f-Aware Conflict Prioritization & Improved Heuristics For Conflict-Based Search

Eli Boyarski[1], Ariel Felner[1], Pierre Le Bodic[2], Daniel Harabor[2], Peter J. Stuckey[2], Sven Koenig[3]
[1]Ben-Gurion University of the Negev
[2]Monash University
[3]University of Southern California
boyarske@post.bgu.ac.il, felner@bgu.ac.il, {pierre.lebodic, peter.stuckey, daniel.harabor}@monash.edu, skoenig@usc.edu

Abstract

Conflict-Based Search (CBS) is a leading two-level algorithm for optimal Multi-Agent Path Finding (MAPF). The main step of CBS is to expand nodes by resolving conflicts (where two agents collide). Choosing the 'right' conflict to resolve can greatly speed up the search. CBS first resolves conflicts where the costs (g-values) of the resulting child nodes are larger than the cost of the node to be split. However, the recent addition of high-level heuristics to CBS and expanding nodes according to $f = g + h$ reduces the relevance of this conflict prioritization method. Therefore, we introduce an expanded categorization of conflicts, which first resolves conflicts where the f-values of the child nodes are larger than the f-value of the node to be split, and present a method for identifying such conflicts. We also enhance all known heuristics for CBS by using information about the cost of resolving certain conflicts with only a small computational overhead. Finally, we experimentally demonstrate that both the expanded categorization of conflicts and the improved heuristics contribute to making CBS even more efficient.

Introduction and Overview

Multi-Agent Path Finding (MAPF) is a coordination problem where the aim is to find a set of collision-free paths for a team of mobile agents, each from its start location to its designated target location. MAPF is a well-known and well-studied topic with numerous real-world applications. For example, MAPF is a core challenge automated warehouse logistics (Wurman, D'Andrea, and Mountz 2008), automated parcel sortation (Kou et al. 2020), automated valet parking (Okoso, Otaki, and Nishi 2019), computer games (Silver 2006) and a variety of other contexts (Ma et al. 2016). Many optimal and suboptimal approaches for MAPF have been proposed; a recent summary is given in (Felner et al. 2017).

In this work, we focus on optimal MAPF, where the goal is to minimize the sum of path costs, and consider improvement strategies for Conflict-Based Search (CBS) (Sharon et al. 2015), a popular and successful MAPF solver. CBS can be described as a two-level algorithm. The low level finds optimal paths for the individual agents. If the paths of two agents collide (i.e., are in conflict), the high level, via a split action, imposes constraints on the agents to avoid the collision, thus resolving the conflict. The search space of CBS is therefore a binary *Constraint Tree* (CT), which the algorithm explores in best-first order. CBS is complete, optimal and often highly performant; e.g., recent variants (Li et al. 2019a,b,c) can solve MAPF problems with more than 100 agents.

CBS is very sensitive to the order in which conflicts are chosen to be resolved. Boyarski et al. (2015b) introduced a scheme for prioritizing conflicts. Highest in the suggested priority ordering are *cardinal conflicts*, as resolving them raises the cost for both child nodes. Next are *semi-cardinal* conflicts, which raise the cost for only one child node, and last are *non-cardinal* conflicts, which do not raise the cost for either child node. This prioritization greatly speeds up the search by decreasing the size of the CT, because child nodes of higher costs are less likely to be expanded.

Later, Felner et al. (2018) and Li et al. (2019a) added heuristics to CBS, which provided further performance improvements. Nodes of the CT are now prioritized based on the sum of their cost and their heuristic value ($f = g + h$).

In this paper, we first propose an enhanced prioritization function that allows CBS to distinguish between conflicts that appeared equivalent when branching. We identify a set of conflicts, called *f-cardinal*, which produce child nodes with increased f-costs. We refer to conflicts that produce child nodes with increased g-costs as *g-cardinal*. Next, we propose a method for identifying f-cardinal conflicts and a new conflict prioritization scheme where f-cardinal conflicts are resolved first, followed by other types of conflicts. Then, we enhance all known heuristics for CBS by using information about the cost of resolving certain conflicts with only a small computational overhead. Finally, we evaluate both contributions experimentally. Our results indicate that this new strategy and the new heuristics increase the efficiency of current versions of CBS.

Background

In classical MAPF, the environment is a graph (usually a grid), and time is discretized to time steps. In each time step, an agent may traverse an edge to move from its vertex to an adjacent vertex, or it may wait at its current vertex. Agents have a vertex conflict if they are planned to occupy the same vertex at the same time, and have an edge conflict if they

are planned to traverse an edge in opposite directions at the same time. Agents are usually assumed to stay at their target when their plan ends until all agents have reached their target (Stern et al. 2019). An agent that has reached its target and finished its plan at time step t and is staying at its target from time step $t+1$ onward will cause a vertex conflict with any agent that passes through its target after time step t. Such conflicts are called *target conflicts* (Li et al. 2020).[1]

The first non-trivial heuristic function for CBS is given by the size of the minimum vertex cover (MVC) of the graph of the (g-)cardinal conflicts for the node (Felner et al. 2018). The (g-)cardinal conflict graph contains a vertex for each agent in CT node N, and an edge exists between two vertices iff the paths of the two corresponding agents have a (g-)cardinal conflict in N. For the purposes of the examples in the next two sections, we assume that the h-values are computed with this heuristic, which we denote as CG.

Cardinal Conflicts Are Insufficient

The integration of heuristics into CBS reduced the effectiveness of its conflict prioritization strategy. More specifically, when a cardinal (resp. semi-cardinal) conflict is resolved, the g-value of each child node (resp. one child node) is guaranteed to increase by 1 relative to its parent, but the f-value might not increase due to a corresponding decrease in the h-value. Currently, CBS does not distinguish between cardinal (resp. semi-cardinal) conflicts that will raise the f-value vs. those that will not, which diminishes the strength of the prioritization scheme and affects the resulting size of the CT. Consider, for example, a node with a single cardinal conflict. This conflict allows us to set the h-value to 1, because if we resolve that conflict, the g-value of the child nodes would increase by at least 1. When we resolve that conflict, the child nodes will have their g-value increased by 1 (or more), and unless new cardinal conflicts are found or the g-value increases by more than 1, the decrease in h at the child results in the same f-value as the parent. Thus, we have the same f-value as before, but two nodes to solve instead of one.

Distribution of Δg, Δh and Δf in Practice

Perhaps the first questions to ask are how often different combinations of f-value and g-value differences occur and whether we should we implement an algorithm to detect them. Hence, we first conduct a study to determine how prevalent each combination is.

Figure 1 shows the breakdown of Δg, Δh and Δf values of CT nodes relative to their parent when running a modern CBS solver with the basic MVC heuristic, which bypasses conflicts (see (Boyarski et al. 2015b)) and prioritizes (g-)cardinal conflicts, on the standard MAPF benchmark suite from (Stern et al. 2019). The values are shown separately for each map type (see the experimental results section for more details).

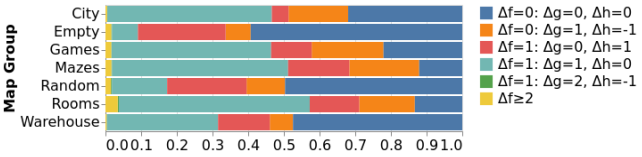

Figure 1: Distribution of Δf values by map types, and their Δg and Δh components, using the CG heuristic.

The orange bars in Figure 1, where $\Delta g = 1$ but $\Delta h = -1$ (i.e., $\Delta f = 0$) illustrate the potential for improvement in CBS. In the previous versions of CBS, conflicts that result in child nodes with values from the orange, teal, and green bars, and some conflicts from the yellow bar, where $\Delta \geq 1$, would be prioritized for resolution in a CT node over conflicts that result in child nodes from blue or red bars. Yet resolving conflicts that result in child nodes with values from the orange bar does not increase the f-value of the child nodes, while resolving conflicts that result in child nodes with values from red bars does. We will be much more interested to resolve conflicts that result in $\Delta f > 0$, as these better represent an increase in the f-value of the child nodes. Thus, the red bars, where $\Delta f = 1$, are preferable to the orange bars. As the figure shows, the portion of such conflicts is substantial.

Different Types of Conflicts

We say that a CT node N is defined by a set of constraints, denoted $N.constraints$, which restrict individual paths that can be assigned to agents. The node also records its g-value (the sum of individual path costs) as $N.g$. When using a heuristic function, N stores in a similar way its corresponding h- and f-values. We are now ready to expand the classification of conflicts from three distinct categories, as given in (Boyarski et al. 2015b), to five possibly-overlapping categories, as follows:

(1) f-Cardinal conflict. A conflict $C = \langle a_1, a_2, v, t \rangle$ is *f-cardinal* for a CT node N iff adding either of the two constraints derived from C (namely, $\langle a_1, v, t \rangle$, $\langle a_2, v, t \rangle$) to $N.constraints$ and invoking the low-level search for the constrained agent a_i, the f-value of the resulting CT node increases compared to $N.f$.

(2) Semi-f-Cardinal conflict. $C = \langle a_1, a_2, v, t \rangle$ is *semi-f-cardinal* for a CT node N iff adding one of the two constraints derived from C to $N.constraints$ increases $N.f$ but adding the other leaves $N.f$ unchanged.

Categories (1-2) may overlap with categories (3-5):

(3) g-Cardinal conflict. A conflict $C = \langle a_1, a_2, v, t \rangle$ is *g-cardinal* (previously simply called cardinal) for a CT node N iff adding either of the two constraints derived from C ($\langle a_1, v, t \rangle$, $\langle a_2, v, t \rangle$) to $N.constraints$ and invoking the low-level on the constrained agent, the cost of its path increases compared to its cost in N.

[1] Li et al. (2020) suggest to resolve a target conflict at time t by branching on the path length l of the agent that is at its target. One child node receives a constraint that $l > t$, and the other child node receives the constraint that $l \leq t$, meaning *all* other agents are constrained from passing through the agent's target after time t. This method is orthogonal to our work, as resolving target conflicts in this way contributes to a smaller CT, but does not change the minimum expected cost increases we discuss in this paper.

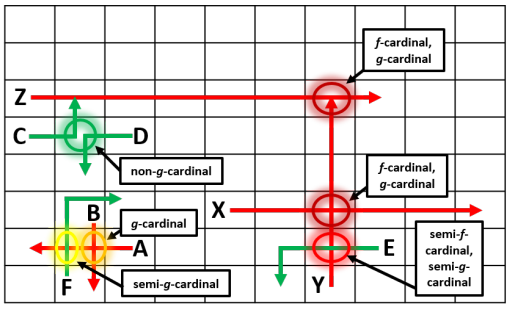

Figure 2: Agents, their paths, and their conflicts

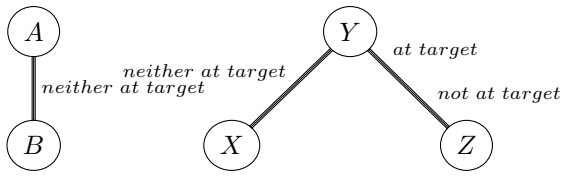

Figure 3: The g-cardinal conflict graph for Figure 2, with labels showing whether agents have finished their plan at the time of the conflict

(4) Semi-g-cardinal conflict. $C = \langle a_1, a_2, v, t \rangle$ is *semi-g-cardinal* (previously called semi-cardinal) for a CT node N iff adding one of the two constraints derived from C to N.*constraints* increases $N.g$ but adding the other one leaves $N.g$ unchanged.

(5) Non-g-cardinal conflict. A conflict C is *non-g-cardinal* for a CT node N iff neither of the two constraints derived from C increases $N.g$.

We propose that f-cardinal conflicts be resolved first, followed by semi-f-cardinal, g-cardinal, semi-g-cardinal and finally non-g-cardinal conflicts. Depending on Δh, a g-cardinal conflict may or may not also be f-cardinal or a semi-f-cardinal, and a semi-g-cardinal conflict may or may not also be semi-f-cardinal.

Figure 2 shows an example with 9 agents and their paths in CT node N. Paths in red have no alternatives of the same cost. Paths in green do. Figure 3 shows the *g-cardinal conflict graph* (Felner et al. 2018) for N. The conflict between agents Y and Z occurs while Y is waiting at its target (unlike the conflict between agents Y and X). The size of the MVC of the g-cardinal conflict graph, and hence the h-value, is 2. We examine different conflicts on this graph next.

The conflict between agents A and B is g-cardinal. Resolving it generates child nodes with $\Delta g = 1$, and the size of the MVC decreases by 1 (unless new cardinal conflicts are caused by the new paths of agents A and B). Thus, based only on information from this conflict graph, the child nodes have $\Delta h = -1$ and $\Delta f = 0$.

Had the conflict between agents Y and Z not been at Y's target, the conflicts between Y and Z and between X and Y would both have been semi-f-cardinal. Replanning X to avoid the conflict with Y would have yielded $\Delta g = 1$, $\Delta h \geq 0$, and $\Delta f \geq 1$. Similarly replanning Z for the conflict (Y, Z). Replanning Y to avoid either the conflict with X or the conflict with Z would have yielded $\Delta g = 1$, $\Delta h \geq -1$, and $\Delta f \geq 0$.

The conflict between agents Y and Z is both f-cardinal and g-cardinal (recall that Y is at its target). If it is resolved in CT node N, the child node that constrains Y will have $\Delta g \geq 2$, because Y is forced away from its target. It will have $\Delta h = -1$, assuming the cost increase would allow Y to also avoid the conflict with X (unless Y's new path causes a new g-cardinal, in which case $\Delta h \geq -1$) and $\Delta f \geq 1$. The child node that constrains Z will have $\Delta g = 1$,$\Delta h \geq 0$ because the MVC of the remaining g-cardinal conflict graph would be of size (at least) 2 and $\Delta f \geq 1$. Thus, this conflict is f-cardinal.

The conflict between X and Y is also both f-cardinal and g-cardinal. If it is resolved in CT node N, the child node that constrains agent X (to avoid the conflict between X and Y) will have $\Delta g = 1$, $\Delta h \geq 0$ and $\Delta f \geq 1$, similarly to the child node that constrains Z in the previous paragraph. The child node that constrains Y will have $\Delta g = 1$, $\Delta h \geq 0$ because Y's cost would not increase enough to avoid the conflict with Z, and $\Delta f \geq 1$.

f-Cardinal Conflicts

In this section, we propose a technique for identifying f-cardinal conflicts. Our contributions generalize previous conflict identification strategies, which focus only on g-increases (Boyarski et al. 2015b).

Target Conflicts Are At Least Semi-f-Cardinal

When CBS resolves a target conflict in CT node N, in one of the child nodes of N, a vertex constraint on an agent's target location is added at time step t' that is *later* than time step t when the agent is planned to reach its target. Such a constraint will result in a cost increase of at least 2 ($\Delta g \geq 2$), because the earliest time step the agent's plan can end in is now $t' + 1 \geq t + 2$. Moreover, under the basic MVC high-level heuristic (discussed further in the next section), replanning the path of an agent can decrease a node's h-value by at most 1 (if its vertex was in the MVC), yielding $\Delta g \geq 2$, $\Delta h \geq -1$ and $\Delta f \geq 1$. Thus, target conflicts are either f-cardinal or semi-f-cardinal under this heuristic. Identifying such target conflicts is trivial.

Identifying f-Cardinal Conflicts

To identify f-cardinal conflicts in a CT node N, we first identify g-cardinal conflicts in the usual way (see (Boyarski et al. 2015b)). That is, for each agent that has a conflict, we build an MDD that represents all of its paths that are of the same cost as its current one. A conflict is g-cardinal neither MDD of its constituent agents contain an alternative that avoids the conflict at the time step of the conflict.

Then, we construct the g-cardinal conflict graph for N and compute N's h-value to be the size of the MVC of the graph. Finally, we iterate over all agents that have edges in the g-cardinal conflict graph. For each such agent A, we temporarily remove all of agent A's edges and compute the size

of the MVC of the remaining graph. This simulates the g-cardinal conflict graph of the child node of N that would be generated if one of A's g-cardinal conflicts were chosen to be resolved and the child node constrained agent A's path. The graph simulates the most optimistic scenario that agent A's new path causes no new g-cardinal conflicts and avoids *all* of A's current conflicts. If the size of the MVC remains unchanged, than all g-cardinal conflicts that agent A participates in are semi-f-cardinal (like agent X and the X-Y-conflict in the example); and, all g-cardinal conflicts that agent A participates in, and where the other conflicting agent is at its target, are f-cardinal (like agent Z and the Y-Z-conflict). Otherwise, the size of the MVC decreases and thus the agent's g-cardinal conflicts remain g-cardinal (but are not f-cardinal in our scenario).

In our implementation, we compute the MVC using a Mixed-Integer Programming (MIP) solver. When we simulate the MVC of child nodes, we 'warm start' the MIP model, removing rows and incrementally adding them back to the model as needed. MIP solvers retain information between consecutive runs on the same model, which speeds up the process of identifying f-cardinal conflicts.

Non-Stable Conflicts

For any g-cardinal conflict identified at CT node N, the conflict will remain g-cardinal at every node in the subtree rooted at N unless resolved in an ancestor. We call such conflicts *stable*. Interestingly, f-cardinal conflicts are not stable, as such a conflict in CT node N might become semi-f-cardinal in a child, despite not being resolved. Similarly, a semi-f-cardinal conflict might become only g-cardinal in a child node.

In Figure 3, if we resolve the conflict between X and Y, the conflict between Y and Z will become semi-f-cardinal in the resulting child nodes, and if we resolve the conflict between Y and Z, the conflict between X and Y will become only g-cardinal in the CT node where the path of Z was replanned (and might be avoided altogether in the CT node where the path of Y was replanned). Due to this observation, when an f-cardinal conflict is identified in a CT node, we propose to terminate the identification procedure and immediately resolve this conflict. Of course, it is not worthwhile to perform additional work to possibly identify other f-cardinal conflicts, as the f-cardinal status of these conflicts would have to be re-checked in the child nodes (since the conflicts are not stable).

Improved Heuristics for CBS

Conflict prioritization affects which nodes CBS generates and adds to the CT and therefore strongly influences both the shape and the size of the CT and thus also the efficiency of CBS. Complementary to this strategy is the CBS high-level heuristic, which influences the order in which CT nodes will be selected for expansion. In this section, we show how to exploit information about expected g-increases, gathered at the conflict classification stage, to improve all current CBS high-level heuristics: CG (Felner et al. 2018), DG and WDG (Li et al. 2019a). Our new and improved heuristics

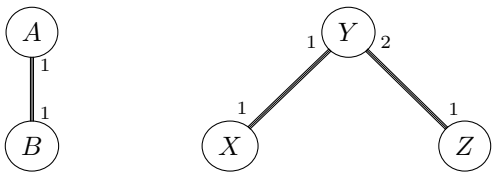

Figure 4: The g-cardinal conflict graph for figure 2, with minimum expected g-increases labeled on the ends of edges

are strictly more informed than their baselines. They may be used in conjunction with our newly proposed conflict prioritization scheme or independently of it.

The NVW-CG Heuristic

Figure 4 shows the g-cardinal conflict graph for the example in Figure 2, with minimum expected g-increases from resolving the g-cardinal conflicts labelled on the ends of the edges[2]. Recall that the h-value of the basic MVC heuristic for this graph is 2.

Our new Near Vertex-Weighted Cardinal Conflict Graph Minimum Vertex Cover (NVW-CG) heuristic for the labelled graph assigns a non-negative integer value x_i for each vertex v_i. The heuristic estimate is the sum of the minimum assignment of values for the vertices that satisfies the following condition for edges: each edge $e = (v_i, v_j)$ with corresponding values (c_{e_i}, c_{e_j}) on its ends is covered by $x_i \geq c_{e_i} \vee x_j \geq c_{e_j}$. For our example, the improved heuristic estimate is 3: either A or B is assigned a value of 1, and either X and Z are assigned a value of 1, or Y a value of 2.

Note that the 1 label on the side of Z is underestimating the cost to really resolve the dependency between agents Y and Z by replanning agent Z's path. The near-vertex weights only take into account the expected g-increases from resolving the current conflict. They are thus trivial to compute, but miss some information. In our example, adding a constraint on Y's target at time step 6 for agent Z would simply cause agent Z to wait one time step before entering the same vertex and creating a new f-cardinal conflict.

The NVW-DG Heuristic

The Dependency Graph Minimum Vertex Cover (DG) heuristic computes the MVC of the *pairwise dependency graph* (Li et al. 2019a), which generalizes the g-cardinal conflict graph. DG edges exist iff all cost-minimal paths of the corresponding two agents have any type of conflict between them (not necessarily g-cardinal). Thus, the DG can have more edges than the CG, and the size of its MVC can be larger. The Near Vertex-Weighted version of the DG heuristic (NVW-DG) is computed as follows: add a weight to the end of each edge in the DG, corresponding to the minimum g-increase from resolving the conflict that the edge

[2]Resolving the conflict between agents Y and Z with length constraints, as discussed earlier, actually increases the cost of the path of agent Z by 2 when it is replanned, but that would only be discovered when that child node is generated.

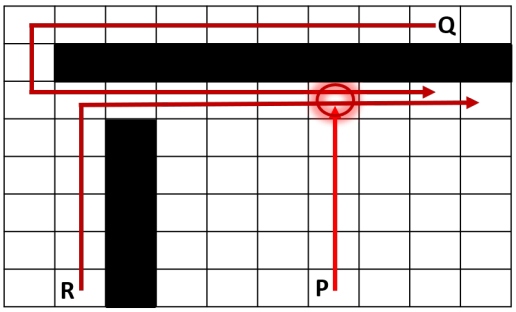

Figure 5: Agents, their paths, and their conflicts

represents by replanning the agent that the vertex represents. Then, compute a weighted minimum vertex cover that satisfies an additional condition: an edge is only considered covered by a weight on a vertex if the weight is at least the same as the weight of the corresponding end of the edge.

The NVW-WDG Heuristic

The (Edge-)Weighted Dependency Graph Minimum Vertex Cover (WDG) heuristic (Li et al. 2019a) is the most advanced heuristic for the high level of CBS to date. It constructs the Weighted Dependency Graph for the agents by setting the weight of each edge between a pair of agents to the difference between the cost of the optimal solution to their 2-agent MAPF subproblem and the sum of the costs of their current paths. Note the graphs used for the previous two basic heuristics are unweighted. Then, it calculates the heuristic value $h = \sum_i x_i$ as a minimal edge-weighted vertex cover of the resulting graph, that is, an assignment of non-negative integers x_i to each vertex (agent) v_i so that, for each edge (v_i, v_j) with cost w_{ij}, $x_i + x_j \geq w_{ij}$.

Even this advanced heuristic can be enhanced with near-vertex weights. Figure 6 shows the weighted dependency graph for Figure 5 with minimum expected cost increases labeling the ends of the edges. The weight of both edges is 2 because agents Q and R can circumnavigate agent P at its target by taking one extra move down before reaching it and one extra move up after passing it. The near-vertex weights for P are 12 on the edge with Q and 6 on the edge with R, because once an agent is forced away from its target, all of the wait actions at its target are added to the cost of its path. Similarly to the previous example, using length constraints to resolve the conflict would increase the cost of the paths of agents P and R by 2 when they are replanned, but that information is not known at this stage. The WDG heuristic gives a value of 2 here, assigning 2 to agent P. Yet, we know that increasing the cost of the path of agent P by 2 would not even resolve the current conflicts. The NVW-WDG heuristic gives a value of 4 here, assigning 2 to Q and R.

Discussion

One might expect fewer f-cardinal conflicts to exist when a more informed heuristic is used, because g-increases in child nodes would often already be factored into the h-value

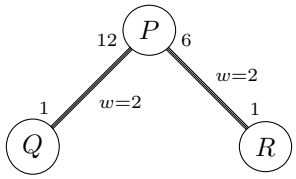

Figure 6: The weighted dependency graph for 5, with expected g-increases labeled on the ends of edges

of the parent, leading to $\Delta g = -\Delta h$. However, our experiments found that the opposite was true. This may be explained by 'bad choices' of constraints that do not readily lead to a conflict-free solution, leading to an unwarranted g-increase in addition to an h-increase.

With a 'good' conflict prioritization strategy, CBS resolves only a minimal number of conflicts *along each branch* before generating a feasible solution. With a 'good' heuristic function, CBS expands only a minimal number of CT nodes *globally* on the way to the target. But, at each expansion step CBS must still decide on what conflict to base the generation of child nodes that would be added to the CT. Suppose now that there exists one g-cardinal conflict at the root of the CT and many non-g-cardinal conflicts. The CG heuristic estimate at the root is therefore 1 (we use CG as an illustrative example; similar examples can be constructed for any heuristic). From the perspective of the heuristic, each non-g-cardinal conflict produces two successors with $\Delta g = 0$ and $\Delta h = 0$, and the g-cardinal conflict produces two successors with $\Delta g = 1$ and $\Delta h = -1$. In other words, it seems that the conflicts could be selected and resolved in any order, since the result from each one appears equally promising. If the g-cardinal conflict is resolved first, the search may quickly find an optimal solution, guided only by the high-level tie-breaking strategy. On the other hand, if the g-cardinal conflict is always resolved last, the search may generate a large CT and time out before expanding a goal node. Thus, we see the two key reasons why conflict prioritization is essential for CBS: (i) to keep the CT small and (ii) to put optimal solutions within reach.

Experimental Results

All our experiments were run on a Linux machine with an Intel Xeon CPU E5-2630 v2 running at 2.60GHz, with memory usage limited to 8GB. We used Gurobi 8.1.1 to solve the MIP models of the MVCs of the various heuristics.

We experimented on the MAPF benchmark (Stern et al. 2019), which contains 32 grids of different types (city maps, grids with random obstacles, mazes, warehouse maps, etc.), each with 25 scenarios that specify the start and target locations for up to 7,000 agents. We increased the number of agents on these scenarios until we reached the runtime limit of 60 seconds. For higher numbers of agents, the solver was considered to have failed implicitly.

The baseline solver in all of our experiments was only aware of g-cardinal conflicts, used the CG heuristic, and implemented bypassing conflicts (Boyarski et al. 2015a).

Type	#Instances	#Solved		#Solved (Hard)	
		CBS	f-cardinal	CBS	f-cardinal
City	36,300	12,767	**13,071**	270	**574**
Empty	48,400	9,209	**9,283**	163	**237**
Games	121,000	20,745	**21,721**	806	**1782**
Mazes	48,400	2,584	**2,728**	117	**261**
Random	48,400	11,454	**11,767**	333	**631**
Rooms	36,300	3,986	**4,231**	149	**394**
Warehouse	48,400	17,357	**17,627**	249	**519**
Total	387,200	78,102	**80,428**	2,087	**4,398**

Table 1: Number of solved instances for CBS and CBS-f-cardinal for all instances and for hard instances

Results for the New Conflict Prioritization

We ran two CBS solvers: The baseline solver prioritized g-cardinal conflicts, and our improved solver identified and prioritized f-cardinal conflicts. Both used the CG heuristic.

Solved instances: Table 1 shows the success rates for the two solvers and each of the different map types. The number of problem instances from each type is also presented. There were 387,200 problem instances and, for each one, each solver either attempted to solve it or failed implicitly. We further distinguish a subset of *hard instances*, which we define as any problem instance where at least one solver failed or required more than half of the allotted time to solve (30 seconds).

There were 2,464 problem instances that were solved only by our improved solver and 153 problem instances that were only solved by the baseline solver. The improvement is statistically significant (McNemar's Test produces $\chi^2 = 2040.78$). Prioritizing f-cardinal conflicts is especially beneficial on hard instances, where it more than doubles the success rate.

Co-solved instances: Figure 7 shows the average runtime and number of generated nodes by the two solvers of CBS, over all instances from each map type that both solvers solved successfully, as a function of the number of agents. Our improved solver consistently generates fewer nodes and runs faster. The error bars represent one standard deviation from the mean for each number of agents. Our solver clearly has lower standard deviations for both the runtimes and the number of generated nodes. The time that it spent on identifying f-cardinal conflicts was negligible.

Results for the New Heuristics

We compare the performance of CBS with the CG (Felner et al. 2018), DG, WDG (Li et al. 2019a), NVW-CG, NVW-DG, and NVW-WDG heuristics. All solvers compute the heuristics lazily and use memoization (Li et al. 2019a) to reduce runtime. The solver used by WDG for the 2-agent subproblems was CBS with the same configuration, except it runs with the CG heuristic, and applies rectangle reasoning (Li et al. 2019c), a technique that further speeds up CBS.

Previous heuristic evaluations have only been performed on a small number of scenarios. Here we examine 111,480 instances. In total, 97,706 problem instances were successfully solved with at least one of the six heuristics. Table 2 shows the number of instances that were solved successfully by each solver for the different map types. Interestingly, while improving the weaker heuristics does not improve success rate, for WDG overall it does. We conjecture this is because with this short time limit, the overhead of new heuristic calculations does not pay off.

Type	#Instances	CG	N-CG	DG	N-DG	WDG	N-WDG
City	19,322	12,599	12,615	15,664	15,634	15,835	**15,972**
Empty	12,197	9,104	9,106	**11,659**	11,626	11,561	11,541
Games	31,450	20,843	20,812	23,686	23,612	25,875	**26,078**
Mazes	3,699	2,766	2,759	2,779	2,774	2,928	**2,950**
Random	16,006	11,645	11,639	**14,490**	14,417	14,387	14,344
Rooms	5,710	4,190	4,169	4,528	4,489	4,825	**4,844**
Warehouse	23,096	17,124	17,117	20,616	20,569	20,765	**20,780**
Total	111,480	78,271	78,217	93,422	93,121	96,176	**96,509**

Table 2: Solved instances with the CG, NVW-CG, DG, NVW-DG, WDG and NVW-WDG heuristics.

Type	CG	N-CG	DG	N-DG	WDG	N-WDG	h^*
City	1.30	1.40	1.32	1.42	2.32	**2.51**	2.62
Empty	0.18	0.18	0.48	0.48	0.62	**0.62**	1.11
Games	1.86	1.96	1.89	1.99	3.41	**3.60**	3.99
Mazes	1.79	1.92	1.80	1.92	4.15	**4.39**	5.64
Random	1.27	1.35	1.52	1.61	2.88	**3.09**	3.79
Rooms	1.80	1.90	1.90	2.01	4.03	**4.32**	5.45
Warehouse	0.50	0.51	0.57	0.58	1.66	**1.70**	1.94
All	1.18	1.25	1.29	1.35	2.51	**2.65**	3.09

Table 3: Average h-values of the root node with each heuristic and with h^* for each map type, on co-solved instances.

Table 3 shows the average h-value of the root CT node on co-solved instances with each heuristic, as well as the average optimal h-value (h^*). The average h-value of each improved heuristic is better than its baseline. Table 4 shows the average number of generated nodes of each solver on co-solved instances, and the average time in milliseconds per generated node, under 60-seconds, 300-seconds and 1800-seconds time limits. Table 4 provides evidence to the fact that the improved heuristics are more important for longer runs.

Figure 8 shows again the breakdown of Δg, Δh, and Δf values of CT nodes relative to their parent, this time when running the same CBS solver with the NVW-WDG heuristic. Note that new colours were added for cases where $\Delta h \leq -2$, which isn't possible with the CG and DG heuristics. Without prioritizing f-cardinal conflicts, conflicts from the orange bars, where $\Delta g = 1$ but $\Delta f = 0$, would be prioritized over conflicts from the teal bars, where $\Delta f = 1$ but $\Delta g = 0$.

Conclusions and Future Work

In this paper, we first expanded the concept of cardinal conflicts, a key concept in CBS, and demonstrated that prioritiz-

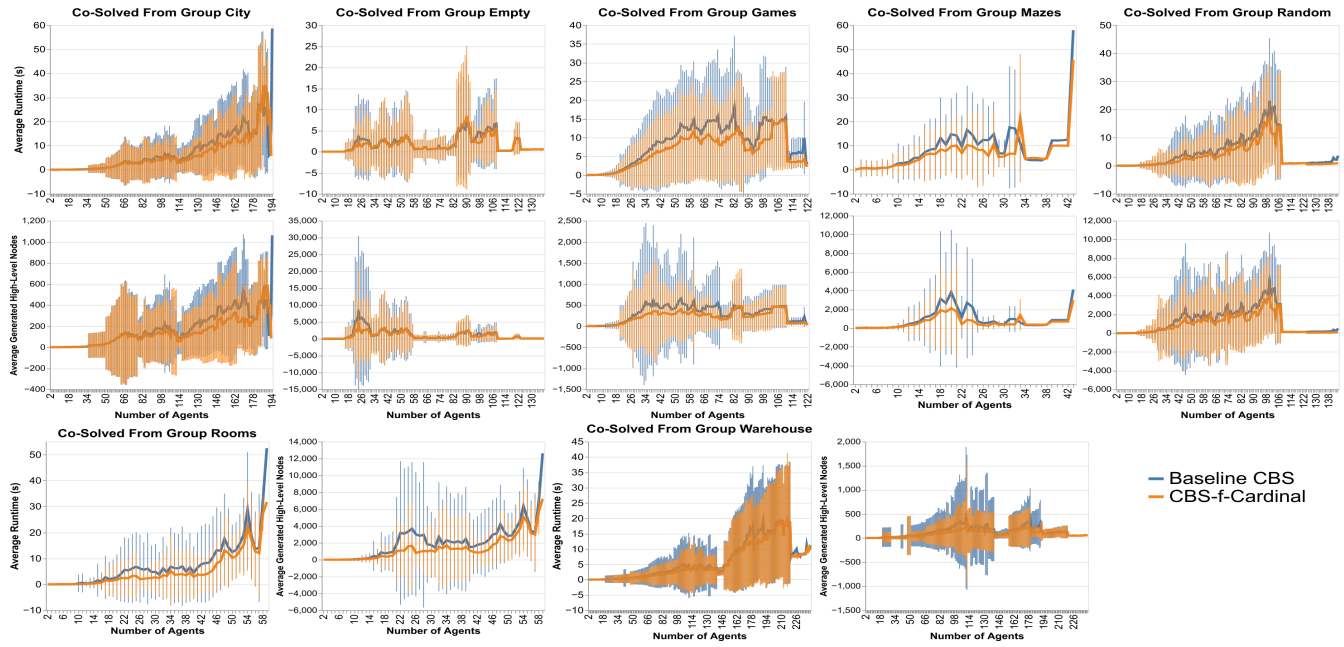

Figure 7: Average runtime and generated nodes per map type for the subset of instances solved by both CBS and CBS-f-cardinal.

Limit(s)	#Instances	CG	NVW-CG	DG	NVW-DG	WDG	NVW-WDG
60	77,728	2.0,**1.17ms**	2.0,1.21ms	0.8,2.35ms	0.8,2.47ms	0.7,2.45ms	**0.7**,2.45ms
300	82,135	7.8,**1.14ms**	7.8,1.19ms	3.2,2.12ms	3.2,2.28ms	2.6,1.72ms	**2.6**,1.68ms
1800	85,735	31.8,**1.16ms**	31.7,1.22ms	9.4,2.63ms	9.3,2.80ms	7.3,1.90ms	**7.1**,1.78ms

Table 4: Average generated nodes in thousands and average time per node in ms using the CG, NVW-CG, DG, NVW-DG, WDG and NVW-WDG heuristics for co-solved instance under three time limits.

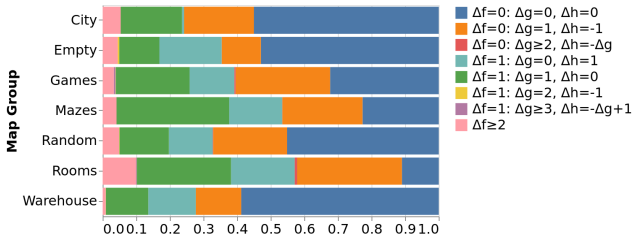

Figure 8: Distribution of Δf values by map type, and their Δg and Δh components, using the NVW-WDG heuristic.

ing the new category of f-cardinal conflicts substantially improves performance. Second, we improved all current high-level heuristics for CBS by exploiting information about expected cost increases that arise from resolving conflicts. Our experimental results demonstrate that our new heuristics are the strongest known heuristics for CBS so far. We demonstrated each enhancement separately on a large suite of benchmarks – the largest comparative evaluation undertaken for CBS specifically and MAPF more generally.

It remains future work to evaluate the joint improvement.

Other future work includes:

- Under *Disjoint Splitting* (Li et al. 2019b), only one of the conflicting agents is constrained in both child nodes. We plan to investigate how to choose an agent that yields higher Δf values in the child nodes.
- Li et al. (2020) define *corridor conflicts*, which occur when two agents attempt to traverse a 1-width corridor in opposite directions. One agent has to wait for the other agent to fully traverse the corridor before entering it, or use a different path. If the earliest times when the agents can reach a corridor entrance are different, then the amount of time they have to wait for the other agent is also different. We plan to encode this asymmetry into a near vertex-weighted heuristic, as we have done for target conflicts.
- We plan to study admissible or inadmissible heuristics for sub-optimal or bounded-sub-optimal CBS-based algorithms (Barer et al. 2014). For example, non-g-cardinal conflicts could be modeled as ϵ-cost edges in the heuristic graph, to represent the amount of work needed to resolve them.

Our code is available at https://github.com/eli-b/fcardinal.

Acknowledgments

The research at Ben-Gurion University of the Negev was supported by the Israel Ministry of Science, ISF grant 844/17 and BSF grant 2017692. The research at Monash University is partially supported by ARC grant DP200100025. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, and 1935712 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. *Proceedings of the Annual Symposium on Combinatorial Search (SoCS-2014)* 19–27. ISSN 09226389. doi:10.3233/978-1-61499-419-0-961.

Boyarski, E.; Felner, A.; Sharon, G.; and Stern, R. 2015a. Don't Split, Try To Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2015)*, 47–51. ISBN 9781577357315. ISSN 23340843.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. S. 2015b. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2015)*, 740–746. ISBN 9781577357384. ISSN 10450823.

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 International Conference on Automated Planning and Scheduling (ICAPS-2018)*, 83–87.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; and Surynek, P. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS-2017)*, 29–37.

Kou, N. M.; Peng, C.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2020. Idle Time Optimization for Target Assignment and Path Finding in Sortation Centers. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2020)*.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2019)*, 442–449. doi:10.24963/ijcai.2019/63.

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 International Conference on Automated Planning and Scheduling (ICAPS-2020)*, 193–201. ISSN 23340843.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019b. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2019)*, 279–283.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019c. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2019)*, 6087–6095. ISSN 2159-5399. doi:10.1609/aaai.v33i01.33016087.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2016. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

Okoso, A.; Otaki, K.; and Nishi, T. 2019. Multi-Agent Path Finding with Priority for Cooperative Automated Valet Parking. In *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC-2019)*, 2135–2140. doi: 10.1109/ITSC.2019.8917112.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 219: 40–66.

Silver, D. 2006. Cooperative Pathfinding. In Rabin, S., ed., *AI Game Programming Wisdom 3*, 99–111. Charles River Media.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS-2019)*, 151–159.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* 29(1): 9–19.