

Multi-Agent Pathfinding with Continuous Time

Anton Andreychuk^{1,3}, Konstantin Yakovlev^{1,2}, Dor Atzmon⁴ and Roni Stern^{4*}

¹Federal Research Center “Computer Science and Control” of Russian Academy of Sciences

²National Research University “Higher School of Economics”

³Peoples’ Friendship University of Russia (RUDN University)

⁴Ben-Gurion University of the Negev

andreychuk@mail.com, yakovlev@isa.ru, {dor,atzmon}@post.bgu.ac.il

Abstract

Multi-Agent Pathfinding (MAPF) is the problem of finding paths for multiple agents such that every agent reaches its goal and the agents do not collide. Most prior work on MAPF was on grids, assumed agents’ actions have uniform duration, and that time is discretized into timesteps. We propose a MAPF algorithm that does not rely on these assumptions, is complete, and provides provably optimal solutions. This algorithm is based on a novel adaptation of Safe interval path planning (SIPP), a continuous time single-agent planning algorithm, and a modified version of Conflict-based search (CBS), a state of the art multi-agent pathfinding algorithm. We analyze this algorithm, discuss its pros and cons, and evaluate it experimentally on several standard benchmarks.

1 Introduction

Multi-Agent Pathfinding (MAPF) is the problem of finding paths for multiple agents such that every agent reaches its goal and the agents do not collide. MAPF has topical applications in warehouse management [Wurman *et al.*, 2008], airport towing [Morris *et al.*, 2016], autonomous vehicles, robotics [Veloso *et al.*, 2015], and digital entertainment [Ma *et al.*, 2017b]. While finding a solution to MAPF can be done in polynomial time [Kornhauser *et al.*, 1984], solving MAPF optimally is NP Hard under several common assumptions [Surynek, 2010; Yu and LaValle, 2013].

Nevertheless, AI researchers in the past years have made substantial progress in finding optimal solutions to a growing number of scenarios and for over a hundred agents [Sharon *et al.*, 2015; Sharon *et al.*, 2013; Wagner and Choset, 2015; Standley, 2010; Felner *et al.*, 2018; Barták *et al.*, 2017; Yu and LaValle, 2013]. However, most prior work assumed that (1) time is discretized into time steps, (2) the duration of every action is one time step, and (3) in every time step each agent occupies exactly a single location. These simplifying assumptions limit the applicability of MAPF algorithm in real-world applications. In fact, most prior work performed empirical evaluation only on 4-connected grids.

*Contact Author

	Actions			Agent		
	N.U.	Cont.	Ang.	Vol.	Opt.	Dist.
CBS-CL [Walker <i>et al.</i> , 2017]	✓	✗	✗	✗	✗	✗
M* [Wagner and Choset, 2015]	✓	✗	✓	✗	✓	✗
E-ICTS [Walker <i>et al.</i> , 2018]	✓	✗	✓	✓	✓	✗
MCCBS [Li <i>et al.</i> , 2019]	✗	✗	✗	✓	✓	✗
POST-MAPF [Ma <i>et al.</i> , 2017a]	✓	✓	✓	✓	✗	✗
ORCA [Snape <i>et al.</i> , 2011]	✓	✓	✓	✓	✗	✓
ALAN [Godoy <i>et al.</i> , 2018]	✓	✓	✓	✓	✗	✓
dRRT* [Dobson <i>et al.</i> , 2017]	✓	✓	✓	✓	✗	✓
AA-SIPP(<i>m</i>) [Yakovlev and Andreychuk, 2017]	✓	✓	✓	✓	✗	✗
TP-SIPPwRT [Liu <i>et al.</i> , 2019]	✓	✓	✓	✓	✗	✗
CCBS	✓	✓	✓	✓	✓	✗

Table 1: Overview: MAPF research beyond the basic setting.

We propose Continuous-time conflict-based search (CCBS), a MAPF algorithm that does not rely on any of these assumptions and is sound, complete, and optimal. CCBS is based on a customized version of Safe interval path planning (SIPP) [Phillips and Likhachev, 2011], a continuous-time single-agent pathfinding algorithm, and an adaptation of Conflict-based search (CBS) [Sharon *et al.*, 2015], a state-of-the-art multi-agent pathfinding algorithm.

CCBS relies on the ability to accurately detect collisions between agents and to compute the *safe intervals* of each agent, that is, the minimal time an agent can start to move over an edge without colliding. In our experiments, we used a closed-loop formulae for collision detection and a discretization-based approach for safe-interval computations. The results show that CCBS is feasible and outputs lower cost solutions compared to previously proposed algorithms. However, since CCBS considers agents’ geometry and continuous time, it can be slower than grid-based solutions, introducing a natural plan cost versus planning time tradeoff.

We are not the first to study MAPF beyond its basic setting. However, to the best of our knowledge, CCBS is the first MAPF algorithm that can handle non-unit actions duration, continuous time, non-grid domains, agents with a volume, and is still optimal and complete. Table 1 provides an overview of how prior work on MAPF relates to CCBS. A more detailed discussion is given in the related work section.

2 Problem Definition

We consider cooperative pathfinding for non-point translating non-rotating agents in 2D workspaces. All agents are assumed (1) to be of the same shape and size, (2) to move

with the same (constant) speed, and (3) to be constrained to the same roadmap of the environment, i.e. there is a single graph $G = (V, E)$ whose vertices correspond to locations agents can occupy (and wait in them) and edges correspond to straight-line trajectories the agents traverse when moving from one location to the other. Inertial effects are neglected and agents start/stop moving instantaneously. Duration of a move is translation speed times the length of the edge. Duration of a wait action can be any positive real number. Prior work referred to this setting as MAPF_R [Walker *et al.*, 2018].

Note that the CCBS algorithm we propose in this work is not limited to assumptions (1), (2), and (3) described above, e.g., it can handle agents moving with different speeds, using different roadmaps, and having complex shapes and sizes. We make these assumptions only to simplify exposition.

A *plan* for an agent i is a sequence of actions π_i such that if i executes this sequence of actions then it will reach its goal. A set of plans, one for each agent, is called a *joint plan*. A solution to a MAPF_R is a joint plan such that if all agents start to execute their respective plans at the same time, then all agents will reach their goal locations without colliding with each other. In this work we focus on finding cost-optimal solutions. To define cost-optimality of a MAPF_R solution, we first define the *cost* of a plan π_i to be the sum of the durations of its constituent actions. Several forms of solution cost-optimality have been discussed in MAPF research. Most notable are *makespan* and *sum of costs (SOC)*, where the makespan is the max over the costs of the constituent plans and SOC is their sum. The problem we address in this work is to find a solution to a given MAPF_R problem that is optimal w.r.t its SOC, that is, no other solution has a lower SOC.

3 CBS with Continuous Times

Next, we introduce CCBS and provide relevant background. CBS [Sharon *et al.*, 2015] is a complete and optimal MAPF solver, designed for standard MAPF, i.e., where time is discretized and all actions have the same duration. It solves a given MAPF problem by finding plans for each agent separately, detecting *conflicts* between these plans, and resolving them by replanning for the individual agents subject to specific *constraints*. The typical CBS implementation considers two types of conflicts: a vertex conflict and an edge conflict. A vertex conflict between plans π_i and π_j is defined by a tuple $\langle i, j, v, t \rangle$ and means that according to these plans agents i and j plan to occupy v at the same time t . An edge conflict is defined similarly by a tuple $\langle i, j, e, t \rangle$, and means that according to π_i and π_j both agents plan to traverse the edge $e \in E$ at the same time, from opposite directions.

A CBS vertex-constraint is defined by a tuple $\langle i, v, t \rangle$ and means that agent i is prohibited from occupying vertex v at t . A CBS edge-constraint is defined similarly by a tuple $\langle i, e, t \rangle$, where $e \in E$. To guarantee completeness and optimality, CBS runs two search algorithms: a low-level search algorithm that finds paths for individual agents subject to a given set of constraints, and a high-level search algorithm that chooses which constraints to add.

CBS: Low-Level Search. The low-level search in CBS can be any pathfinding algorithm that can find an optimal plan

for an agent that is consistent with a given set of CBS constraints. To adapt single-agent pathfinding algorithms such as A* to consider CBS constraints, the search space must also consider the time dimension since a CBS constraint $\langle i, v, t \rangle$ blocks location v only at a specific time t . For MAPF problems, where time is discretized, this means that a state in this single-agent search space is a pair (v, t) , representing that the agent is in location v at time t . Expanding such a state generates states of the form $(v', t + 1)$, where v' is either equal to v , representing a wait action, or equal to one of the locations adjacent to v . States generated by actions that violate the given set of CBS constraints, are pruned. Running A* on this search space will return the lowest-cost path to the agent’s goal that is consistent with the given set of CBS constraints, as required. This adaptation of textbook A* is very simple, and indeed most papers on CBS do not report it and just say that the low-level search of CBS is A*.

CBS: High-Level Search. The high-level search algorithm in CBS works on a Constraint Tree (CT), which is a binary tree, in which each node represents a set of CBS constraints imposed on the agents and a joint plan consistent with these CBS constraints. For a CT node N , we denote its constraints and joint plan by $N.constraints$ and $N.\Pi$, respectively. A CT node N is generated by first setting its constraints ($N.constraints$) and then computing $N.\Pi$ by running the low-level solver, which finds a plan for each agent subject to the constraints relevant to it in $N.constraints$. If $N.\Pi$ does not contain any conflict, then N is a goal. Else, one of the CBS conflicts $\langle i, j, x, t \rangle$ (where x is either a vertex or an edge) in $N.\Pi$ is chosen and two new CT nodes are generated N_i and N_j . Both nodes have the same set of constraints as N , plus a new constraint: N_i adds the constraint $\langle i, x, t \rangle$ and N_j adds the constraint $\langle j, x, t \rangle$. CBS searches the CT in a best-first manner, expanding in every iteration the CT node N with the lowest-cost joint plan.

3.1 From CBS to CCBS

CCBS follows the CBS framework. The main differences between CCBS and CBS are:

- To detect conflicts, CCBS uses a geometry-aware collision detection mechanism.
- To resolve conflicts, CCBS uses a geometry-aware unsafe-interval detection mechanism.
- CCBS adds constraints over pairs of actions and time ranges, instead of location-time pairs.
- For the low-level search, CCBS uses a version of SIPP adapted to handle CCBS constraints.

Next, we explain these differences in details.

Conflict Detection in CCBS

In CCBS, agents can have any geometric shape and agents’ actions can have any duration. Therefore, conflicts can occur between agents traversing different edges, as well as vertex-edge conflicts, which occurs when an agent moving along an edge collides with an agent waiting at a vertex [Li *et al.*, 2019]. Thus, a CCBS conflict is defined as conflicts between *actions*. Formally, a CCBS conflict is a tuple $\langle a_i, t_i, a_j, t_j \rangle$,

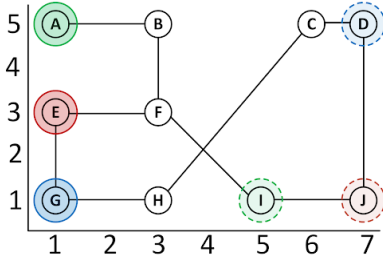


Figure 1: Our running example: a MAPF_R problem with 3 agents.

representing that if agent i executes a_i at t_i and agent j executes a_j at t_j then they will collide.

Collision detection for arbitrary-shaped moving objects is a non-trivial problem extensively studied in computer graphics, computational geometry and robotics [Jiménez *et al.*, 2001]. For the setting used in our experiments, there is a fast closed-loop collision detection mechanism [Guy and Karamouzas, 2015]. In general, CCBS as a MAPF algorithm is agnostic to the exact collision detection procedure used.

Resolving Conflicts in CCBS

The high-level search in CCBS runs a best-first search like regular CBS, selecting in every iteration a leaf N of the CT that has the joint plan with the smallest cost. The collision detection mechanism checks if N is a goal node. If not, the high-level search expands N by choosing one of the CCBS conflicts $\langle a_i, t_i, a_j, t_j \rangle$ detected in N .II and generating two new CT nodes, N_i and N_j . To compute the constraints to add to N_i and N_j , CCBS computes for each action its *unsafe intervals* w.r.t the other action. The unsafe interval of action a_i w.r.t. action a_j is the maximal time interval starting from t_i in which performing a_i will create a collision with performing a_j at time t_j . CCBS adds to N_i the constraint that agent i cannot perform a_i in its unsafe interval w.r.t to a_j , and adds to N_j the constraint that agent j cannot perform a_j in its unsafe interval w.r.t to a_i . For example, consider the MAPF_R problem illustrated in Figure 1. There are three disk-shaped ($r=0.5$) agents, a_1 , a_2 , and a_3 , where A , E , and G are their starts and I , J , and D are their goals, respectively. In the first CT node, the plans for a_2 and a_3 are $E \rightarrow F \rightarrow I \rightarrow J$ and $G \rightarrow H \rightarrow C \rightarrow D$, respectively. There is a conflict between the second actions in both plans, i.e., $\langle (F, I), 2, (H, C), 2 \rangle$ is a conflict. The unsafe interval for a_2 is $[2, 3.74)$ and for a_3 is $[2, 3.31)$. CCBS will generate two new CT nodes: one with the additional constraint $\langle a_2, (F, I), [2, 3.74) \rangle$ and one with the additional constraint $\langle a_3, (H, C), [3, 3.31) \rangle$.

SIPP for Planning with CCBS Constraints

The low-level solver of CCBS is based on SIPP, which is a single-agent pathfinding algorithm designed to handle continuous time and moving obstacles. SIPP computes for every location $v \in V$ a set of *safe intervals*, where a safe interval is a maximal contiguous time interval in which an agent can stay or arrive at v without colliding with a moving obstacle. A safe interval is *maximal* in the sense that extending it to either direction yields a collision. The key observation in SIPP is that the number of safe intervals is finite and often small.

To find a plan, SIPP runs an A*-based algorithm, searching in the space of (location, safe interval) pairs. SIPP is complete and returns time-minimal solutions.

To adapt SIPP to be used as a low-level solver of CCBS, we modify it so actions that violate the constraints are prohibited, as follows. Let $\langle i, a_i, [t_i, t_i^u) \rangle$ be a CCBS constraint imposed over agent i . To adapt SIPP to plan for agent i subject to this constraint, we distinguish between two cases:

a_i is a move action. Let v and v' be the source and target destinations of a_i . If the agent arrives to v in time step $t \in [t_i, t_i^u)$ then we remove the action that moves it from v to v' at time t , and add an action that represents waiting at v until t_i^u and then moving to v' .

a_i is a wait action. Let v be the vertex in which the agent is waiting in a_i . We forbid the agent from waiting at v in the range $[t_i, t_i^u)$ by splitting the safe intervals of v accordingly. For example, if v is associated with a single safe interval: $[0, \infty)$, then splitting it to two intervals $[0, t_i]$ and $[t_i^u, \infty)$.¹

To demonstrate these two modifications to SIPP, consider again the MAPF_R problem in Figure 1. As mentioned above, in the root CT node there is a conflict between a_2 and a_3 , and one of the constraints added to resolve it $\langle a_2, (F, I), [2, 3.74) \rangle$. This is a constraint over a move action, and thus we replace the action that moves a_2 from F to I with an action that waits for a duration of 1.74 in F and then moves to I . The optimal plan for a_2 under this constraint is indeed to use the modified action. However, this plan has a conflict with the a_1 (green), which has the plan $A \rightarrow B \rightarrow F \rightarrow I$. To resolve it, the constraint added to a_2 is $\langle a_2, (F, F), [3, 4) \rangle$. This is a wait action, and so we split safe intervals of F from the default $[0, \infty)$ to two safe intervals: $[0, 3]$ and $[4, \infty)$.

Lemma 1. *Running the adapted SIPP described above with a set of CCBS constraints C_1, \dots, C_m returns the lowest-cost path that satisfies these constraints.*

The correctness of Lemma 1 follows from SIPP’s optimality and the fact that our adaptations only prohibit moves that directly break the given CCBS constraint. A formal proof is omitted due to space constraints.

3.2 Theoretical Properties

Next, we prove that CCBS is sound, complete, and optimal. Our analysis is based on Lemma 1 and the notion of a *sound* pair of constraints, defined by Atzmon *et al.* [2018].

Definition 1 (Sound Pair of Constraints). *A pair of constraints is sound iff in every optimal solution it holds that at least one of these constraints hold.*

Lemma 2. *For any CCBS conflict $\langle a_i, t_i, a_j, t_j \rangle$ and corresponding unsafe intervals $[t_i, t_i^u)$ and $[t_j, t_j^u)$ the pair of CCBS constraints $\langle i, a_i, [t_i, t_i^u) \rangle$ and $\langle j, a_j, [t_j, t_j^u) \rangle$ is a sound pair of constraints.*

Proof. By contradiction, assume that there exists $\Delta_i \in (0, t_i^u - t_i]$ and $\Delta_j \in (0, t_j^u - t_j]$ such that perform a_i at $t_i + \Delta_i$ and a_j at $t_j + \Delta_j$ does not create a conflict. That is, $\langle a_i, t_i + \Delta_i, a_j, t_j + \Delta_j \rangle$ is not a conflict.

¹Note that the first interval includes t_i . This is because while the agent cannot perform wait action in t_i , it can perform a move action.

By definition, of t_j^u :

$$\forall t \in [t_j, t_j^u) : \langle a_i, t_i, a_j, t \rangle \text{ is a conflict.}$$

$$\forall t \in [t_j + \Delta_j, t_j^u) : \langle a_i, t_i + \Delta_j, a_j, t \rangle \text{ is a conflict.}$$

By definition of Δ_i and Δ_j :

$$\langle a_i, t_i + \Delta_i, a_j, t_j + \Delta_j \rangle \text{ is not a conflict}$$

Therefore, $\Delta_i < \Delta_j$. Similarly, by definition of t_i^u :

$$\forall t \in [t_i, t_i^u) : \langle a_i, t, a_j, t_j \rangle \text{ is a conflict.}$$

$$\forall t \in [t_i + \Delta_i, t_i^u) : \langle a_i, t, a_j, t_j + \Delta_i \rangle \text{ is a conflict.}$$

Therefore, by definition of Δ_i and Δ_j we have that $\Delta_j < \Delta_i$, which leads to a contradiction. \square

Theorem 1. *CCBS sound, complete, and is guaranteed to return an optimal solution.*

Proof. Soundness follows from performing conflict detection on every joint plan. Completeness and optimality is due to Lemma 2 and Atzmon et al.’s [2018] proof for k -robust CBS. In details, let N be a CT node with children N_1 and N_2 , generated by the sound pair of constraints C_1 and C_2 , respectively. $\pi(N)$ denotes all joint plans that satisfy N .constraints. Since C_1 and C_2 is a sound pair of constraints, it holds that $\pi(N) = \pi(N_1) \cup \pi(N_2)$. Thus, splitting a CT node does not lose any valid joint plan. Due to Lemma 1 and the fact that the CT is searched in a best-first manner over the cost and adding constraints can only increase cost, we are guaranteed that CCBS returns an optimal solution. \square

4 Practical Aspects

The analysis above relies on having accurate collision detection and unsafe interval detection mechanisms. That is, a collision is detected iff one really exists, and the maximal unsafe interval is used for every given pair of actions. However, constructing such accurate mechanisms is not trivial. There are various ways to detect collisions between agents with volume in a continuous space, including closed-loop geometric computations as well as sampling-based approaches. See Jiménez et al. [2001] for an overview and [Tang et al., 2014] for an example of a particular collision detection procedure. For the constant velocity disk-shaped agents we used in our experiments, there exists a closed-loop accurate collision detection mechanism described in [Guy and Karamouzas, 2015].

Computing the unsafe interval of an action w.r.t to another action also requires analyzing the kinematics and geometry of the agents. However, unlike collision detection, which has been studied for many years and can be computed with a closed-loop formula in some settings, to the best of our knowledge no such closed loop formula are known for computing the unsafe intervals. A general method for computing unsafe intervals is to apply the collision detection mechanism multiple time, starting from t_i and incrementing by some small $\Delta > 0$ until the collision detection mechanism reports that the unsafe interval is done. This approach is limited in that the resulting unsafe interval may be larger than the real unsafe interval.

4.1 Conflict Detection and Selection Heuristics

As noted above, conflict detection in CCBS is more complex than in regular CBS. Indeed, in our experiments we observed that conflict detection took a significant portion of time. To speed up the conflict detection, we only checked conflicts between actions that overlap in time and may overlap geometrically. In addition, we implemented two heuristics for speeding up the detection process. We emphasize that these heuristics do not compromise our guarantee for soundness, completeness, and optimality.

The first heuristic we used, which we refer to as the *past-conflicts heuristic*, keeps track of the number of times conflicts have been found between agents i and j , for every pair of agents (i, j) . Then, it checks first for conflicts between pair of agents with a high number of past conflicts. Then, when a conflict is found the search for conflicts is immediately halted. That found conflict is then stored in the CT node, and if that CT node will be expanded then it will generate CT nodes that are aimed to resolve this conflict. This implements the intuition that pairs of agents that have conflicted in the past are more likely to also conflict in the future.

We have found this heuristic to be very effective in practice for reducing the time allocated for conflict detection. Using this heuristic, however, has some limitations. Prior work has established that to intelligently choosing which conflict to resolve when expanding a CT node can have a huge impact on the size of the CT and on the overall runtime [Boyarski et al., 2015]. Specifically, Boyarski et al. [2015] introduced the notion of *cardinal conflicts*, which are conflicts that resolving them results increases the SOC. *Semi-cardinal conflicts* are conflicts that resolving them by replanning for one of the involved agents will increase the solution cost, but replanning for the other involved agents do not increase solution cost.

For CBS, choosing to resolve first cardinal conflicts, and then semi-cardinals, yielded significant speed ups [Boyarski et al., 2015]. However, to detect cardinal and semi-cardinal conflicts, one needs to identify all conflicts, while the advantage of the heuristic is that we can halt the search for conflicts before identifying all conflicts.

To this end, we proposed a second hybrid heuristic approach. Initially, we detect all conflicts and choose only cardinal conflicts. However, if a node N does not contain any cardinal or semi-cardinal conflict, then for all nodes in the CT subtree beneath it we switch to use the past-conflicts heuristic. This hybrid approach worked well in our experiments, but fully exploring this tradeoff between fast conflict detection and smart conflict selection is a topic for future work.

5 Experimental Results

We conducted experiments on grids, where agents can move from the center of one grid cell to the center of another grid cell. The size of every cell is 1×1 , and the shape of every agent is an open disk whose radius equals $\sqrt{2}/4$. This specific value was chosen to allow comparison with CBS, since it is the maximal radius that allows agents to safely perform moves in which agents follow each other.

To allow non-unit edge costs, we allowed agents to move in a single move action to every cell located in their 2^k neigh-

k	SOC				Success Rate			
	2	3	4	5	2	3	4	5
4	25.7	21.2	20.4	20.3	1.00	1.00	0.97	0.95
6	38.2	31.6	30.5	30.2	0.99	1.00	0.88	0.83
8	49.2	40.7	39.3	39.0	0.98	0.97	0.74	0.57
10	61.0	50.5	48.8	48.4	0.95	0.94	0.54	0.42
12	78.0	64.7	-	-	0.94	0.86	-	-
14	90.8	75.3	-	-	0.88	0.64	-	-
16	102.4	85.2	-	-	0.76	0.53	-	-
18	118.7	-	-	-	0.62	-	-	-
20	131.7	-	-	-	0.46	-	-	-

Table 2: Results for CCBS on 10×10 open grid.

borhood, where k is a parameter [Rivera *et al.*, 2017]. Moving from one cell to the other is only allowed if the agent can move safely to the target cell without colliding with other agents or obstacles, where the geometry of the agents and obstacles are considered. The cost of a move corresponds to the Euclidean distance between the grid cells centers.

5.1 Open Grids

For the first set of experiments we used a 10×10 open grid, placing agents’ start and goal locations randomly. We run experiments with 4, 5, . . . , 20 agents. For every number of agents we created 250 different problems. Each problem was solved with CCBS with $k = 2, 3, 4,$ and 5 . Table 2 shows the results of this set of experiments. Every row shows results for a different number of agents, as indicated on the left-most column. The four right-most columns show the success rate, i.e., the ratio of problems solved by the CCBS under a timeout of 60 seconds, out of a total of 250 problems. Data points marked by “-” indicate settings where the success rate was lower than 0.4. The next four columns show the average SOC, averaged over the problems solved by all CCBS instances that had a success rate larger than 0.4.

The results show that increasing k yields solutions with lower SOC, as expected. The absolute difference in SOC when moving from $k = 2$ to $k = 3$ is the largest, and it grows as we add more agents. For example, for problems with 14 agents, moving from $k = 2$ to $k = 3$ yields an improvement of 15.5 SOC, and for problems with 16 agents the gain of moving to $k = 3$ is 17.2 SOC. Increasing k further exhibits a diminishing return effect, where the largest average SOC gain when moving from $k = 4$ to $k = 5$ is at 0.5.

Increasing k , however, has also the effect of increasing the branching factor, which in turns means that path-finding becomes harder. Indeed, the success rate of $k = 5$ is significantly lower compared to $k = 4$. An exception to this is the transition from $k = 2$ to $k = 3$, where we observed a slight advantage in success rate for $k = 3$ for problems with a small number of agents. For example, with 6 agents the success rate of $k = 2$ is 0.99 while it is 1.00 for $k = 3$. An explanation for this is that increasing k also means that plans for each agent can be shorter, which helps to speed up the search. Thus, increasing k introduces a tradeoff w.r.t. the problem-solving difficulty: the resulting search space for the low-level search is shallower but wider. For denser problems, i.e., with more agents, $k = 2$ is again better in terms of success rate, as more involved paths must be found by the low-level search.

We also compared the performance of CCBS with $k = 2$

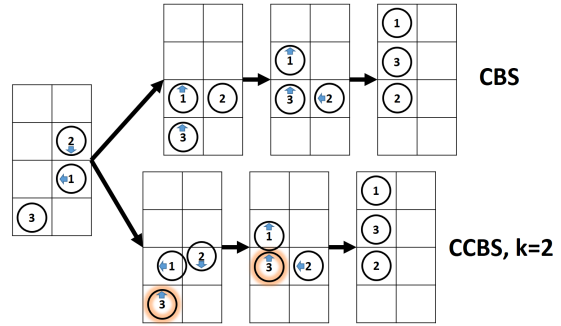


Figure 2: Example: CCBS for $k=2$ finds a better solution than CBS.

and a standard CBS implementation. CBS was faster than CCBS, as its underlying solver is A* on a 4-connected grid, detecting collisions is trivial, and it has only unit-time wait actions. However, even for $k = 2$, CCBS is able to find better solutions, i.e., solutions of lower SOC. This is because, an agent may start to move after waiting less than a unit time step. Figure 2 illustrates such a scenario. An animation of this example is given in <https://tinyurl.com/ccbs-vs-cbs2>.

5.2 Dragon Age Maps

Next, we experimented with a larger grid, taken from the Dragon Age: Origin (DAO) game and made available in the movingai repository [Sturtevant, 2012]. We used the den520d map, shown to the right of Table 3, which was used by prior work [Sharon *et al.*, 2015]. Start and goal states were chosen randomly, and we create 250 problems for every number of agents. Table 3 shows the results obtained for CCBS with $k = 2, 3,$ and 4 , in the same format as Table 2. The same overall trends are observed: increasing k reduces the SOC and decreases the success rate.

5.3 Conflict Detection and Resolution Heuristics

In all the experiments so far we used CCBS with the hybrid conflict detection and selection heuristic described earlier in the paper. Here, we evaluate the benefit of using this heuristic. We compared CCBS with this heuristic against the following: (1) Vanilla: CCBS that chooses arbitrarily which actions to check first for conflicts, (2) Cardinals: CCBS that identifies all conflicts and chooses cardinal conflicts, and (3) PastConf: CCBS that uses the past-conflicts heuristic to choose where to search for conflicts first, and resolves the first conflict it finds.

Table 4 shows results for the den520d DAO map for 20 agents with $k = 2, 3,$ and 4 ; and 25 agents with $k=2$ and $k=3$. For every configuration we create and run CCBS on 1,000 instances. The table shows the success rate (the row labelled “Success”) and the average number of high-level nodes expanded by CCBS (“HL exp.”). The results show that the pro-

k	SOC			Success Rate		
	2	3	4	2	3	4
10	1,791	1,515	1,460	0.96	0.93	0.86
15	2,598	2,198	2,118	0.94	0.84	0.70
20	3,347	2,829	2,726	0.79	0.72	0.50
25	4,049	3,426	3,304	0.58	0.58	0.32




Table 3: Results for CCBS on the den520d DAO map.

		Vanilla	PastConf	Cardinals	Hybrid
$k=2$	Success	0.72	0.74	0.75	0.82
Agents=20	HL exp.	765	712	453	452
$k=3$	Success	0.67	0.68	0.75	0.76
Agents=20	HL exp.	152	141	51	47
$k=4$	Success	0.39	0.4	0.48	0.50
Agents=20	HL exp.	564	516	232	270
$k=2$	Success	0.39	0.43	0.38	0.53
Agents=25	HL exp.	1762	1730	968	990
$k=3$	Success	0.44	0.45	0.60	0.61
Agents=25	HL exp.	313	270	81	72

Table 4: Comparing conflict detection and selection methods.

posed hybrid heuristic enjoys the complementary benefits of PastConf and Cardinals, expanding as few CT nodes as Cardinals and having the highest success rate.

5.4 Comparison to E-ICTS

Finally, we compared the performance of CCBS and E-ICTS [Walker *et al.*, 2018], a MAPF_R algorithm based on the Increasing Cost Tree Search (ICTS) framework [Sharon *et al.*, 2013]. E-ICTS can handle non-unit edge cost, and handles continuous time by discretizing it according to a minimal wait time parameter Δ . Figure 3 shows the success rate of the two algorithms on open 10×10 grids with different numbers of agents and $k = 2, 3, 4$, and 5. We thank the E-ICTS authors who made their implementation publicly available (<https://github.com/nathanstt/hog2>).

The results show that for $k = 2$ and $k = 3$, CCBS works better in most cases, while E-ICTS outperforms CCBS for $k = 4$ and $k = 5$. The reason for this is that as k increases, more actions conflict with each other, resulting in a significantly larger CT. Developing pruning techniques for such CT is a topic for future work. We also compared CCBS to ICTS over larger dragon age maps. The results where that in most cases E-ICTS solved more instances.

Given an accurate unsafe interval detection mechanism, CCBS handles continuous time directly, and thus can return better solutions than E-ICTS. However, the unsafe interval detection mechanism we implemented did, in fact, discretize time. Also, we used different implementations for CCBS and E-ICTS. Thus, we do not presume to infer when each algorithm is better. This is an open question even for basic MAPF.

CCBS is applicable beyond grid domains. To show this, we created a roadmap with 878 vertices and 14,628 edges based on the den520d DAO map using the OMPL library (<http://ompl.kavrakilab.org>). For 250 problems with 10, 15,

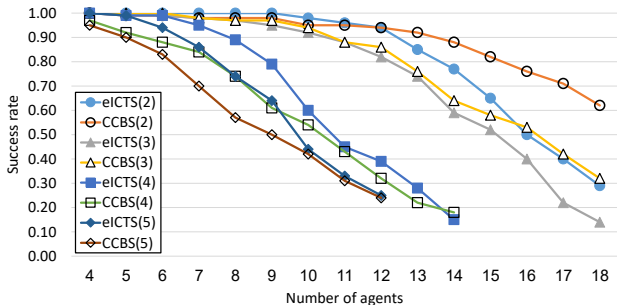


Figure 3: Success rate of CCBS and E-ICTS in 10×10 open grids.

and 20 agents, the success rate was 0.89, 0.60, and 0.22, with SOC of 1,459, 2,082, and 2,688, respectively.

6 Related Work

AA-SIPP(m) is an any-angle MAPF algorithm based on SIPP that adopts a prioritized planning approach [Yakovlev and Andreychuk, 2017]. Ma *et al.* [2019] also used SIPP in a prioritized planning framework for lifelong MAPF. Both algorithms do not guarantee completeness or optimality. Multi-Constraint CBS (MCCBS) is a CBS-based algorithm for agents with a geometrical shape that may have different configuration spaces [Li *et al.*, 2019]. They assumed all actions have a unit duration and did not address continuous time. CBS-CL is a CBS-based algorithm designed to handle non-unit edge costs and hierarchy of movement abstractions [Walker *et al.*, 2017]. It does not allow reasoning about continuous time and does not return provably optimal solutions. MAPF-POST is a post-processing step that adapts a MAPF solution to different action durations that due to kinematic constraints [Hönig *et al.*, 2017]. ORCA [Van Den Berg and Overmars, 2005; Snape *et al.*, 2011] and ALAN [Godoy *et al.*, 2018] are fast and distributed MAPF algorithms for continuous space. None of these algorithms guarantee optimality. dRRT* is a sample-based MAPF algorithm designed for continuous spaces [Dobson *et al.*, 2017]. dRRT* is asymptotically complete and optimal while CCBS is optimal and complete, and is designed to run over a discrete graph.

Table 1 provides a differential overview of related work on MAPF beyond its basic setting. Columns “N.U.,” “Cont.,” “Ang.,” “Vol.,” “Opt.,” and “Dist.,” means support for non-uniform action durations, actions with arbitrary continuous duration, actions beyond the 4 cardinal moves, agents with a volume (i.e., some geometric shape), returns a provably optimal solution, and distributed algorithm, respectively. Rows correspond to different algorithms or family of algorithms.

7 Conclusion and Future Work

We proposed CCBS, a sound, complete, and optimal MAPF algorithm that supports continuous time, actions with non-uniform duration, and agents and obstacles with a geometric shape. CCBS follows the CBS framework, using an adapted version of SIPP as a low-level solver, and unique types of conflicts and constraints in the high-level search. To the best of our knowledge, CCBS is the first MAPF algorithm to provide optimality guarantees for such a broad range of MAPF settings. Experimental evaluation highlighted that conflict detection becomes a bottleneck when solving MAPF_R problems. We suggested a hybrid heuristic for reducing this cost. Future work may apply meta-reasoning to decide when and how much to invest in conflict detection.

Acknowledgments

This research is supported by ISF grants no. 210/17 to Roni Stern and by RSF grant no. 16-11-00048 to Konstantin Yakovlev and Anton Andreychuk.

References

- [Atzmon *et al.*, 2018] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou. Robust multi-agent path finding. In *Symposium on Combinatorial Search (SOCS)*, 2018.
- [Barták *et al.*, 2017] R. Barták, N.-F. Zhou, R. Stern, E. Boyarski, and P. Surynek. Modeling and solving the multi-agent pathfinding problem in picat. In *International Conference on Tools with Artificial Intelligence (ICTAI)*, 2017.
- [Boyarski *et al.*, 2015] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony. ICBS: The improved conflict-based search algorithm for multi-agent pathfinding. In *Symposium on Combinatorial Search (SoCS)*, 2015.
- [Dobson *et al.*, 2017] A. Dobson, K. Solovey, R. Shome, D. Halperin, and K. E. Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2017.
- [Felner *et al.*, 2018] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, TK S. Kumar, and S. Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, 2018.
- [Godoy *et al.*, 2018] Julio Godoy, Tiannan Chen, Stephen J Guy, Ioannis Karamouzas, and Maria Gini. Alan: adaptive learning for multi-agent navigation. *Autonomous Robots*, pages 1–20, 2018.
- [Guy and Karamouzas, 2015] S. J. Guy and I. Karamouzas. Guide to anticipatory collision avoidance. In Steven Rabin, editor, *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, chapter 19. AK Peters/CRC Press, 2015.
- [Hönig *et al.*, 2017] W. Hönig, TK S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig. Summary: multi-agent path finding with kinematic constraints. In *IJCAI*, pages 4869–4873, 2017.
- [Jiménez *et al.*, 2001] P. Jiménez, F. Thomas, and C. Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.
- [Kornhauser *et al.*, 1984] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Symposium on Foundations of Computer Science*, 1984.
- [Li *et al.*, 2019] J. Li, P. Surynek, A. Felner, and H. Ma. Multi-agent path finding for large agents. In *AAAI*, 2019.
- [Liu *et al.*, 2019] M. Liu, H. Ma, J. Li, and S. Koenig. Task and path planning for multi-agent pickup and delivery. In *AAMAS*, 2019.
- [Ma *et al.*, 2017a] H. Ma, S. Kumar, and S. Koenig. Multi-agent path finding with delay probabilities. In *AAAI*, 2017.
- [Ma *et al.*, 2017b] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, 2017.
- [Ma *et al.*, 2019] H. Ma, W. Hönig, T. K. S. Kumar, N. Ayanian, and S. Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *AAAI*, 2019.
- [Morris *et al.*, 2016] R. Morris, C. S. Pasareanu, K. S. Luckow, W. Malik, H. Ma, TK S. Kumar, and S. Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop*, 2016.
- [Phillips and Likhachev, 2011] M. Phillips and M. Likhachev. Sipp: Safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- [Rivera *et al.*, 2017] N. Rivera, C. Hernández, and J. A. Baier. Grid pathfinding on the 2k neighborhoods. In *AAAI*, pages 891–897, 2017.
- [Sharon *et al.*, 2013] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 2013.
- [Sharon *et al.*, 2015] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 2015.
- [Snape *et al.*, 2011] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.
- [Standley, 2010] T. s. Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 2010.
- [Sturtevant, 2012] N. R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.
- [Surynek, 2010] P. Surynek. An optimization variant of multi-robot path planning is intractable. In *AAAI*, 2010.
- [Tang *et al.*, 2014] M. Tang, r. Tong, Z. Wang, and D. Manocha. Fast and exact continuous collision detection with bernstein sign classification. *ACM Transactions on Graphics (TOG)*, 2014.
- [Van Den Berg and Overmars, 2005] J. P. Van Den Berg and M. H. Overmars. Prioritized motion planning for multiple robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 430–435, 2005.
- [Veloso *et al.*, 2015] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, page 4423, 2015.
- [Wagner and Choset, 2015] G. Wagner and H. Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.
- [Walker *et al.*, 2017] T. T. Walker, D. Chan, and N. R. Sturtevant. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.
- [Walker *et al.*, 2018] T. T. Walker, N. R. Sturtevant, and A. Felner. Extended increasing cost tree search for non-unit cost domains. In *IJCAI*, pages 534–540, 2018.
- [Wurman *et al.*, 2008] P. R. Wurman, R. D’Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 2008.
- [Yakovlev and Andreychuk, 2017] K. Yakovlev and A. Andreychuk. Any-angle pathfinding for multiple agents based on SIPP algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.
- [Yu and LaValle, 2013] J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 2013.