Symmetry Breaking for k-Robust Multi-Agent Path Finding

Zhe Chen,[1] **Daniel Harabor,**[1] **Jiaoyang Li,**[2*] **Peter J. Stuckey**[1]
[1]Monash University
[2]University of Southern California
{zhe.chen,daniel.harabor,peter.stuckey}@monash.edu, jiaoyanl@usc.edu

Abstract

During Multi-Agent Path Finding (MAPF) problems, agents can be delayed by unexpected events. To address such situations recent work describes k-Robust Conflict-Based Search (k-CBS): an algorithm that produces a coordinated and collision-free plan that is robust for up to k delays for any agent. In this work we introduce a variety of pairwise symmetry breaking constraints, specific to k-robust planning, that can efficiently find compatible and optimal paths for pairs of colliding agents. We give a thorough description of the new constraints and report large improvements to success rate in a range of domains including: (i) classic MAPF benchmarks, (ii) automated warehouse domains, and (iii) on maps from the 2019 Flatland Challenge, a recently introduced railway domain where k-robust planning can be fruitfully applied to schedule trains.

1 Introduction

Multi-Agent Path Finding (MAPF) is a coordination problem where we need to find collision-free paths for a team of cooperating agents and is known to be NP-hard on graphs and grids (Yu and LaValle 2013; Banfi, Basilico, and Amigoni 2017). When MAPF problems are solved in practice, agents can sometimes be unexpectedly delayed during plan execution; e.g. due to exogenous events or mechanical problems. Currently, there exist two principal approaches to handle such delays. The first approach involves *robust execution policies* (Ma, Kumar, and Koenig 2017; Hönig et al. 2019). Here dependencies are introduced to guarantee that agents execute their plans in a specific and compatible order. Another approach is to reason about potential delays at the planning stage which involves computing *robust plans*.

Following (Atzmon et al. 2018) we say that a plan is k-robust if the individual path of each agent remains valid for up to k unexpected delays of that agent. In other words, provided each agent waits for no more than k timesteps on the way to its target, its plan is guaranteed to be collision-free. In addition to their execution benefits, k-robust plans are valuable in application areas where agents must maintain minimum safety distances. Such constraints appear in rail scheduling, quay crane scheduling, planning for warehouse robots and others. In these settings a k-robust plan naturally provides k timesteps of distance between agents that are are moving, while also allowing agents to stay close if they are not moving. Furthermore, k-robust plans can be used in conjunction with *robust execution policies* to benefit from both methods (Atzmon et al. 2020).

To compute k-robust plans, Atzmon et al. (2018) propose k-robust Conflict-Based Search (k-CBS), a robust variant of the popular and well known branch-and-replan strategy used for MAPF (where k = 0) (Sharon et al. 2015), and a SAT-based solution, which cannot solve problems on large grids like brc202d DAO map(Stern et al. 2019). A main problem with CBS is that the algorithm is extremely inefficient when reasoning *equivalent permutations* of conflicts that can occur between pairs of agents, such as rectangle symmetry (Li et al. 2019), corridor symmetry, and target symmetry (Li et al. 2020). Further complicating the situation is that the reasoning techniques proposed to handle these situations, do not always extend straightforwardly to the k-robust case.

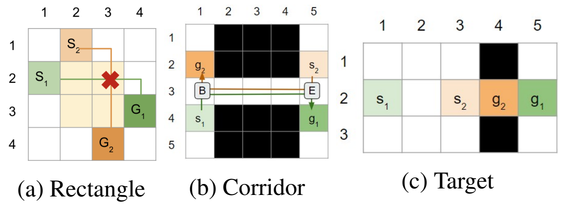
To address this gap in the literature we introduce a variety of specialised k-robust symmetry breaking constraints that dramatically improve performance for the k-CBS algorithm. Experimental results show very large gains in success rate for k-CBS; not only on classic MAPF benchmarks but also in two application specific domains: in *warehouse logistics*, where k-robust plans are desirable and in *railway scheduling* where k-robust plans are mandatory.

2 Problem Definition

We consider a multi-agent coordination problem where the operating environment is an undirected (e.g. gridmap) or directed (e.g. rail network) graph $G = (V, E)$. We restrict graph G to be a 4-neighbour grid and we place upon it m agents $\{a_1...a_m\}$. Every agent a_i is assigned an initial vertex s_i and a goal vertex g_i. Time is discretised into unit-size steps. In each timestep, agents can move to an adjacent vertex or wait at the current location. Each move or wait action has an associated unit cost.

We say that a path is k-robust if for each location v visited at time t by agent a no other agent visits the location in the time interval $[t, t + k]$. We call a k-delay *vertex conflict* the situation where agent a_i at timestep t and a_j at timestep $t' = t + \Delta, \Delta \in [0, k]$, visit the same location v. We de-

*Jiaoyang Li performed the research during her visit at Monash University.



(a) Rectangle (b) Corridor (c) Target

Figure 1: Examples of rectangle, corridor and target conflicts when k=0, reproduced from (Li et al. 2020).

note such a conflicts by $\langle a_i, a_j, v, t, \Delta\rangle$. When $k = 0$ it is further possible for two agents to cross the same edge in opposite directions at the same time, resulting in a so-called *edge conflict*. Notice however that with $k > 0$ such a crossing will always result in a vertex conflict. Thus for $k > 0$ we do not need to model edge conflicts.

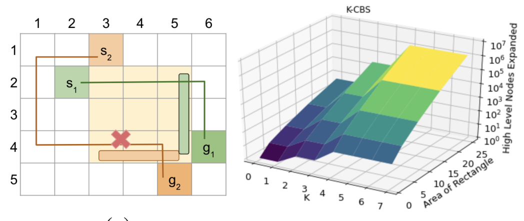
A solution to the problem (equiv. a *k-robust plan*) is a set of paths, one for each agent a_i, which moves each agent a_i from its start location s_i to its goal location g_i such that there are no k-delay conflicts with the plans of any other agent. Our objective is to find a k-robust plan which minimises the sum of all individual path costs (SIC).

3 k-Robust Conflict-Based Search

k-Robust Conflict-Based Search (k-CBS) (Atzmon et al. 2018) is a two-level search algorithm specialised from classical MAPF (k=0) (Sharon et al. 2015). At the high-level, k-CBS searches (in a best-first way) a binary constraint tree (CT) where each node is a complete assignment of paths to agents (i.e. a plan). The process of finding such single agent paths constitutes the low level of k-CBS. Here individual agents find paths (via A*) from start to target while subject to a set of collision-avoiding *constraints*.

The search process of k-CBS proceeds as follows: At each iteration k-CBS expands the CT node with lowest f-cost. If the current node is conflict-free then that node is a goal and the search terminates having found a least-cost feasible plan. Otherwise, the current node must contain at least one pair of agents that are in collision. Suppose for example that the conflict is $\langle a_i, a_j, v, t, \Delta\rangle$. This situation occurs when agent a_i at timestep t and a_j at timestep $t' = t + \Delta, \Delta \in [0, k]$, visit the same location v. To resolve the conflict k-CBS replans each of the two affected agents and thus generates two new candidate plans. To each child node is added a *time range constraint* which resolves the situation now and in all future descendant nodes derived from each respective child. The constraint $\langle a_i, v, [t, t + k]\rangle$ says that agent a_i is not allowed to occupy vertex v at any timestep in the range $[t, t + k]$. The second child node, where a_j is replanned, receives a similar constraint: $\langle a_j, v, [t, t + k]\rangle$.

k-CBS continues in this way, splitting and searching, while the current CT node contains any conflict. This approach is solution complete and optimal. It guarantees to find a k-robust plans (Atzmon et al. 2018), if any such plan exists, since the union of valid plans permitted by the two child nodes is the same as at the parent node (i.e. adding constraints does not eliminate valid solutions).



(a) (b)

Figure 2: (a)A collision-free solution, orange path, is eliminated, if we simply put "thick" barrier constraints to resolve a k-delay rectangle conflict when $k = 4$. (b) How number of high-level nodes expanded by k-CBS increases with k and the size of the rectangle area.

Conflict selection strategies: Deciding which conflict to resolve next is critical to the success of (k-)CBS. In this work we follow Boyarski et al. (2015) where authors classify conflicts as *cardinal*, *semi-cardinal* and *non-cardinal*:

- A conflict C is *cardinal* if replanning for any agent involved in the conflict increases the SIC.
- A conflict C is *semi-cardinal* if replanning for one agent involved in the conflict always increases the SIC while replanning for the other agent does not.
- A conflict C is *non-cardinal* otherwise.

In Boyarski et al. (2015) it is shown that resolving cardinal conflicts first can dramatically reduce the size of the resulting CBS tree. After all cardinal conflicts are resolved we choose semi-cardinal conflicts and finally non-cardinal. Similar to that work we use a Multi-valued Decision Diagram (MDD) to classify conflicts. Each MDD records all nodes (i.e., vertex-time pairs) that can appear on an optimal path for each agent. (Semi-)cardinal conflicts require the conflict node to be a *singleton* for (resp. one) both agents, i.e. all optimal paths must pass through the node.

CBS Heuristics (CBSH): Different from (Atzmon et al. 2018) but following (Felner et al. 2018), we also exploit known cardinal conflicts to derive a *minimal cost increase* heuristic for our high-level A* search. This strategy is known to improve the performance of CBS and has become a common approach in leading MAPF (k = 0) solvers.

4 k-Symmetries

Symmetries in MAPF occur when two agents repeatedly run into one another along equivalent individually optimal paths. Figure 2a shows an example for $k = 4$. Notice that each agent has available a number of optimal-cost paths. However every optimal path of agent a_1 is in collision with *every* optimal path of agent a_2 and vice versa. Without detecting such situations k-CBS will repeatedly split, node after node, growing the CT tree in order to enumerate all possible collisions in the highlighted rectangle area. Figure 2b shows the result: each time k-CBS splits, it potentially doubles the amount of search yet remaining to reach the goal node. The size of the CT tree grows exponentially as the size of the rectangle area increases, causing k-CBS to return timeout failure. Note that as k increases, the size of the CT tree also grows exponentially, which makes the problem extremely difficulty to solve. All this difficulty can be avoided by recognising there exists a simple optimal strategy: one of the two agents has to wait. For $k = 0$ one agent must wait for one timestep. For $k > 0$ one agent may need to wait for more than one timestep. However, as k grows large the problem can permit optimal-cost bypass routes that allow one agent to avoid the rectangle area entirely. For example in Figure 2a, when $k \geq 4$ the solid-orange-line path becomes optimal and agent a_2 can reach its target without waiting.

In this work we consider three distinct symmetric situations which can appear in k-robust MAPF:

- Rectangle Symmetries, as illustrated in Figure 1a.
- Corridor Symmetries, as illustrated in Figure 1b.
- Target Symmetries, as illustrated in Figure 1c.

Rectangle symmetries have previously been studied in the context of MAPF (k = 0) (Li et al. 2019) while Corridor and Target Symmetries have only recently been introduced (Li et al. 2020), again in the context of MAPF (k = 0). Each time authors show that symmetries are common in a range of standard benchmarks and they report dramatic gains in performance when these symmetric situations are resolved via specialised reasoning techniques. We adapt each of these ideas to k-robust planning and we report similarly strong results. Generalising these constraints is not simply academic. As we show in the experimental section, k-robust plans are needed for important practical problems. Without the development of suitable algorithmic techniques such problems will remain out of reach to MAPF planners.

5 k-Robust Rectangle Reasoning

Rectangle symmetries arise in k-CBS when the paths of two agents cross topologically. The agents are heading in the same directions (e.g. down and right in Figure 2a) and there exists for each many different but equally shortest paths which arise from re-ordering their individual moves. In such cases the standard replanning strategy of k-CBS does not help to immediately resolve the problem.

Definition 1 (k-delay rectangle conflict)**.** *A k-delay rectangle conflict between two agents occurs if all paths with cost between optimal and optimal+k (inclusive) for the two agents that enter a given rectangular area have a k-delay vertex conflict in the rectangular area.*

Rectangle conflicts pose substantial challenges for k-robust planning in general and for k-CBS in particular because:

- the agents do not have to reach positions at exactly the same time to have a conflict;
- the delay caused by the conflict can be up to $k + 1$; and
- for $k \geq 2$ an agent can leave and enter the conflicting area without adding more than k steps to its path.

To address these challenges we follow Li et al. (2019) where authors develop *barrier constraints*: a pruning strategy that can efficiently resolve rectangle conflicts for CBS with $k = 0$ in a single branching step. With respect to Figure 2a, one barrier constraint prohibits agent a_1 from occupying cells (5, 2), (5, 3), and (5,4) at timesteps 3, 4, and 5, respectively. Similarly, the other barrier constraint prohibits agent a_2 from occupying cells (3, 4), (4, 4), and (5, 4) at timesteps 3, 4, and 5, respectively. Notice that each barrier blocks all equivalent shortest paths and forces one agent or the other to wait, thus resolving the conflict.

A straightforward idea for extending this strategy to $k > 0$ would be to increase the "thickness" (number of timesteps) of the barrier constraints. We therefore introduce *temporal barrier constraints*, which unify the (temporal) range constraints of (Atzmon et al. 2018) and the (spatial) barrier constraints of (Li et al. 2019).

Definition 2 (Temporal barrier constraint)**.** *Given a vertex-time pair $p = (u, t)$ we denote with $ot(p, v)$ the* optimal time *to reach vertex $v = (v_x, v_y)$ from vertex $u = (u_x, u_y)$ starting at timestep t. We compute the optimal time by adding to t the Manhattan distance from u to v: $ot(p, v) = t + |v_x - u_x| + |v_y - u_y|$. A w* temporal barrier constraint, *denoted $B(a_x, V, p, w)$, forbids an agent a_x, currently at vertex-time pair p, from visiting any vertex $v \in V$ at its optimal time or up to w timesteps later. That is, the barrier constraint $B(a_x, V, p, w)$ is the set of time-range vertex constraints $\langle a_x, v, [ot(p, v), ot(p, v) + w]\rangle, v \in V$.*

To resolve the k-robust rectangle conflict in Figure 2a, one might think that we can replace the two barrier constraints used by (Li et al. 2019) with two k temporal barrier constraints at the exit of the rectangle. However, this approach is not complete! For example, when $k = 4$, the orange line is a collision-free path for agent a_2, no matter what path agent a_1 takes. But this solution is eliminated by the $w = 4$ temporal barrier constraints. Therefore, we propose a novel approach that enlarges the rectangle area being reasoned about and adding temporal barrier constraints based on it with adjusted thickness so that we can preserve the completeness and optimality by taking into consideration such bypasses.

5.1 Enlarging Rectangle and Shifting Borders

Given a 4-neighbor grid map, we define a *rooted rectangle R* as the set of vertices occurring in the rectangle defined by the two corner points $D = (D_x, D_y)$ the root corner, and $E = (E_x, E_y)$ the opposite corner. As shown in Figure 3, the illustrated rectangle is defined by $D = (4, 3)$ and $E = (5, 4)$. Given D and E, we define four sides: $S1(R) = \{(D_x, D_y)..(E_x, D_y)\}$, $S2(R) = \{(D_x, D_y)..(D_x, E_y)\}$, $S3(R) = \{(E_x, D_y)..(E_x, E_y)\}$, and $S4(R) = \{(D_x, E_y)..(E_x, E_y)\}$. We define the *shifted side $Sj(R)_l, j \in [1, 4], l \in [0, \lfloor \frac{k}{2} \rfloor]$*, as the side $Sj(R)$ shifted away from the center of R by l grid locations, note the $Sj(R)$ cannot be shifted beyond the start or goal locations of agents involved in the conflict. Define the y-shifted start location D^y_l as the root corner D shifted l locations along y-axis away from R, and the x-shifted start location D^x_l as the start location shifted l locations along y-axis away from R. Similarly define E^y_l and E^x_l.

Given these definitions, the shifted barrier constraints with some particular thickness can define a *k-delay rectangle conflict*, that is all paths for two agents a_1 and a_2 that

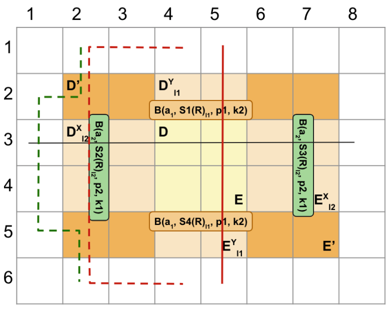


Figure 3: (1) Example of temporal barrier constraints. In this case $k_1 = 2$, $S1(R)_{l_1}$ and $S4(R)_{l_1}$ are shifted for $l_1 = 1$ grid location away from $S1(R)$ and $S4(R)$, and $k_2 = 4$ so $S2(R)_{l_2}$ and $S3(R)_{l_2}$ are shifted for $l_2 = 2$ timesteps from $S2(R)$ and $S3(R)$. (2) The yellow area is the rectangle R. Adding the light and dark orange areas gives rectangle R'. (3) The solid red line is a path traversing rectangle R and must have k-delay conflicts. The dashed red line is a path bypassing the rectangle R, but traversing rectangle R' and must have k-delay conflicts as well. The green dashed line is a collision-free path bypassing the rectangle R'. (4) Extended constraints (Step Temporal Barrier Constraint) defined in Section 5.2 fill the gaps in the barriers for the dark orange area.

cross these barriers must inevitably result in a k-delay conflict.

Theorem 1. *Consider a k-robust MAPF problem with an arbitrary rooted rectangle R defined by corners D and E and two arbitrary integers $0 \leq k_1, k_2 \leq k$. Let root time rt be the minimum of the timestep when agent a_1 or a_2 reaches the root corner D, $l_1 = \lfloor \frac{k_1}{2} \rfloor$, $l_2 = \lfloor \frac{k_2}{2} \rfloor$, $p_1 = (D^y_{l_1}, rt - l_1)$, and $p_2 = (D^x_{l_2}, rt - l_2)$. If the paths for agents a_1 and a_2 violate all of the four constraints:*

- a_1 Entrance: $B(a_1, S1(R)_{l_1}, p_1, k_2)$,
- a_1 Exit: $B(a_1, S4(R)_{l_1}, p_1, k_2)$,
- a_2 Entrance: $B(a_2, S2(R)_{l_2}, p_2, k_1)$,
- a_2 Exit: $B(a_2, S3(R)_{l_2}, p_2, k_1)$,

the paths of the two agents have a k-delay vertex conflict.

Proof. Assume that $S1(R)$ is the top of the rooted rectangle R, $S4(R)$ the bottom, $S2(R)$ the left, and $S3(R)$ the right. The other cases follow similarly. $S1(R)_{l_1}$, $S4(R)_{l_1}$, $S2(R)_{l_2}$, $S3(R)_{l_2}$ are corresponding sides shifted away from R. These shifted sides define a new larger rectangle R' with two corner points D' and E', as shown in Figure 3.

In order to violate the first two constraints, agent a_1 has to enter the top of rectangle R' through a vertex v on $S1(R)_{l_1}$ at timestep $ot(p_1, v)$ or up to k_2 timesteps later, and leave from the bottom through a vertex v on $S4(R)_{l_1}$ at timestep $ot(p_1, v)$ or up to k_2 timesteps later, so it can wait for at most k_2 timesteps in R' but cannot leave R', since this would require at least extra $2l_2 + 2$ timesteps comparing with a shortest path across R' but $2l_2 + 2 > k_2$. Every vertex v it visits in R' is visited within the time range $[ot(p_1, v), ot(p_1, v) + k_2]$.

Similarly agent a_2 enters the left of rectangle R' through a vertex v on $S2(R)_{l_2}$ at timestep $ot(p_2, v)$ or up to k_1 timesteps later, and leaves the right through a vertex v on $S3(R)_{l_2}$ at timestep $ot(p_2, v)$ or up to k_1 timesteps later. Again it cannot leave R', since this would require at least extra $2l_1 + 2$ timesteps and it can take at most k_1 wait in R'. Every vertex v it visits in R' is visited within the time range $[ot(p_2, v), ot(p_2, v) + k_1]$.

Now since agent a_1 crosses from top to bottom and agent a_2 from left to right, their paths must cross, say at vertex $v \in R'$. Assume that agent a_1 visits vertex v at $t_1 \in [ot(p_1, v), ot(p_1, v) + k_2]$ while agent a_2 visits vertex v at $t_2 \in [ot(p_2, v), ot(p_2, v) + k_1]$. Since $ot(p_1, v)$ and $ot(p_2, v)$ share the same root time rt, $ot(p_1, v) = ot(p_2, v)$, and k_1 and k_2 are both less than or equal to k, which make $|t_1 - t_2| \leq k$, there is a k-delay conflict $\langle a_1, a_2, v, t_1, t_2 - t_1 \rangle$ (if $t_1 \leq t_2$) or $\langle a_2, a_1, v, t_2, t_1 - t_2 \rangle$ (if $t_1 > t_2$). □

5.2 Extending Temporal Barrier Constraints

The temporal barrier constraints proposed in Theorem 1 do not cover all situations where two agents must have a k-delay vertex conflict in a rectangle area because they do not cover the entire circumference of rectangle R'. As the red dashed line path shown in Figure 3 illustrates, if agent a_1 enters the top of rectangle R' through corner D' at the optimal time, and leaves from left-bottom vertex at the optimal time. This path (red dashed line) traverses through $S2(R)_{l_2}$ and is $2\lfloor \frac{k}{2} \rfloor$ timesteps longer than paths entering through $S1(R)_{l_1}$ optimally and leaving through $S4(R)_{l_1}$ optimally. Now agent a_2 enters R' via $S2(R)_{l_2}$ at the optimal time or up to k_2 timesteps later, so they still have a k-delay conflict.

We cannot simply extend temporal barrier constraints to the entire circumference of R' to eliminate such conflicts, as it would also eliminate the green dashed collision-free path in Figure 3. We thus consider a stronger reasoning method.

Definition 3 (Step temporal barrier constraint)**.** *Denoted by $B_{step}(a_x, S, p, k')$ this constraint reduces the temporal width of the barrier $B(a_x, S, p, k')$ by a value of 2 for each step away from the original barrier. Let $l' = \lfloor \frac{k'}{2} \rfloor$, we add range constraints $\langle a_x, v, [ot(p, v), ot(p, v) + k' - 2d] \rangle$ for each vertex v in the same line as S at distance $d \in [1, l']$ from the original barrier.*

For the rectangle R' in Figure 3, the step temporal barrier constraint $B_{step}(a_1, S1(R)_{l_1}, p_1, k_2)$ consists of $B(a_1, S1(R)_{l_1}, p_1, k_2)$ and the additional range constraints $\langle a_1, (2, 2), [rt + 1, rt + 1] \rangle$, $\langle a_1, (3, 2), [rt, rt + 2] \rangle$, $\langle a_1, (6, 2), [rt + 1, rt + 3] \rangle$, and $\langle a_1, (7, 2), [rt + 2, rt + 2] \rangle$. Notice how the time ranges shrink further from the original barrier. This prevents the red dashed path in Figure 3. We can now extend Theorem 1: if the paths of agents a_1 and a_2 violate all of the four following constraints:

- a_1 *Entrance:* $B_{step}(a_1, S1(R)_{l_1}, p_1, k_2)$
- a_1 *Exit:* $B_{step}(a_1, S4(R)_{l_1}, p_1, k_2)$
- a_2 *Entrance:* $B_{step}(a_2, S2(R)_{l_2}, p_2, k_1)$
- a_2 *Exit:* $B_{step}(a_2, S3(R)_{l_2}, p_2, k_1)$

the paths of the two agents have a k-delay vertex conflict. The proof is similar.

5.3 Resolution of k-Delay Rectangle Conflicts

We can always resolve a rectangle conflict by four-way branching adding to each branch one of the constraints: a_1 Entrance, a_1 Exit, a_2 Entrance, and a_2 Exit; since we know that one of them must be violated in any solution. But in many cases, we can correctly resolve the rectangle conflict by two-way branching on the constraints a_1 Exit and a_2 Exit, if both agents satisfy the following condition:

Condition 1 All possible paths that traverse the exit barrier must also traverse the entrance barrier.

We can use a k-MDD to check that the condition is satisfied. A k-MDD for agent a_i is a modified Multi-Valued Decision Diagram (MDD) (Boyarski et al. 2015) that stores all paths of agent a_i from start to goal with path length no more than k above the optimal. MDDs are widely used in CBS algorithms to store all optimal paths, the k-MDD is a direct extension.

If either agent's k-MDD shows that paths that bypass the entrance barrier and traverse the exit barrier exist, the given conflict cannot be resolved by two-way branching on the given barriers, as it may eliminate conflict free paths that bypass the entrance barrier. If any combination of k_1 and k_2 lead to **Condition 1** being satisfied, the given conflict can resolved by two-way branching on the exit barriers. If none of the combinations satisfy **Condition 1**, the given conflict will be resolved as a normal conflict (i.e. we never do four-way branching).

We can classify k-delay rectangle conflicts as cardinal, semi-cardinal and non-cardinal using the k-MDD:

- A k-delay rectangle conflict is *cardinal*, if all paths in the k-MDDs of both agents traverse the exit barrier, which means that replanning for any agent involved in the conflict increases the SIC.
- A k-delay rectangle conflict is *semi-cardinal*, if only one agent has all paths in its k-MDD traverse the exit barrier, which again means that replanning this agent involved always increases the SIC while replanning for the other agent does not.
- A k-delay rectangle conflict is *non-cardinal* if both agents have paths in their k-MDD bypass their exit barriers.

We can then prioritize selecting conflict based on cardinality.

5.4 Detecting Rectangle Conflicts

When we detect a vertex conflict during CBS we need to recognise that it's actually a rectangle conflict, in order to perform rectangle symmetry breaking.

Assume that agents a_1 and a_2 have a k-delay vertex conflict $\langle a_1, a_2, v, t, \Delta\rangle$. Let d_1 and d_2 be the moving directions when agents a_1 and a_2 enter vertex v, respectively. If they are the same directions, then there is an earlier vertex conflict (where they both came from). If they are opposite directions, then there is no rectangle conflict.

So assume d_1 and d_2 are orthogonal directions. Let $(B1_x, B1_y)$ be the earliest vertex in the path of agent a_1 where all moves from here to v are in direction d_1 or d_2,

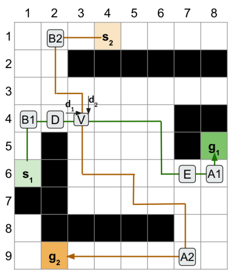


Figure 4: Detecting a rectangle conflict when $k = 2$.

similarly define $(B2_x, B2_y)$ for agent a_2. Let t_{b1} be the earliest timestep that a_1 visits $(B1_x, B1_y)$ and t_{b2} be the earliest timestep that a_2 visits $(B2_x, B2_y)$. Let $(A1_x, A1_y)$ be the latest vertex in the path of agent a_1 where all moves from v to here are in directions d_1 or d_2, similarly define $(A2_x, A2_y)$ for agent a_2.

Define D_x to be the closer of $B1_x$ and $B2_x$ to v_x, similarly for D_y. Define E_x to be the closer of $A1_x$ and $A2_x$ to v_x, similarly for E_y. Figure 4 shows an example.

Let $rt_1 = t_{b1} + |B1_x - D_x| + |B1_y - D_y|$ and $rt_2 = t_{b2} + |B2_x - D_x| + |B2_y - D_y|$. We define the root time $rt = \min(rt_1, rt_2)$.

We then for each value $k_1 \in \{0, 1, \ldots, k\}$ and $k_2 \in \{0, 1, \ldots, k\}$ check if the agents satisfy **Condition 1** using the Step Temporal Barriers defined by these values. We do so by examining the k-MDD for each agent, temporarily blocking its entrance barrier and seeing if its exit barrier is still reachable. If not then **Condition 1** holds. We try the values for k_1 and k_2 in decreasing order to find the strongest blocking conditions possible. If $k_1 = a, k_2 = b$ satisfies the conditions we don't investigate any pairs (a', b') where $a' \leq a$ and $b' \leq b$.

Figure 4 shows an example of the rectangle detection. In this example, agents a_1 and a_2 have a 2-delay conflict $\langle a_1, a_2, (3, 4), 4, 2\rangle$ with d_1 pointing right and d_2 pointing down. Then B_1 is located at $(1, 4)$ with $t_{b1} = 2$ and $rt_1 = 3$, and B_2 located at $(2, 1)$ with $t_{b2} = 2$ and $rt_2 = 5$. Root time $rt = min(t_{b1}, t_{b2}) = 3$. We detect that for the case $k_1 = k_2 = 2$ the Step Temporal Barriers defined by these values satisfy **Condition 1**.

We call the detection of rectangles and the associated branching K-CBSH-RM (it is a generalization of the RM technique defined by Li et al. (2019)).

Theorem 2. *K-CBSH-RM is correct.*

Proof. Since **Condition 1** holds for the chosen Step Temporal Barriers, and using (extended) Theorem 1, each pair of paths that violate the exit barriers must conflict. Hence the two-way branching removes no solutions, and advances the search since it is violated by the current paths. □

6 k-Robust Corridor Reasoning

A *corridor* from B to E is a chain of nodes C where nodes in C except B and E each have exactly two neighbours, and B and E have exactly one neighbour in C. Figure 1b shows a corridor where $B = (3, 1)$, $C =$

$\{(1,3),(2,3),(3,3),(4,3),(5,3)\}$ and $E = (5,3)$. A *corridor conflict* (Li et al. 2020) occurs when two agents have a vertex or edge conflict occurring in a corridor. In the example agents a_1 and a_2 conflict at vertex (3, 3). Simply adding a vertex conflict constraint will not resolve the conflict, they will continue to conflict in the corridor. Li et al. (2020) introduce corridor symmetry breaking constraints which we extend here for k-delay conflicts.

The difference between corridor conflicts for k-robust CBS and normal corridor conflicts is that agents need to occupy vertexes for extra timesteps to avoid k-delay conflicts. Assuming there is a corridor with length of l between vertex B and vertex E, and a k-delay conflict in the corridor with agent a_1 (a_2) moving from B to E (resp. E to B). Let t_1 (resp. t_2) be the earliest timestep when agent a_1 (a_2) is able to reach vertex E (resp. B).

Clearly when planning a k-robust solution, any path of a_1 using the corridor that reaches vertex E at or before timestep $t_2 + l + k$ must conflict with any paths of a_2 using the corridor that reach vertex B at or before timestep $t_1 + l + k$. But there may be alternate paths the agents can take to reach B or E. Assuming agent a_1 can reach vertex E at timestep t'_1 without using the corridor and agent a_2 can reach vertex B at timestep t'_2 without traversing the corridor. Hence in planning a k-robust solution, any path of a_1 that reaches vertex E at or before timestep $min(t'_1 - 1 + k, t_2 + l + k)$ must conflict with any path of a_2 that reaches vertex B at or before timestep $min(t'_2 - 1 + k, t_1 + l + k)$.

Hence the constraint $\langle a_1, E, [0, min(t'_1 - 1 + k, t_2 + l + k)] \rangle \vee \langle a_2, B, [0, min(t'_2 - 1 + k, t_1 + l + k)] \rangle$ must hold in all solutions. To handle the corridor constraint we branch on this disjunction. Clearly

Theorem 3. *k-robust Corridor Reasoning is correct.* □

7 k-Robust Target Reasoning

A *target conflict* (Li et al. 2020) occurs when one agent a_2 reaches its goal vertex g_2 at timestep l, and another agent a_1 conflicts with agent a_2 at vertex g_2 at some later timestep $t, t \geq l$. Consider Figure 1c where agent a_2 reaches its goal cell (4,2) at timestep 1, and then agent a_1 tries to traverse cell (4,2) at timestep 3. Simply adding the constraint $\langle a_1, (4,3), [3,3] \rangle$ causes a_1 to wait before entering cell (4,3) at timestep 4 and then the conflict reoccurs.

To avoid this Li et al. (2020) resolve the conflict by branching on $et_2 \leq t \vee et_2 > t$ where et_2 is the end time for agent a_2. In the first case, since agent a_2 finishs before or at timestep t, agent a_1 can never use location g_2 at timestep t or after. In the second case agent a_2 cannot finish before timestep $t + 1$ freeing up the location for agent a_1.

Planning a k-robust solution requires the avoiding of k-delay conflict. Therefore, we branch on $et_2 \leq t + k \vee et_2 > t + k$. The first case forces agent a_2 to finish before timestep $t + k$ preventing agent a_1 (or any other agent) from using vertex g_2 at timestep t, the second case forces agent a_2 not to finish earlier so vertex g_2 at timestep t is freed up for agent a_1. Again clearly

Theorem 4. *k-Robust Target Reasoning is correct.* □

Note that to handle target symmetries we have to update the low-level path finder for agents to take into account new kinds of constraints where we restrict the end time of an agent, and where we prevent any agent from using a location from some time point onwards. Both are straightforward additions. See Li et al. (2020) for details.

8 Experiments

The implementation is based on CBS with rectangle, corridor and target reasoning of Li et al. (2020) and support for k-robust planning is added on top of it. It is programmed in C++ and experiments were performed on a server with AMD Opteron 63xx class CPU and 32 GB RAM. For each map, we keep increasing the number of agents, and for each number of agent we solve 25 different instances. The time limit is set to 90s for each instance. In result plots, K-CBS is the current state of art algorithm to plan k-robust plan proposed by Atzmon et al. (2018),which is selected as the baseline, K-CBSH is our extension of K-CBS with heuristics (Section 3), RM adds k-robust Rectangle reasoning, C adds k-robust Corridor Reasoning, and T adds k-robust Target Reasoning.

Experiment 1: Game Maps The MAPF research community have developed a series of benchmark maps from games (Stern et al. 2019). They are available from movingai.com. We use 25 even scenarios from movingai.com to evaluate our algorithms. We run experiments on following representative maps: *Brc202d*, *Den520d*, *Random-32-32-10*, and *maze-128-128-1*.

Figure 5 shows the experiment results on grid game maps. K-CBSH-RM based algorithms shows significant higher success rate compared with K-CBS on *Brc202d*, *Den520d*, and *Random-32-32-10*. Although k-robust corridor reasoning does not help on these three maps and k-robust target reasoning slightly helps, they effectively improves the success rate on *maze-128-128-1*. As k increases, the problem becomes harder, and the success rate drops, but the symmetry-breaking algorithms still show significant advantages over K-CBS.

Experiment 2: Warehouse Map We use a *31×79 Warehouse map* (Li et al. 2020) with randomly generated problems to evaluate the performance of robots in warehouse system. Figure 5e shows that k-CBSH significantly improves success rate, k-robust target reasoning and corridor reasoning helps to further improve the success rate of K-CBSH-RM. Here target reasoning is clearly very important.

Experiment 3: Simplified Railway System The Flatland challenge is a railway scheduling challenge (Swiss Federal Railways 2019) provides a simplified railway simulator using a directed grid map, where trains cannot move backwards. Railway systems have headway control, one train cannot start to enter a railway block if another train currently occupies the block. Hence the railway domain requires $k = 1$ robust plans. Our experiments use flatland-rl v2.0.0 to generate experiment problems. Note, no target conflicts can occur in the Flatland challenge scenarios since trains "disappear" when reaching their destination.

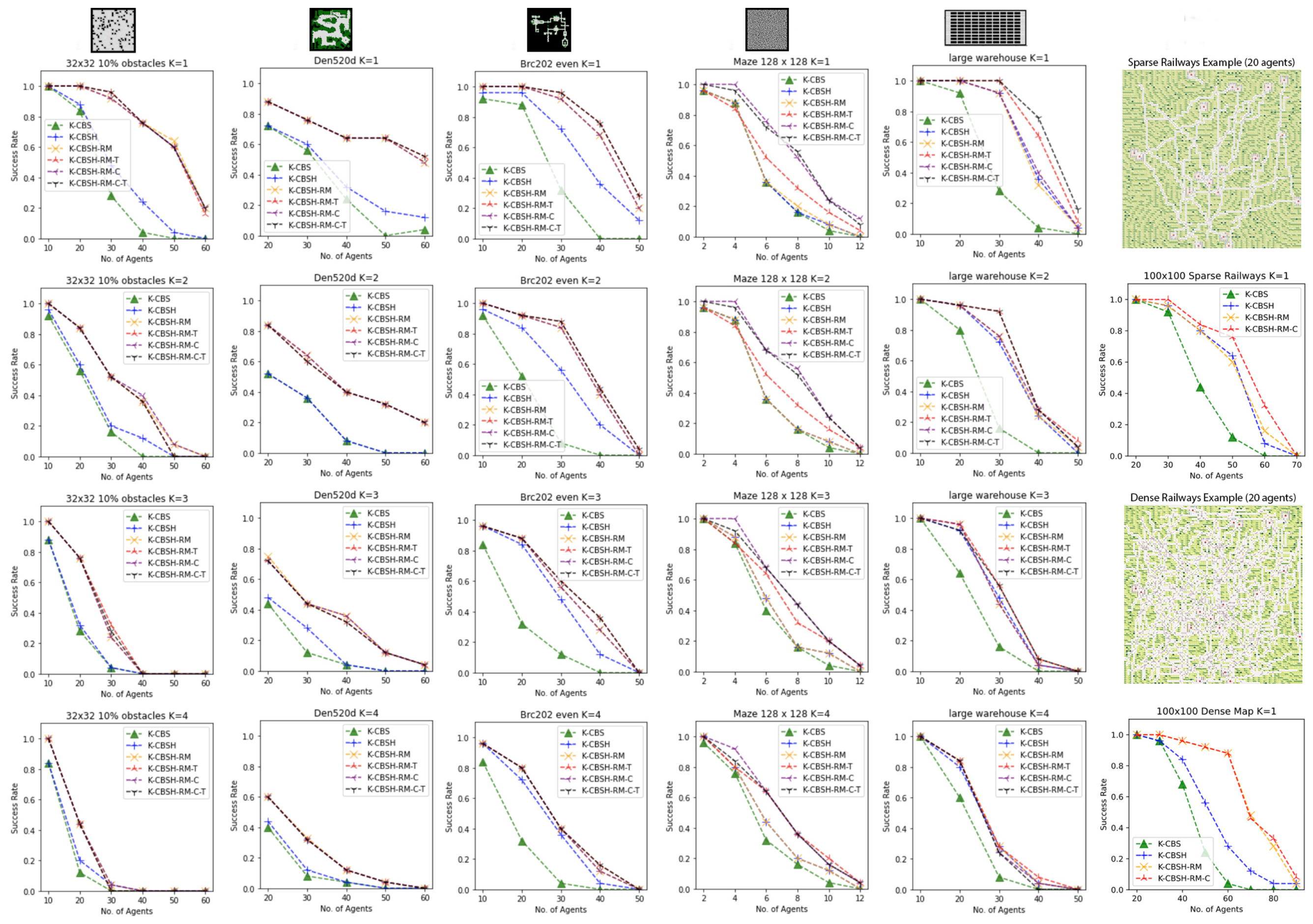

(a) Random 32x32 (b) Den520d (c) Brc202d (d) Maze 128x128 (e) Warehouse map (f) Flatland Railways

Figure 5: Success rate versus number of agents on different problems.

We have two settings for evaluation: (1) a *100×100 fixed dense map* contains fixed 140 start and goal locations, and each pair of start and goal are connected by 5 railways; and (2) a *100×100 incremental sparse map* has one start and goal location per agent, and each pair of start and goal are only connected by 1 railway.

The experiment on railway maps, Figure 5f shows that K-CBSH and K-CBSH-RM performs substantially better than K-CBS. k-robust corridor reasoning helps further improve performance on sparse railways map. Clearly the rectangle methods are more important on the denser map, where more symmetric conflicts are possible, and k-robust corridor reasoning is more important on sparse maps.

Ratio of Rectangle Conflicts The Figure 6 shows the percentage of k-delay rectangle conflicts among all resolved conflicts using K-CBSH-RM-C-T as k increases. The statistics on conflicts are derived from Experiment 1 and Experiment 2. Clearly, as k increases, the percentage of k-delay rectangle conflicts rises on the maps where rectangle conflicts can occur, hence the importance of k-robust rectangle reasoning is demonstrated.

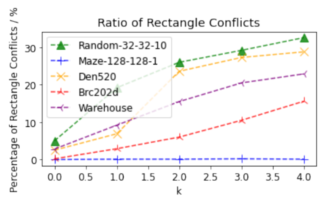


Figure 6: Percentage of k-delay rectangle conflicts among all conflicts as k increases when using K-CBSH-RM-C-T.

9 Conclusions and Future Work

This research introduces symmetry resolution methods for generating k-robust plans, which are vital for robust (e.g. warehouse robotics) and safe (e.g. railway scheduling) multi-agent plans. Symmetry reasoning methods improve dramatically on K-CBS, with k-robust rectangle reasoning being the most important, while k-robust corridor and target reasoning can further improve the performance.

References

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2018. Robust multi-agent path finding. In *Eleventh Annual Symposium on Combinatorial Search*.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2020. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research* 67: 549–579.

Banfi, J.; Basilico, N.; and Amigoni, F. 2017. Intractability of time-optimal multirobot path planning on 2d grid graphs with holes. *IEEE Robotics and Automation Letters* 2(4): 1941–1947.

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*.

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. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*, 83–87.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and robust execution of mapf schedules in warehouses. *IEEE Robotics and Automation Letters* 4(2): 1125–1131.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019. Symmetry Breaking Constraints for Grid-based Multi-Agent Path Finding. In *Proceedings of the National Conference on Artificial Intelligence, Honolulu, HI, USA*, volume 27.

Ma, H.; Kumar, T. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 3605–3612.

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

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *arXiv preprint arXiv:1906.08291* .

Swiss Federal Railways. 2019. Flatland Challenge. Https://www.aicrowd.com/challenges/flatland-challenge.

Yu, J.; and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.