EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding

Jiaoyang Li,[1][*] **Wheeler Ruml,**[2] **Sven Koenig**[1]

[1]University of Southern California,
[2]University of New Hampshire
jiaoyanl@usc.edu, ruml@cs.unh.edu, skoenig@usc.edu

Abstract

Multi-Agent Path Finding (MAPF), i.e., finding collision-free paths for multiple robots, is important for many applications where small runtimes are necessary, including the kind of automated warehouses operated by Amazon. CBS is a leading two-level search algorithm for solving MAPF optimally. ECBS is a bounded-suboptimal variant of CBS that uses focal search to speed up CBS by sacrificing optimality and instead guaranteeing that the costs of its solutions are within a given factor of optimal. In this paper, we study how to decrease its runtime even further using inadmissible heuristics. Motivated by Explicit Estimation Search (EES), we propose Explicit Estimation CBS (EECBS), a new bounded-suboptimal variant of CBS, that uses online learning to obtain inadmissible estimates of the cost of the solution of each high-level node and uses EES to choose which high-level node to expand next. We also investigate recent improvements of CBS and adapt them to EECBS. We find that EECBS with the improvements runs significantly faster than the state-of-the-art bounded-suboptimal MAPF algorithms ECBS, BCP-7, and eMDD-SAT on a variety of MAPF instances. We hope that the scalability of EECBS enables additional applications for bounded-suboptimal MAPF algorithms.

1 Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding collision-free paths for a team of agents in a known environment while minimizing the sum of their travel times. It is inspired by real-world applications such as warehouse logistics (Ma et al. 2017), airport operations (Li et al. 2019d), UAV traffic management (Ho et al. 2019), automated valet parking (Okoso, Otaki, and Nishi 2019), and video games (Li et al. 2020b).

CBS (Sharon et al. 2015) is a leading two-level search algorithm for solving MAPF optimally. Its central idea is to plan a path for each agent independently and then resolve collisions between two agents by branching. Each branch is a new candidate plan wherein one agent or the other is forced to find a new path that avoids the chosen collision. Researchers have made significant progress on speeding up CBS (Boyarski et al. 2015a,b, 2020a,b; Felner et al. 2018;

Gange, Harabor, and Stuckey 2019; Li et al. 2019b,c, 2020a; Zhang et al. 2020) and designing compilation-based algorithms that use ideas similar to CBS (Surynek 2019; Lam et al. 2019; Lam and Le Bodic 2020). However, for many applications, we need to coordinate hundreds of agents with limited computational resources. The fact that MAPF is NP-hard to solve optimally (Yu and LaValle 2013) motivates finding bounded-suboptimal solutions in order to reduce the runtime of the search.

Enhanced CBS (ECBS) (Barer et al. 2014) is a bounded-suboptimal variant of CBS that is guaranteed to find solutions whose costs are no more than a user-specified factor away from optimal. Its bounded suboptimality is achieved by replacing the best-first search in the high and low levels of CBS with focal search (Pearl and Kim 1982). Focal search uses an admissible heuristic for bounding the solution cost and another heuristic for determining the distance of nodes to the goal nodes. We will demonstrate that ECBS becomes inefficient if these heuristics are negatively correlated.

In this paper, we propose a new bounded-suboptimal variant of CBS, called Explicit Estimation CBS (EECBS). EECBS replaces focal search with Explicit Estimation Search (Thayer and Ruml 2011) on the high level and uses online learning (Thayer, Dionne, and Ruml 2011) to learn an informed but potentially inadmissible heuristic to guide the high-level search. ECBS and EECBS differ from CBS only in the node selection rules used in their high- and low-level searches. Hence, the ideas behind many improvements of CBS, such as bypassing conflicts (Boyarski et al. 2015a), prioritizing conflicts (Boyarski et al. 2015b), symmetry reasoning (Li et al. 2019c, 2020a), and focusing the high-level search with admissible heuristics (Li et al. 2019a), might improve ECBS and EECBS as well, and we show how they can be adapted to them. We empirically evaluate how each improvement affects performance, finding that their combination is best. EECBS with the improvements runs significantly faster than ECBS and BCP-7 (Lam and Le Bodic 2020) and eMDD-SAT (Surynek et al. 2018), two other state-of-the-art bounded-suboptimal MAPF algorithms.

2 Preliminaries

In this section, we formalize the definition of MAPF and introduce the baseline algorithms CBS and ECBS.

2.1 Multi-Agent Path Finding (MAPF)

MAPF has many variants. In this paper, we focus on the variant defined by Stern et al. (2019) that (1) considers vertex and swapping conflicts, (2) uses the "stay at target" assumption, and (3) optimizes the sum of costs. Formally, we define MAPF by a graph and a set of m agents $\{a_1, \ldots, a_m\}$. Each agent a_i has a start vertex s_i and a target vertex g_i. Time is discretized into timesteps. At each timestep, every agent can either move to an adjacent vertex or wait at its current vertex. A *path* p_i for agent a_i is a sequence of vertices that are pairwise adjacent or identical (indicating a wait action), starting at the start vertex s_i and ending at the goal vertex g_i. The *cost* of a path p_i is its length $|p_i|$. Agents remain at their goal vertices after they complete their paths. We say that two agents have a *conflict* iff they are at the same vertex or traverse the same edge in opposite directions at the same timestep. A *solution* is a set of conflict-free paths $\{p_1, \ldots, p_m\}$, one for each agent. An *optimal solution* is a solution with the minimum *sum of costs* $\sum_{i=1}^{m} |p_i|$.

2.2 Conflict-Based Search (CBS)

Vanilla CBS CBS (Sharon et al. 2015) is a two-level search algorithm for solving MAPF optimally. On the high level, CBS performs a best-first search on a binary *constraint tree* (CT). Each CT node N contains a set of constraints $N.constraints$ that are used to coordinate agents to avoid conflicts and a set of paths $N.paths$, one for each agent, that satisfy the constraints. We use $N.paths[i]$ to denote the path of agent a_i. The *cost* of N is the sum of costs of its paths, i.e., $cost(N) = \sum_{i=1}^{m} |N.paths[i]|$. The root CT node contains an empty set of constraints and a set of shortest paths.

When expanding a CT node, CBS checks for conflicts among its paths. If there are none, then the CT node is a goal CT node, and CBS terminates. Otherwise, CBS chooses one of the conflicts and resolves it by *splitting* the CT node into two child CT nodes. In each child CT node, one agent from the conflict is prohibited from using the conflicting vertex or edge at the conflicting timestep by way of an additional constraint. The path of this agent does not satisfy the new constraint and is replanned with A* on the low level. CBS guarantees completeness by eventually exploring both ways of resolving each conflict. It guarantees optimality by performing best-first searches on both its high and low levels.

Recent Improvements of CBS Researchers have proposed many techniques for speeding up CBS while preserving its completeness and optimality.

Bypassing conflicts (Boyarski et al. 2015a) is a conflict-resolution technique that, instead of splitting a CT node, modifies the paths of the agents involved in the chosen conflict. When expanding a CT node N and generating child CT nodes, if the cost of a child CT node N' is equal to $cost(N)$ and the number of conflicts of $N'.paths$ is smaller than the number of conflicts of $N.paths$, then it replaces the paths in N with the paths in N' and discards all generated child CT nodes. Otherwise, it splits N as before. It has been shown that bypassing conflicts often produces smaller CTs and decreases the runtime of CBS.
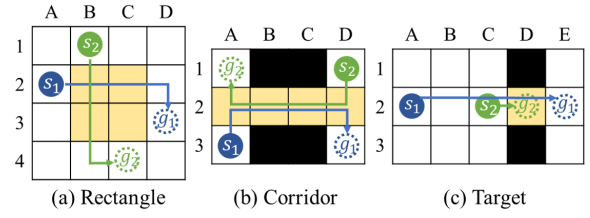

Figure 1: Examples of symmetries on 4-neighbor grids.

Prioritizing conflicts (Boyarski et al. 2015b) is a conflict-selection technique. A conflict is *cardinal* iff, when CBS uses the conflict to split a CT node N, the costs of both child CT nodes are larger than $cost(N)$. It is *semi-cardinal* iff the cost of one child CT node is larger than $cost(N)$ and the cost of the other child CT node is equal to $cost(N)$. It is *non-cardinal* iff the costs of both child CT nodes are equal to $cost(N)$. The cardinality of a conflict is determined by building a Multi-Valued Decision Diagram (MDD) (Sharon et al. 2013) for each agent (i.e., an acrylic directed graph that consists of all shortest paths of the agent). CBS can significantly improve its efficiency by resolving cardinal conflicts first, then semi-cardinal conflicts, and finally non-cardinal conflicts, because generating CT nodes with larger costs first typically improves the lower bound of the CT (i.e., the minimum cost of the CT nodes in the open list) faster and thus produce smaller CTs.

Symmetry reasoning (Li et al. 2019c, 2020a) is a technique for avoiding to resolve conflicts between the same pair of agents repeatedly due to symmetric paths and conflicts. Figure 1(a) shows an example of rectangle symmetry. There exist multiple shortest paths for each agent, each of which can be obtained by changing the order of the individual RIGHT and DOWN moves. Any shortest path for one agent is in conflict with any shortest path for the other agent. CBS has to try multiple combinations of these paths before realizing that one of the agents has to wait for one timestep. The size of the CT grows exponentially as the size of the yellow rectangular area (i.e., the region where conflicts can occur) increases. Similar behaviors of CBS can be observed in corridor and target symmetries (Figure 1(b) and (c)). Symmetry reasoning identifies each symmetry efficiently and resolves it by a single splitting action with specialized constraints, producing smaller CTs and decreasing the runtime of CBS.

Weighted Dependency Graph (WDG) heuristic (Li et al. 2019a) is an admissible heuristic for the high-level search of CBS. It proceeds by building a weighted dependency graph for each CT node N, whose vertices represent agents, whose edges represent that the two corresponding agents are dependent, i.e., the minimum sum of costs of their conflict-free paths that satisfy $N.constraints$ (which is computed by solving a 2-agent MAPF instance using CBS) is larger than the sum of costs of their paths in $N.paths$ (which are the shortest paths that satisfy $N.constraints$ but are not necessarily conflict-free), and whose edge weights represent the difference between the minimum sum of costs of the conflict-free paths that satisfy $N.constraints$ and the

sum of costs of their paths in $N.paths$. The value of the edge-weighted minimum vertex cover of the graph is then used as an admissible heuristic for the high-level search. Despite the runtime overhead of building the weighted dependency graphs and finding their edge-weighted minimum vertex cover, the addition of the WDG heuristics often produces smaller CTs and decreases the runtime of CBS.

2.3 Enhanced CBS (ECBS)

Focal search is a bounded-suboptimal search algorithm based on A_ϵ^* (Pearl and Kim 1982). It maintains two lists of nodes: OPEN and FOCAL. OPEN is the regular open list of A*, sorted according to an admissible cost function f. Let $best_f$ be the node in OPEN with the minimum f value and w be a user-specified suboptimality factor. FOCAL contains those nodes n in OPEN for which $f(n) \leq w \cdot f(best_f)$, sorted according to a function d that estimates the *distance-to-go*, i.e., the number of hops from node n to a goal node. Focal search always expands a node with the minimum d value in FOCAL. Since $f(best_f)$ is a lower bound on the optimal solution cost C^*, focal search guarantees that the returned solution cost is at most $w \cdot C^*$.

ECBS (Barer et al. 2014) is a bounded-suboptimal variant of CBS that uses focal search with the same suboptimality factor w on both the high and low levels. The low level of ECBS finds a bounded-suboptimal path for agent a_i that satisfies the constraints of CT node N and minimizes the number of conflicts with the paths of other agents. It achieves this by using a focal search with $f(n)$ being the standard $f(n) = g(n) + h(n)$ of A* and $d(n)$ being the number of conflicts with the paths of other agents. When it finds a solution, it returns not only the path but also the minimum f value $f_{\min}^i(N)$ in OPEN, indicating a lower bound on the cost of the shortest path for agent a_i. For clarity, we denote the cost of the shortest path for agent a_i as $f_{opt}^i(N)$, but this value is in general unknown during search. Thus,

$$f_{\min}^i(N) \leq f_{opt}^i(N) \leq |N.paths[i]| \leq w \cdot f_{\min}^i(N). \quad (1)$$

Unlike usual bounded-suboptimal searches, the focal search used on the low level of ECBS is to speed up the high-level search, instead of the low-level search itself, as it tries to reduce the number of conflicts that need to be resolved by the high-level search.

The high level of ECBS uses a modified focal search. OPEN is the regular open list of A*, which sorts its CT nodes N according to $lb(N) = \sum_{i=1}^{m} f_{\min}^i(N)$, indicating a lower bound on the minimum cost of the solutions below CT node N. From Equation (1), we know that $lb(N) \leq cost(N) \leq w \cdot lb(N)$. Let $best_{lb}$ be the node in OPEN with the minimum lb value. FOCAL contains those CT nodes N in OPEN for which $cost(N) \leq w \cdot lb(best_{lb})$, sorted according to the number of conflicts $h_c(N)$ of $N.paths$, roughly indicating the distance-to-go for the high-level search, i.e., the number of splitting actions required to find a solution below CT node N. Since $lb(best_{lb})$ is a lower bound on the optimal sum of costs, the cost of any CT node in FOCAL is no larger than w times the optimal sum of costs. Thus, once a solution is found, its sum of costs is also no larger than w times the optimal sum of costs.

w		1.04	1.08	1.12	1.16	1.20
ECBS	Δlb	0.63	0.63	0.64	0.52	0.56
EECBS	Δlb	5.21	2.32	1.40	0.71	0.64
	Cleanup	39.6%	14.4%	11.1%	0.7%	0.0%

Table 1: Lower-bound improvement Δlb, i.e., the value of $lb(best_{lb})$ when the algorithm terminates minus the lb value of the root CT node. Cleanup represents the percentage of expanded CT nodes that are selected from CLEANUP.

3 Explicit Estimation CBS (EECBS)

We first analyze the behavior of the high-level focal search of ECBS. We then present our new algorithm EECBS, which uses Explicit Estimation Search on the high level and online learning to estimate the solution cost.

To evaluate the effectiveness of each technique that we introduce, we test it on 200 standard MAPF benchmarks (Stern et al. 2019) with a time limit of one minute per instance. In particular, we use map `random-32-32-20`, a 32×32 4-neighbor grid with 20% randomly blocked cells, shown in Figure 2, with the number of agents varying from 45 to 150 in increments of 15. We use the "random" scenarios of the benchmarks, yielding 25 instances for each number of agents. We vary the suboptimality factor from 1.02 to 1.20 in increments of 0.02.

3.1 Limitations of ECBS

Figure 3 shows the typical behavior of ECBS on a hard MAPF instance, revealing two drawbacks of its high-level focal search. The first drawback is that, when selecting CT nodes for expansion, ECBS only considers the distance-to-go and requires the cost of the selected CT node N to be within suboptimality bound $w \cdot lb(best_{lb})$ but ignores the fact that the cost of the solution below CT node N is likely to be larger than $cost(N)$ and thus could also be larger than the suboptimality bound. In the example of Figure 3, ECBS first keeps expanding CT nodes roughly along a branch (i.e., CT nodes $1 \rightarrow 2 \rightarrow 3 \rightarrow \cdots \rightarrow 487$ in the left diagram), so the cost of the selected CT node keeps increasing and its h_c value keeps decreasing until it expands a CT node N both of whose child CT nodes are not qualified for inclusion in FOCAL (i.e., CT node 487 in the left diagram). As a result, it then expands a neighboring CT node N' whose cost is slightly smaller than $cost(N)$ and whose h_c value is slightly larger than $h_c(N)$. It keeps expanding CT nodes below CT node N', but, after several iterations, expands another CT node both of whose child CT nodes are not qualified for inclusion in FOCAL (i.e., CT node 506 in the left diagram). It repeats numerous times, indicated by the right charts, and as a result, as shown in the middle diagram, ECBS is stuck in a local area of the CT and never gets a chance to explore other parts of the CT. This thrashing behavior, in which the negative correlation of $h_c(N)$ and $cost(N)$ causes focal search to repeatedly abandon the children of expanded CT nodes, was noted by Thayer, Ruml, and Kreis (2009).

The second drawback is that the lower bound of ECBS rarely increases, as shown by the flat red line in the first chart of Figure 3. This is because, when expanding a CT
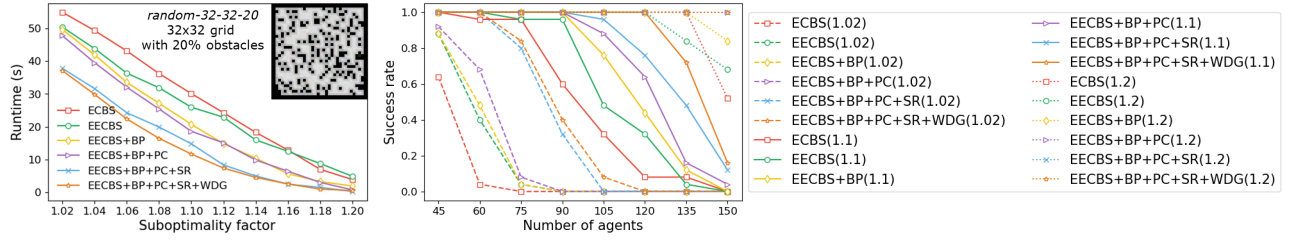
Figure 2: Performance. The left figure plots the average runtime of the algorithms over 200 instances. One minute is included in the average for each unsolved instance. The right figure plots the *success rate* (i.e., the percentage of the instances solved within the time limit) of the algorithms with suboptimality factors 1.02, 1.10, and 1.20. BP, PC, SR, and WDG are short for bypassing conflicts, prioritizing conflicts, symmetry reasoning, and using the WDG heuristic, respectively.
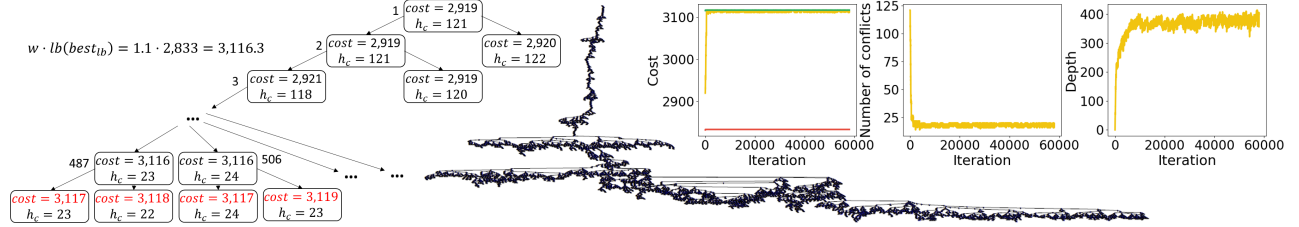

Figure 3: Performance of ECBS on a hard MAPF instance. The left and middle diagrams illustrate the CT tree of ECBS after 506 and 5,000 iterations (i.e., after expanding 506 and 5,000 CT nodes). The right three charts plot the cost, the h_c value, and the depth of the selected CT node at each iteration, respectively, with the red and green lines in the first chart being the lower bound $lb(best_{lb})$ and the suboptimality bound $w \cdot lb(best_{lb})$ at each iteration.

node N, ECBS resolves a conflict and adds new constraints to the generated child CT nodes, so the lb values of the child CT nodes tend to be equal to or larger than $lb(N)$ while the h_c values of the child CT nodes tend to be smaller than $h_c(N)$. As a result, $best_{lb}$ tends to have a large h_c value. This results in $best_{lb}$ being expanded only when FOCAL is almost empty. Since there are many CT nodes of the same cost, ECBS rarely empties FOCAL, and thus its lower bound $lb(best_{lb})$ rarely increases. More statistics can be found in Table 1 (second row). Therefore, if the optimal sum of costs is not within the initial suboptimality bound, ECBS can have difficulty finding a solution within a reasonable time. While this problematic behavior is similar to that in the first drawback, in that both involve a negative correlation of node values, it is subtly different as it involves the lower bound rather than the cost.

3.2 Explicit Estimation Search (EES)

EES (Thayer and Ruml 2011) is a bounded-suboptimal search algorithm designed in part to overcome the poor behavior of focal search. It introduces a third function $\hat{f}$ that estimates, potentially inadmissibly, the cost of the solution below a given node. EES combines estimates of the distance-to-go and the solution cost to predict the expansion of which nodes will lead most quickly to a solution within the given suboptimality factor. If the nodes of interest are not within the current suboptimality bound, it expands the node with the minimum f value to improve the current suboptimality bound. Formally, EES maintains three lists of nodes: CLEANUP, OPEN, and FOCAL. CLEANUP is the regu-

lar open list of A*, sorted according to an admissible cost function f. Let $best_f$ be the node in CLEANUP with the minimum f value and w be the user-specified suboptimality factor. OPEN is also another regular open list of A*, sorted according to a more informed but potentially inadmissible cost function $\hat{f}$. Let $best_{\hat{f}}$ be the node in OPEN with the minimum $\hat{f}$ value. FOCAL contains those nodes n in OPEN for which $\hat{f}(n) \leq w \cdot \hat{f}(best_{\hat{f}})$, sorted according to a distance-to-go function d. $\hat{f}(best_{\hat{f}})$ is an estimate of the cost of an optimal solution, so EES suspects that expanding the nodes in FOCAL can lead to solutions that are no more than w times away from optimal. Let $best_d$ be the node in FOCAL with the minimum d value. When selecting nodes for expansion, EES first considers $best_d$, as expanding nodes with nearby goal nodes should lead to a goal node fast. To guarantee bounded suboptimality, EES selects $best_d$ for expansion only if $f(best_d) \leq w \cdot f(best_f)$. If $best_d$ is not selected, EES next considers $best_{\hat{f}}$, as it suspects that $best_{\hat{f}}$ lies along a path to an optimal solution. To guarantee bounded suboptimality, EES selects $best_{\hat{f}}$ for expansion only if $f(best_{\hat{f}}) \leq w \cdot f(best_f)$. If neither $best_d$ nor $best_{\hat{f}}$ is selected, EES selects $best_f$, which can raise the suboptimality bound $w \cdot f(best_f)$, allowing EES to consider $best_d$ or $best_{\hat{f}}$ in the next iteration.

3.3 Explicit Estimation CBS (EECBS)

We form EECBS by replacing focal search with EES on the high level of ECBS. Formally, the high level of EECBS

maintains three lists of CT nodes: CLEANUP, OPEN, and FOCAL. CLEANUP is the regular open list of A*, sorted according to the lower bound function lb. OPEN is another regular open list of A*, sorted according to a potentially inadmissible cost function $\hat{f}$, which estimates the minimum cost of a solutions below a CT node. We use $\hat{f}(N) = cost(N) + \hat{h}(N)$, where $\hat{h}(N)$ is the cost-to-go. We introduce $\hat{h}$ in Section 3.4. FOCAL contains those CT nodes N in OPEN for which $\hat{f}(N) \leq w \cdot \hat{f}(best_{\hat{f}})$, sorted according to the distance-to go function h_c. EECBS selects CT nodes from these three lists using the SELECTNODE function:

1. if $cost(best_{h_c}) \leq w \cdot lb(best_{lb})$, then select $best_{h_c}$ (i.e., selected from FOCAL);

2. else if $cost(best_{\hat{f}}) \leq w \cdot lb(best_{lb})$, then select $best_{\hat{f}}$ (i.e., selected from OPEN);

3. else select $best_{lb}$ (i.e., selected from CLEANUP).

Like EES, EECBS selects a CT node N only if its cost is within the current suboptimality bound, i.e.,

$$cost(N) \leq w \cdot lb(best_{lb}), \qquad (2)$$

which guarantees bounded suboptimality. EECBS uses the same focal search as ECBS on its low level.

EECBS overcomes the first drawback from Section 3.1 by taking the potential cost increment below a CT node into consideration in Steps 1 and 2 and selecting a CT node N whose estimated cost $\hat{f}(N)$ is within estimated suboptimality bound $w \cdot \hat{f}(best_{\hat{f}})$. It overcomes the second drawback by selecting the CT node with the minimum lb value in Step 3 to raise the lower bound. Table 1 (third row) shows its empirical behavior in comparison with ECBS. Unlike ECBS, whose lower-bound improvement is always around 0.6, the lower-bound improvement of EECBS increases as the suboptimality factor w decreases. The smaller w is, the less likely a solution is within the initial suboptimality bound and thus the more frequently EECBS selects CT nodes from CLEANUP (as shown in Table 1 (fourth row)). Figure 2 shows the runtimes and success rates of ECBS and EECBS. As expected, EECBS (green lines) has a smaller runtime and larger success rate than ECBS (red lines). The improvement increases as w decreases.

3.4 Online Learning of the Cost-To-Go $\hat{h}$

Our estimate of the minimum cost of the solutions below a given CT node N uses online learning since it does not require preprocessing and allows for instance-specific learning. Thayer, Dionne, and Ruml (2011) present a method for learning the cost-to-go during search using the error experienced during node expansions. Consider a node n with an admissible cost heuristic h and a distance-to-go heuristic d. The error ϵ_d of the distance-to-go heuristic d, called *one-step distance error*, is defined as $\epsilon_d(n) = d(bc(n)) - (d(n) - 1)$, where $bc(n)$ is the *best child node* of node n, i.e., the child node with the smallest $\hat{f}$ value, breaking ties in favor of the child node with the smallest d value. Similarly, the error ϵ_h of the cost function f, called *one-step cost error*, is defined as $\epsilon_h(n) = h(bc(n)) - (h(n) - c(n, bc(n)))$, where

$c(n, bc(n))$ is the cost of moving from node n to node $bc(n)$. These errors can be calculated after every node expansion. They use a *global error model* that assumes that the distribution of one-step errors across the entire search space is uniform and can be estimated as an average of all observed one-step errors. Therefore, the search maintains a running average of the one-step errors observed so far, denoted by $\bar{\epsilon}_d$ and $\bar{\epsilon}_h$. Then, they prove that the true cost-to-go of node n can be approximated by $\hat{h}(n) = h(n) + \frac{d(n)}{1 - \bar{\epsilon}_d(n)} \cdot \bar{\epsilon}_h(n)$ (Thayer, Dionne, and Ruml 2011).

We apply this method to EECBS. Since EECBS does not have an admissible cost heuristic h, we define ϵ_d of CT node N as $\epsilon_d(N) = h_c(bc(N)) - (h_c(N) - 1)$ and ϵ_h of CT node N as $\epsilon_h(N) = cost(bc(N)) - cost(N)$. Then, $\hat{h}(N) = \frac{h_c(N)}{1 - \bar{\epsilon}_d(N)} \cdot \bar{\epsilon}_h(N)$. Thus $\hat{h}(N)$ is linear in $h_c(N)$, indicating that the larger the number of conflicts of a CT node, the higher the potential cost increments below the CT node could be.

Algorithm 1 shows the pseudo-code of the high-level search of EECBS. For now, we ignore the blue portions (i.e., Lines 2, 8-13, and 19-22) since they will be introduced in the next section. Compared to the high-level search of ECBS, EECBS changes the PUSHNODE and SELECTNODE functions and adds an UPDATEONESTEPERRORS function at the end of each iteration to update the one-step errors (Line 26).

4 Bringing CBS Improvements to EECBS

We now show how we can incorporate recent CBS improvements into EECBS. We introduce these techniques one by one and, for each one, evaluate its effectiveness by adding it to the best version of EECBS so far and showing the resulting performance in Figure 2.

4.1 Bypassing Conflicts

In CBS, the paths of every CT node N are the shortest paths that satisfy $N.constraints$. However, in EECBS, the paths can be bounded-suboptimal. Therefore, we can resolve more conflicts with the bypassing conflicts technique in EECBS if we relax the conditions of accepting bypasses. In addition, since bypassing conflicts resolves conflicts without adding any constraints, it does not change the lb value of a CT node. Therefore, it may not be helpful if the purpose of expanding a CT node N is to improve the lower bound, i.e., CT node N is selected from CLEANUP. Formally, when expanding a CT node N and generating its child CT nodes, EECBS replaces the paths of N with the paths of a child CT node N' and discards all generated child CT nodes iff (1) CT node N is not selected from CLEANUP, (2) the cost of every path of CT node N' is within the suboptimality bound of the corresponding agent in CT node N, i.e., $\forall i \; |N'.paths[i]| \leq w \cdot f^i_{\min}(N)$, (3) the cost of CT node N' is within suboptimality bound $w \cdot lb(best_{lb})$, and (4) the number of conflicts decreases, i.e., $h_c(N') < h_c(N)$. The first condition avoids wasting time in applying the bypassing conflicts technique to CT nodes that are selected from CLEANUP. The second and third conditions ensure that Equations (1) and (2) hold after replacing the paths, which in turn guarantees bounded suboptimality. The last

Algorithm 1: High-level search of EECBS.

```
1  Generate root CT node R;
2  COMPUTEWDGHEURISTIC(R);
3  PUSHNODE(R);
4  while OPEN is not empty do
5      P ← SELECTNODE();
6      if P.conflicts is empty then
7          return P.paths;        // P is a goal node
8      if P is selected from CLEANUP and the WDG
          heuristic of P has not been computed yet then
9          COMPUTEWDGHEURISTIC(P);
10         PUSHNODE(P);
11         continue;
12     CONFLICTPRIORITIZATION(P.conflicts);
13     SYMMETRYREASONING(P.conflicts);
14     conflict ← CHOOSECONFLICT(P.conflicts);
15     constraints ← RESOLVECONFLICT(conflict);
16     children ← ∅;
17     for constraint in constraints do
18         Q ← GENERATECHILD(P, constraint);
19         if P is not selected from CLEANUP and
             ∀i |Q.paths[i]| ≤ w · f_min^i(P) and
             cost(Q) ≤ w · lb(best_lb) and h_c(Q) < h_c(P)
             then                         // Bypassing
20             P.paths ← Q.paths;
21             P.conflicts ← Q.conflicts;
22             Go to Line 6;
23         Add Q to children;
24     for Q in children do
25         PUSHNODE(Q);
26     UPDATEONESTEPERRORS(P, children);
27 return "No solutions";
```

w	1.04	1.08	1.12	1.16	1.20
CBS bypassing	0.086	0.107	0.110	0.116	0.108
Relaxed bypassing	0.091	0.114	0.104	0.131	0.126

Table 2: Average number of accepted bypasses per CT node.

condition avoids deadlocks, as in the standard CBS bypassing technique. See Lines 19-22 of Algorithm 1.[1]

Empirically, we compare the effectiveness of EECBS with relaxed bypassing and EECBS with CBS bypassing. Table 2 reports the average number of accepted bypasses per expanded CT node. Relaxed bypassing accepts more bypasses than CBS bypassing, and the difference increases as the suboptimality factor w increases. The yellow and green lines in Figure 2 show the performance of EECBS with and without relaxed bypassing. Relaxed bypassing improves the performance of EECBS for all values of w and all numbers of agents. In general, the improvement increases with w.

---

[1]There are many other ways to relax the conditions. For example, one can omit the third condition and re-insert CT node N into the appropriate lists (instead of keeping expanding it) after applying the bypassing conflicts technique. We leave it for future work to evaluate different ways of relaxing the conditions of accepting bypasses for EECBS.

4.2 Prioritizing Conflicts

In CBS, the prioritizing conflicts technique (PC) tries to improve the costs of CT nodes faster since the cost of a CT node also serves as a lower bound on the minimum cost of the solutions below this CT node. In EECBS, however, the cost of a CT node is different from its lower bound. Therefore, we reformulate cardinal, semi-cardinal, and non-cardinal conflicts. A conflict is *cardinal* iff, when CBS uses the conflict to split CT node N and generates two child CT nodes N' and N'', both $\sum_{i=1}^{m} f_{opt}^i(N')$ and $\sum_{i=1}^{m} f_{opt}^i(N'')$ are larger than $\sum_{i=1}^{m} f_{opt}^i(N)$. The changes to the definitions of semi-cardinal and non-cardinal conflicts are similar. Like CBS, EECBS performs PC before it chooses conflicts (Line 12) and uses MDDs to classify conflicts. Since the construction of MDDs induces runtime overhead and not all cardinal conflicts are important to EECBS, EECBS classifies a conflict between agents a_i and a_j only when (1) the CT node N is selected from CLEANUP or (2) at least one of the path costs is equal to its lower bound, i.e., $|N.paths[i]| = f_{min}^i(N)$ or $|N.paths[j]| = f_{min}^j(N)$. PC is applied in Case (1) because resolving cardinal conflicts tends to raise the lower bound in this case. It is applied in Case (2) because resolving cardinal conflicts can increase the agents' path costs in this case (which makes the costs of the resulting child CT nodes closer to the sum of costs of the solutions below them). EECBS selects cardinal conflicts first, then semi-cardinal conflicts, then non-cardinal conflicts, and finally unclassified conflicts. By comparing the purple and yellow lines in Figure 2, we see that PC improves the performance of EECBS.

4.3 Symmetry Reasoning

We adapt symmetry reasoning techniques to EECBS with almost no changes. EECBS performs symmetry reasoning before it chooses conflicts (Line 13). Rectangle symmetry occurs only if the paths of both agents are the shortest because, otherwise, one of the agents can simply perform a wait action to avoid the rectangle symmetry. Therefore, for a given conflict between agents a_i and a_j at CT node N, we apply rectangle symmetry reasoning only when the paths of both agents are provably the shortest, i.e., $|N.paths[i]| = f_{min}^i(N)$ and $|N.paths[j]| = f_{min}^j(N)$. Corridor and target symmetries can occur even if the paths of the agents are suboptimal. Therefore, we apply corridor and target symmetry reasoning to all conflicts. By comparing the blue and purple lines in Figure 2, we see that the symmetry reasoning technique significantly improves the performance of EECBS.

4.4 WDG Heuristic

Since a path of a CT node N in EECBS is not necessarily the shortest one and, for each agent a_i, $f_{min}^i(N)$ could be smaller than $f_{opt}^i(N)$, we need to modify the WDG heuristic as follows. When EECBS computes the WDG heuristic for a CT node N, it builds a weighted dependency graph $G = (V, E)$. Each vertex $i \in V$ corresponds to an agent a_i. The edge weight on each edge $(i, j) \in E$ is equal to the minimum sum of costs of the conflict-free paths for agents a_i and a_j that satisfy $N.constraints$ minus $f_{opt}^i(N) + f_{opt}^j(N)$

w		1.04	1.08	1.12	1.16	1.20
No WDG	Δlb	36.1	23.4	12.9	7.4	3.9
	Cleanup	84.0%	74.0%	56.6%	40.4%	11.6%
CBS WDG	Δlb	54.5	45.0	37.0	30.1	25.8
	Cleanup	49.9%	46.5%	25.1%	17.8%	5.7%
	WDG time	87.9%	84.5%	80.1%	76.4%	65.3%
Adaptive WDG	Δlb	54.8	45.2	36.8	29.1	25.3
	Cleanup	53.2%	44.0%	27.2%	14.4%	8.7%
	WDG time	77.6%	62.6%	56.9%	43.2%	24.1%

Table 3: *Lower-bound improvement Δlb*, i.e., the $lb + h$ value of the best CT node in CLEANUP when EECBS terminates minus the lb value of the root CT node. WDG time is the percentage of runtime spent on computing WDG heuristics.

(instead of $|N.paths[i]| + |N.paths[j]|$). Like CBS, we compute each edge weight by solving a 2-agent MAPF instance (with the constraints in $N.constraints$) using CBS: $f_{opt}^i(N)$ and $f_{opt}^j(N)$ are equal to the costs of the paths of agents a_i and a_j of the root CT node of CBS, and the minimum sum of costs of the conflict-free paths for agents a_i and a_j is equal to the cost of the solution returned by CBS. Since computing the edge weights for all pairs of agents is time-consuming, we follow CBS by computing the weight for an edge (i, j) only when the paths $N.paths[i]$ and $N.paths[j]$ are in conflict (as the weight is more likely to be larger than 0 in this case). We delete the other edges and the vertices that have no edges. Let $h_{\text{WDG}}(N)$ be the value of the edge-weighted minimum vertex cover of G (Li et al. 2019a). By the proof in (Li et al. 2019a), we know that $\sum_{i=1}^m f_{opt}^i(N) + h_{\text{WDG}}(N)$ is a lower bound on the minimum sum of costs of the solutions below CT node N. Since

$$\sum_{i=1}^m f_{opt}^i(N) + h_{\text{WDG}}(N)$$
$$= lb(N) + \sum_{i=1}^m (f_{opt}^i(N) - f_{\min}^i(N)) + h_{\text{WDG}}(N)$$
$$\geq lb(N) + \sum_{i \in V}(f_{opt}^i(N) - f_{\min}^i(N)) + h_{\text{WDG}}(N),$$

$h(N) = \sum_{i \in V}(f_{opt}^i(N) - f_{\min}^i(N)) + h_{\text{WDG}}(N)$ is admissible, and we can thus use $lb(N) + h(N)$ to sort the CT nodes in CLEANUP and compute the lower bound.

Since computing the WDG heuristic for a CT node is time-consuming, Li et al. (2019a) suggest to calculate the heuristic lazily by computing a cheaper but less informed heuristic (e.g., using pathmax) when generating a CT node N. Only when CT node N is selected for expansion is the WDG heuristic computed and N re-inserted into the appropriate lists. Here, we follow the same scheme, as shown on Lines 8-11. But, instead of computing the WDG heuristic for all CT nodes, we only compute it for CT nodes selected from CLEANUP, as the purpose of computing the WDG heuristic is to improve the lower bound. In addition, we also compute the WDG heuristic for the root CT node (Line 2), as this can provide a higher lower bound to begin with.

Table 3 reports the lower-bound improvements of EECBS without the WDG heuristic technique (denoted as No WDG), with the WDG heuristic being computed for all CT nodes (denoted as CBS WDG), and with the WDG heuristic technique introduced above (denoted as Adaptive WDG). Compared to No WDG, Adaptive WDG selects CT nodes less frequently from CLEANUP but obtains a larger lower-bound improvement. Compared to CBS WDG, Adaptive
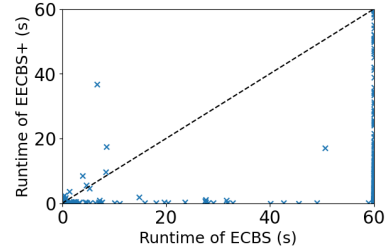

Figure 4: Runtimes of ECBS and EECBS+ on the random map with m varying from 45 to 150 and w varying from 1.02 to 1.20. The runtime of each unsolved instance is set to 60 seconds. Among the 2,000 instances, 475 instances are solved by neither algorithm; 444 instances are solved only by EECBS+; and 0 instances are solved only by ECBS.

WDG obtains similar lower-bound improvements but spends less time on computing the WDG heuristic. By comparing the orange and blue lines in Figure 2, we see that adaptive WDG improves the performance of EECBS for small and moderately large suboptimality factors.

5 Empirical Evaluation

We evaluate the algorithms on 6 maps of different sizes and structures from the MAPF benchmark suite (Stern et al. 2019) with 8 different numbers of agents per map. We use the "random" scenarios, yielding 25 instances for each map and number of agents. We evaluate 10 different values of w for each setting, and the values of w decrease as the map size increases. The algorithms are implemented in C++, and the experiments are conducted on Ubuntu 20.04 LTS on an Intel Xeon 8260 CPU with a memory limit of 16 GB and a time limit of 1 minute.

As shown in Figure 5, EECBS (green lines) outperforms ECBS (red lines) on some maps but shows a similar or even slightly worse performance on other maps. The four improvements improve the performance of both ECBS and EECBS, but EECBS benefits more from them. As a result, EECBS+ (EECBS with all improvements, purple lines) significantly outperforms the other algorithms on all six maps in terms of both runtime and success rate. When comparing the dashed lines in the success rate figures, we observe that, in many cases (e.g., for 60 agents on the random map and 180 agents on the empty map), EECBS+ is able to solve almost all instances within the runtime limit while ECBS solves only a few or even no instances. For a given success rate, EECBS+ is able to solve instances with up to twice the number of agents than ECBS (e.g., on map den520d).

We provide more details of the experiment on the random map in Figure 4. EECBS+ runs faster than ECBS in most cases and never fails to solve an instance that is solved by ECBS. In addition, when comparing the solution quality, the average solution costs of ECBS and EECBS+ over the 1,081 instances solved by both algorithms are 1,967 and 1,958, respectively, indicating that the improvements used in EECBS+ do not sacrifice the solution quality.

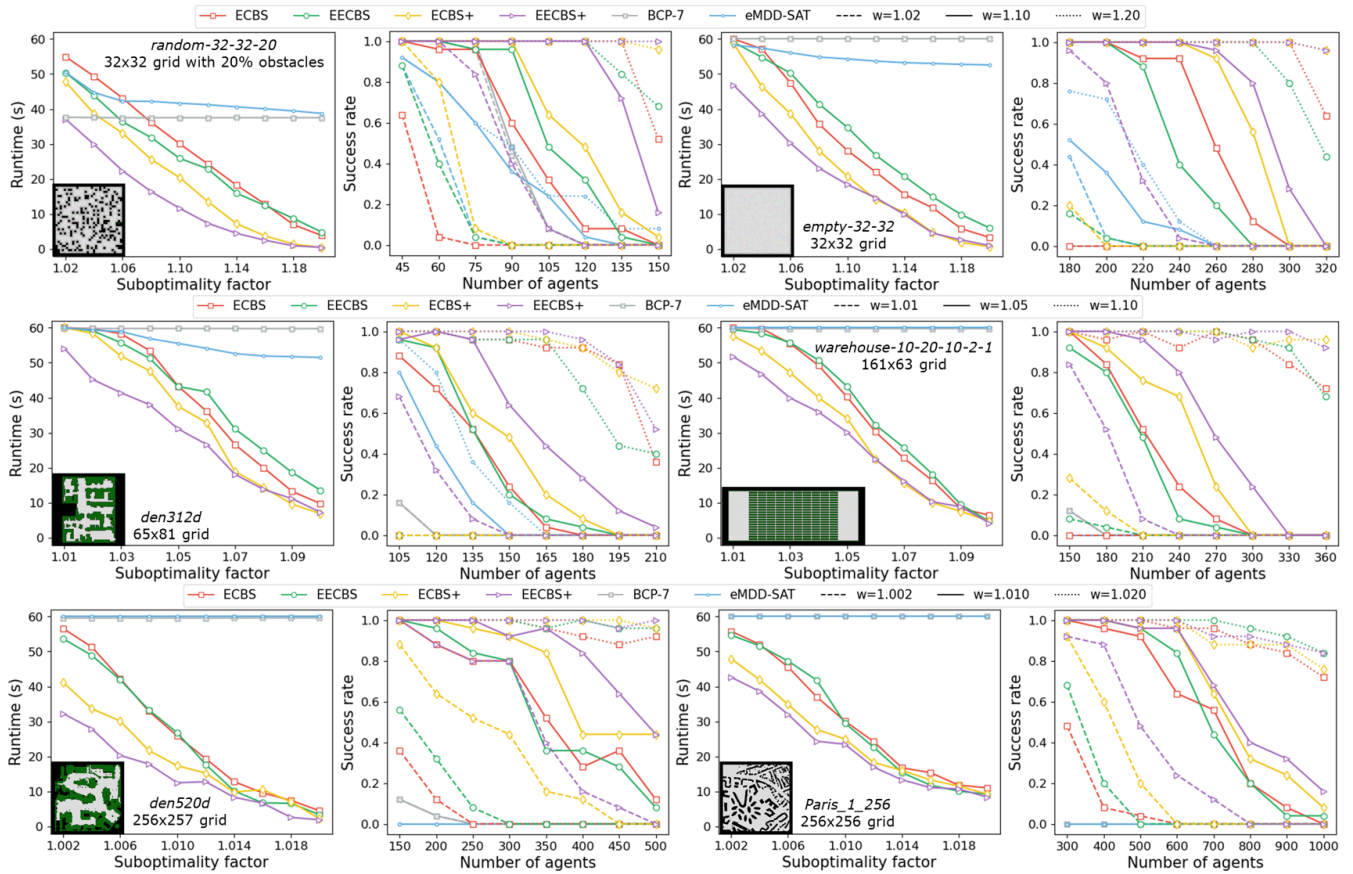We omit comparisons with the many search-based

Figure 5: Performance of ECBS, EECBS, ECBS+ (ECBS with all improvements), EECBS+ (EECBS with all improvements), BCP-7, and eMDD-SAT. All results are presented in the same format as in Figure 2. The legend for each row is shown above it. The algorithms are indicated by the legend with the same color and the same marker style, while the suboptimality factors of the lines in the success rate figures are indicated by the legend with the same line style. Since some algorithms solve (almost) zero instances, their lines overlap at the top of the runtime figures and the bottom of the success rate figures: the grey lines are hidden by the blue lines in the runtime figures of maps `warehouse-10-2-10-2-1`, `den520d`, and `Pairs_1_256`; the red and green dashed lines are hidden by the yellow dashed line in the success rate figure of map `den312d`; and many of the grey/ blue dashed/solid/dotted lines are hidden by other lines at the bottom of many of the success rate figures.

bounded-suboptimal MAPF algorithms that have already been shown to perform worse than ECBS (Aljalaud and Sturtevant 2013; Barer et al. 2014; Wagner 2015). However, there are two recent reduction-based bounded-suboptimal MAPF algorithms, namely BCP-7 (Lam and Le Bodic 2020) and eMDD-SAT (Surynek et al. 2018). BCP-7 can outperform CBS when finding optimal solutions (Lam and Le Bodic 2020), and eMDD-SAT can outperform ECBS in some domains (Surynek et al. 2018). We therefore compare them with our algorithms. We modify the search of the ILP solver used by BCP-7 from a best-first search (which is more beneficial for finding optimal solutions) to a depth-first search with restarts (which is more beneficial for finding suboptimal solutions). As shown in Figure 5, BCP-7 and eMDD-SAT outperform ECBS on the two small maps when the suboptimality factor is small. But for larger suboptimality factors or larger maps, they perform worse than ECBS. EECBS+ performs better than them on all six maps.

6 Conclusion

We proposed a new bounded-suboptimal MAPF algorithm EECBS that uses online learning to estimate the cost of the solution below each high-level node and uses EES to select high-level nodes for expansion. We further improve it by adding bypassing conflicts, prioritizing conflicts, symmetry reasoning, and using the WDG heuristic. With these improvements, EECBS significantly outperforms the state-of-the-art bounded-suboptimal MAPF algorithms ECBS, BCP-7, and eMDD-SAT. Within one minute, it is able to find solutions that are provably at most 2% worse than optimal for large MAPF instances with up to 1,000 agents, while, on the same map, state-of-the-art optimal algorithms can handle at most 200 agents (Lam and Le Bodic 2020). We hope that the scalability of EECBS enables additional applications for bounded-suboptimal MAPF algorithms.

Acknowledgement

References

Aljalaud, F.; and Sturtevant, N. R. 2013. Finding Bounded Suboptimal Multi-Agent Path Planning Solutions Using Increasing Cost Tree Search (Extended Abstract). In Helmert, M.; and Röger, G., eds., *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 203–204.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 19–27.

Boyarski, E.; Felner, A.; Harabor, D.; Stuckey, P. J.; Cohen, L.; Li, J.; and Koenig, S. 2020a. Iterative-Deepening Conflict-Based Search. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, 4084–4090.

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)*, 47–51.

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

Boyarski, E.; Harabor, D.; Stuckey, P. J.; Le Bodic, P.; and Felner, A. 2020b. F-Cardinal Conflicts in Conflict-Based Search. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 123–124.

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

Gange, G.; Harabor, D.; and Stuckey, P. J. 2019. Lazy CBS: Implict Conflict-Based Search Using Lazy Clause Generation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 155–162.

Ho, F.; Salta, A.; Geraldes, R.; Goncalves, A.; Cavazza, M.; and Prendinger, H. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 131–139.

Lam, E.; and Le Bodic, P. 2020. New Valid Inequalities in Branch-and-Cut-and-Price for Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 184–192.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1289–1296.

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)*, 442–449.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020a. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201.

Li, J.; Harabor, D.; Stuckey, P. J.; Felner, A.; Ma, H.; and Koenig, S. 2019b. Disjoint Splitting for Conflict-Based Search for Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 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)*, 6087–6095.

Li, J.; Sun, K.; Ma, H.; Felner, A.; Kumar, T. K. S.; and Koenig, S. 2020b. Moving Agents in Formation in Congested Environments. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 726–734.

Li, J.; Zhang, H.; Gong, M.; Liang, Z.; Liu, W.; Tong, Z.; Yi, L.; Morris, R.; Pasareanu, C.; and Koenig, S. 2019d. Scheduling and Airport Taxiway Path Planning under Uncertainty. In *Proceedings of the AIAA Aviation Forum*.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 837–845.

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)*, 2135–2140.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 4(4): 392–399.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 195: 470–495.

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

Surynek, P. 2019. Lazy Compilation of Variants of Multi-robot Path Planning with Satisfiability Modulo Theory (SMT) Approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3282–3287.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2018. Sub-Optimal SAT-Based Approach to Multi-Agent Path-Finding Problem. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 90–105.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning Inadmissible Heuristics During Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 250–257.

Thayer, J. T.; and Ruml, W. 2011. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 674–679.

Thayer, J. T.; Ruml, W.; and Kreis, J. 2009. Using Distance Estimates in Heuristic Search: A Re-Evaluation. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*.

Wagner, G. 2015. *Subdimensional Expansion: A Framework for Computationally Tractable Multirobot Path Planning*. Ph.D. thesis, Carnegie Mellon University.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1444–1449.

Zhang, H.; Li, J.; Surnek, P.; Koenig, S.; and Kumar, T. K. S. 2020. Multi-Agent Path Finding with Mutex Propagation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 323–332.