster). Algorithmically, WXG-A^* is simply XG-A^* guided by f_w rather than f_x.

Algorithm 4: SR-$A^*(G, s_i, g_i, C, P_{-i}, B)$.

```
1  TO ← plan2TimedObs(P_{-i});
2  Q ← {s_i};
3  while Q not empty do
4      c ← Q.highestPriority();
5      if c.loc = g_i then
6          return π_i ← c.Path()
7      else
8          Q.pop(c); N ← expand(c, G, C, B, TO) ;
9          for every n ∈ N do
10             if n.parent.gScore+1 ≤n.gScore then
11                 n.gScore ← n.parent.gScore+1;
12                 Q.add(n);
```

We note that careful consideration is needed in choosing a value for w. An intuition is that f_a (path length) is typically much greater than f_x (number of segments). Unless w is very large, f_a is dominant and f_x acts more like a tie-breaker. In Section 6, we empirically show how varying w changes the behavior of XG-CBS. Finally, note that WXG-A^* exhausts the same search space as XG-A^*, differing only in the order of the search. Therefore, Theorem 4 still holds for WXG-A^*, i.e., WXG-A^* is complete.

4.2.3.3. SR-A^* – segmentation respecting A^* While WXG-A^* can theoretically provide a good balance (trade-off) between efficiency and index minimization, it suffers from two drawbacks. First, it is difficult to choose an appropriate weight w *a priori* to achieve a good balance, since it is highly instance dependent. Second, WXG-A^* needs to maintain the history of the path (as in XG-A^*) in order to perform segmentation, resulting in a slow search algorithm. We propose a new low-level algorithm that does not keep track of history, thus obtaining a significant speedup.

Recall from Remark 7 that XG-A^* computes paths that fit within the existing segmentation of P_{-1} by keeping track of the index of P_{-1} combined with the new path, which requires keeping the history of the path from the last segmentation point. A coarse way of eliminating the need to keep the history is to make sure the planned path completely avoids all paths in P_{-1}, and so does not contribute to segmentation. This, however, likely results in no plans being found, as it amounts to keeping the agents disjoint. Our proposed algorithm, dubbed *segmentation-respecting* A^* (SR-A^*), refines this idea, by making sure that the planned path is disjoint from all paths *within the current segment*. The algorithm is shown in Algorithm 4. Intuitively, SR-A^* treats every disjoint segment within P_{-1} as time dependent obstacles. That is, existing paths within a segment become obstacles only for the time window of the segment. The resulting behavior is an efficient graph search algorithm that is dedicated to fitting within an existing segmentation.

Formally, consider a plan P_{-1} with a disjoint decomposition $t_0 < t_1 < \ldots < t_r$, and a planning query for Agent 1. The search space is now modified by adding a "timed obstacle" at vertex v at time t as follows. Let $1 \leq i \leq r$ be the segment such that $t_i \leq t \leq t_{i+1}$, then we add a timed obstacle if there is a path of P_{-1} that visits vertex v at the interval $[t_i, t_{i+1}]$. For example, if P_{-1} contains the segment v_1, v_2, v_3 at times $3, 4, 5$, respectively, then vertices v_1, v_2 and v_3 are all obstacles at times $[3, 5]$. This procedure results in a set of timed obstacles (Line 1 of Algorithm 4). The obstacles are checked against during the expand procedure (Line 8 of Algorithm 4).

Observe that crucially, if Agent 1 does not intersect with any timed obstacle, then it also does not create new segments, and hence "respects" the segmentation of P_{-1}. In particular, SR-A^* breaks the completeness of XG-CBS. Indeed, the restriction of the search space means that some paths are never explored. From an efficiency perspective, however, SR-A^* both limits the search space, and eliminates the tracking of history, rendering this search comparable to A^*. In Section 6, we demonstrate that SR-A^* performs exceedingly well, both in terms of efficiency and plan index.

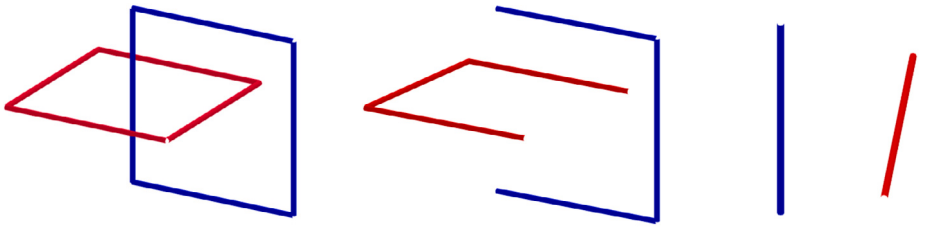
5. The 3D setting

Disjoint decompositions are defined with respect to paths in an arbitrary graph, and indeed, their graph-theoretic study in Sections 2, 3, and 4 makes no assumptions on the underlying graph. Recall that part of our initial motivation, however, asserts that disjoint decompositions can be easily visualized as non-intersecting paths. This is only true for planar graphs drawn on a plane. While planarity is a natural assumption when dealing with agents that move on a plane, in the case of agents that move in a 3D domain (e.g., UAV's, aircraft, etc.), there is no reason for disjoint paths to seem non-intersecting when viewed as a 2D "image" from a certain viewpoint. We demonstrate this issue in Fig. 5(a).

Thus, in order to maintain our motivation of explainability using a small number of "images" (i.e., 2D projections), we refine our notion of disjoint decompositions by including a viewpoint for each segment, such that the projection of the paths on the viewpoint yields non-intersecting segments.

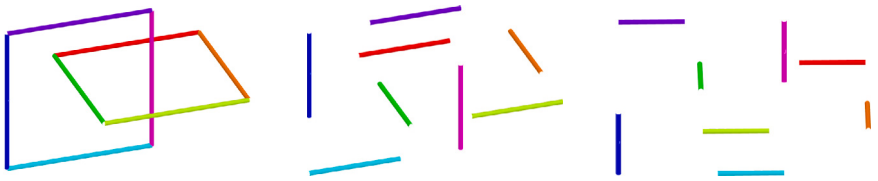
Before delving into the details, we remark that, as it will shortly be evident, the analysis of the 3D setting requires mathematical machinery from algebraic geometry (e.g., projections, linear operators, and quantifier elimination for the first order theory of the reals) which was absent from the 2D analysis. The reason for this additional mathematical difficulty is that in the 2D setting, the definition of explanations is purely graph-theoretic, and is therefore analyzed using discrete, graph-related techniques. In the 3D setting, however, we also consider the embedding of the graph in $\mathbb{R}^3$, which then inherently calls for tools to reason about $\mathbb{R}^3$.

We start with some definitions. Given a unit vector $w \in \mathbb{R}^3$ (i.e., $|w| = 1$), the *plane normal to* w is the set $\mathcal{H}_w = \{x \in \mathbb{R}^3 \ : \ x^\intercal w = 0\}$. Given a point $z \in \mathbb{R}^3$, the *projection* of z on the plane $\mathcal{H}_w$ is given by $\mathsf{Proj}(z, \mathcal{H}_w) = z - (z^\intercal w) \cdot w$. This is lifted to a set $S \subseteq \mathbb{R}^3$



(a) Full plan (b) Part 1: time 0–2 (c) Part 2: time 2–3

Fig. 5. Two non-intersecting paths for agents on a 3D grid. The blue agent's path is $(0,0,0)-(0,2,0)-(2,2,0)-(2,0,0)-(0,0,0)$ and the red agent's path is $(1,1,1)-(1,-1,1)-(1,-1,-1)-(1,1,-1)-(1,1,1)$. The paths are clearly non-intersecting, but from any viewpoint, the projected image does have intersecting lines. The two decomposition parts have slightly different viewpoints (hence the different apparent "slope" of the red segment).



(a) Full plan (b) Part 1: time 0–0.6 (c) Part 2: time 0.6–1

Fig. 6. Paths for 8 agents, taking one time unit, and a respective decomposition at time 0.6.

by $\mathsf{Proj}(S,\mathcal{H}_w)=\{\mathsf{Proj}(z,\mathcal{H}_w)\ :\ z\in S\}$. Projection is a linear operator (indeed, it is easy to verify that $\mathsf{Proj}(z,\mathcal{H}_w)=(I-ww^{\intercal})z$). Thus, for a segment $L=\{\alpha u+(1-\alpha)v\ :\ \alpha\in[0,1]\}$ we have $\mathsf{Proj}(L,\mathcal{H}_w)=\{\alpha\mathsf{Proj}(u,\mathcal{H}_w)+(1-\alpha)\mathsf{Proj}(v)\ :\ \alpha\in[0,1]\}$.

Consider a graph $G=\langle V,E\rangle$ embedded in $\mathbb{R}^3$. That is, $V\subseteq\mathbb{R}^3$ and with $(u,v)\in E$ we associate the segment $L_{(u,v)}=\{\alpha u+(1-\alpha)v\ :\ \alpha\in[0,1]\}$.

Let $P=\{\pi_1,\ldots,\pi_k\}$ be a set of non-colliding paths of length T. We treat each path as a continuous piecewise-linear trajectory $\pi_1:[0,T]\to\mathbb{R}^3$, where traversing each edge corresponds takes length 1 (hence a path of length T corresponds to the domain $[0,T]$).

A *viewpoint-disjoint decomposition* of P consists of an ordered list ℓ of *real* numbers $0=t_0<t_1<\ldots<t_r=T$ and a list of unit vectors $w_1,w_2,\ldots,w_r$ such that for every $0<i\le r$ the projections $\{\mathsf{Proj}(\pi_j[t_{i-1},t_i),\mathcal{H}_{w_i})\ :\ 1\le j\le k\}$ are disjoint (where $\pi_j[a,b]=\{\pi_j(t)\ :\ a\le t\le b\}$).

Thus, a viewpoint-disjoint decomposition splits the trajectories into segments such that within each segment, the projection of the segment on its prescribed viewpoint is disjoint, and the viewpoint may change from one segment to another. We demonstrate a viewpoint-disjoint decomposition in Fig. 5.

Note that unlike the definition in Section 2, we do not require the segmentation to take place in natural times (i.e., on the vertices), but rather allow the segmentation to cut in "middle" of edges. As we now demonstrate, this is crucial.

Example 6. Consider the setting depicted in Fig. 6, where 8 agents are traversing one edge each, to form a similar shape as that of Fig. 5. Clearly there is no decomposition with natural time units, as the entire trajectory takes 1 time unit. However, it is possible to find a viewpoint-disjoint decomposition, by splitting at time 0.6, and using two different viewpoints.

Example 6 also shows that changing the viewpoint may make it difficult to convince that the images are indeed consecutive and follow the paths of the agents. One way to overcome this is by using a video transition to demonstrate the change in the viewpoint. We provide a demonstration video[6] of the decomposition of Fig. 6.

5.1. Algorithmic aspects of viewpoint disjoint decompositions

In the graph-theoretical setting of Section 3, a decomposition segment is "legal" iff the paths within the segment were disjoint. This property is algorithmically easy (indeed, almost trivial) to check. In the 3D setting, however, a decomposition segment must also admit a viewpoint from which the projection is disjoint. This, it turns out, is much harder to verify.

We thus start with the geometric problem of determining whether a suggested segment can serve as part of a viewpoint-disjoint decomposition.

Problem 5 *(3D Viewpoint Existence).* Given piecewise-linear trajectories $\pi_1,\ldots,\pi_k$ where $\pi_j:[0,T]\to\mathbb{R}^3$ for every $1\le j\le k$, determine whether there exists a unit vector $w\in\mathbb{R}^3$ such that $\{\mathsf{Proj}(\pi_j[0,T],\mathcal{H}_w)\ :\ 1\le j\le k\}$ are disjoint.

[6] https://youtu.be/hcfwJqeniMA.

Problem 5 is almost identical to the problem of finding a *crossing-free parallel projection*, which is shown to be solvable in polynomial time in [35]. The only difference between the problems is that in [35], the projection must be such that no two line segments intersect, whereas in our setting, line segments that belong to the same trajectory are allowed to intersect.

Fortunately, the techniques developed in [35] can be easily adapted to our setting, as follows. Consider a pair of non-intersecting line segments, then the set of viewpoints from which these lines appear to intesect forms a spherical quadrilateral, called a *forbidden quadrilateral*. The algorithm in [35] starts by computing the $O(n^2)$ forbidden quadrilaterals corresponding to each pair of segments, and then proceeds to check whether its union covers the entire sphere. Any point not covered by this union can serve as a viewpoint from which the segments are non intersecting. Thus, in our setting, the initial set of forbidden quadrilaterals is computed by only considering pairs of segments that do not belong to the same trajectory. Then, the algorithm proceeds without change.

As mentioned above, Problem 5 is merely a basic step in solving the general problem of finding a vertex disjoint decomposition:

Problem 6 *(3D Viewpoint Decomposition)*. Given piecewise-linear trajectories $\pi_1, \ldots, \pi_k$ where $\pi_j : [0,T] \to \mathbb{R}^3$ for every $1 \leq j \leq k$ and an index m, decide whether there exists a viewpoint-disjoint decomposition P of index at most m, and output one if exists.

In order to solve Problem 6, we intuitively need to solve Problem 5 for a set of trajectories parameterized by a variable $t \in [0,1]$ which represents the end of the first segment. To our knowledge, a parameterized version of the algorithm of [35] was not studied. Nonetheless, we are able to obtain a polynomial upper bound on a solution for Problem 6.

Theorem 5. *Problem 6 can be solved in polynomial time.*[7]

Proof. We start by showing how to find the maximal time $t_1 \in [0,1]$ such that the trajectories admit a disjoint viewpoint on the time interval $[0,t_1]$. Then, we can iteratively find t_2 by cropping the trajectories in time $[0,t_1]$, and so on. If at any point the number of segments exceeds m, the algorithm terminates. The correctness of this procedure follows the greedy approach presented in Theorem 1.

In order to find t_1, we describe it using a sentence in the first order theory of the reals, as follows.

Consider piecewise-linear trajectories $\pi_1, \ldots, \pi_k$ where $\pi_j : [0,T] \to \mathbb{R}^3$ for every $1 \leq j \leq k$. Given $t \in [0,T]$, we define the first-order formula:

$$\Phi(t) = \exists w \in \mathbb{R}^3 \ : \ \bigwedge_{1<i<j<k} \mathsf{Proj}(\pi_i[0,t], \mathcal{H}_w) \cap \mathsf{Proj}(\pi_j[0,t], \mathcal{H}_w) = \emptyset$$

Thus, $\Phi(t)$ is true iff there is a viewpoint w such that the projection on $\mathcal{H}_w$ of the trajectories up to time t is disjoint. We now consider the set

$$\mathcal{M} = \{t \in [0,T] \ : \ \Phi(t) \wedge \forall s > t, \neg\Phi(s)\}$$

Thus, $\mathcal{M}$ contains the unique number t_1 for which the segments up to t_1 admit a disjoint viewpoint, but no greater number satisfies this property. That is, t_1 is as required above.

In order to extract t_1 from the description of $\mathcal{M}$, observe that $\Phi(t) \wedge \forall s > t, \neg\Phi(s)$ is a quantified sentence over 5 variables (s, t, and the three entries of $w \in \mathbb{R}^3$) consisting of a polynomial number of linear equalities and inequalities (namely the explicit description of $\mathsf{Proj}(\pi_i[0,t], \mathcal{H}_w) \cap \mathsf{Proj}(\pi_j[0,t], \mathcal{H}_w) = \emptyset$). Indeed, this description consists only of linear equalities and inequalities, since trajectories are piecewise linear.

We can thus use Fourier-Motzkin Quantifier Elimination [36] to compute a point in $\mathcal{M}$, namely t_1. Moreover, the complexity of Fourier-Motzkin Elimination is $O(k^{2^n})$ where k is the number of linear constraints, and n is the number of variables. Since in our case we have $n = 5$, we conclude that the overall complexity is $O(k^{32})$ where k is the description length of the input. □

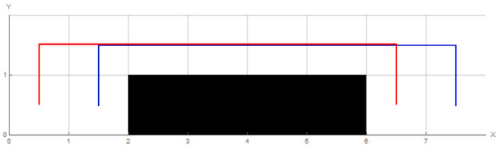
While Theorem 5 establishes a polynomial algorithm, in practice it might not perform well (as the degree of the polynomials involved is potentially 32). An alternative, approximate approach, is to take a discrete approximation, and search for t_1 e.g. using binary search, repeatedly solving Problem 5.

We remark that the general problem of planning with explanations in mind, the analogue of Problem 4, is clearly at least **NP-Hard**, and being harder than the 2D case, is still beyond our current reach for practical purposes.

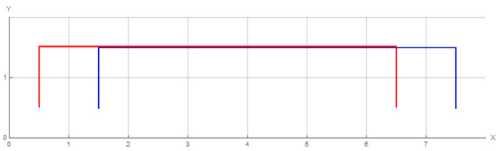
Remark 9. A dual notion of viewpoint-disjoint decomposition asks whether we can fix a single viewpoint $w \in \mathbb{R}^3$ such that the projection of the trajectories along w admits a disjoint decomposition with a certain number of segments.

Technically, the quantifier-elimination approach used in Theorem 5 can be adapted to this setting as well. However, this requires adding a variable t_i for each segment, thus rendering the complexity of the solution in exponential time. In future research, we will investigate the complexity of this version.

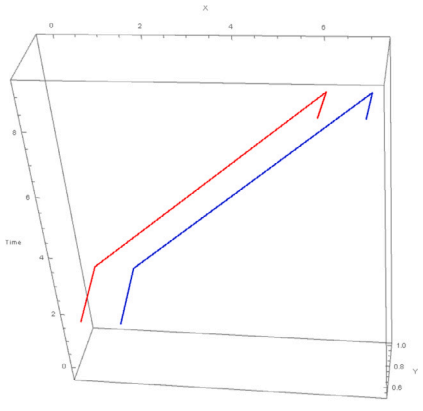
[7] We remark that the polynomial dependency is with respect to m (the segment bound) represented in unary.



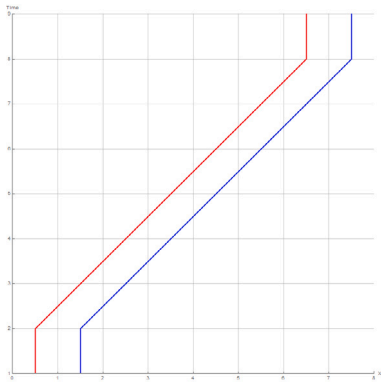
(a) Agents in a corridor



(b) Same plan without obstacles



(c) A 3D view



(d) Projection on (x, time) plane

Fig. 7. A non-colliding plan for two agents in a corridor.

5.2. *Projection along spatial dimensions*

A surprising application of the 3D setting, and in particular of Theorem 5 is using 3D viewpoints in order to reduce the number of segments required for explaining 2D plans. Intuitively, the idea is as follows. In the 2D setting, we think of trajectories as functions $\pi_i : [0,1] \to \mathbb{R}^2$, and two trajectories π_i, π_j collide if there exists $t \in [0,1]$ such $\pi_i(t) = \pi_j(t)$. Thus, π_i and π_j do not collide iff their *graphs* are disjoint, where the graph of π is $\Gamma(\pi_i) = \{(\pi_i(t), t) : t \in [0,1]\} \subseteq \mathbb{R}^3$. Thus, we can witness the fact that paths do not collide using a viewpoint from which the graphs of the trajectories are disjoint (if such a point exists). The tools developed in Theorem 5 allow us to find such a viewpoint.

On a conceptual level, in the 2D setting we always project the plan on the (x, y) plane (i.e., along the time axis), whereas now we allow projecting along any axis. We demonstrate this with an example.

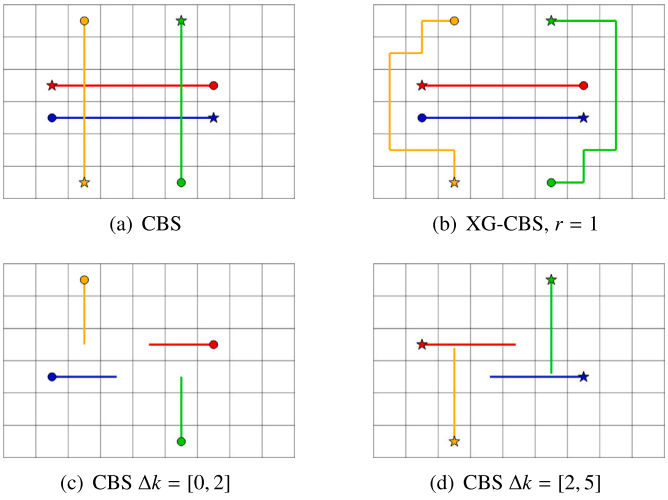
Example 7. Consider the environment depicted in Fig. 7(a), where two agents need to go through a corridor.

As we mention in [1], corridor settings give a notoriously bad tradeoff between efficiency and explainability: the quickest plan, where one agent remains a step behind the other, requires a segment for each step. Hence, the minimal explanation for this plan has 7 segments. For clarity, we show the same plan without the obstacles, in Fig. 7(b).

By rendering this plan in 3D (Fig. 7(c)), we now get two disjoint paths (since the agents do not collide). Moreover, notice that the agents never arrive at the same x coordinate at the same time. Thus, if we project along the y axis, we obtain a 2D image whose axes are the x coordinate and the time coordinate (Fig. 7(d)), in which the paths are disjoint. We thus obtain a single-image explanation for the plan.

Formally, the setting applies also to the 3D setting. We formalize it uniformly as follows. Consider a trajectory $\pi_i : [0,1] \to \mathbb{R}^d$ (for $d \in \{2,3\}$), and its corresponding *graph* $\Gamma(\pi_i) = \{(\pi_i(t), t) : t \in [0,1]\} \subseteq \mathbb{R}^{d+1}$. Then, π_i, π_j do not collide iff $\Gamma(\pi_i) \cap \Gamma(\pi_j) = \emptyset$, so we have reduced the problem of showing that two paths do not collide to the problem of showing that their graphs are disjoint, in a higher dimension.

Unfortunately any attempt at solving the 3D problem is computationally intractable even for the simplest of instances. Thus, we do not present any algorithmic solution to Problem 6.



(a) CBS (b) XG-CBS, $r = 1$

(c) CBS $\Delta k = [0, 2]$ (d) CBS $\Delta k = [2, 5]$

Fig. 8. Road crossing: solutions via CBS and XG-CBS.

6. Experimental results

In this section, we present a series of case studies to evaluate the proposed explanation scheme and algorithms in the 2D setting. We first illustrate the efficacy of the proposed explanation scheme and trade-off it causes between the length of plans and their minimal decomposition on some self-designed problems. The self-designed spaces exhibit interesting behaviors that are unique to the Explainable MAPF problem. Next, we evaluate the performance of XG-CBS on standard MAPF benchmark problems available in [37]. The results show the advantages and disadvantages of the proposed planning algorithms in various environments and scenarios.

Our implementation of the algorithms is in C++ and is available on GitHub [38]. The results presented below were obtained on a machine with an AMD Ryzen 7 3.9 GHz CPU and 64 GB of RAM.

6.1. Illustrative examples

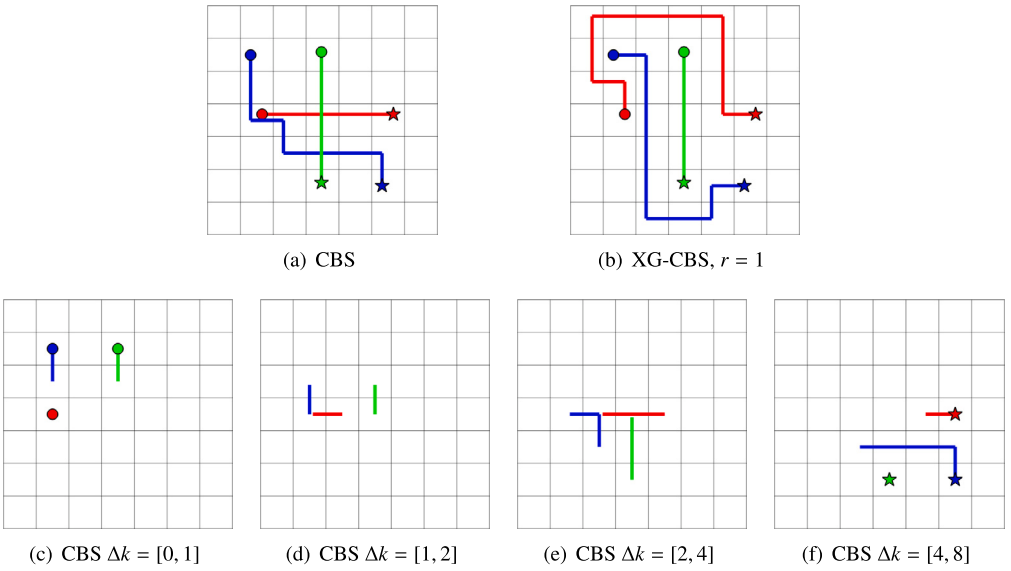
To gain insight into the effectiveness of our explanation method and how the solutions to the Explainable MAPF differ from the classical MAPF, here, we showcase a few illuminating examples.

Fig. 8(a) shows a CBS solution of MAPF, where four agents need to cross an intersection. Abstracting away the time dimension altogether makes it impossible to verify that the plan is collision free. However, using our explanation scheme, which decomposes the plan into two disjoint segments in Fig. 8(c) and 8(d), this becomes easy. We can further simplify this verification by using our explainable planner XG-CBS. Fig. 8(b) shows the plan obtained by XG-CBS with index 1. This example also demonstrates the trade-off between plan length and explanations: the shortest plan requires index 2, while index 1 can be achieved with a longer plan. Performance-wise, XG-CBS with A^*, as proposed in Section 4.2.2, timed out after a 15 minute threshold, whereas XG-CBS with XG-A^* arrived at an index-1 solution in 0.05 seconds. This difference can be attributed to the facts that the set of index-1 plans is comparatively sparse in the set of plans, and that index-1 plans greatly deviate from the shortest plan. These factors have a significant effect on the efficacy of each algorithm. We further discuss this in the next section. Since this setting is relatively small, a coupled approach using A^* was also able to obtain an index-1 plan. This, however, took 239 seconds – about 4 orders of magnitude worse than XG-CBS with XG-A^*.

Our next use case is depicted in Fig. 9(a). One may notice a possible collision between the red and green agents. However, it becomes clear in the explanation (Figs. 9(c)-9(f)) that the red agent does, in fact, wait at the first vertex, thus avoiding collision. An improved explanation can be obtained using XG-CBS with XG-A^* as shown in Fig. 9(b). This solution was obtained in 0.5 seconds, whereas XG-CBS with A^* again timed out. Again, since this is a small setting with few agents, coupled A^* was able to find an index-1 plan in 49 seconds – two orders of magnitude worse than XG-CBS with XG-A^*.

In Fig. 10 we see one example where XG-CBS with classical A^* performs significantly better than XG-CBS with XG-A^*. Here, XG-CBS with XG-A^* does not return a 2-segment solution within 15 minutes. However, XG-CBS with A^* returns the preferred solution in 0.44 seconds. Notice that the 6-segment plan returned by CBS (Fig. 10(a)) and the 2-segment plan returned by XG-CBS with A^* (Fig. 10(h)) look very similar. This is different from the above examples, where decreasing segments require a heavy deviation from the shortest paths for individual agents. Experiments show that in examples like these, where we can greatly simplify the explanation with minor tweaking of the shortest-path plan, XG-CBS with A^* outperforms XG-CBS with XG-A^*. We attribute this to the fact that using A^* quickly makes minor changes to the shortest paths while XG-A^* enables large deviations from shortest paths but suffers from computation time as a result. Finally, coupled A^* does manage to find an index-4 plan in 2.22 seconds, but times out when searching for an index-3 plan.

For more case studies with various environment sizes and agent numbers, we refer the reader to [38].



(a) CBS (b) XG-CBS, $r = 1$

(c) CBS $\Delta k = [0, 1]$ (d) CBS $\Delta k = [1, 2]$ (e) CBS $\Delta k = [2, 4]$ (f) CBS $\Delta k = [4, 8]$

Fig. 9. Apparent collision in short plan vs. optimal index.

6.2. Benchmark evaluations

Here, we evaluate the performance of our proposed Explainable MAPF algorithms introduced in Section 4. For the coupled algorithm, as described in Section 4.1, the exponential blowup both in the size of the graph and in the number of agents makes the algorithm prohibitively inefficient.

The results presented in Section 6.1 validate that this algorithm does not scale well. Thus, we focus on the comparison of the decoupled methods, namely XG-CBS with A*, XG-A*, WXG-A*, and SR-A* as the low-level search algorithms, on a large set of MAPF benchmarks from [37]. We also evaluate CBS as a baseline. The comparison is with respect to four metrics: *computation time*, *segmentation index*, *plan length* (average cost, i.e., sum-of-costs divided by number of agents), and *success rate* (percentage of the solved instances).

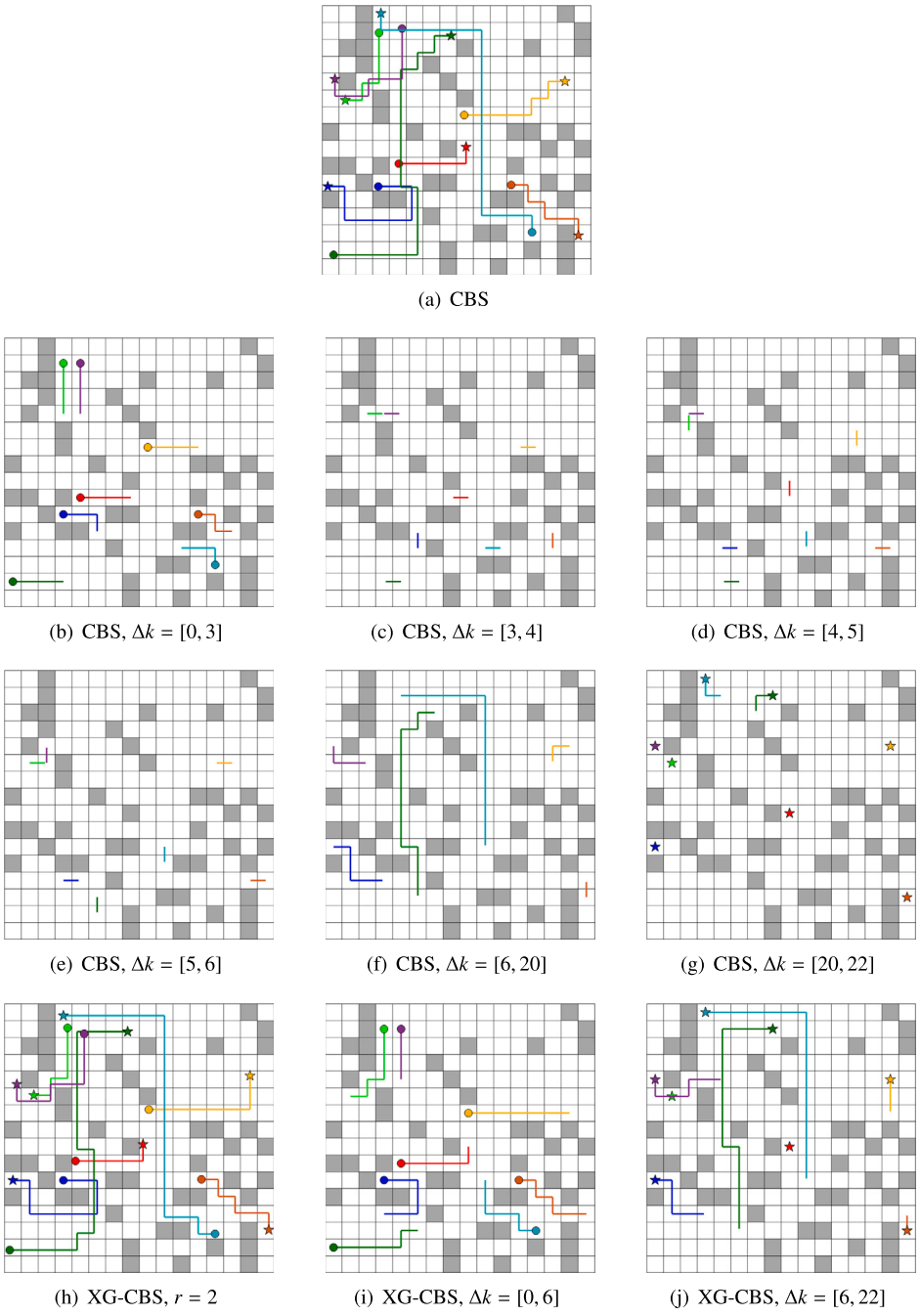
Our experiments are run as follows. For each benchmark, we run CBS. If CBS finds a plan, we segment it and use the index as an upper bound for XG-CBS. We then repeatedly lower the bound in XG-CBS, until it times out. We refer to the former result as *first* and to the latter as *best*. In case CBS does not terminate, we run XG-CBS with an initial bound of ∞. We remark that whenever an algorithm times out without a solution, we do not include this in the computation time.

Our benchmarks were on randomly generated MAPF instances of the following structure: 9×9 grid world with 12 obstacles and 4, 8, 10, and 12 agents, 16×16 grid world with 30 obstacles and 5, 10, 15, and 20 agents, and 33×33 with 200 obstacles and 10, 20, and 30 agents. For each map and agent number combination, we ran 100 unique experiments, where each experiment consisted of CBS, followed by XG-CBS with each of the low-level algorithms. The timeout for a single algorithm on a single benchmark was 5 minutes (while this may seem like a high threshold, recall that Explainable MAPF is computationally harder than MAPF, cf. Remark 4). We performed the aforementioned experimental set-up twice. Once using the sum-of-costs metric and another using the makespan cost metric. The results are presented as boxplots and barplots in Figs. 11, 12, and 13 (sum-of-costs) and Figs. 14, 15, and 16 (makespan). For 33×33 environments, XG-A^* and WXG-A^* nearly always time out, and hence not evaluated.

All six figures mentioned in the previous paragraph are divided into sub-plots based on the number of agents considered (see the vertical black lines that divide the plots). Within every sub-plot, every column denotes a particular algorithm. Note that the column names for every sup-plot are identical. Clear trends can be seen when looking at the sub-plots.

For the most part, the results match our expectations: vanilla CBS offers the best tradeoff between plan length and computation time (except for the 33×33 environment with 30 agents, which is discussed below), but invariably outputs plans with high index. Of the two extremities A^* and XG-A^*, the speed of A^* allows it to eventually find smaller index plans than XG-A^*, with comparable path length. However, XG-A^*, being guided towards minimal index plans, often outputs a lower index plan initially (cf., *first* column). Moreover, as the environment becomes smaller and more congested (8×8, 12 agents), XG-A^* outperforms A^*. Unfortunately, the history-dependence of XG-A^* means that it does not scale to larger environments and times out. In particular, this rules out the use of WXG-A^*, which is also history-dependent, for larger environments.

The surprising results come from SR-A^*. Despite being theoretically incomplete, in practice it offers an excellent performance. It has high success rate, matching CBS in most cases and even outperforming it in the 33×33 environment, and computes the smallest plan index in every benchmark problem. In most cases, it was able to reduce the index by a significant amount compared to CBS (e.g., in 33×33 with 30 agents, the reduction is from roughly 24 segments to 6 segments!). Moreover, its limited search space allows it to match CBS in computation time and sometimes even outperform it. The tradeoff naturally comes in the path length, which increases to allow small index plans.



(a) CBS

(b) CBS, $\Delta k = [0, 3]$

(c) CBS, $\Delta k = [3, 4]$

(d) CBS, $\Delta k = [4, 5]$

(e) CBS, $\Delta k = [5, 6]$

(f) CBS, $\Delta k = [6, 20]$

(g) CBS, $\Delta k = [20, 22]$

(h) XG-CBS, $r = 2$

(i) XG-CBS, $\Delta k = [0, 6]$

(j) XG-CBS, $\Delta k = [6, 22]$

Fig. 10. Example of XG-CBS with 2-segment solution vs 6-segment solution of CBS.

Another pleasant surprise comes from A^*, which has mostly similar computation times to CBS for the first solution and only slightly larger computation times for the best solution. In return, it always computes smaller index plans than CBS, even in larger environments (see Figs. 13 and 16). Another interesting observation is that, while we expected generally lower success rates, XG-CBS with A^* success rates are similar to CBS in the 9×9 environment (Figs. 11 and 14) and better than CBS in the 33×33 environment (Figs. 13 and 16). Moreover, since A^* uses the distance to the goal as a heuristic, the plans found by A^* are typically shorter than those of SR-A^* (but usually have a higher cost, since lowering the cost eventually causes it to time out).

On smaller environments, WXG-A^* sometimes finds smaller index plans than XG-A^* on the first try. In addition, it has a higher success rate (often higher than all other versions, including CBS) due to being guided in part by A^*. However, since finding the best weight parameter is instance-dependent (cf., Section 4.2.3.2), it is not clear whether batch experiments capture the performance of WXG-A^*.

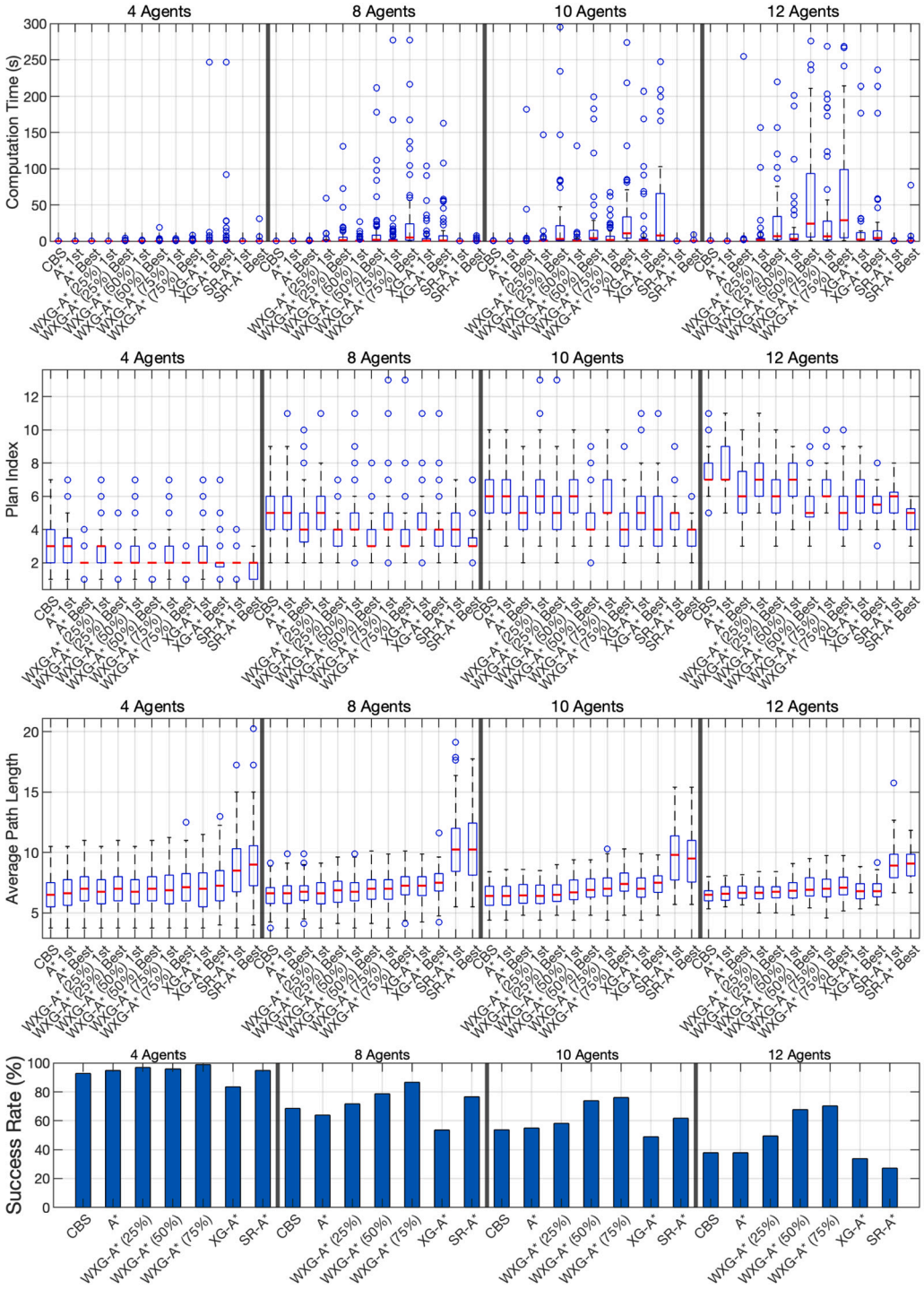


Fig. 11. Benchmark results for 9×9 environments using the sum-of-costs metric.

To summarize the results, on larger environments, if one wishes to optimize explainability, then SR-A^* is a clear winner. On smaller environments, A^* usually works fairly well, but XG-A^* can offer smaller index on congested environments. Finally, by carefully tuning a combined weight, one can obtain better explanations in small environments using WXG-A^*. It however remains an open question how to methodically find the optimal weight for WXG-A^* other than the trial-and-error method, which is far from ideal.

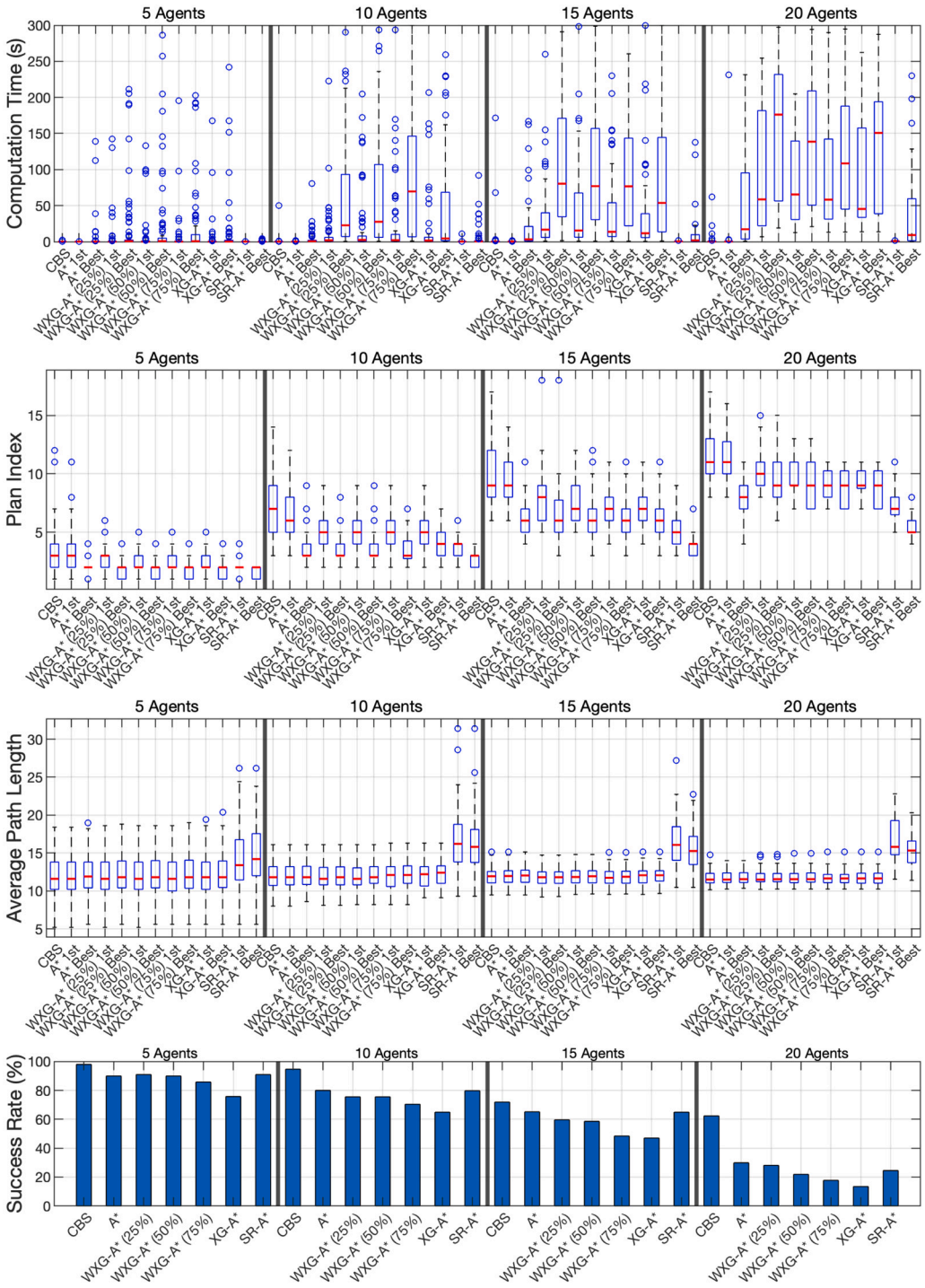


Fig. 12. Benchmark results for 16×16 environments using the sum-of-costs metric.

7. Final remarks and future work

In this work, we have laid down principles for devising an explanation scheme for algorithms. We demonstrate these principles in the context of MAPF, where we devise an explanation scheme that is sound and offers a tradeoff between completeness and simplicity. We have shown that in the two-dimensional setting, merely explaining the result of existing MAPF algorithms is tractable and, in fact, can be achieved greedily. This implies that our scheme can be readily incorporated in existing frameworks.

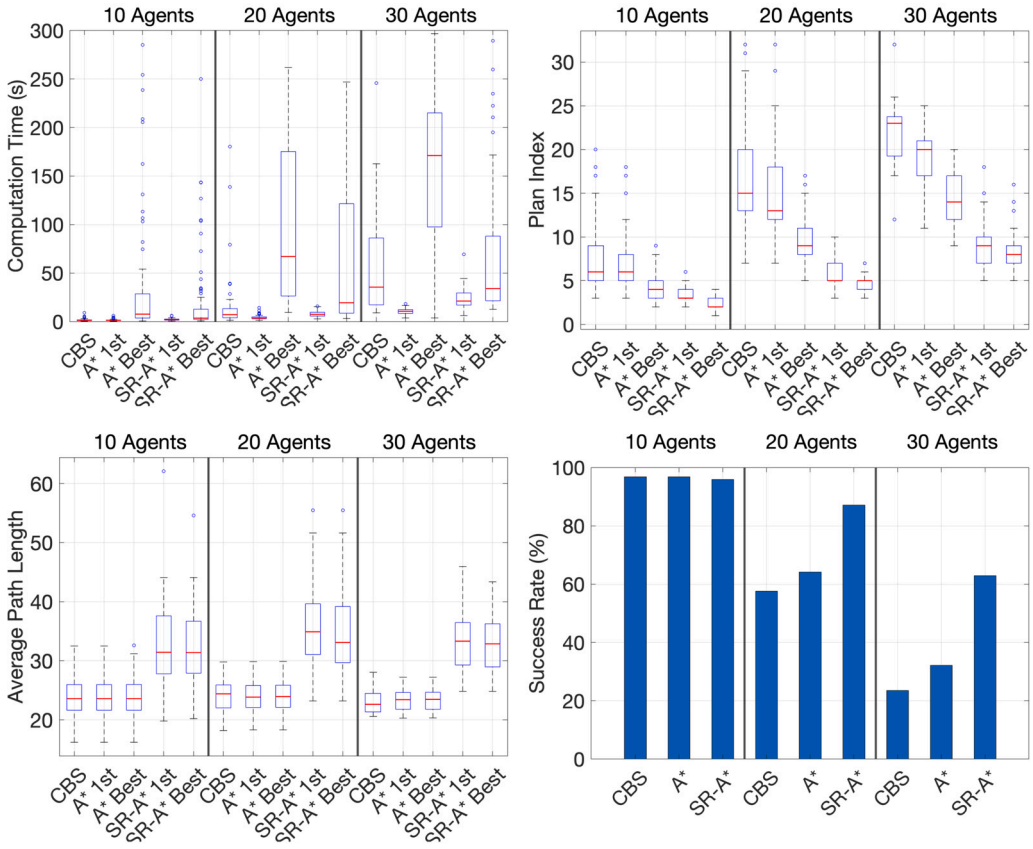


Fig. 13. Benchmark results for 33×33 environments using the sum-of-costs metric.

We have also shown that the "deeper" problem of finding plans that admit a short explanation (Explainable MAPF) is **NP-Complete**, and that solving it using search-based methods introduces a blowup in the state space that is not present in traditional MAPF. To this end, we introduced XG-CBS, a CBS-based decoupled algorithm for Explainable MAPF that is scalable. We also describe new low-level search algorithms, namely XG-A^*, WXG-A^* and SR-A^* that are oriented toward low index plans. While the former two yields a complete algorithm, our benchmark results show they do not generally perform well in large problems (large environments with large number of agents). WXG-A^* offers better scalability than XG-A^* when a "good" weight is picked, but finding that weight is non-trivial. SR-A^*, despite not being complete, scales well and is often equally and sometimes even more efficient than CBS, while yielding plans with a short explanation. In addition, we have illustrated a tradeoff between the length of a plan and the length of an explanation for it, meaning that longer plans usually have shorter explanations, and vice-versa.

In the three-dimensional setting, our explanation scheme is still theoretically valid, but the disjoint representations that were admitted in the two-dimensional setting are less relevant for visualization purposes. Hence, we consider an adaptation of the problem by adding viewpoints. We show that while finding a decomposition in this adaptation can be done in polynomial time, our solution requires some quantifier elimination, which currently renders it impractical. Nonetheless, the three-dimensional approach gives rise to new ways of performing segmentation, where instead of projecting the time axis, we allow projection also along spatial dimensions.

Finally, we remark that Explainable MAPF has potential applications beyond visualization and minimization of a description of the plan. Indeed, disjoint decompositions can be used during the actual execution of the plan, in case the agents' paths must not cross. For example in tethered robotics (e.g., tethered UAVs), we wish to minimize tangling of the tethers as they constrain robot motion. This essentially means we desire plans with a small index, which XG-CBS allows to achieve. Similarly, in applications of MAPF to 3D pipe routing [39], small segmentations may allow for simpler routing. Other applications can be found in multi-layered circuit board design. In particular, such applications show that even the reduction of one segment from the index may have beneficial financial applications, which may be significant.

7.1. Future work

In future research, we plan to adapt other MAPF algorithms to the Explainable settings, such as Priority-Based Search [9], and SAT-based solutions. The latter specifically poses interesting challenges because it first requires an encoding of the problem as a SAT or CSP instance and then using an existing solver [40–43]. The crux of these methods lies in clever encodings of the problem, often relying on domain-specific encoding. In the explainable setting, the additional difficulty is that the constraint imposed on

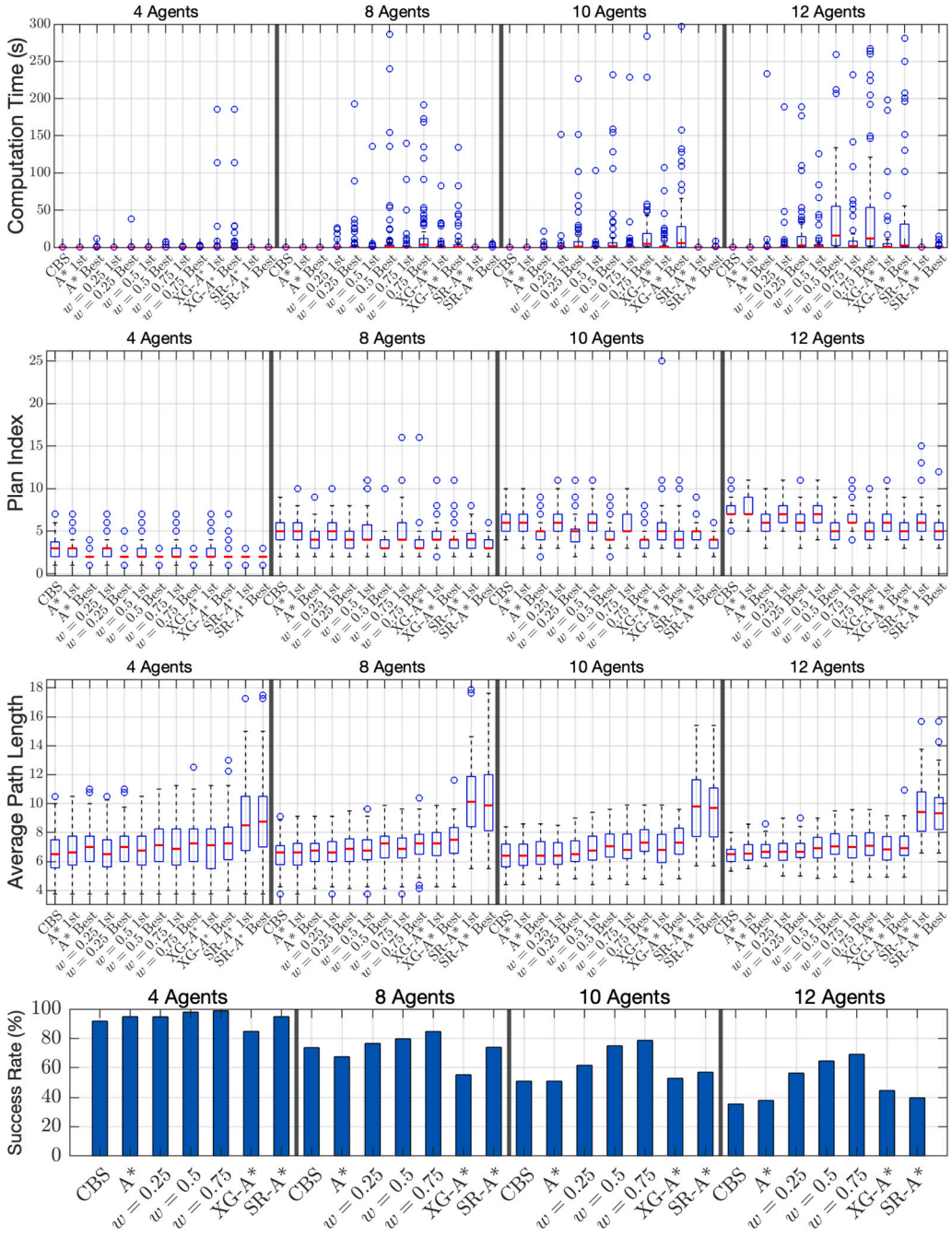


Fig. 14. Benchmark results for 9×9 environments using the makespan cost metric.

each segment in the decomposition should state that the paths are disjoint within the segment. This involves quadratically many statements in the length of the segment (i.e., no two agents are on the same vertex at any time in the segment). Unlike the blowup in the number of agents (which is typically small, albeit significant), a blowup in the size of the environment is potentially much bigger. This relates to the fact that Problem 4 remains **NP-Hard** even for two agents (Lemma 1).

Another future direction is to add explainability as an optimization goal for search-based planners, and use adaptations of A* for multi-objective search, such as focal search or double-queue A*. Naturally, this would require further optimizations, but analysis of these approaches can show what new bottlenecks arise in the presence of explainability.

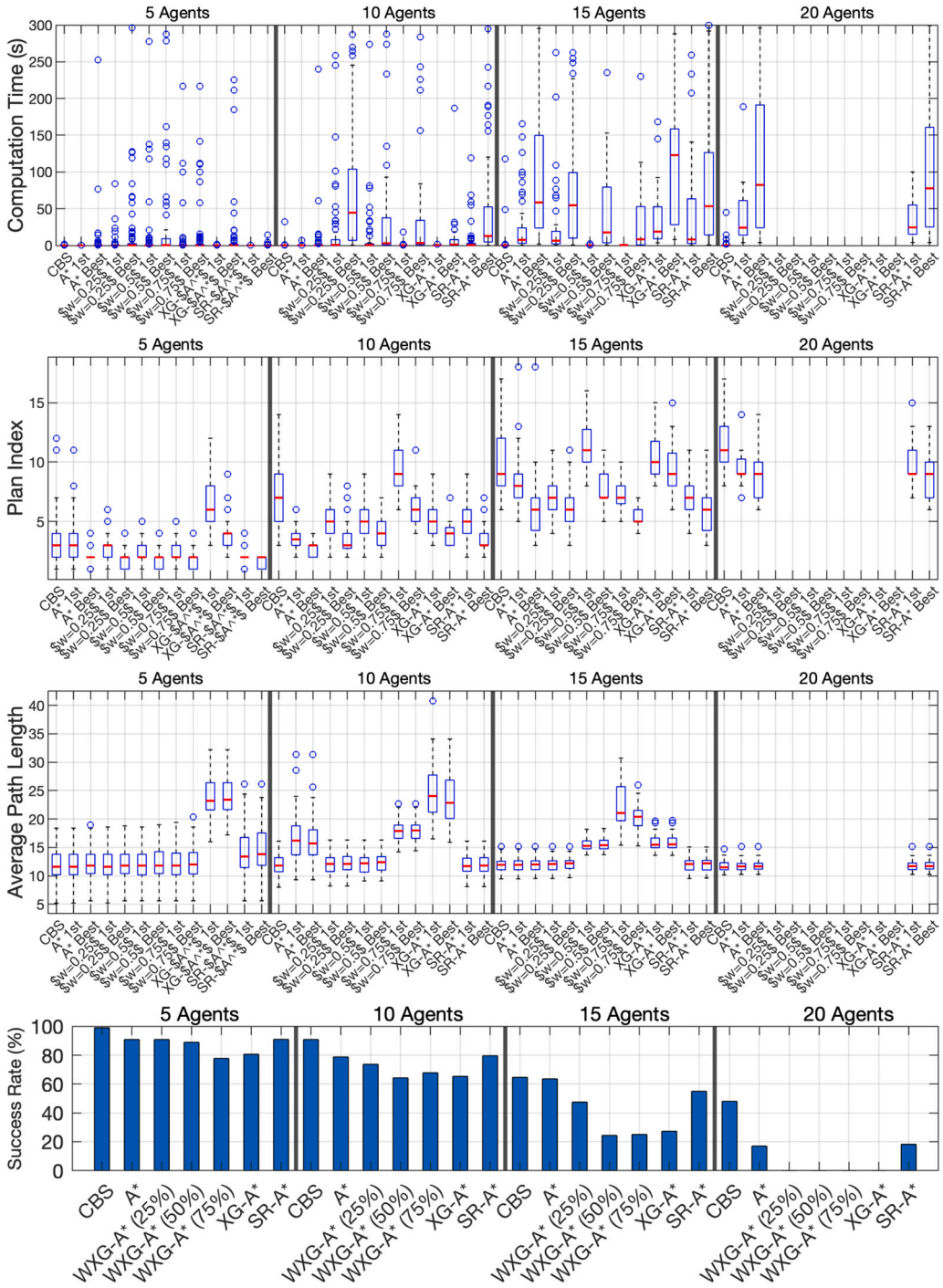


Fig. 15. Benchmark results for 16×16 environments using the makespan cost metric.

CRediT authorship contribution statement

Shaull Almagor: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Justin Kottinger:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Morteza Lahijanian:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

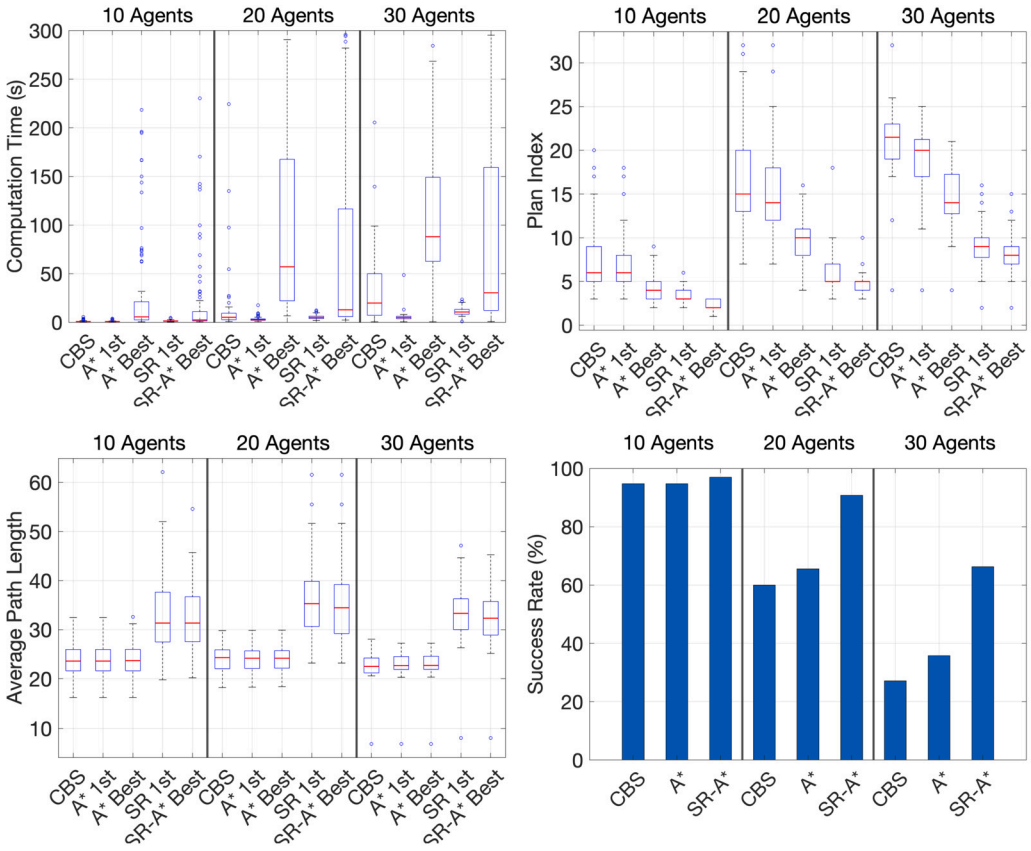


Fig. 16. Benchmark results for 33×33 environment using the makespan cost metric.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data shown in our results section can be made available upon request. The code used to reproduce the data is already publicly available and cited in the manuscript.

Acknowledgements

S. Almagor has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327. J. Kottinger and M. Lahijanian have received funding from the University of Colorado Boulder.

References

[1] S. Almagor, M. Lahijanian, Explainable multi agent path finding, in: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS'20, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 34–42, https://dl.acm.org/doi/abs/10.5555/3398761.3398771.

[2] J. Kottinger, S. Almagor, M. Lahijanian, Conflict-based search for explainable multi-agent path finding, in: Int'l Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2022.

[3] R. Stern, N.R. Sturtevant, D. Atzmon, T. Walker, J. Li, L. Cohen, H. Ma, T.K.S. Kumar, A. Felner, S. Koenig, Multi-agent pathfinding: Definitions, variants, and benchmarks, Symposium on Combinatorial Search (SoCS), 2019, pp. 151–158.

[4] T.S. Standley, Finding optimal solutions to cooperative pathfinding problems, in: Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.

[5] A. Felner, R. Stern, E. Shimony, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, P. Surynek, Search-based optimal solvers for the multi-agent pathfinding problem: summary and challenges, in: Proceedings of the Symposium on Combinatorial Search (SoCS), 2017, pp. 28–37.

[6] P. Surynek, A. Felner, R. Stern, E. Boyarski, An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives, in: Proceedings of the Symposium on Combinatorial Search (SoCS), 2016, pp. 145–147.

[7] R. Bartak, J. Svancara, M. Vlk, A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2018, pp. 748–756.
[8] L. Cohen, S. Koenig, S. Kumar, G. Wagner, H. Choset, D. Chan, N. Sturtevant, Rapid randomized restarts for multi-agent path finding: preliminary results, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2018, pp. 1909–1911.
[9] H. Ma, D. Harabor, P. Stuckey, J. Li, S. Koenig, Searching with consistent prioritization for multi-agent path finding, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2019 (in print).
[10] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66, https://doi.org/10.1016/j.artint.2014.11.006, https://www.sciencedirect.com/science/article/pii/S0004370214001386.
[11] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, E. Shimony, Icbs: improved conflict-based search algorithm for multi-agent pathfinding, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
[12] J. Li, D. Harabor, P.J. Stuckey, A. Felner, H. Ma, S. Koenig, Disjoint splitting for multi-agent path finding with conflict-based search, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 29, 2019, pp. 279–283.
[13] J. Li, A. Felner, E. Boyarski, H. Ma, S. Koenig, Improved heuristics for multi-agent path finding with conflict-based search, in: IJCAI, vol. 2019, 2019, pp. 442–449.
[14] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T.S. Kumar, S. Koenig, Adding heuristics to conflict-based search for multi-agent path finding, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 28, 2018.
[15] J. McMahon, N. Ahmed, M. Lahijanian, P. Amorese, T. Deka, K. Muvvala, T. Slack, S. Wakayama, Expert-informed autonomous science planning for in-situ observations and discoveries, in: 2022 IEEE Aerospace Conference (AERO), IEEE, 2022, pp. 1–11.
[16] D.H. Hubel, T.N. Wiesel, Receptive fields of single neurones in the cat's striate cortex, J. Physiol. 148 (1959) 574–591.
[17] S. Tang, T.S. Lee, M. Li, Y. Zhang, Y. Xu, F. Liu, B. Teo, H. Jiang, Complex pattern selectivity in macaque primary visual cortex revealed by large-scale two-photon imaging, Curr. Biol. 28 (2018) 38–48.
[18] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, K.-R. Müller, Unmasking clever hans predictors and assessing what machines really learn, Nat. Commun. 10 (2019) 1096.
[19] S. Almagor, D. Chistikov, J. Ouaknine, J. Worrell, O-minimal invariants for linear loops, in: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
[20] D. Gunning, Explainable Artificial Intelligence (XAI) DARPA-BAA-16-53, Technical report, Defense Advanced Research Projects Agency (DARPA), 2016.
[21] F.K. Došilović, M. Brčić, N. Hlupić, Explainable artificial intelligence: a survey, in: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, 2018, pp. 0210–0215.
[22] R. Eifler, M. Cashmore, J. Hoffmann, D. Magazzeni, M. Steinmetz, Explaining the space of plans through plan-property dependencies, in: Proceedings of the 2nd Workshop on Explainable Planning (XAIP 2019), 2019.
[23] M. Brandao, D. Magazzeni, Explaining plans at scale: scalable path planning explanations in navigation meshes using inverse optimization, in: Workshop on XAI, IJCAI-PRICAI, 2020.
[24] M. Brandao, G. Canal, S. Krivić, D. Magazzeni, Towards providing explanations for robot motion planning, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 3927–3933.
[25] S. Kambhampati, Synthesizing explainable behavior for human-ai collaboration, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1–2.
[26] M. Fox, D. Long, D. Magazzeni, Explainable planning, in: Explainable Planning, International Joint Conferences on Artificial Intelligence, 2017.
[27] A. Bogatarkan, E. Erdem, Explanation generation for multi-modal multi-agent path finding with optimal resource utilization using answer set programming, Theory Pract. Log. Program. 20 (2020) 974–989, https://doi.org/10.1017/S1471068420000320.
[28] M. Brandao, G. Canal, S. Krivić, P. Luff, A. Coles, How experts explain motion planner output: a preliminary user-study to inform the design of explainable planners, in: 2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN), IEEE, 2021, pp. 299–306.
[29] M. Middendorf, F. Pfeiffer, On the complexity of the disjoint paths problem, Combinatorica 13 (1993) 97–107.
[30] B.A. Reed, N. Robertson, A. Schrijver, P.D. Seymour, Finding disjoint trees in planar graphs in linear time, Contemp. Math. 147 (1993) 295.
[31] D.K.G.M.P. Spiralris, Coordinating Pebble Motion O_N Graphs, the Diameter of Permutation Groups, and Applications, 1984.
[32] L. Cohen, S. Koenig, Bounded suboptimal multi-agent path finding using highways, in: IJCAI, 2016, pp. 3978–3979.
[33] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. Sturtevant, J. Schaeffer, R. Holte, Partial-expansion a* with selective node generation, in: Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012.
[34] M. Goldenberg, A. Felner, R. Stern, G. Sharon, J. Schaeffer, A* variants for optimal multi-agent pathfinding, in: Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012.
[35] P. Bose, P. Ramos, F. Gomez, G. Toussaint, Drawing nice projections of objects in space, in: International Symposium on Graph Drawing, Springer, 1995, pp. 52–63.
[36] J.B.J. Fourier, Solution d'une question particuliere du calcul des inégalités, Nouv. Bull. Sci. Soc. Philomat. Paris 99 (1826) 100.
[37] A. Bose, I. Markelov, Multi-agent path planning in python, https://github.com/atb033/multi_agent_path_planning, 2019.
[38] J. Kottinger, Explanation-guided conflict-based search for explainable mapf, https://github.com/aria-systems-group/Explainable-CBS, 2021.
[39] G. Belov, W. Du, M.G. De La Banda, D. Harabor, S. Koenig, X. Wei, From multi-agent pathfinding to 3d pipe routing, in: Thirteenth Annual Symposium on Combinatorial Search, 2020.
[40] P. Surynek, Towards optimal cooperative path planning in hard setups through satisfiability solving, in: Pacific Rim International Conference on Artificial Intelligence, Springer, 2012, pp. 564–576.
[41] P. Surynek, A. Felner, R. Stern, E. Boyarski, Boolean satisfiability approach to optimal multi-agent path finding under the sum of costs objective, in: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1435–1436.
[42] J. Wang, J. Li, H. Ma, S. Koenig, T. Kumar, A new constraint satisfaction perspective on multi-agent path finding: preliminary results, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2253–2255.
[43] P. Surynek, J. Švancara, A. Felner, E. Boyarski, Integration of independence detection into sat-based optimal multi-agent path finding-a novel sat-based optimal mapf solver, in: International Conference on Agents and Artificial Intelligence, vol. 2, SCITEPRESS, 2017, pp. 85–95.