Prioritizing Conflicts in Conflict-Based Search with Disjoint Splitting

Grigorios Mouratidis[1][0009−0009−6797−0013], Bernhard Nebel[1][0000−0002−6833−6323], and Sven Koenig[2][0000−0002−5458−094X]

[1] University of Freiburg
mouratig@cs.uni-freiburg.de
nebel@uni-freiburg.de
[2] University of California, Irvine
sven.koenig@uci.edu

Abstract. Multi-agent path finding (MAPF) is the problem of finding collision-free paths for a team of agents. Conflict-based search (CBS) is a state-of-the-art algorithm for solving MAPF optimally that repeatedly picks a collision (known as conflict) between two agents and resolves it by constraining the movement of the agents. One of the earliest runtime enhancement of CBS categorized conflicts into three groups (cardinal, semi-cardinal, and non-cardinal) and resolved them in that priority order. When multiple collisions share the same type, CBS typically breaks ties by selecting one at random. In this work, we show that CBS with disjoint splitting (which is another CBS' runtime enhancement) is further improved by introducing a more nuanced conflict prioritization strategy when multiple cardinal (or semi-cardinal) conflicts are present. More specifically, we prioritize conflicts that involve agents that participate in multiple cardinal (or semi-cardinal) conflicts. Our experiments show that the new conflict prioritization strategy reduces both the number of node expansions and the runtime of CBS with disjoint splitting for many instances from the standard MAPF benchmark suite.

Keywords: MAPF · Heuristic Search · CBS

1 Introduction

Multi-Agent Path Finding (MAPF) is a widely studied problem in AI planning that has attracted significant attention due to its computational complexity and wide range of real-world applications, including robotics, logistics, and automated transportation systems. In MAPF, a set of agents must compute collision-free paths from their respective start locations to their goal locations. The objective is to find solutions efficiently, despite the exponential growth of the joint state space as the number of agents increases. Consequently, numerous optimal and suboptimal approaches have been proposed to address this challenge.

Conflict-Based Search (CBS) is a leading optimal algorithm for solving MAPF. It follows a two-level search framework: the low-level search independently computes shortest paths for each agent, while the high-level search repeatedly resolves collisions (known as conflicts) that emerge between these paths. When a conflict is detected, the high level introduces constraints to prevent the conflict and the paths of the affected agents are replanned at the low level. This iterative process continues until a conflict-free solution is returned. Due to its efficiency in finding optimal solutions, CBS has become one of the most prominent methods in MAPF research.

Originally, when multiple conflicts were present, CBS selected the next conflict to resolve randomly. Subsequent research proposed more informed conflict-selection techniques [1, 2, 4] that guide the search more effectively, thereby improving overall performance. In this work, we introduce a novel conflict-prioritization strategy for CBS with disjoint splitting (which is one of CBS' enhancements) designed to further improve efficiency by minimizing redundant branching in the search process.

In the following sections, we first present a theoretical example demonstrating how the proposed conflict-prioritization strategy reduces unnecessary branching. Next, we describe the algorithmic modifications required to integrate our approach into CBS. Finally, we evaluate our approach on six different MAPF benchmark maps and present the results

2 MAPF

Formally, MAPF is often described as a problem on an undirected graph $G = (V, E)$ as follows: Let A be a set of agents and let $s : A \to V$ and $g : A \to V$ denote injective functions assigning every agent $a_i \in A$ a start and a goal vertex respectively. The objective is to find a *valid* solution, which is a set of conflict-free paths that navigates the agents from their starting vertices to their goal vertices.

In classic MAPF, time is divided into discrete timesteps. At each timestep, an agent may either move to any adjacent vertex or remain at its current location. Every action requires one timestep to complete and incurs a cost of one. A *path* is a sequence of locations that an agent occupies to arrive to its goal. Finally, the *cost* of an agent's path is equal to the sum of costs of all its actions until it arrives to its goal.

The majority of MAPF problem descriptions considers only two types of conflicts, which are the vertex conflicts and the edge conflicts. *Vertex conflicts* occur when two agents are located at the same vertex at the same timestep, and *edge conflicts* occur when two agents move along the same edge but in different directions at the same timestep.

MAPF can be solved optimally or (bounded) suboptimally with respect to different objective functions. In this work we focus on optimal valid solutions with respect to the sum-of-costs (SoC) which is the sum of costs of all agents'

paths. Solving MAPF optimally with respect to the sum-of-costs objective is NP-complete [13].

3 Conflict-Based Search

CBS [9] is a two-level search and its high-level search creates a binary search tree, called the constraint tree (CT). Each node $N \in CT$ contains the following attributes:

- A *set of constraints* for all agents,
- A *solution* (not necessarily valid) that respects the current set of constraints,
- A *cost*, which is the cost of the solution with respect to the given objective.

The high-level search performs a best-first search with respect to the nodes' cost and uses a priority queue. At the beginning, it creates the root node that has an empty constraint set and invokes the low-level search, which is often a simple A* search, to find and return the optimal paths for all the agents. Then, the root node cost is computed, and the root node is inserted in the priority queue. Subsequently, the high level decides the next best CT node to expand (initially only the root node is in the priority queue) and searches for possible conflicts in its current solution. If no conflict is found, the current node solution is valid and optimal and is returned.

In case there is a vertex conflict (a_i, a_j, u, t) (or an edge conflict) between two agents a_i and a_j at vertex u and timestep t, the current node is removed from the priority queue and CBS performs a *split* action. The split action creates two child CT nodes that inherit the constraints of the current node (parent node) and an extra negative constraint is added to each of them to resolve the conflict. For the aforementioned conflict, the constraint $(a_i \not\rightarrow (u, t))$ is added to one of the children and the constraint $(a_j \not\rightarrow (u, t))$ is added to the other one. The negative constraint $(a_i \not\rightarrow (u, t))$ means that agent a_i is not allowed to be at vertex u at timestep t. Then the low-level search is invoked again for each child node to compute the new optimal path for the agent affected by the additional constraint. Finally, the new cost of the child nodes is computed and the nodes are added to the priority queue where the high level performs a best-first search with respect to the node cost.

This process continues till no conflict exists any longer and the solution returned by CBS is guaranteed to be valid and optimal with respect to the given objective.

4 Related Work

CBS is very sensitive to the single-agent paths returned by the low level. This happens because there are often many different optimal paths for an individual agent on the low level but some of them increase the amount of conflicts found in the high level. Since CBS' runtime is in the worst case exponential to the

number of conflicts that it has to resolve, these bad decisions will significantly increase the runtime. This is why a *conflict avoidance table* (CAT) [11] is used when calculating the single-agents paths on the low level. CAT is a lookup table that stores the position and time of every agent whose path has already been calculated and when the low level searches for the optimal path of a new agent, ties on low-level nodes (that have the same A* search f-value) are broken in favor of nodes with lower number of conflicts in CAT.

CBS' high-level chooses randomly the next conflict (in case there are many) to resolve. These decisions significantly affect the performance of the algorithm. To mitigate this issue, improved CBS (ICBS) [2] was developed and categorized conflicts into three different types, namely *cardinal*, *semi-cardinal* and *non-cardinal*. The cardinality of a conflict depends on the cost of the child nodes that are created when this conflict is resolved. When cardinal conflicts are resolved, the node cost of both child nodes is increased compared to the parent node. In semi cardinal conflicts, the cost of only one child node is increased and finally, when resolving a non-cardinal conflict, none of the costs of the child nodes increases. In ICBS, cardinal conflicts have the highest priority, then follow semi-cardinal conflicts and in the end the non-cardinal ones. This prioritization improved the runtime of CBS because, when cardinal (or semi-cardinal) conflicts are prioritized, the solution cost of both (or one, respectively) child nodes increases and this makes the search more informed.

Another improvement on the CBS algorithm (called ICBS-h) was accomplished by adding admissible heuristics (CG heuristic) [3] on the high-level nodes. These admissible heuristic values are calculated by aggregating the cardinal conflicts between disjoint pairs of agents and make the high-level search more informed. ICBS-h further improved the runtime compared to ICBS.

In the same line of work, two more informative admissible heuristics, namely *DG* and *WDG* were developed to guide CBS' high-level search [5]. The DG heuristic is more informative than the CG heuristic because it doesn't only take into account the cardinal conflicts between the agents to compute the heuristic value, but also investigates their whole possible paths to find out if all their cost-minimal paths have an inevitable conflict with each other, resulting to higher heuristic values. WDG is even more informative that DG because it also captures the minimum extra cost needed to resolve all conflicts between pairs of agents. Although DG and WDG incur a computation overhead, they manage to reduce substantially the amount of high-level nodes expanded and thus sped up CBS.

The standard splitting action in CBS creates two negative constraints that are propagated to the child nodes. This approach creates some duplicate search effort because some solutions are shared in both child nodes. To mitigate the solution overlap and the respective computational overhead, disjoint splitting was introduced. Disjoint splitting [7] splits a node by creating one positive and one negative constraint for the same agent. *Positive constraints* e.g. $a_i \rightarrow (u, t)$ force an agent to be at a specific vertex (or edge) at a specific timestep and simultaneously prohibit all other agents from being at this vertex (edge) at this

timestep (e.g. $a_j \not\rightarrow (u,t)$, $\forall a_j \in A \backslash \{a_i\}$). This type of splitting sped up CBS and CBS variants by up to 2 orders of magnitude.

Reasoning techniques have been also leveraged to identify cases where CBS split actions are inefficient because they create a big number of nodes that can exponentially increase the high-level search space in some cases. Those techniques identify *rectangle* [8], *corridor* and *target* conflicts [6] between pair of agents and resolve them with one single split action, resulting to lower runtimes.

Despite categorizing conflicts into three categories (cardinal, semi-cardinal and non-cardinal) with different priorities, CBS still randomly decides about what conflict to choose and resolve next in case there are multiple conflicts with the same priority. Since these choices can heavily influence the efficiency of the algorithm, researchers employed a linear ranking function [4], learned using machine learning techniques, to rank conflicts at a search node based on their estimated potential to lead to a faster solution. This conflict prioritization approach achieved a runtime reduction of 10% to 65%, depending on the specific map in the MAPF suite.

Another research direction [1] that further investigated the possibility of prioritizing conflicts focused on prioritizing conflicts that increase the f-values of the resulting child nodes. By prioritizing such conflicts the search becomes more informed and since the computational cost needed to find such conflicts was low, this approach led to reduced node expansions and shorter runtimes.

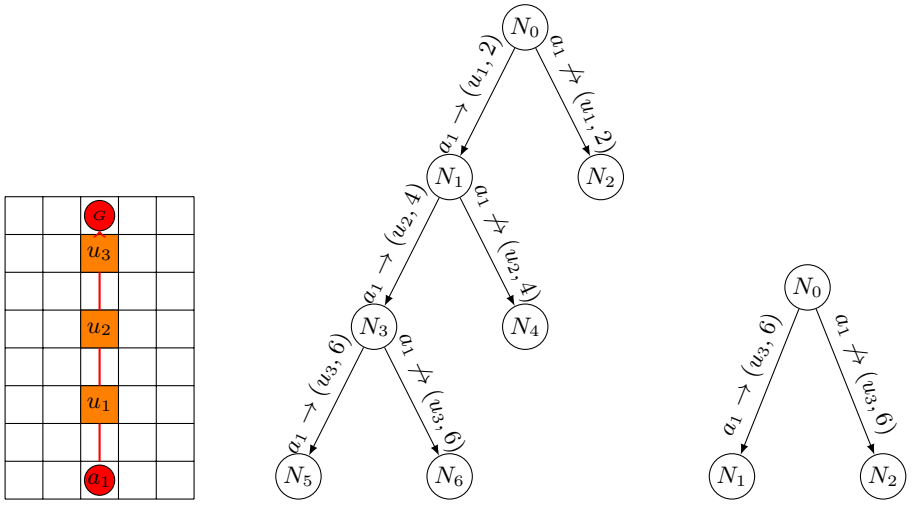
5 Conflict Prioritization

CBS with disjoint splitting prioritizes cardinal conflicts, followed by semi-cardinal and finally non-cardinal. Though, in case of multiple conflicts with the same priority, there is no clear guideline on which conflict to resolve first and how this affects the efficiency of the algorithm because the aforementioned research directions [4, 1] that prioritize certain conflicts were specifically engineered for non-disjoint splitting and they cannot be easily extended for non-disjoint splitting. Thus, since no typical guideline exists when multiple conflicts have the same priority, some researchers decided to break ties in favor of conflicts with the earliest timestep [8].

5.1 Example

In the following, we will consider an example that demonstrates that there are some cases where resolving conflicts earlier in time is inefficient because it causes the CT to branch unnecessarily many times and create superfluous high-level nodes. The increased node expansion may lead to a larger CT and thus runtime, which could have been avoided if we had chosen a different conflict resolution order.

Figure 1(a) shows an agent that faces three cardinal conflicts while following its fastest path to its goal. Figure 1(b) shows the generated CT, if we decide to break ties in favor of conflicts earlier in time. In that case, all the cardinal



(a) MAPF instance

(b) Break ties in favor of conflicts with the earliest timestep

(c) New conflict prioritization strategy

Fig. 1. Motivation example that demonstrates the importance of conflict prioritization when an agent faces multiple cardinal conflicts at different timesteps.

conflicts of a_1 have to be resolved one after the other by generating six high-level nodes. Whereas, Figure 1(c) shows the respective CT if we decide to break ties in favor of the cardinal conflict latest in time for a_1. This CT generates only two high-level nodes because by adding the constraint $a_1 \not\rightarrow (u_3, 6)$, CBS' low-level invokes A^* with conflict avoidance and therefore it finds a path that avoids the rest of the cardinal conflicts.

Furthermore, the new constraint $a_1 \rightarrow (u_3, 6)$ that was added in node N_1 in Figure 1(c) silently imposes two extra constraints because of proposition 1.

Definition 1. *A multi-value decision diagram (MDD_i^c) [10] is a compact representation of all possible paths of agent a_i with path length c that lead the agent to its goal vertex. The MDD_i^c is organized into levels, where each level t contains all vertices that the agent can occupy at timestep t along some path of cost c. The first level contains only the start vertex of the agent, and the last level contains only its goal vertex.*

Definition 2. *A node (v, t) in MDD_i^c is called a singleton node if it is the only node in level t.*

Proposition 1. *Let MDD_i^c where c is the length of a shortest path of agent a_i from its start vertex to its goal vertex, have a singleton node (u_3, t_3) at level t_3. A positive constraint $a_i \rightarrow (u_3, t_3)$ implies $a_i \rightarrow (v, t)$, $\forall (v, t) \in MDD_i^c$ that are singleton nodes and at level $t \leq t_3$.*

Proof. By contradiction, if $a_i \not\rightarrow (v, t)$ where (v, t) is a singleton node and $t \leq t_3$. Then there is no feasible path for a_i to be at (u_3, t_3) since all its feasible paths cross (v, t).

Therefore, $a_1 \rightarrow (u_3, 6)$ implies $a_1 \rightarrow (u_1, 2)$ and $a_1 \rightarrow (u_2, 4)$ too. Hence, to make the CBS' heuristics more efficient but also to avoid unnecessary node expansions later, the constraints $a_1 \rightarrow (u_1, 2)$ and $a_1 \rightarrow (u_2, 4)$ with their respective negative constraints for all other agents could also be added in node N_1 in Figure 1(c).

This tie-breaking technique can also be leveraged for agents that are involved in multiple semi-cardinal conflicts when the location-time of these conflicts are singleton nodes for the same agent. Let's assume that we have the same example as in Figure 1(a) but the conflicts of a_1 are now semi-cardinal and also $(u_1, 2)$, $(u_2, 4)$ and $(u_3, 6)$ are singleton nodes for a_1. Then we would get the same CT expansions as in Figures 1(b) and 1(c) for the two different aforementioned tie-breaking strategies. Moreover, since the location-times are singleton nodes for a_1, we can again add the silently imposed constraints $a_1 \rightarrow (u_1, 2)$ and $a_1 \rightarrow (u_2, 4)$ in node N_1 in Figure 1(c).

5.2 Algorithmic Extension

The algorithmic extension of CBS for our conflict prioritization is described in Algorithm 1. CBS already finds all conflicts in every high-level node in order to compute its high-level heuristics. For our conflict prioritization, we only need to do some extra bookkeeping. Whenever a new cardinal conflict between two agents a_i and a_j is found, it is added to the set of cardinal conflicts of both agents. Likewise, for the semi-cardinal conflicts with the only difference that it is added to the set of semi-cardinal conflicts for the agent that the location-time of the conflict is a singleton.

Subsequently, in each high-level node, we find the agent a_k with the most cardinal (or semi-cardinal singletons, if no more cardinal conflicts exist) conflicts and a multi-conflict is created. This multi-conflict includes all the cardinal (or semi-cardinal singletons) conflicts of a_k and it is chosen for the next conflict resolution. Finally, by resolving this multi-conflict, two new child nodes are created. The left child node adds positive constraints to a_k for all the conflicts that are included in the multi-conflict and the other one adds one negative constraint to a_k for the conflict with the latest timestep.

6 Experimental Results

To evaluate empirically the effect of our new conflict prioritization strategy to CBS, we developed two different variants of CBS with disjoint splitting in Python[3], namely a CBS that uses the CG heuristic and corridor reasoning

[3] https://github.com/GrFr/CBSH-CG-DG-WDG-RTC/tree/CBS-CG-WDG-RTC-multiconflict

Algorithm 1 Conflict Prioritization

Require: CardinalList : List of all cardinal conflicts
Require: SemiCardinalList : List of all semi-cardinal conflicts
Require: Singletons[a_i] : Set of all singleton nodes $\forall a_i \in A$
Cardinals[a_i] = $\emptyset$, $\forall a_i \in A$
SemiCardinals[a_i] = $\emptyset$, $\forall a_i \in A$
for each vertex or edge conflict $c = (a_i, a_j, u, t)$ **do**
  if c ∈ CardinalList **then**
    Cardinals[a_i].add(c)
    Cardinals[a_j].add(c)
  else if c ∈ SemiCardinalList **then**
    if $(u, t) \in$ Singletons[a_i] **then**
      SemiCardinals[a_i].add(c)
    else if $(u, t) \in$ Singletons[a_j] **then**
      SemiCardinals[a_j].add(c)
    end if
  end if
end for
if CardinalList $\neq \emptyset$ **then**
  $a_1 \leftarrow \arg\max_a$ |Cardinals[a]|
  NextConflict ← MultiConflict(Cardinals[a_1])
else if SemiCardinalList $\neq \emptyset$ **then**
  $a_1 \leftarrow \arg\max_a$ |SemiCardinals[a]|
  NextConflict ← MultiConflict(SemiCardinals[a_1])
end if
return NextConflict

(CBSH-CG-C) and a CBS that uses the most informative WDG heuristic and rectangle, corridor and target reasoning (CBSH-WDG-RTC). Moreover, in CBS with disjoint splitting, there are 3 different strategies (random, singleton and width) to decide which agent to constrain when a split action is performed and we tested all of them in our experiments. We used 4 maps from the standard MAPF benchmark suite [12] which are empty-16-16, random-32-32-10, random-32-32-20 and lak303d and we also created two new grid maps with dimensions 32 x 32 and obstacle densities of 30% and 40%. These maps were named random-32-32-30 and random-32-32-40 respectively.

For the small maps empty-16-16, random-32-32-10, random-32-32-20 and the big game map lak303d, we varied the agents from 10 to 60 and for the maps random-32-32-30 and random-32-32-40 we varied the agents from 10-35 since finding solutions in those heavily constrained maps gets hard really fast by increasing the number of agents. We generated 50 instances with randomly generated start and goal positions per number of agent and map and we set the time limit for finding a solution to 30 minutes. Finally the experiments were performed on a machine with an AMD EPYC 9655 (96-core) processor, with each run restricted to a single CPU core and 16 GB of RAM.

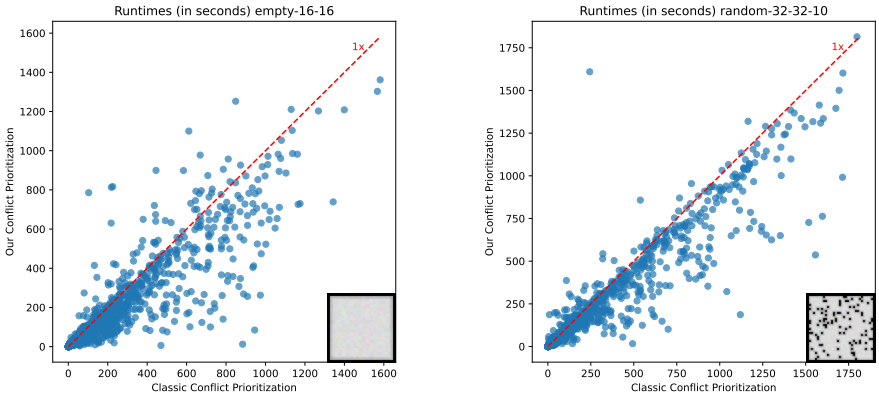
6.1 Results for CBSH-CG-C

In Figure 2 the runtime comparison for the two different conflict prioritization strategies is presented for every instance that is solved by both conflict prioritization strategies for CBS-CG-T. We named as "Classic Conflict Prioritization" the tie-braking in favor of conflicts earlier in time and "Our Conflict Prioritization" the conflict prioritization strategy described in the previous section. From these plots, it is apparent that our new conflict prioritization generally yields better runtimes across the majority of instances, notably on the smaller maps (empty-16-16, random-32-32-10, random-32-32-20, random-32-32-30, random-32-32-40). This observation is further supported by the numerical evaluation in Table 1 where we present the average runtimes and nodes expanded across all instances for specific numbers of agents. Our new conflict prioritization achieves up to 40% lower runtimes and node expansions.

Furthermore, in Table 2 you can see the frequency of resolving a multi-conflict (cardinal or semi-cardinal) per map. It is expected that multiconflicts emerge more often in smaller maps that are often more congested such as empty-16-16 and all the 32 x 32 maps with random obstacles. On the contrary in large maps like lak303d, multiconflicts emerge rarely and this is probably the reason that our new technique doesn't show any improvement in this map.

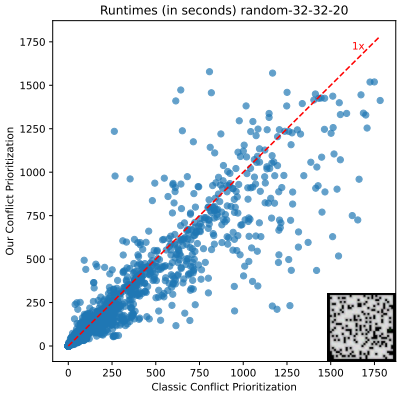
6.2 Results for CBSH-WDG-RTC

However, when we combine our new conflict prioritization technique with CBS-WDG-RTC, the previous advantage disappears as it is shown in Figure 3. This can be attributed to the use of symmetry reasoning techniques together with the WDG heuristic in CBS. In this CBS variant, conflicts such as rectangle and target

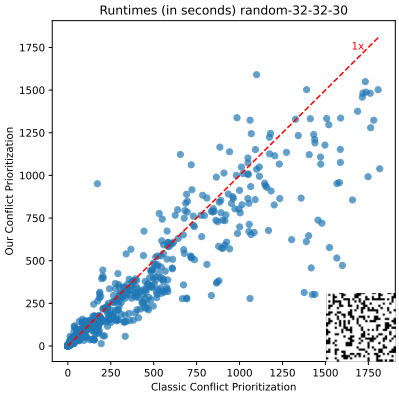


(a) empty-16-16

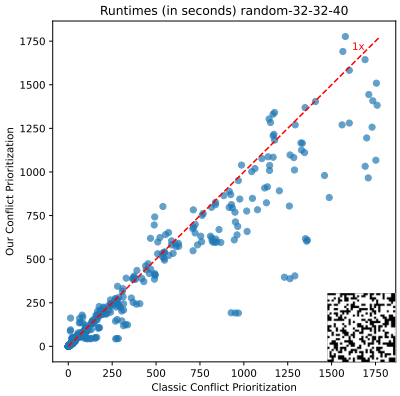
(b) random-32-32-10



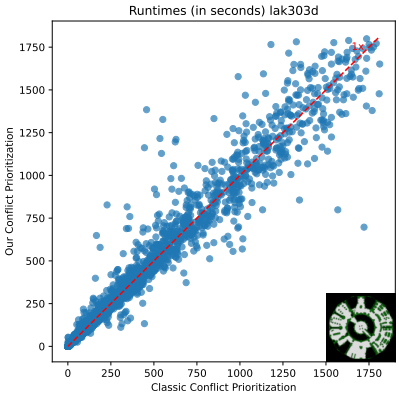
(c) random-32-32-20



(d) random-32-32-30



(e) random-32-32-40



(f) lak303d

Fig. 2. Runtime comparison for the two different conflict prioritization strategies with CBSH-CG-C

Table 1. Comparison of average high-level nodes expanded and runtimes for the two conflict prioritization strategies and CBSH-CG-C on different maps with varying number of agents

Map	Agents	Expanded Nodes		Runtimes (s)	
		Classic CP	Our CP	Classic CP	Our CP
empty-16-16	15	15.56	**14.25**	0.15	**0.14**
	25	756.40	**613.75**	18.79	**15.49**
	35	1560.38	**1123.56**	56.77	**40.59**
	45	3748.62	**2530.69**	188.50	**133.70**
	55	5170.43	**4368.00**	354.56	**322.40**
random-32-32-10	15	3.63	**3.57**	**0.10**	**0.10**
	25	349.09	**314.26**	15.01	**13.43**
	35	775.53	**699.14**	42.21	**38.20**
	45	1426.50	**1159.45**	119.59	**98.87**
	55	1901.27	**1607.71**	222.47	**190.75**
random-32-32-20	15	10.40	**9.69**	0.53	**0.45**
	25	81.82	**80.84**	**4.39**	4.64
	35	776.32	**702.93**	78.02	**68.54**
	45	**2111.40**	2205.95	**242.80**	246.49
	55	3185.04	**2304.57**	418.48	**318.85**
random-32-32-30	10	17.97	**14.43**	1.83	**1.43**
	20	537.91	**466.24**	95.35	**79.36**
	30	3056.67	**2655.75**	402.69	**319.60**
random-32-32-40	10	443.46	**364.82**	225.59	**182.24**
	15	624.71	**378.90**	280.72	**180.16**
	20	**7342.00**	7386.00	831.28	**811.66**
lak303d	20	**33.51**	33.99	125.16	**121.40**
	30	56.88	**55.08**	**412.86**	420.35
	40	**99.50**	102.83	**538.78**	553.37
	50	165.22	**149.89**	**694.13**	844.66

conflicts are identified and resolved before vertex and edge conflicts. Therefore the frequency of standard vertex and edge conflict resolutions is reduced, but so does their impact in the efficiency of CBS because they are typically resolved deeper in the CT. At that stage, avoiding unnecessary branching yields smaller performance gains, making the overall improvement less significant.

7 Conclusion and Future Work

Conflict prioritization plays an important role in the efficiency of CBS and in this work we developed a new conflict prioritization technique for CBS with disjoint splitting that reduces node expansions and runtimes for CBS-CG-C for the majority of the tested MAPF maps. Our new technique prioritizes the resolution of conflicts for agents that are involved in multiple cardinal (and semi-cardinal conficts) by prioritizing the resolution of their cardinal (or semi-cardinal) conflict with the latest timestep. However, when this technique is combined with

Table 2. Frequency of cardinal and semi-cardinal multi-conflict resolutions per map

Map	Cardinal multi-conflicts	Semi-cardinal multi-conflicts	Total Nodes	Ratio
empty-16-16	12007	486646	5062345	0.10
random-32-32-10	3363	438599	3683513	0.12
random-32-32-20	30427	233437	3828778	0.07
random-32-32-30	50378	45973	1300140	0.07
random-32-32-40	16435	8708	317007	0.08
lak303d	2686	4973	175691	0.04

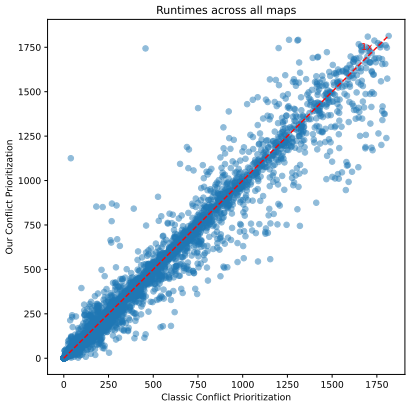


Fig. 3. Comparison of the runtimes for all maps and CBS-WDG-RTC

CBS-WDG-RTC, the results show no clear advantage, likely because the prioritization of vertex and edge conflicts becomes less significant since rectangular and and target conflicts are prioritized.

In future work, we aim to investigate the possibility of developing techniques that further prioritize other type of conflicts such as rectangle, corridor and target conflicts and by doing so, we may be able to improve the efficiency of CBSH-WDG-RTC too.

References

1. Boyarski, E., Felner, A., Bodic, P.L., Harabor, D.D., Stuckey, P.J., Koenig, S.: f-aware conflict prioritization & improved heuristics for conflict-based search. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 12241–12248. AAAI Press (2021)
2. Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., Shimony, S.E.: ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 740–746. AAAI Press (2015)
3. Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T.K.S., Koenig, S.: Adding heuristics to conflict-based search for multi-agent path finding. In: de Weerdt, M., Koenig, S., Röger, G., Spaan, M.T.J. (eds.) Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018. pp. 83–87. AAAI Press (2018)
4. Huang, T., Koenig, S., Dilkina, B.: Learning to resolve conflicts for multi-agent path finding with conflict-based search. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 11246–11253. AAAI Press (2021)
5. Li, J., Felner, A., Boyarski, E., Ma, H., Koenig, S.: Improved heuristics for multi-agent path finding with conflict-based search. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 442–449. ijcai.org (2019)
6. Li, J., Gange, G., Harabor, D., Stuckey, P.J., Ma, H., Koenig, S.: New techniques for pairwise symmetry breaking in multi-agent path finding. In: Beck, J.C., Buffet, O., Hoffmann, J., Karpas, E., Sohrabi, S. (eds.) Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020. pp. 193–201. AAAI Press (2020)
7. Li, J., Harabor, D., Stuckey, P.J., Ma, H., Koenig, S.: Disjoint splitting for multi-agent path finding with conflict-based search. In: Benton, J., Lipovetzky, N., Onaindia, E., Smith, D.E., Srivastava, S. (eds.) Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019. pp. 279–283. AAAI Press (2019)

8. Li, J., Harabor, D., Stuckey, P.J., Ma, H., Koenig, S.: Symmetry-breaking constraints for grid-based multi-agent path finding. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 6087–6095. AAAI Press (2019)
9. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: Hoffmann, J., Selman, B. (eds.) Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada. pp. 563–569. AAAI Press (2012)
10. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013)
11. Standley, T.S.: Finding optimal solutions to cooperative pathfinding problems. In: Fox, M., Poole, D. (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. pp. 173–178. AAAI Press (2010)
12. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. IEEE Trans. Comput. Intell. AI Games **4**(2), 144–148 (2012)
13. Yu, J., LaValle, S.M.: Multi-agent path planning and network flow. In: Frazzoli, E., Lozano-Pérez, T., Roy, N., Rus, D. (eds.) Algorithmic Foundations of Robotics X - Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2012, MIT, Cambridge, Massachusetts, USA, June 13-15 2012. Springer Tracts in Advanced Robotics, vol. 86, pp. 157–173. Springer (2012)