The Eleventh International Symposium on Combinatorial Search
(SoCS 2018)

Robust Multi-Agent Path Finding

Dor Atzmon
Roni Stern
Ariel Felner
Ben-Gurion University
Israel

Glenn Wagner
Carnegie Mellon University
USA

Roman Barták
Charles University
Czech Republic

Neng-Fa Zhou
CUNY Brooklyn College
USA

Abstract

In the multi-agent path-finding (MAPF) problem, the task is to find a plan for moving a set of agents from their initial locations to their goals without collisions. Following this plan, however, may not be possible due to unexpected events that delay some of the agents. We explore the notion of k-*robust* MAPF, where the task is to find a plan that can be followed even if a limited number of such delays occur. k-*robust* MAPF is especially suitable for agents with a control mechanism that guarantees that each agent is within a limited number of steps away from its pre-defined plan. We propose sufficient and required conditions for finding a k-robust plan, and show how to convert several MAPF solvers to find such plans. Then, we show the benefit of using a k-robust plan during execution, and for finding plans that are likely to succeed.

1 Introduction and Overview

The *Multi-Agent Path Finding* (MAPF) problem is defined by a graph, $G = (V, E)$ and a set of n agents labeled $a_1 \dots a_n$, where each agent a_i has a start position $s_i \in V$ and a goal position $g_i \in V$. At each time step, an agent can either *move* to an adjacent location or *wait* in its current location. The task is to find a sequence of move/wait actions for each agent a_i that moves it from s_i to g_i such that agents do not *conflict*, i.e., occupy the same location at the same time. MAPF has practical applications in video games, traffic control, and robotics (see (Felner et al. 2017) for a survey). In many cases, there is also a requirement to minimize some cumulative cost function such as the sum of costs incurred by all agents before reaching their goals. Solving MAPF optimally is NP-Hard (Yu and LaValle 2013b; Surynek 2010). Nonetheless, efficient optimal algorithms exist, some are even capable of finding optimal plans for more than a hundred agents (Wagner and Choset 2015; Boyarski et al. 2015; Surynek 2012; Yu and LaValle 2013a).

In practice, unexpected events may delay some of the agents, preventing them from following their pre-determined sequence of move/wait actions. When such an event occurs, one may need to adjust the plan of one or more agents to avoid collisions. Such re-planning can be very costly or even impossible, as it may require computing and communication capabilities as well as time that the agents may not have. This is especially common in safety-critical settings such as air traffic control. Thus, it is often desirable to generate a plan that can withstand unexpected delays, without replanning if they occur.

In this work, we explore a novel form of *robustness* for MAPF problems called k-robust MAPF (kR-MAPF) which is designed to produce MAPF plans that can be followed even if unpredictable delays occur. In kR-MAPF we seek a plan that is robust to k delays per agent during plan execution, i.e., each agent may be delayed up to k times during plan execution and the plan would still be safe (no collisions). Standard MAPF is a special case of kR-MAPF with $k = 0$. k-robustness is useful in various settings, including cases where agents' localization is not perfect, and cases where synchronization between the agents is imperfect, e.g., when an agent may not be able to track its path exactly, but each agent can guarantee remaining within k steps of its nominal trajectory. We describe sufficient and required conditions for a k-robust MAPF solution and show that a natural adaptation of a A*-based MAPF solver to kR-MAPF results in a state space that grows exponentially with k. We then present two algorithms for solving kR-MAPF more efficiently. The first is based on the Conflict-Based Search (CBS) framework (Sharon et al. 2015). The second is based on expressing the problem in Picat (Zhou, Kjellerstrand, and Fruhman 2015), a declarative constraint programming language, and solving it using a suitable general-purpose solver.

We evaluate experimentally the proposed algorithms on 8x8 and 32x32 grids, as well as on larger grids from a commercial video game. Our results show that finding a k-robust solution indeed results in fewer re-plans during execution, and the overall increase in plan cost is small. The Picat-based solver performed well for small grids, while the CBS-based solver performed better for large grids, being able to find k-robust solutions for some problems with more than 90 agents.

Another form of robustness is probabilistic robust, where the desire is be able to perform the plan (without re-planning) within a given probability p. We briefly discuss this variant in the end of the paper but a deeper handling of probabilistic robust is left for future work.

2 k-Robust MAPF

A solution to a MAPF problem is a plan $\pi = \{\pi_1, \ldots, \pi_n\}$ such that $\forall i \in [1, n]$, π_i is a sequence of move/wait actions that move agent a_i from s_i to g_i. The location of a_i after executing the first t move/wait actions in π without experiencing any delay is denoted by $\pi_i(t)$. Thus, $\pi_i(0) = s_i$ and $\pi_i(|\pi_i|) = g_i$.

Definition 2.1 (Conflict)**.** A conflict $\langle a_i, a_j, t\rangle$ in a plan π occurs iff agents a_i and a_j are located in the same location at time step t, i.e., when $\pi_i(t) = \pi_j(t)$, or when they traverse the same edge when moving from time $t-1$ to t, i.e., $\pi_i(t-1) = \pi_j(t) \wedge \pi_i(t) = \pi_j(t-1)$. A conflict due to a shared location is called a *vertex conflict* and a conflict due to a shared edge is called an *edge conflict*.

We say that π is a **valid plan** if it is conflict-free. A MAPF solver is *sound* if it outputs a valid plan. A *delay* in a plan π is defined by a tuple $\langle \pi, i, t\rangle$, representing that agent a_i did not perform the move action defined for it in π_i and instead stayed at time t in location $\pi_i(t-1)$. A plan is *robust* to a delay if the delayed agent can continue to follow its plan after the delay without causing a collision. Formally, for a plan π and a delay $D = \langle \pi, i, t_D\rangle$, let $D(\pi)$ be the plan that is equivalent to π, except for replacing π_i with

$$\pi_i'(t) = \begin{cases} s_i & t = 0 \\ \pi_i(t) & t < t_D \\ \pi_i(t-1) & otherwise \end{cases}.$$

A plan π is *robust* to a delay D if $D(\pi)$ is valid. π is robust to a set of delays $\mathcal{D}$ iff applying any subset of $\mathcal{D}$ to π will yield a valid plan, i.e., for every set of delays $\{D_1, \ldots D_T\} \subseteq \mathcal{D}$, the plan $D_T(D_{T-1}(\ldots D_1(\pi)\ldots))$ is valid.

Definition 2.2 (k-robust plan)**.** A plan is k-robust iff it is valid and it is robust to any set of delays that contains at most k delays for each agent.

Having a k-robust plan is desirable in many cases, especially when agents' localization is not perfect, and control mechanisms are used to verify that it is k-steps from the given plan.

Note that there cannot be edge conflicts in a k-robust plan for any $k > 0$.[1]

Definition 2.3 (k-delay Conflict)**.** A k-delay conflict $\langle a_i, a_j, t\rangle$ in a plan π occurs iff there exists $\Delta \in [0, k]$ such that agents a_i and a_j are located in the same location in time steps t and $t + \Delta$, respectively, i.e, $\pi_i(t) = \pi_j(t + \Delta)$.

Observation 1. A plan is k-robust iff it does not contain any k-delay conflicts.

There can be more than one k-robust plan for a given MAPF problem instance, and plans with lower *costs* are usually preferred. We focus on a common MAPF cost function called the *sum of costs*, which is the summation of the number of time steps required by each agent to reach its goal (Standley 2010; Standley and Korf 2011; Sharon et al. 2013; 2015). A k-robust plan is **optimal** if there is no other k-robust plan with a lower cost. Next, we explore several algorithms for finding optimal k-robust plans.

[1] Also, $k > 0$ disallows agents to move to a location that was occupied by another agent in the previous time step. Such a train-like motion is not allowed in some MAPF formulation.

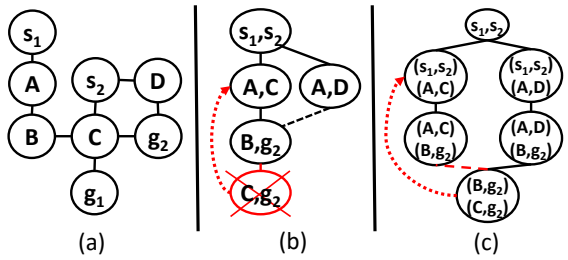

Figure 1: A MAPF problem (a) its search tree (b), and its k-robust search tree (c). The red lines show k-delay conflicts.

3 A*-based Solutions

Many MAPF algorithms (Silver 2005; Standley 2010; Goldenberg et al. 2012; Wagner and Choset 2015) are based on the well-known A* algorithm (Hart, Nilsson, and Raphael 1968). These algorithms search in a n-agent state space, which includes all the possible ways to place n agents into $|V|$ vertices, one agent per vertex. The *start* and *goal* states are $(s_1, \ldots s_n)$ and $(g_1, \ldots g_n)$, respectively. An action in this state space represents n single-agent move/wait actions, one single-agent action per agent. An action is applicable if its constituent single-agent actions do not create conflicts. Hence, a path in this n-agent state space from the start state $(s_1, \ldots s_n)$ to the goal state $(g_1, \ldots g_n)$ corresponds to a valid plan.

One way to adapt A* solvers to return k-robust plans is to modify state generation to prevent combinations of single-agent actions that lead to k-delay conflicts. However, this may lead to the solver returning non-optimal plans. For example, consider finding a 2-robust plan for the problem in Figure 1(a). The optimal 2-robust solution is $\pi_1 = (s_1, A, B, C, g_1)$ and $\pi_2 = (s_2, D, g_2)$, with a cost of 6. Consider running A* on this problem. First, A* expands state (s_1, s_2), generating two children (A, C) and (A, D). Assume that (A, C) was expanded first, generating state (B, g_2) with cost 4 (2 per agent). Next, (B, g_2) is expanded. Since a_2 was in C at $t = 1$, state (C, g_2) will not be generated due to the 2-robustness constraint. Next, state (A, D) is expanded. It will not generate (B, g_2), as this state was already reached via state (A, C) with the same cost (see Figure 1(b)). Thus, while there is a plan in which state (B, g_2) generates state (C, g_2), this specific run of A* will not find it. This results in finding a suboptimal plan of cost 7.

To remedy this, the n-agent state space needs to be modified to keep track of the last k steps of each agent in each state. Thus, in this state space a state represents a possible way to place n agents into $|V|$ vertices, one agent per vertex, *over* $k-1$ *consecutive time steps*. Figure 1(c) shows the search tree of this extended state space. An A*-based MAPF solver over this state space can find an optimal k-robust plan, but the size of this state space is much larger than the classic n-agent state space, since its size grows exponentially with

k. In the next section, we describe k-robust algorithms that mitigate this problem.

4 Conflict-Based Search Solutions

Conflict-based search (CBS) (Sharon et al. 2015) is a commonly used MAPF solver (Ma, Kumar, and Koenig 2017; Boyarski et al. 2015) that does not explicitly search the n-agent state space. CBS finds a plan by searching for a path for each agent separately. Conflicts are avoided by imposing a set of *constraints* of the form $\langle a_i, v, t\rangle$, representing that agent a_i is prohibited from occupying vertex v at time step t. A plan $\pi = \{\pi_1, \dots \pi_n\}$ is called *consistent* with a set of constraints if its constituent sequence of move/wait actions satisfy this set of constraints, i.e., for every constraint $\langle a_i, v, t\rangle$ in this set it holds that $\pi_i(t) \neq v$.

CBS works by searching a *constraint tree* (CT) for a set of constraints such that a plan consistent w.r.t. this set of constraints is valid and optimal. The CT is a binary tree, in which each node N contains: **(1)** a set of constraints imposed on the agents ($N.constraints$), **(2)** a single plan ($N.\pi$) consistent with these constraints, and **(3)** the cost of $N.\pi$ ($N.cost$). The root of the CT contains an empty set of constraints (thus, every plan is consistent with the root). A successor of a node in the CT inherits the constraints of the parent node and adds a single new constraint for one agent. Generating a successor node N means finding a plan consistent with $N.constraints$ and identifying the conflicts in this plan, if they exist. Since N was generated by adding a single new constraint, only one agent needs to replan, which can be done with any optimal single-agent path-finding algorithm such as A*. The algorithm used for this purpose is referred to as the CBS low-level solver. A CT node N is a goal node when $N.\pi$ is valid. To search the CT for a goal node CBS runs a best-first search where nodes are ordered by their costs ($N.cost$).

Next, we describe two key components of CBS: how it identifies conflicts in N, and how to choose which constraint to add when expanding N and generating its successors.

Identifying conflicts in a consistent plan.

Once a consistent plan has been found by the low-level solver, it is *validated* by simulating the movement of the agents along their planned paths ($N.\pi$). If all agents reach their goals without any conflict, N is declared as the goal node, and $N.\pi$ is returned. Otherwise, a conflict is found the node is declared a non-goal.

Resolving a conflict: When a non-goal CT node N is chosen in the best-first search of the CT, CBS generates its successor CT nodes. This is done by attempting to resolve a conflict in $N.\pi$. Let $\langle a_i, a_j, t\rangle$ be a conflict in $N.\pi$, and let v be the location of this conflict, i.e., $v = N.\pi_i(t) = N.\pi_j(t)$.

CBS *splits* N and generates two new CT nodes as children of N, adding the constraint $\langle a_i, v, t\rangle$ to one child and the constraint $\langle a_j, v, t\rangle$ to the other child.

Note that for each (non-root) CT node the low-level search is activated only for one agent – the agent for which the new constraint was added.

4.1 k-Robust CBS

Next, we introduce *k-robust CBS* (kR-CBS), an adaptation of CBS designed to return optimal k-robust plans. kR-CBS differs from CBS in how it identifies and resolves conflicts.

Identifying k-delay conflicts. After the low-level solver returned a consistent plan for a CT node N, kR-CBS scans $N.\pi$ for k-delay conflicts by simulating the paths and checking for conflicts with the k-last locations of all other agents.

Thus, finding a k-delay conflict $\langle a_i, a_j, t\rangle$ means that $N.\pi_i(t) = N.\pi_j(t+\Delta)$ for $\Delta \in [0, k]$. Node N is a goal CT node iff it has no k-delay conflicts. This process of checking conflicts and identifying a goal is easy to implement, but its runtime is larger by a factor of k than the equivalent plan validation step in CBS.

Resolving conflicts (splitting CT nodes). Let N be a non-goal node in the CT selected to be expanded next by kR-CBS, and let $\langle a_i, a_j, t\rangle$ be a k-delay conflict in N. This means that there is a vertex v and a value $\Delta \in [0, k]$ such that $v = N.\pi_i(t) = N.\pi_j(t + \Delta)$. Note that there is no k-robust plan in which a_i is at v at time t while a_j is at v at time $t + \Delta$. Therefore, at least one of the constraints, $\langle a_i, v, t\rangle$ or $\langle a_j, v, t+\Delta\rangle$, must be added to the CT and must be satisfied by the low-level solvers. Consequently, kR-CBS generates two children to N, each having one of these constraints.

kR-CBS is sound, because it only halts when generating a CT node that has no k-delay conflicts. It is complete in the sense that if a solution exists then it will find it, because splitting a CT node never loses any valid plans. Similarly, kR-CBS returns optimal plans, as it searches the CT in a best-first order according to the nodes' costs, and the cost of a node N is a lower bound on the cost of any valid plan consistent with $N.constraints$.

Example. Consider a 2-robust MAPF problem on the graph in Figure 2(a), with two agents whose start-goal pairs are s_1-g_1 and s_2-g_2, respectively. Figure 2(b) shows the first two levels of the CT generated by kR-CBS, where every node N shows $N.constraints$ (labeled Con), $N.\pi_1$, $N.\pi_2$, and $N.cost$.

The plan in the root is valid, but it is not 2-robust since it has a 2-delay conflict $\langle a_2, a_1, 1\rangle$ at location B for $\Delta = 1$ (since $\pi_2(1) = \pi_1(2) = B$). To try to resolve this conflict, kR-CBS adds the constraint $\langle a_2, B, 1\rangle$ to the left child and the constraint $\langle a_1, B, 2\rangle$ to the right child. Both children of the root node are also not goal nodes. In fact, in this example we will need to generate a total of 7 CT nodes before finding an optimal plan. As we show next, it is possible to improve kR-CBS such that it will find the goal sooner.

4.2 Improved k-Robust CBS

In Figure 2(b), the 2-delay conflict $\langle a_2, a_1, 1\rangle$ in the root CT node is resolved by adding constraints such that either a_2 is not in B at time 1 (left child), or a_1 is not in B at time 2 (right child). Imposing these constraints is correct because in every 2-robust plan either a_1 is not at B at time 1 or a_2 is not in B at time 2. This argument can be extended: in every 2-robust plan either a_2 is not in B at times 1 and 2 or a_1 is not in B at times 2 and 3. Thus, we can impose a stricter constraint on the left subtree by adding the constraint $\langle a_2, B, 2\rangle$, and add

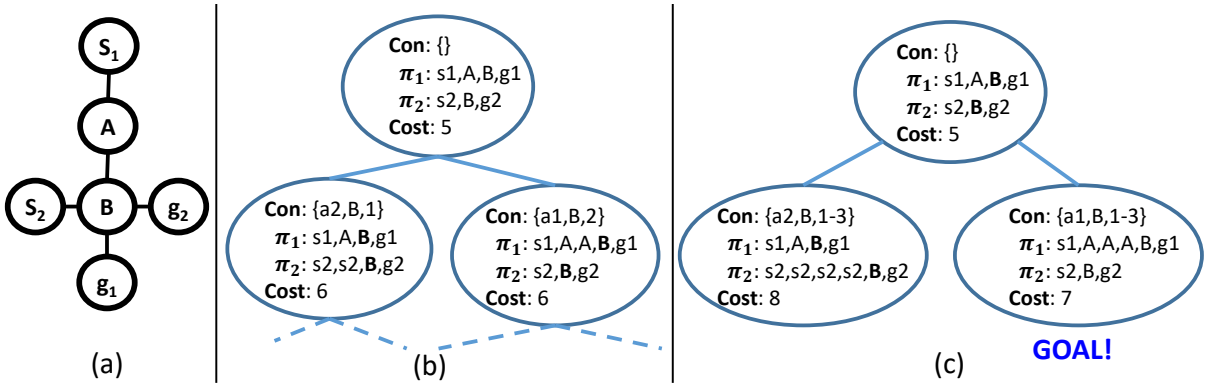

Figure 2: (a) The graph (b) The CT using the original time/location constraints (c) CT using the range constraints

the constraint $\langle a_1, B, 3\rangle$ to the right subtree. Imposing more constraints per CT node can reduce the size of the CT tree, and consequently the overall runtime.

To exploit this understanding, we introduce the Improved kR-CBS (IkR-CBS) that resolves conflicts in a CT node N by imposing *range constraints* on its successors. A *range constraint* is defined by the tuple $\langle a_i, v, [t_1, t_2]\rangle$ and represents the constraint that agent a_i must avoid vertex v from time step t_1 to time step t_2.

Ideally, we would like to construct range constraints as large as possible, to minimize the size of the CT tree. However, over-constraining CT nodes may result in losing completeness and optimality. The key question is thus which pairs of range constraints to use to resolve conflicts without losing completeness and optimality.

Definition 4.1 (Sound Range Constraints)**.** A pair of range constraints for a given k-delay conflict is called *sound* iff all optimal k-robust plans satisfy at least one of the constraints.

Proposition 4.1. A kR-CBS variant that resolves conflicts with sound pairs of range constraints is sound, and is guaranteed to return an optimal k-robust plan if such exists.

Proof: For a CT node N, let N_1 and N_2 be its children, generated by the sound pair of range constraints R_1 and R_2, respectively. Now, $\pi(N)$ denotes all the k-robust plans that do not violate $N.constraints$. Observe that $\pi(N_1)$ contains all the plans in $\pi(N)$ that satisfy R_1, and similarly $\pi(N_2)$ contains all the plans in $\pi(N)$ that satisfy R_2. Since R_1 and R_2 are a sound pair of constraints, it holds that $\pi(N) = \pi(N_1) \cup \pi(N_2)$. Thus, splitting CT nodes by resolving conflicts with a sound pair of constraints does not lose any plans and thus preserves the soundness and optimality, and is guaranteed to return a k-robust plan if exists.□

We call a pair of range constraints *symmetric* if they constrain the same vertex and the same time range.

Corollary 4.2 (Symmetric range constraints)**.** For any time step t, vertex v, and agents a_i and a_j, the range constraints $\langle a_i, v, [t, t+k]\rangle$, $\langle a_j, v, [t, t+k]\rangle$ are sound for solving a k-robust MAPF problem.

Proving Corollary 4.2 is straightforward. Note that setting a symmetric range constraint on a range larger than k is in general not sound. Thus, Corollary 4.2 gives an upper bound on the size of the largest pair of symmetric range constraints that is sound. Nevertheless, there are more than one k-sized symmetric pairs of range constraints for a given k-delay conflict $\langle a_i, a_j, t\rangle$ over vertex v. In our implementation, we used the time range $[t, t+k]$, but the pair of range constraints $[t-k, t]$ is also sound.

A pair of sound range constraints can also be *asymmetric*, i.e., constrain one agent to a longer time range than the other agent. For example, consider a conflict $\langle a_i, a_j, t\rangle$ over vertex v and pair of range constraints $R_1 = \langle a_i, v, [t-k, t+k]\rangle$ and $R_2 = \langle a_j, v, [t]\rangle$. R_1 and R_2 are a sound pair of constraints, because a solution must satisfy either R_1 or R_2, since violating both results in a k-delay conflict. R_1 and R_2 are extremely asymmetric, but one can imaging asymmetric range constraints that are more balanced. An open question for asymmetric range constraints is how to choose on which agent to impose the more restrictive constraint.

4.3 Experimental Results

Next, we experiment with kR-CBS and IkR-CBS using symmetric and asymmetric pairs of range constraints.

8x8 open grid 60 random problem instances were generated in an open 8x8 grid. Then a kR-MAPF solver for $k = 0, 1$, and 2 was executed and the resulting plan cost and CPU runtime were measured.

Table 1 shows the average plan cost and CPU runtime when finding k-robust solutions using kR-CBS (labeled KR) and IkR-CBS with the asymmetric and with the symmetric range constrains (labeled IKR(A) and IKR(S), respectively) for 4, 6, 7, 8, 9, and 10 agents (different rows). Since all solvers return optimal k-robust plans, we show the plan cost only once for every value of k. Note that $k = 0$ is standard CBS.

First, consider the plan costs. As can be seen, the k-robust plans are often more costly than a plan that are not robust ($k = 0$), but the added cost is relatively small. In fact, the largest relative increase in cost when moving from $k = 0$ to $k = 1$ was observed for 4 agents, where for $k = 0$ the

	Plan cost			Plan time (ms)						
				k=0	k=1			k=2		
n	k=0	k=1	k=2	All	KR	IKR(A)	IKR(S)	KR	IKR(A)	IKR(S)
4	21	22	22	6	15	**14**	15	193	110	**67**
6	31	32	32	5	28	26	**20**	990	388	**94**
7	36	37	39	7	31	26	**17**	1,618	826	**184**
8	41	41	43	6	29	23	**20**	2,625	1,051	**229**
9	48	49	51	9	379	218	**76**	20,006	4,408	**556**
10	49	51	53	41	162	124	**78**	22,464	7,097	**875**

Table 1: Average plan cost and planning runtime for different CBS-based k-robust solvers, on an 8x8 open grid.

average cost was 21 and for $k = 1$ the average cost increased to 22, which amounts to an increase of less than 5%.

Second, the runtime of finding a k-robust solution increase as k increases. For example, finding an optimal valid plan with kR-CBS for 9 agents required 9 milliseconds but finding an optimal 1-robust plan required 379 milliseconds. As expected, both IkR-CBS variants run much faster than kR-CBS and this improvement increases when increasing k and when there are more agents. When comparing the symmetric and the asymmetric range constraints, we see a clear advantage for the symmetric range constraints. For example, consider finding an optimal 2-robust solution for 9 agents: it required 20,006 milliseconds for kR-CBS, 4,408 for IkR-CBS with asymmetric constraints, and only 556 milliseconds for IkR-CBS with the symmetric constraints. We conjuncture that this is due to the arbitrary way in which we choose which agent to constrain more when using the asymmetric range constraints; better ways of choosing asymmetric range constraints may exist.

brc202.d Map We also performed some experiments on a larger map from the Dragon Age Origins (DAO) video game, which is available in the `movingai` repository (Sturtevant 2012). Specifically, we generated 50 randomly generated instances with 30 agents on the `brc202d` map, which has 43,151 vertices. The results show the same trends observed in the 8x8 grid results reported above: increasing k results in slightly higher plan costs and longer runtime. Since this map is very large, the optimal k-robust plan often has the same cost as the optimal plan that is not robust. When averaging over 50 random instances, the average plan cost was 3,818.35, 3,818.43, and 3,818.53 for $k = 0$, 1, and 2, respectively. The increase in runtime when increasing k was also more modest compared to the 8x8 results: 213, 284, and 381 seconds, for $k = 0$, 1, and 2, respectively. This lower impact of k on plan cost and runtime is because the map's large size provides more room to find alternative plans that are more robust.

Increasing k Larger values for k increase the ranges that are checked and has the potential to increase the number of conflicts that should be checked. Nevertheless, the effect of increasing k is moderate and problems with large values of k can be solved in relatively reasonable time. Table 2 shows the number of instances out of 50 that could be solved within 5 minutes when trying to solve problems on 16x16 open grid with IkR-CBS with the symmetric constraints for different values of k (rows) and different number of agents n (columns). As can be seen, increasing k and increasing n both have an effect on the success rate. However, unless both were large, the majority of the problem instances could be solved.

#agents	5	10	15
k=0	50	50	49
k=1	50	50	45
k=2	50	46	35
k=3	50	43	19
k=4	47	34	13
k=5	47	29	8
k=6	45	21	3
k=7	41	15	1

Table 2: Number of instances solved within the allocated time out. The grids used are 16x16 grids

5 A Declarative Solution

An alternative approach to solve MAPF problems is to compile them into other known NP-hard problems that have mature and effective general purpose solvers (Surynek 2012; Yu and LaValle 2013a; Erdem et al. 2013; Surynek et al. 2016).

Generally speaking, these approaches express a set of constraints that define the MAPF problem in some *declarative* language and then call a general purpose solver, e.g., a SAT solver or a Mixed Integer Linear Program (MILP) solver, to obtain the solution. Adapting such solvers to the k-robust variant is relatively simple, requiring a simple modification to the constraints. To demonstrate this, we implemented a MAPF solver using Picat (Zhou, Kjellerstrand, and Fruhman 2015), a logic-based programming language that has three constraint modules. The encoding we used is based on Surynek's SAT-based MAPF solver (Surynek et al. 2016), in which there is a Boolean variable for every triplet (a, t, v) of agent (a), time (t), and location (v), where this variable is true iff agent a occupies location v at time t. A set of constraints is imposed on these variables, namely:

1. Each agent occupies exactly one vertex at each time step.
2. No two agents occupy the same vertex at any time.
3. In every time step an agent may only transition between two adjacent locations.

For producing k-robust solutions, the second constraint is extended such that no two agents occupy the same vertex in time steps that are closer than k from each other. The exact Picat model we developed is available at https://tinyurl.com/kRobust. The advantage of using Picat to encode MAPF is that the model can be compiled to SAT, to a constraint program (CP), or to MILP. In our experiments we only run a SAT compilation.

5.1 Experimental Results

Next, we experimentally evaluated our Picat-based solver, and compared it with IkR-CBS with symmetric range constraints, which is the CBS-based solver that performed best in our experiments.

Table 3 shows the average plan cost and planning runtime for 50 problem instances on a 8x8 grid with 6 agents and $k = 0, 1,$ and 2. We experimented here with problem instances with a different number of randomly allocated obstacles (the "Obs." column). As expected, increasing k results in plans of higher cost and higher runtime. For both algorithms, the impact of varying the number of obstacles on the planning time follows a classical easy-hard-easy pattern: with either a few or many obstacles is easy, and it becomes harder for the middle-ground, where the problem is not under- or over-constrained.

Now we compare the results of kR-CBS and the Picat-based solvers. Since both Picat and kR-CBS solvers solve kR-MAPF optimally, their solution cost is the same, and the comparison between them is in the runtime of finding an optimal solution.[2] For this small grid, the Picat-based solver shows better performance in most settings. Indeed, compilation-based approaches are known to perform well for small and relatively dense grids (Surynek et al. 2016).

Next, we compared the Picat-based and CBS-based solvers on a larger, 32x32 grids, with 20% obstacles at random locations, with 10, 15, 20, 25 and 30 agents, and $k = 0, 1,$ and 2. These problems were harder to solve and thus we set a 5 minutes timeout for every problem instance. Table 4 shows the number of instances solved under this timeout, out of a total of 50 problem instances. Here, the results of both solvers are very similar, and there is no clear advantage to either.

Finally, we experimented with the `brc202d` DAO map described earlier, which is much larger than the 32x32 grid used in Table 4. Here, the Picat-based solver was not able to solve any problem instance, even with only 5 agents. By contrast, the CBS-based solver was able to find even optimal 2-robust solutions for some instances with 95 agents. Figure 3 shows the number of problems solved within the 5 minutes timeout from a total of 50 problem instances, with $k = 0, 1,$ and 2 and 5, 10, 15, .. ,100 agents.

In conclusion, there is no universal winner: for small grids the Picat-based solver is best, while for very large grids the CBS-based solver is much better. This trend was also observed in prior works in regular MAPF (Surynek et al. 2016; Zhou et al. 2017).

[2]For $k = 0$, the problem formulation used by kR-CBS and the Picat-based solver differs slightly, as our Picat mode do not consider edge conflict. Thus, the runtime results for $k = 0$ are not directly comparable. As discussed earlier, for $k > 0$ edge conflicts are not allowed anyhow, and the results can be compared safely.

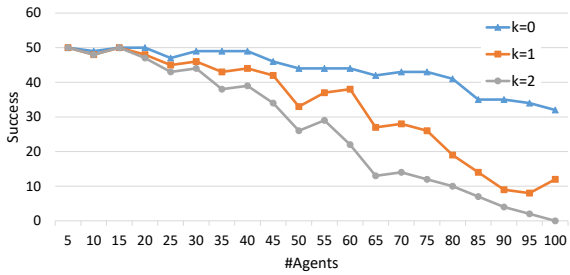

Figure 3: Success rate for kR-CBS over the `brc202d` DAO map.

6 Robust Planning and Execution

An important motivation for creating robust plans is to avoid the need to replan during execution when experiencing unexpected delays. Recent work by Ma et al. 2017 proposed an *execution policy* called MCP that is designed to minimize the number of times agents need to communicate and re-plan during execution to avoid conflicts from occurring.

As an application of robust MAPF, we implemented MCP so that the initial plan it is given is k-robust, and evaluated experimentally whether or not this results in fewer replans needed during execution. The results of these experiments are given in Figure 4. The x-axis shows the number of agents and the y-axis shows the average number of re-plans needed. The different curves represent different values of k. The results clearly show the benefit of using a k-robust plan: increasing k indeed reduces the number of modifications dramatically. For example, when $k = 2$ the average number of modifications is less than 0.5 even for 12 agents, while it is almost 3.5 when using an optimal plan (which is not robust, i.e., $k = 0$).

7 Probabilistic Robust MAPF

In some cases, it is possible to estimate the probability that agents will be delayed. This raises a probabilistic form of robustness, where the aim is to find a low-cost plan that keeps the *probability* of collisions below a given threshold p. Wagner and Choset (2017) studied this form of probabilistic robustness and proposed a corresponding solver based on M* (Wagner and Choset 2015).

We started to adapt our ideas for finding k robust solutions on this setting, as follows. Instead of defining a k-delay conflict that either exists or not, we define a *potential conflict*, which occurs if two agents have a k delay conflict for any k. That is, a pair of agents do not have a potential conflict only if their paths are vertex and edge disjoint. Using the given delay probabilities, we compute for each potential conflict the probability that it will occur. Next, we used a CBS-based approach, starting with an initial plan created by having each agent plan separately. Then, potential conflicts

Obstacles	Cost			Planning time (ms)					
	0	1	2	0		1		2	
				CBS	Picat	CBS	Picat	CBS	Picat
12	33.06	35.16	36.30	**22**	447	1,511	**1,026**	**454**	1,805
16	36.78	39.24	40.68	3,413	**431**	4,142	**1,401**	6,572	**2,590**
19	36.38	39.67	42.47	631	**434**	3,362	**1,506**	8,727	**3,235**
22	30.66	34.20	36.91	594	**341**	6,979	**1,439**	14,042	**3,104**
25	24.42	28.26	29.52	1,691	**292**	9,834	**1,206**	12,957	**1,813**
32	14.46	16.28	18.73	5,632	**94**	**95**	322	1,488	**810**

Table 3: Solution cost and runtime results for 8x8 grids with 6 agents, and a different number of obstacles.

#agents		10	15	20	25	30	35	40	45	50
k=0	Picat	**50**	**50**	**49**	**48**	**44**	**29**	15	1	**2**
	CBS	**50**	49	**49**	45	42	26	**22**	**9**	0
k=1	Picat	**50**	**50**	**47**	**35**	15	2	0	0	0
	CBS	**50**	49	42	27	**22**	**7**	**1**	0	0
k=2	Picat	**50**	**49**	**38**	**9**	0	0	0	0	0
	CBS	**50**	45	31	6	**4**	0	0	0	0

Table 4: Number of instances solved within the allocated time out. The grids used are 32x32 grids with 20% obstacles.

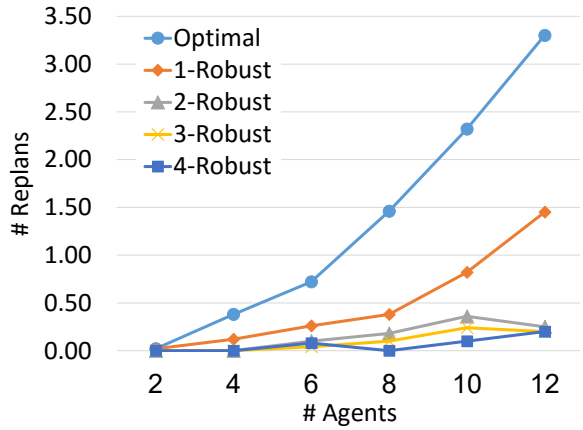

Figure 4: figure
Number of replans when using a k-robust plan

are identified and the most-likely conflict is chosen. To resolve this potential conflict, we split it to two CT nodes by setting a symmetric constraint of size k to each of the conflicting agents. To determine the correct k to use, we consider the difference between the planned arrival time of the two agents to the potential conflict's location. This process is repeated until a solution with the desired probabilistic robustness has been found. While our initial results for this approach are promising, a detailed exploration of this form of robustness is beyond the scope of this paper.

8 Related Work

As part of their work on execution policies for MAPF with delays, Ma, Kumar, and Koenig (2017) also proposed a CBS-based algorithm that aims to minimize the expected makespan. Unlike our work, they assumed prior knowledge of delay probabilities and do not provide any guarantee on the returned plan.

kR-MAPF can be viewed as a special case of *conformant planning* (Cimatti and Roveri 2000; Brafman and Hoffmann 2006), where the task is to find a plan that will be successful regardless of imperfect information about the initial state and action outcomes and without sensing capabilities. kR-MAPF is different from Nguyen et al.'s (2017) robust planning, which is for planning and not MAPF, and address a setting in which an incomplete model of the domain is available and the task is to find a plan that is likely to succeed.

Many improvements to CBS have been introduced throughout the years (Boyarski et al. 2015; Cohen et al. 2016).

Most can be applied on top of our k-robust CBS without further adjustments. An exception is the *Meta-agent CBS* (MA-CBS) (Sharon et al. 2015) algorithm, where agents with many mutual conflicts are merged into a *meta-agent* that is then treated as a joint composite agent by the low-level solver. A k-robust version of MA-CBS requires a low-level solver that is also k-robust for meta-agents consisting two or more agents. This is a topic for future work.

9 Conclusions and Future Work

In this paper, we explored a form of robustness for MAPF called k-robust, where a k-robust plan is a plan in which each agent can experience up to k delays while still preserving the ability to follow the generated plan. Several ways to obtain k-robust plan were proposed, including solvers based on A*, CBS, and constraint programming. Our experimental results show that finding a k-robust plan is possible, results in fewer re-plans during execution, and requires only a minimal increase in plan cost. There are many possible lines of future work, including adapting other MAPF solvers such as the ICTS algorithm ((Sharon et al. 2013)) to find k-robust plans, and a deeper integration of robust MAPF plans with execution policies.

10 Acknowledgements

This research was supported by the Israel Ministry of Science, the Czech Ministry of Education, and by ISF grants #844/17 to Ariel Felner and #210/17 to Roni Stern.

References

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Shimony, E.; Bezalel, O.; and Tolpin, D. 2015. Improved conflict-based search for optimal multi-agent path finding. In *IJCAI*.

Brafman, R. I., and Hoffmann, J. 2006. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence* 170(6):507–541.

Cimatti, A., and Roveri, M. 2000. Conformant planning via symbolic model checking. *J. Artif. Intell. Res.(JAIR)* 13:305–338.

Cohen, L.; Uras, T.; Kumar, T. S.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Improved solvers for bounded-suboptimal multi-agent path finding. In *IJCAI*, 3067–3074.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*.

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

Goldenberg, M.; Felner, A.; Stern, R.; and Schaeffer, J. 2012. A* Variants for Optimal Multi-Agent Pathfinding. In *Workshop on Multi-agent Path finsing. Colocated with AAAI-2012*.

Hart, P.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4:100–107.

Ma, H.; Kumar, S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *AAAI*.

Nguyen, T.; Sreedharan, S.; and Kambhampati, S. 2017. Robust planning with incomplete domain models. *Artif. Intell.* 245:134–161.

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

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

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

Standley, T. S., and Korf, R. E. 2011. Complete algorithms for cooperative pathfinding problems. In *IJCAI*, 668–673.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*.

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

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

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *AAAI*.

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

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

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

Yu, J., and LaValle, S. M. 2013a. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*.

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

Zhou, N.-F.; Kjellerstrand, H.; and Fruhman, J. 2015. *Constraint solving and planning with Picat*. Springer.