Conflict-Based Search for Explainable Multi-Agent Path Finding

Justin Kottinger[1], Shaull Almagor[2], Morteza Lahijanian[1,3]

[1]Department of Aerospace Engineering Sciences, University of Colorado Boulder, USA
[2]The Henry and Marilyn Taub Faculty of Computer Science, Technion, Israel
[3]Department of Computer Science, University of Colorado Boulder, USA
{justin.kottinger, morteza.lahijanian}@colorado.edu, shaull@cs.technion.ac.il

Abstract

The goal of the Multi-Agent Path Finding (MAPF) problem is to find non-colliding paths for agents in an environment, such that each agent reaches its goal from its initial location. In safety-critical applications, a human supervisor may want to verify that the plan is indeed collision-free. To this end, a recent work introduces a notion of explainability for MAPF based on a visualization of the plan as a short sequence of images representing time segments, where in each time segment the trajectories of the agents are disjoint. Then, the problem of *Explainable MAPF via Segmentation* asks for a set of non-colliding paths that admit a short-enough explanation. Explainable MAPF adds a new difficulty to MAPF, in that it is NP-hard with respect to the size of the environment, and not just the number of agents. Thus, traditional MAPF algorithms are not equipped to directly handle Explainable MAPF. In this work, we adapt Conflict Based Search (CBS), a well-studied algorithm for MAPF, to handle Explainable MAPF. We show how to add explainability constraints on top of the standard CBS tree and its underlying A^* search. We examine the usefulness of this approach and, in particular, the trade-off between planning time and explainability.

Introduction

Multi-Agent Path Finding (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 agents are involved, such as air-traffic control, UAVs, warehouse robots, autonomous cars, etc. While MAPF is generally intractable, the importance of this problem has generated a significant body of work over the past decade (Stern et al. 2019; Standley 2010; Felner et al. 2017; Surynek et al. 2016; Bartak, Svancara, and Vlk 2018; Cohen et al. 2018; Ma et al. 2019a), 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)* (Sharon et al. 2015), which is a decentralized approach, and has extensions with various heuristics (Boyarski et al. 2015; Li et al. 2019b,a; Felner et al. 2018).
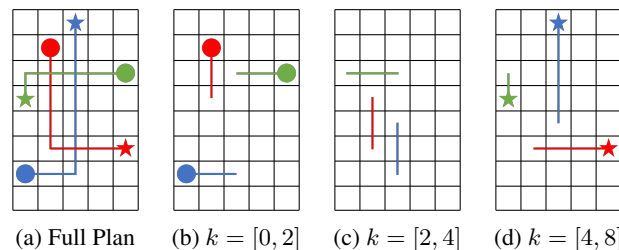
Figure 1: A plan for three agents (a), and a corresponding explanation via disjoint decomposition (b)-(d). The circles and stars mark the start and goal vertices, respectively.

(a) Full Plan (b) $k = [0, 2]$ (c) $k = [2, 4]$ (d) $k = [4, 8]$

A major barrier in adopting such capable MAPF algorithms in *safety-critical* applications, as with many algorithms in AI, is *trust* (or lack thereof) between the designers of such algorithms, and their potential user. That is, in heavily-regulated applications (e.g., air-traffic control, hazardous-materials warehouses), automated planning has to be trusted before acting upon in order to maintain legal and ethical accountability. Designers can gain trust in their algorithms through studying, developing, and exhaustive testing. The same trust building methods may not be available for the user. To combat this dilemma in the context of MAPF, the current practice is to suggest a computed plan to a human supervisor, who has to verify the correctness of it to allow its execution (Fines, Sharpanskykh, and Vert 2020). This poses an additional problem in MAPF – *explainability* of plans. In other words, plans must be presented to the supervisor in some humanly-understandable manner. In particular, the presentation (explanation) should enable the supervisor to understand the paths taken by the agents, and to easily verify that the agents do not collide, as otherwise the supervisor would not necessarily approve the plan. This study focuses on the problem of Explainable MAPF and aims to develop a scalable algorithm whose solutions to the MAPF problem are easily-interpretable and verifiable by humans.

Explainable AI (XAI) is an active area of research in recent years. Many studies focus on explaining decisions made by machine learning algorithms, in both categories of classification and regression (Arrieta et al. 2020). In those works, various forms of explanations are explored, but visual expla-

nations seem to be dominant for their ease of interpretability, especially for classifiers (Lapuschkin et al. 2019). In other cases, such as fault detection algorithms, explanations are typically in the form of witness executions (Mari, Dang, and Gössler 2021). In the planning community, explanations are mostly studied in the context of the single agent problem, and they often take a non-visual form. For example, explanations are given based on alternative plans (Eifler et al. 2019), minimal differences between plans (Kambhampati 2019), or reasoning on quantitative advantage of one plan over another (Fox, Long, and Magazzeni 2017). None of these studies, however, focus on the MAPF problem.

Recent works (Almagor and Lahijanian 2020; Kottinger, Almagor, and Lahijanian 2021) propose an explanation scheme for MAPF by means of visualization. There, the idea is to decompose a non-colliding plan into time segments, such that within each segment the paths of the agents are disjoint. Then, by depicting each segment separately (see Figure 1), it is easy for a human supervisor to verify that the agents do not collide. Indeed, recognizing line intersections takes place early in the visual cognitive process (Hubel and Wiesel 1959; Tang et al. 2018), making it easy to verify that the depicted lines in each segment are disjoint. The usefulness of such explanations is also supported by the findings of the survey study (Brandao et al. 2021). While decomposition can be readily used on any MAPF plan obtained by any algorithm, it is not guaranteed that doing so results in a small number of segments. In case the number of segments is very high, this undermines the explanation scheme. Thus, the central problem in *Explainable MAPF via Segmentation* is to find a plan for the agents that can be decomposed to a small number of segments (and hence can be explained with a small number of pictures).

Unfortunately, Explainable MAPF is much harder than standard MAPF, in the sense that, unlike MAPF, it is NP-hard already for two agents. In particular, the hardness of Explainable MAPF is with respect to the size of the environment (as well as the number of agents), rendering the runtime of algorithms for Explainable MAPF exponential in the size of the environment. In contrast, the complexity of classical MAPF is polynomial in size of the environment, making the problem much easier, especially with a low number of agents. Unsurprisingly, centralized algorithms for Explainable MAPF do not scale well, as shown in (Almagor and Lahijanian 2020).

In this work, we consider a decentralized approach to the Explainable MAPF problem. Specifically, we adapt CBS, a two-level algorithm that, in its low-level, plans individually for each agent, and in its high-level, identifies collisions between the agents and places constraints to resolve them in the next low-level iteration. Our *main contribution* is accordingly split to two levels: at the high-level, we show how we can use similar constraints as those used by CBS to capture *segmentation conflicts*, namely plans whose minimal decompositions have too many segments. We then discuss how to adapt CBS to compute and place these constraints during its search, thus obtaining our new algorithm, dubbed *Explanation-Guided CBS (XG-CBS)*.

We then turn our attention to the low-level planner of XG-

CBS. As discussed below, standard A^* seems, intuitively, ill-fitted to work with XG-CBS. Indeed, minimizing the number of disjoint segments of a plan often requires lengthening the plan, which A^* is reluctant to do since it minimizes the path length. Thus, at the low level, our contribution focuses on developing appropriate search algorithms, that are guided towards plans with small decompositions, which are appropriate for XG-CBS. To this end, we propose three low-level search algorithms. The first algorithm, dubbed XG-A^*, guides the search toward a plan with minimum number of disjoint segments, while maintaining the completeness of XG-CBS. Moreover, it can be combined with standard A^* to improve performance through a meta-parameter, resulting in the algorithm WXG-A^*.

We discuss how the optimal value for this parameter is highly dependant on the instance of the problem, and hence, difficult to choose *a priori*. Also, both XG-A^* and WXG-A^*, due to their completeness, are subject to the inherent difficulty of Explainable MAPF with respect to the environment size. This is manifested by the need to track the history of paths within the search space. To address these problems, we propose another low-level algorithm, SR-A^* that uses the segmentation information in a coarse way such that it does not need to track history, yet obtains solutions with small number of decomposition. Theoretically, SR-A^* sacrifices completeness, but our experimental results show that with SR-A^*, XG-CBS has comparable computation time to *vanilla* CBS (and even outperforms it), while obtaining plans with much smaller decompositions. This is despite solving a much harder problem.

Thus, our overall contribution is a decentralized algorithm for the Explainable MAPF problem. To the best of our knowledge, this is the first algorithm of its class that scales, significantly outperforming previous algorithms. We show properties of our algorithm and further evaluate it on many benchmarks, comprising examples that demonstrate specific intricacies of Explainable MAPF, as well as standard MAPF benchmarks. Overall, this work illustrates the unique computational challenges faced in Explainable MAPF and paves the way for further algorithmic exploration of this problem.

Problem Statement

Consider $n \in \mathbb{N}$ agents, acting in a directed graph $G = \langle V, E \rangle$ where each agent $i \in \{1, \ldots, n\}$ has a source $s_i \in V$ and a goal $g_i \in V$. A *path* in G is a sequence of vertices $\pi = v_1 v_2 \ldots v_m$ such that $(v_k, v_{k+1}) \in E$ for all $1 \leq k < m$.

Given paths $\pi_1 = v_1 v_2 \ldots v_m$ and $\pi_2 = u_1 u_2 \ldots u_m$ in G for some $m > 1$, we say that π_1 and π_2 are *non-colliding* if the following conditions are satisfied for all $1 \leq k < m$:

(i) $v_k \neq u_k$ (i.e., no vertex collisions),

(ii) $(v_k, v_{k+1}) \neq (u_{k+1}, u_k)$ (i.e., no edge collisions).

We extend the definition to paths of different lengths by truncating the longer path. Intuitively, this means that once a path ends, the respective agent "disappears"[1].

---

[1]Changing this to have the agents remain at the target location does not impact our results in any significant way.

Given n agents on a graph G and two lists $s_1, \ldots, s_n$ and $g_1, \ldots, g_n$ of source and goal vertices, respectively, a *plan* $P = \{\pi_1, \ldots, \pi_n\}$ is a set of non-colliding paths (i.e., π_i and π_j are non-colliding for all $i, j \in \{1, \ldots, n\}$ and $i \neq j$) such that π_i drives agent i from s_i to g_i for every $i \in \{1, \ldots, n\}$. The *length* of the plan is the maximal length of a path in P. The classical *Multi-Agent Path Finding (MAPF)* problem is to find a plan[2] P on G.

We now turn to recap the definitions of Explainable MAPF via Segmentation from (Almagor and Lahijanian 2020). Consider a path $\pi = v_1 \ldots v_m$ and $t_1 \leq t_2$. We define $\pi[t_1, t_2] = v_{t_1} \ldots v_{t_2}$ to be the segment of π between t_1 and t_2. If either t_1 or t_2 are not within the range $\{1, \ldots, m\}$, we simply disregard the out-of-bounds vertices.

A set of paths (better thought of as path segments) $\{\tau_1, \ldots, \tau_n\}$ where each $\tau_i = v_{i1} \ldots v_{ik_i}$ is *vertex disjoint* if for all $i \neq j$ we have $\{v_{i1}, \ldots, v_{ik_i}\} \cap \{v_{j1}, \ldots, v_{jk_j}\} = \emptyset$. Next, consider a plan $P = \{\pi_1, \ldots, \pi_n\}$ as above and let $K = \max_i m_i$ be its length, where m_i is the length of $\pi_i \in P$. A *vertex-disjoint decomposition* of P is an ordered list of natural numbers $1 = t_0 < t_1 < \ldots < t_r = K + 1$ such that for every $1 \leq k \leq r$, the path segments $\{\pi_j[t_{k-1}, t_k-1]\}_{j=1}^n$ are vertex-disjoint. We refer to r as the *index* of the decomposition. The minimal index of a vertex-disjoint decomposition of P is referred to as the *index* of P. As shown in (Almagor and Lahijanian 2020) (and recapped below), computing a minimal-index decomposition can be done in polynomial time using a greedy algorithm, hence, we only consider minimal index decompositions here. We now present the formal definition of the Explainable MAPF via Segmentation problem.

Problem 1 (Explainable MAPF via Segmentation). *Given a graph $G = \langle V, E \rangle$ with lists $s_1, \ldots, s_n$ and $g_1, \ldots, g_n$ of source and goal vertices, respectively, and bound $r \in \mathbb{N}$, find a plan P for the agents with index of at most r or answer that the instance is* unsolvable *– no such plan exists.*

Almagor and Lahijanian (2020) proved that (the decision version of) Problem 1 is NP-complete, even for 2 agents (unlike MAPF, which is in P for a fixed number of agents). They propose a centralized algorithm for the problem, but demonstrate that it does not scale. To this end, the goal of this paper is to develop a *decentralized* algorithm that is capable of solving Problem 1 and scaling to a large number of agents.

Explanation-Guided CBS

Our solution to Problem 1 extends from CBS (Sharon et al. 2015), a decentralized MAPF algorithm. Here, we first review this algorithm and then present our extensions to it to obtain Explanation-Guided CBS (XG-CBS).

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

---

[2]Typically, the plan is required to be optimal with respect to some cost function, e.g., makespan or sum-of-costs.

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 decentralized 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.

We (partially) demonstrate CBS in Figure 2. 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 1 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 index typically incurs a tradeoff with plan length. Below, we build upon CBS to plan for explainability, in order to address Problem 1.

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 below.

We remark that we focus on the classical CBS algorithm, as opposed to improvements thereof (e.g., ICBS (Boyarski et al. 2015)), as our goal is to study the efficacy of the well-understood constraint-tree method to explanations.

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 $P = \{\pi_1, \ldots, \pi_n\}$. In (Almagor and Lahijanian 2020), it is shown 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 $t_0 = t_1 = 0$ 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
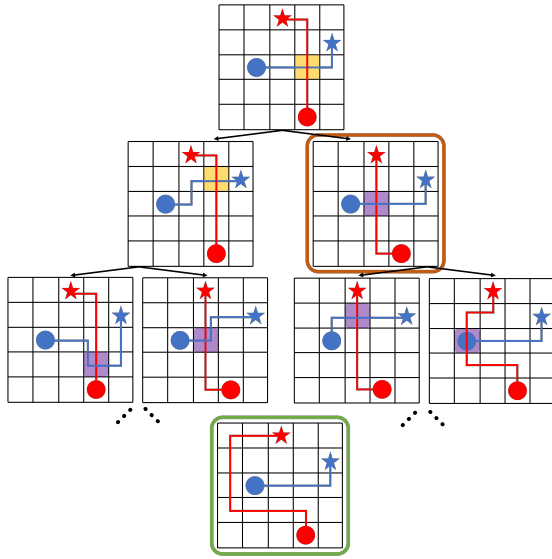
Figure 2: Illustration of XG-CBS with A^* as the low-level planner. Yellow and purple colors indicate collision and segmentation conflicts, respectively.

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 as above. 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.

In Figure 2, 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.

XG-CBS We now 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 the following section and only assume that the low-level search algorithm is sound and complete, e.g., A^*.

XG-CBS algorithm proceeds as follows. Initially, the low-level algorithm is called for each agent separately to obtain an initial plan. If the initial solution does not have any conflicts (collision nor segmentation), the plan and its decomposition are returned as the solution. If conflicts exist, they are resolved by extending the tree according to the constraints as above. Once a new node is created with a constraint on Agent i, the low-level algorithm is called to replan for Agent i. Each new node is assigned a cost (as we discuss below) and added to a priority queue. At the next iteration, the minimum cost plan is popped from the queue, and gets evaluated for conflicts. This process repeats until either a satisfactory plan is found or the search is exhausted. We refer the reader to the supplementary material for the pseudocode presentation of the XG-CBS algorithm.

An important remark is that during the low-level planning, an upper bound is set on the length of the path. The bound originates from the proof of membership in NP of Problem 1, and serves to bound the constraint tree.

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.

Figure 2 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 1 (XG-CBS Completeness). *XG-CBS always terminates, and given a solvable instance of Explainable MAPF via Segmentation, XG-CBS will terminate with a valid solution.*

We refer the reader to Kottinger, Almagor, and Lahijanian (2022) for the proof of Theorem 1.

Low-Level Search

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 Figure 2, 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 discussed in the next section, 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 later.

The three approaches above maintain the completeness of XG-CBS. Our final low-level planner, however, 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.

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 $\mathcal{C}$ 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 (c.f., Theorem 1).

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 $\mathcal{C}$), 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.

Consider the root node of Figure 2 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 Figure 2) 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^*. 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

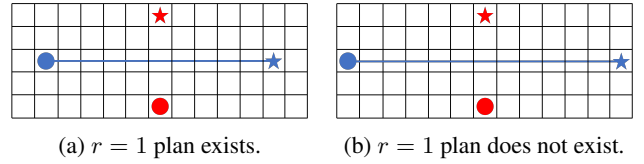(a) $r = 1$ plan exists. (b) $r = 1$ plan does not exist.

Figure 3: XG-A^* drawback of Remark 2.

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 $\mathcal{C}$, and if they are not constrained, H and i are computed as per the greedy approach described previously. Thus, XG-A^* starts by exploring all 1-segment plans, and only once these are exhausted, moves on to 2-segments, etc. We refer the reader to the supplementary material for the pseudocode of XG-A^* and two methods of speeding it up. We now make two important observations about XG-A^*.

Remark 1. *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" 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 2. *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 Figure 3, where XG-A^* is used to compute a path for the red agent given the existing path of the blue agent. In Figure 3a, 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 Figure 3b, 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 mentioned previously, this difficulty is inherent to Explainable MAPF via Segmentation.*

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 2 (XG-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 $\mathcal{C}$, 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 $\mathcal{C}$, then XG-A^* will terminate with such a path π_i that minimizes the index of $\{\pi_1, \ldots, \pi_n\}$.*

WXG-A^* – Weighted Explanation Guided A^*

As demonstrated in Remark 2, 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.

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. 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 2 still holds for WXG-A^*, i.e., WXG-A^* is complete.

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 1 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

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

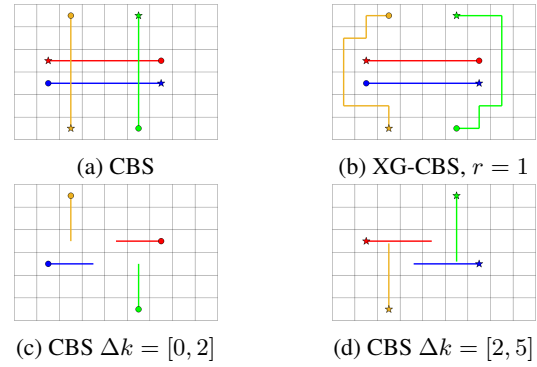(c) CBS $\Delta k = [0, 2]$ (d) CBS $\Delta k = [2, 5]$

Figure 4: Road crossing: solutions via CBS and XG-CBS

this idea, by making sure that the planned path is disjoint from all paths *within the current segment*. 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 $1 \le i \le r$ be the segment such that $t_i \le t \le t_{i+1}$, 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 $[3, 5]$.

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^*. We demonstrate that SR-A^* performs exceedingly well, both in terms of efficiency and plan index.

Case Studies

We evaluate the performance of XG-CBS on a combination of self-designed problems and standard MAPF benchmark problems from Bose and Markelov (2019). The self-designed spaces exhibit interesting behaviors that are unique to Problem 1. The benchmark results show the advantages and disadvantages of the proposed algorithms in various scenarios. All experiments were performed on a machine with an AMD Ryzen 7 3.9GHz CPU and 64 GB of RAM. Our implementation is available on GitHub (Kottinger 2021).

Illustrative Examples

To gain insight into XG-CBS, we showcase it on settings that present unique explanation challenges. Figure 4a shows a CBS solution of MAPF, where four agents need to cross an

(a) CBS   (b) CBS $\Delta k = [0,1]$ (c) CBS $\Delta k = [1,2]$

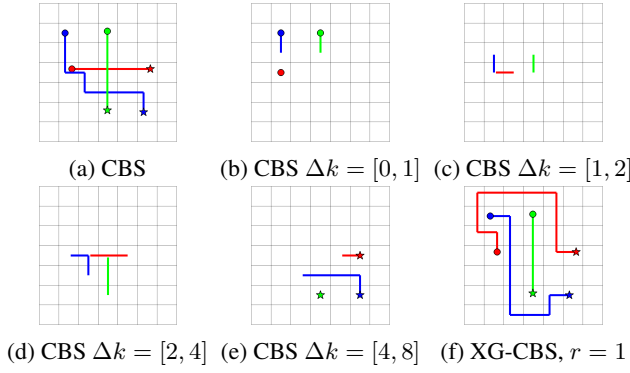(d) CBS $\Delta k = [2,4]$ (e) CBS $\Delta k = [4,8]$ (f) XG-CBS, $r = 1$

Figure 5: Apparent collision in short plan vs. optimal index

intersection. Visually verifying that the plan is collision free is difficult. It becomes easy using the explanation scheme, which decomposes the plan into two disjoint segments in Fig. 4c and 4d. Using XG-CBS, we obtain a plan with index 1, as depicted in Fig. 4b, which is much easier to verify. This 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 previously, 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. As we discuss in in the next section, these factors have a significant effect on the efficacy of each algorithm.

Our next use case is depicted in Figure 5a. A human examining the plan may notice a possible collision between the red and green agents. However, it becomes clear in the explanation (Figures 5b-5e) 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. 5f. This solution was obtained in 0.5 seconds, whereas XG-CBS with A^* again timed out. For more case studies, we refer the reader to the supplementary material.

Benchmark Evaluation

We now evaluate XG-CBS with the different low-level algorithms on a large set of MAPF benchmarks from Bose and Markelov (2019). Our comparison of the algorithms is along three axes: computation time, segmentation index, and plan length (average cost, i.e., sum-of-costs divided by number of agents). We also evaluate CBS as a baseline.

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 upper bound for XG-CBS. We then repeatedly lower the bound in XG-CBS, until failure. 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 grid worlds with the following sizes and number of agents: 9×9 with 4, 8, 10, and 12

agents, 16×16 with 5, 10, 15, and 20 agents, and 33×33 with 10, 20, and 30 agents. For each grid size and agent number combination, we ran 100 unique experiments. The timeout for a single algorithm on a single benchmark was 5 minutes (the high threshold is due to the fact that Problem 1 is computationally harder than MAPF). The results are partially presented in Figures 6, 7, and 8. We refer to Kottinger, Almagor, and Lahijanian (2022) for the full set of benchmark results. For 33×33 environments, XG-A^* and WXG-A^* nearly always time out, and hence not evaluated.

For the most part, the results match our expectations: vanilla CBS offers the best trade-off between plan length and computation time, 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 (c.f., *first* column). Moreover, as the environment becomes smaller and more congested (9×9, 12 agents), XG-A^* outperforms A^*. Unfortunately, the history-dependence of XG-A^* means it cannot scale to larger environments. 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 success-rate (matching CBS), and invariably reduces the index of the plan (compared to CBS) by a significant amount (e.g., for 33×33, 30 agents, the reduction is from roughly 24 segments to 6 segments!). Moreover, it's limited search space allows it to match CBS in computation time, and sometimes even outperform it. The trade-off, naturally, comes in the path length, which increases.

Another pleasant surprise comes from A^*, which despite the expected increase in computation time, does manage to give some decrease in the index, even on larger environments. 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, it is not clear whether batch experiments capture the performance of WXG-A^*.

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 indices on congested environments. Finally, by carefully tuning a combined weight (e.g., by trying different options), one can obtain better explanations in small environments using WXG-A^*.

Discussion and Future Work

We introduced a CBS-based decentralized algorithm for Explainable MAPF via Segmentation. Our technical contribution is twofold: first, we describe the extension XG-CBS, which can be readily implemented on top of existing CBS
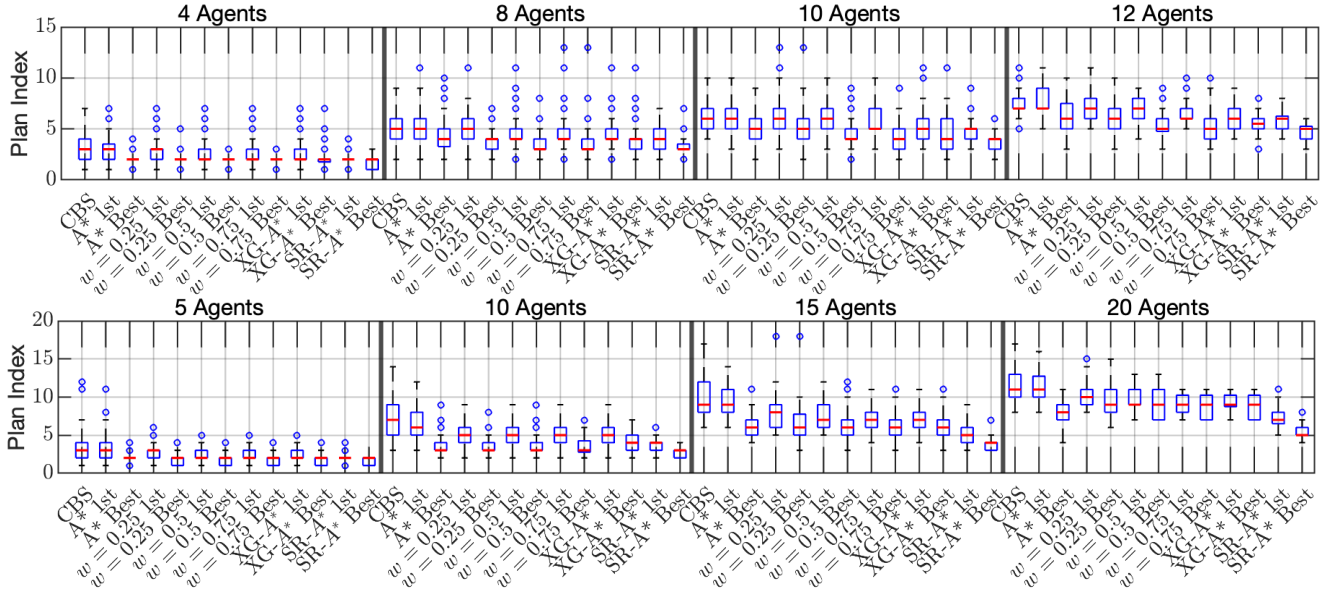
Figure 6: Plan Index for 9×9 (top) and 16×16 (bottom) environments.
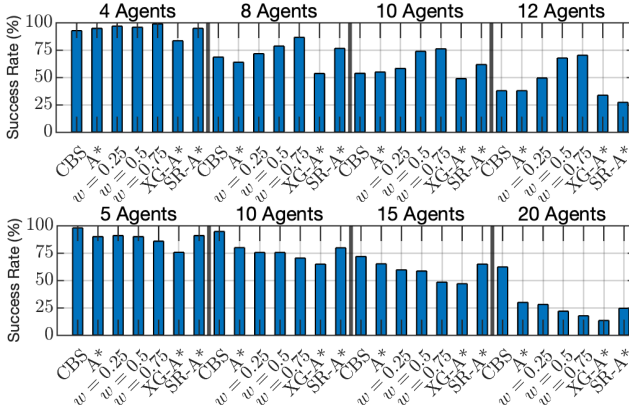

Figure 7: Success Rate for 9×9 (top) and 16×16 (bottom) environments.

Figure 8: Benchmark results for 33×33 environment.

implementations. Second, we describe new low-level search algorithms, namely XG-A^*, WXG-A^* and SR-A^* oriented toward low index plans. While the former two are complete algorithms, we show they do not scale well. The latter, despite not being complete, scales well and is often as efficient as CBS, while yielding easily explainable plans.

Future research will adapt other MAPF algorithms to the explainable settings, such as Priority-Based Search (Ma et al. 2019b), and SAT-based solutions, as well as extensions and improvements of "vanilla" CBS. Finally, we remark that Explainable MAPF via Segmentation has potential applications beyond gaining trust. Disjoint decompositions can be used during the actual execution of the plan, in case the agents' paths must not cross. For example in tethered robots, 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 enables us to achieve. Similarly, 3D pipe routing (Belov et al. 2020), small explana-
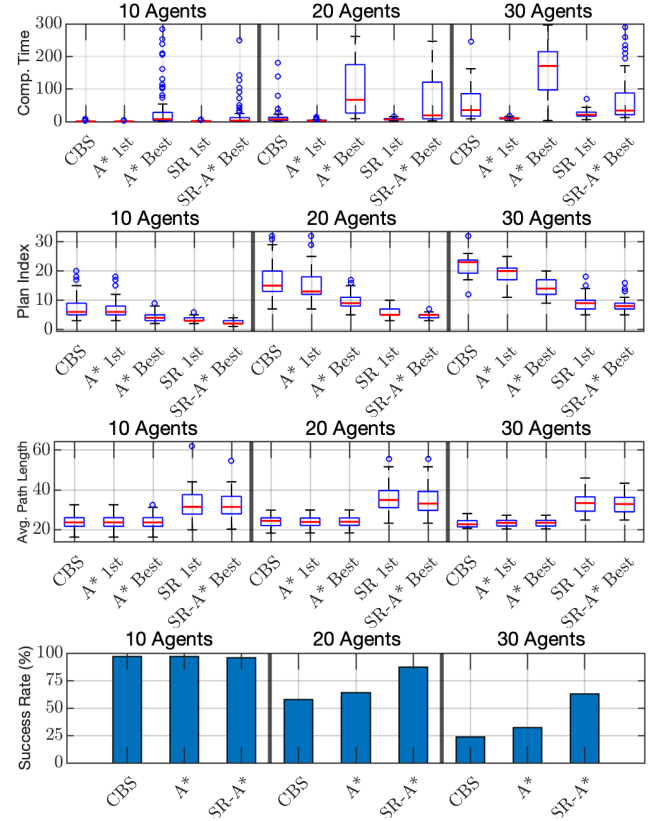
tions may allow for simpler routing. Other applications can be found in multi-layered circuit board design. In particular, such applications show that even an index reduction of one may by financially beneficial.

References

Almagor, S.; and Lahijanian, M. 2020. Explainable Multi Agent Path Finding. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, 34–42. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450375184.

Arrieta, A. B.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; García, S.; Gil-López, S.; Molina, D.; Benjamins, R.; et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58: 82–115.

Bartak, R.; Svancara, J.; and Vlk, M. 2018. 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)*, 748–756.

Belov, G.; Du, W.; De La Banda, M. G.; Harabor, D.; Koenig, S.; and Wei, X. 2020. From multi-agent pathfinding to 3D pipe routing. In *Thirteenth Annual Symposium on Combinatorial Search*.

Bose, A.; and Markelov, I. 2019. Multi-Agent Path Planning in Python. https://github.com/atb033/multi_agent_path_planning. Accessed: 2021-08-01.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. 2015. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Brandao, M.; Canal, G.; Krivić, S.; Luff, P.; and Coles, A. 2021. 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)*, 299–306. IEEE.

Cohen, L.; Koenig, S.; Kumar, S.; Wagner, G.; Choset, H.; Chan, D.; and Sturtevant, N. 2018. Rapid Randomized Restarts for Multi-Agent Path Finding: Preliminary Results. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1909–1911.

Eifler, R.; Cashmore, M.; Jorg, H.; Magazzeni, D.; and Steinmetz, M. 2019. Explaining the Space of Plans through Plan-Property Dependencies. *Proceedings of the 2nd Workshop on Explainable Planning (XAIP)*.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28(1): 83–87.

Felner, A.; Stern, R.; Shimony, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; and Surynek, P. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 28–37.

Fines, K.; Sharpanskykh, A.; and Vert, M. 2020. Agent-based distributed planning and coordination for resilient airport surface movement operations. *Aerospace*, 7(4): 48.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. arXiv:1709.10256.

Hubel, D. H.; and Wiesel, T. N. 1959. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3).

Kambhampati, S. 2019. Synthesizing Explainable Behavior for Human-AI Collaboration. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 1–2. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099.

Kottinger, J. 2021. Explanation-Guided Conflict-Based Search for Explainable MAPF. https://github.com/aria-systems-group/Explanation-Guided-CBS. Accessed: 2021-12-01.

Kottinger, J.; Almagor, S.; and Lahijanian, M. 2021. MAPS-X: Explainable Multi-Robot Motion Planning via Segmentation. In *Proceedings of the IEEE International Conference on Robotics and Automation*. Xi'an, China: IEEE.

Kottinger, J.; Almagor, S.; and Lahijanian, M. 2022. Conflict-Based Search for Explainable Multi-Agent Path Finding. https://arxiv.org/abs/2202.09930.

Lapuschkin, S.; Wäldchen, S.; Binder, A.; Montavon, G.; Samek, W.; and Müller, K.-R. 2019. Unmasking Clever Hans predictors and assessing what machines really learn. *Nature Communications*, 10(1).

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *IJCAI*, volume 2019, 442–449.

Li, J.; Harabor, D.; Stuckey, P. J.; Felner, A.; Ma, H.; and Koenig, S. 2019b. Disjoint splitting for multi-agent path finding with conflict-based search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 279–283.

Ma, H.; Harabor, D.; Stuckey, P.; Li, J.; and Koenig, S. 2019a. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, (in print).

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019b. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 7643–7650.

Mari, T.; Dang, T.; and Gössler, G. 2021. Explaining Safety Violations in Real-Time Systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 100–116. Springer.

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

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Stern, R.; Sturtevant, N. R.; Atzmon, D.; Walker, T.; Li, J.; Cohen, L.; Ma, H.; Kumar, T. K. S.; Felner, A.; and Koenig, S. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. 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)*, 145–147.

Tang, S.; Lee, T. S.; Li, M.; Zhang, Y.; Xu, Y.; Liu, F.; Teo, B.; and Jiang, H. 2018. Complex Pattern Selectivity in Macaque Primary Visual Cortex Revealed by Large-Scale Two-Photon Imaging. *Current Biology*, 28(1): 38–48.e3.