# Robust Multi-Agent Path Finding

Dor Atzmon, Roni Stern, Ariel Felner
Ben Gurion University of the Negev, Israel

Glenn Wagner
Carnegie Mellon University, USA

Roman Barták
Charles University, Czech Republic

Neng-Fa Zhou
CUNY Brooklyn College, USA

**ACM Reference Format:**
Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Barták, and Neng-Fa Zhou. 2018. Robust Multi-Agent Path Finding. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 3 pages.

## INTRODUCTION AND DEFINITIONS

In the multi-agent path-finding problem (MAPF) a plan is needed to move a set of $n$ agents from their initial location to their goals without collisions. A solution to a MAPF problem is a plan $\pi = \{\pi_1, \ldots \pi_n\}$ such that $\forall i \in [1, n]$, $\pi_i$ is a sequence of move/wait actions that move agent $a_i$ from $s_i$ to $g_i$. $\pi_i(t)$ denotes the location of $a_i$ after executing the first $t$ move/wait actions in $\pi$ without experiencing any delay. So, $\pi_i(0) = s_i$ and $\pi_i(|\pi_i|) = g_i$. We introduce and study a new type of MAPF problem where we seek a plan that is robust to $k$ unexpected delays per agent. This form of robustness is especially suitable for agents with a control mechanism that guarantees each agent is at most $k$ steps from its pre-defined plan.

*Definition 1 ($k$-delay Conflict).* A $k$-delay conflict $\langle a_i, a_j, t \rangle$ in a plan $\pi$ occurs iff $\exists \Delta \in [0, k]$ such that $\pi_i(t) = \pi_j(t + \Delta)$.

A plan is a called a *valid $k$-robust* plan if it does not have any $k$-delay conflicts. Informally, this means that no conflicts will occur even if some of the agents are delayed by up to $k$ time steps. Next, we show how to find a plan $\pi$ that is valid $k$ robust and has the minimal sum-of-costs ($\sum_{\pi_i \in \pi} |\pi_i|$), i.e., a plan that has the minimal sum-of-costs among all plans that are valid $k$ robust.

## A*-BASED SOLUTIONS

A*-based MAPF algorithms [2, 4, 5, 9] search for a plan in a $n$-agent state-space, which includes all the possible ways to place $n$ agents into $|V|$ vertices, one agent per vertex.

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.

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. To preserve optimality, however, the $n$-agent state-space needs to be modified to keep track of the last $k$ steps of each agent in each state. An A*-based MAPF solver over this state-space can find an optimal $k$-robust plan, but the size of this state space grows exponentially with $k$.

*Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.* 

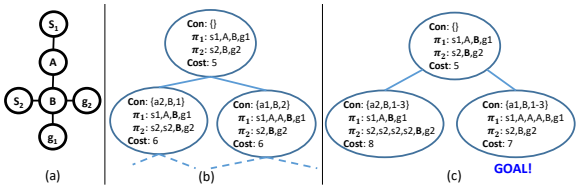


**Figure 1: a MAPF instance and its two CTs**

| | Plan cost | | | Plan time (ms) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | k=0 | k=1 | | | k=2 | | |
| m | 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**

## CONFLICT-BASED SEARCH SOLUTIONS

MAPF algorithms based on *Conflict-Based Search* (CBS) [3] find an optimal plan by iteratively identifying and resolving conflicts. Each agent in CBS is associated with 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 *consistent path* for agent $a_i$ is a path that satisfies *all* of $a_i$'s constraints, and a *consistent plan* is a plan composed only of consistent paths. CBS finds consistent plans by invoking a low-level solver for each of the agents individually. Such a plan may have conflicts, and CBS resolves them by imposing constraints on the conflicting agents. CBS generates a *constraint tree* (CT) to search for combinations of constraints and agents that will result in a plan that is conflict-free and optimal.

### $k$-Robust CBS

We propose *$k$-robust CBS* ($k$R-CBS), a $k$-robust version of CBS. It differs from CBS in how it identifies and resolves conflicts.

**Identifying $k$-delay conflicts.** CBS adds constraints to resolve conflicts, while $k$R-CBS adds constraints to resolve $k$-delay conflicts. This means that after the low-level solver returns a consistent path, $k$R-CBS simulates the resulting plan and checks for conflicts with the $k$-last locations of all other agents.

**Resolving conflicts (splitting CT nodes).** Let $N$ be a node in the CT selected to be expanded next by $k$R-CBS, and let $\langle a_i, a_j, t \rangle$ be a $k$-delay conflict in $N$. 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. $k$R-CBS generates two children to $N$, each having one of these constraints.

**Example.** Figure 1(a) shows a 2-robust MAPF problem with two agents whose start-goal pairs are $s_1$-$g_1$ and $s_2$-$g_2$, respectively.

| #agent | | 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 2: Number of instances solved within the allocated time out. The grids used are 32x32 grids with 20% obstacles.**

Figure 1(b) shows the first two levels of the CT generated by $k$R-CBS, where every node $N$ shows $N.constraints$ (labeled Con), $N.\pi_1$, $N.\pi_2$, and $N.cost$. Observe that the plan in the root has a 2-delay conflict $\langle a_2, a_1, 2\rangle$ at location $B$ for $\Delta = 1$, since $\pi_1(2) = \pi_2(1) = B$. To resolve this conflict, $k$R-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.

## Improved $k$-Robust CBS (I-$k$R-CBS)

I-$k$R-CBS 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 $t_2$.

*Definition 2 (Sound Range Constraints).* A pair of range constraints are called *sound* for $k$ robust iff all optimal $k$-robust plans satisfy at least one of these constraints.

Corollary 3. *$k$R-CBS variants that resolves conflicts only with sound pairs of range constraints are sound, complete, and return optimal $k$-robust plans.*

Corollary 4 (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 $k$-robust.*

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 there are asymmetric range constraints that are more balanced. An open question for asymmetric range constraints is how to choose which agent to impose the more restricted constraint upon.

## A DECLARATIVE SOLUTION

Some MAPF solvers express 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 [1, 7, 8, 10]. Adapting such solvers requires simple modifications. To demonstrate this, we implemented a MAPF solver using Picat [12], a logic-based programming language that has three constraint modules. The encoding we used is based on Surynek's SAT-based MAPF solver [8], 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 are 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 (For producing $k$-robust solutions: no two agents occupy the same vertex in time steps that are closer than $k$ from each other). **(3)** In every time step an agent may only transition between two adjacent locations.

## EXPERIMENTAL RESULTS

We experimented with $k$R-CBS and I-$k$R-CBS using extreme asymmetric and symmetric pairs of range constraints. Random MAPF problem instances were generated in an open 8x8 grid. Then a solver for $k = 0, 1$, and 2 was executed and the resulting plan cost and the CPU runtime were measured.

Table 1 shows the average plan cost and average CPU runtime when finding $k$-robust solutions using $k$R-CBS (labeled KR) and I-$k$R-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). Note that the plan cost was identical for all solvers and that $k = 0$ is equivalent to a standard CBS.

The $k$-robust plans do not cost much more than a plan for the basic definition of MAPF ($k = 0$). This suggests that searching for $k$-robust plans is advisable if one needs a safety zone or expects delays. Next, as expected, both I-$k$R-CBS variants runs much faster than $k$R-CBS and this improvement increases when increasing $k$ and when more agents exist. Symmetric constrains clearly outperform asymmetric 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.

We also experimented on 90 randomly generated instances with 30 agents on the `brc202d` map [6] which has 43,151 vertices. This map is very large and the cost of a $k$-robust plan is often similar to the cost of a basic plan ($k = 0$). The average plan cost was 3,818.35, 3,818.43, and 3,818.53 for $k = 0$, 1, and 2, respectively. This emphasizes the usefulness of finding a $k$-robust plan, as one can be found in such a domain without extensive cost increase. Nevertheless, finding $k$-robust plans is more time consuming. In the above experiments, 213, 284, and 381 seconds were required for $k = 0$, 1, and 2, respectively.

We also compared our Picat-based solver, with our best CBS-based solver (IKR(S)) on 32x32 grids, with 20% random obstacles with $10, 15, \ldots 50$ agents, and $k = 0, 1$, and 2. Table 2 shows the number of instances solved under a 5 minutes timeout, out of a total of 50 problem instances. The results of both solvers are very similar, and there is no clear advantage to either. We also experimented on 8x8 grids. Here, the Picat-based solver was superior in most settings. Indeed, compilation-based approaches are known to perform well for small and relatively dense grids [8]. Finally, we experimented the large `brc202d` DAO map described earlier. Here, the Picat-based solver was not able to solve any problem instance, even with 5 agents. By contrast, the CBS-based solver was able to find even optimal 2-robust solutions for some instances with 95 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 [8, 11].

## 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


[1] Esra Erdem, Doga G. Kisa, Umut Oztok, and Peter Schueller. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*.

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

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

[4] David Silver. 2005. Cooperative Pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 117–122.

[5] Trevor S. Standley. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*.

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

[7] Pavel Surynek. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*. 564–576.

[8] P. Surynek, A. Felner, R. Stern, and E. Boyarski. 2016. Efficient SAT Approach to Multi-Agent Path Finding Under the Sum of Costs Objective. In *ECAI*.

[9] Glenn Wagner and Howie Choset. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence* 219 (2015), 1–24.

[10] Jingjin Yu and Steven M. LaValle. 2013. Planning optimal paths for multiple robots on graphs. In *ICRA*. 3612–3617.

[11] Neng-Fa Zhou, Roman Barták, Roni Stern, Eli Boyarski, and Pavel Surynek. 2017. Modeling and Solving the Multi-Agent Pathfinding Problem in Picat. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*.

[12] Neng-Fa Zhou, Håkan Kjellerstrand, and Jonathan Fruhman. 2015. *Constraint solving and planning with Picat*. Springer.