ECBS with Flex Distribution for Bounded-Suboptimal Multi-Agent Path Finding

Shao-Hung Chan,[1] **Jiaoyang Li,**[1] **Graeme Gange,**[2] **Peter J. Stuckey,**[2]
Daniel Harabor, [2] **Sven Koenig,**[1]
[1]University of Southern California, USA
[2]Monash University, Australia

Abstract

Multi-Agent Path Finding (MAPF) is the problem of finding collision-free paths for multiple agents on a map. Conflict-Based Search (CBS) is a leading MAPF solver that solves the MAPF instances completely and optimally. Based on CBS, Enhanced Conflict-Based Search (ECBS) is a bounded suboptimal MAPF solver that uses focal search to speed up CBS. The costs of solutions generated by ECBS are within a given factor away from the optimal. In this paper, we propose Flexible ECBS (FECBS), which is an ECBS variant that uses the range between the costs of paths and its upper bound to speed up while still providing solutions within the upper bound. The empirical evaluations show that using our approach can lead to 7.2% higher success rates than the state-of-the-art ECBS variant in the small map, and 10.4% higher in the large map.

1 Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding collision-free paths as a *solution* for multiple agents on the map. The travel time of an agent is known as the *path cost* of the agent, and the sum of the travel time is known as the *sum of cost (SoC)*. If a solution whose SoC is minimum, then it is known as an *optimal* solution. On the other hand, if a solution whose SoC is away from the optimal value within a user-specified bound, then it is known as a *bounded suboptimal* solution. Real-world applications of MAPF include autonomous warehouse (Li et al. 2020), UAV traffic management (Ho et al. 2019), video game characters (Ma et al. 2017), and office robots (Veloso et al. 2015).

Conflict-Based Search (CBS) (Sharon et al. 2015) is a leading two-level MAPF solver for solving MAPF optimally. On the low level, it plans paths for single agents. On the high level, it resolves collisions using the best-first search in the space of collision resolutions, with each branch being a new candidate plan that forces one agent or the other find new path to avoid the collision. However, since solving MAPF optimally is a NP-hard problem (Yu and LaValle 2013), CBS suffers from the scalability problem, which motivates finding bounded-suboptimal solvers to speed up the search. Enhanced CBS (ECBS) (Barer et al. 2014) is a variant of CBS that aims to find the bounded-suboptimal MAPF solution. That is, given user-specified suboptimal factor, ECBS finds the solution with cost at most the factor away from optimal. ECBS replaces the best-first search with *focal search* (Pearl and Kim 1982) on both high and low levels of CBS. Once the factor is sufficiently large, leading to sufficiently large upper bound, ECBS can quickly find collision-free paths where agents are allowed to take some delays or detours, since it does not have to prove that solutions of lower costs do not exist.

While solving MAPF, ECBS may generate candidate plans whose SoC is less than the acceptable upper bound. Thus, in this paper, we proposed Flexible ECBS (FECBS), an ECBS variant that uses the range between the upper bound and SoC to improve the efficiency.

2 Problem Definition

A MAPF problem, defined in (Stern et al. 2019), consists of an undirected graph $G = (V, E)$ and a set of k agents $\{a_1, a_2, \ldots, a_k\}$. Each agent a_i has a unique start vertex $s_i \in V$ and a unique goal vertex $g_i \in V$. Time is discretized into timesteps. At every timestep, an agent performs one of the two actions: the *move* action is to move from the current vertex to an adjacent vertex, and the *wait* action is to stay at the current vertex. An agent continues to exist after it has reached its goal vertex.

A path p_i is a sequence of vertices that starts at start vertex s_i and ends at goal vertex g_i. Each vertex in p_i corresponds to the vertex where agent a_i stays at each timestep. Every two adjacent vertices in path p_i are either adjacent vertices in G, meaning that agent a_i *moves*, or identical, meaning that agent a_i *waits*. The path cost of path p_i is the number of timestep, or the traveling time, for agent a_i moving from start vertex s_i to goal vertex g_i, which is also the path length. The timesteps that agent a_i terminally waits at goal vertex g_i are excluded while calculating the cost of path p_i. A *collision* (or ,equivalently, *conflict*) between two agents a_i and a_j belongs to one of the categories. It can be a *vertex conflict* $\langle a_i, a_j, v, t\rangle$, where agents a_i and a_j occupy the same vertex $v \in V$ at the same timestep t. It can also be an *edge conflict* $\langle a_i, a_j, u, v, t\rangle$, where agents a_i and a_j traverse the same edge $(u, v) \in E$ in opposite directions from timestep t to timestep $t + 1$. A solution is a set of conflict-free paths, one for each agent. An optimal solution is a solution with minimum SoC of the paths. In this paper, the graphs are 4-neighbor grids with vertices corresponding to the unblocked

cells and edges corresponding to the connections between adjacent unblocked cells in the four main compass directions.

3 Existing MAPF Solvers

In this section, we introduce Conflict-Based Search (CBS) and Enhanced CBS (ECBS). Both of them rely on a two-level architecture where the low level plans paths for single agents and the high level resolves conflicts between two agents.

3.1 Conflict-Based Search

Before introducing CBS (Sharon et al. 2015), we define *constraints*, that are used in CBS to resolve conflicts between two agents. A vertex conflict $\langle a_i, a_j, v, t\rangle$ can be resolved by prohibiting either agent a_i or agent a_j from staying at vertex v at timestep t, leading to *vertex constraints* $\langle a_i, v, t\rangle$ and $\langle a_j, v, t\rangle$. An edge conflict $\langle a_i, a_j, u, v, t\rangle$, on the other hand, can be resolved by prohibiting either agent a_i or agent a_j from traversing edge (u, v) in the opposite direction from timestep t to timestep $t + 1$, leading to *edge constraints* $\langle a_i, u, v, t\rangle$ and $\langle a_j, v, u, t\rangle$.

On the low level, CBS plans the path of agent a_i by viewing vertex $v \in V$ and timestep t as a spatio-temporal node $n = (v, t)$ and runs A* to find a minimum cost path that satisfies the constraints given by the high level. That is, A* on the low level uses an open list OPEN and sorts the spatio-temporal nodes in OPEN in increasing order of their f values $f^i(n) = g^i(n) + h^i(n)$, where $g^i(n)$ is the number of timesteps for agent a_i to move from its start vertex s_i to vertex v and $h^i(n)$ is an admissible heuristic that estimates the distance from vertex v to its goal vertex g_i. The A* on the low level breaks ties in favor of a path that has the fewest conflicts with the paths of other agents.

On the high level, CBS runs a best-first search on a binary *constraint tree* (CT). Each CT node N has two components: (1) a set of paths of all agents generated by the low-level search $N.paths$, with the path of agent a_i being $N.paths[i]$ and its cost being $|N.paths[i]|$, and (2) a set of vertex and edge constraints, $N.constraints$, that coordinate agents to avoid conflicts. The *cost* of CT node N, $N.cost$, is the SoC of $N.paths$, that is,

$$N.cost = \sum_{i=1}^{k} |N.paths[i]|. \tag{1}$$

CBS begins its high-level search at the *root* CT node, that contains a minimum cost path for each agent and an empty set of constraints. While expanding CT node N, if there are no conflicts among the paths of N, CBS returns the paths of N and terminates. Otherwise, it picks one of the conflicts and resolves it by *branching*, also known as splitting N into two child CT nodes. In each child CT node, CBS adds a vertex or edge constraint to one of the conflicting agents to prohibit it from utilizing the conflicted vertex or edge, respectively, at the conflicted timestep. It then performs A* on the low level to replan the path of the agent with the additional constraint and leaves all other paths unchanged. In other words, CBS only replans one agent in a child CT node. CBS solves MAPF optimally by performing best-first searches on both high and low levels.

3.2 Enhanced CBS

Enhanced CBS (ECBS) (Barer et al. 2014) uses focal search on both the high and low levels to speed up CBS significantly, which is a bounded suboptimal search instead of the best-first search. On the low level, ECBS uses an open list OPEN and sorts its spatio-temporal nodes n in increasing order of their f values $f_1^i(n)$, which is identical to function $f^i(n)$ of CBS, to find a path for agent a_i. Let $best^i$ be the node n with the minimum $f_1^i(n)$ value in OPEN and w be the user-specified suboptimality factor. At the beginning of the low-level focal search, only the *root spatio-temporal node* $n_i^0 = (s_i, 0)$ is in OPEN, thus $n_i^0 = best^i$ and the *initial minimum* $f_1^i(n)$ value is set to $f_1^i(n_i^0)$. The low-level focal search also uses a focal list FOCAL that contains all spatio-temporal nodes $n = (v, t)$ in OPEN with $f_1^i(n) \leq w \cdot f_1^i(best^i)$. The spatio-temporal nodes in FOCAL are sorted in increasing order of their different f values $f_2^i(n)$, which specifies the number of conflicts of the path of agent a_i with the paths of other agents in $N.paths$ while agent a_i moves from its start vertex s_i to vertex v. The low-level focal search expands a node n with the minimum $f_2^i(n)$ value in FOCAL. We define the *low-level lower bound* on the cost of the optimal path of agent a_i as $N.lb[i]$. Since $f_1^i(n)$ uses an admissible heuristic, $N.lb[i] = f_1^i(best^i)$ for ECBS. Thus, the low-level focal search always returns a path for agent a_i with a cost of at most w times the optimal path cost c_i^*, meaning that,

$$N.lb[i] \leq |N.paths[i]| \leq w \cdot N.lb[i] \leq w \cdot c_i^*. \tag{2}$$

Here, we let $N.ub[i] = w \cdot f_1^i(best^i)$ as the *low-level upper bound* of agent a_i in CT node N. The low-level focal search also returns a lower bound on the cost of the optimal path for agent a_i, which is the $f_1^i(best^i)$ value when the low-level search terminates.

On the high level, ECBS also runs focal search on CT. Given a CT node N, we define its lower-bound value as $LB(N) = \sum_{i=1}^{k} f_1^i(best^i)$, and we define its upper-bound value as $UB(N) = w \cdot \sum_{i=1}^{k} f_1^i(best^i)$. We define the number of conflicts between all the pairs of paths in CT node N to be $f_2(N)$. Let $LB = \min(LB(N) \mid N \in \text{OPEN})$. Since $LB(N)$ is a lower bound on the minimum SoC of the solutions below CT node N, LB is a lower bound on the optimal cost, denoted as C^*. The high-level focal search sorts the CT nodes N in OPEN according to $LB(N)$ and adds the CT nodes N with the costs of at most $w \cdot LB$ into FOCAL, where w is the same suboptimality factor as the low level. ECBS expands the CT node with the minimum f_2 value in FOCAL, namely the CT node with the fewest number of conflicts. Given a MAPF instance with optimal cost C^*, since ECBS only selects CT nodes whose costs are at most $w \cdot LB$ and $LB \leq C^*$, it finds a solution whose cost is at most $w \cdot C^*$, that is,

$$LB \leq N.cost \leq w \cdot LB \leq w \cdot C^*. \tag{3}$$

Here, we define $UB = w \cdot LB$ as the upper bound of high-level focal search. We define UB as the high-level upper

bound. As the value of w increases, ECBS has more sub-optimal solutions to choose from, which might decrease its runtime.

4 Flexible ECBS (FECBS)

In ECBS, each CT node N has path $N.paths[i]$ of agent a_i and the low-level lower bound of the path $N.lb[i]$. Given a user-specified factor w, the low-level upper bound of path $N.paths[i]$ is $w \cdot N.lb[i]$. We define the *flex* of agent a_i, or $N.flex[i]$, as the difference between the low-level upper bound of the path and the path cost, that is,

$$N.flex[i] = w \cdot N.lb[i] - |N.paths[i]|. \quad (4)$$

One of the limitations of ECBS is that each agent a_i, while being replanned on the low level, only considers its upper bound given by its own lower bound $f_1^i(best^i)$. Thus, we propose Flexible ECBS (FECBS), an ECBS variant that increases the low-level upper bound of agent a_i by adding the flex values from other agents, known as the *flex distribution* scheme. The low-level upper bound of agent a_i of FECBS is the summation of $w \cdot N.lb[i]$ and the sum of the flex values from other agents, that is,

$$N.ub[i] = w \cdot N.lb[i] + \sum_{a_j \in A \setminus a_i} N.flex[j], \quad (5)$$

where $\sum_{a_j \in A \setminus a_i} N.flex[j]$ is the sum of the flex values from other agents aside from agent a_i, indicating we distribute all the flex values from other agents to the agent that is going to be replanned. Thus, the low-level focal search of FECBS adds spatio-temporal node n whose $f_1^i(n)$ is within low-level upper bound $N.ub[i]$ to FOCAL. Also, FECBS should keep track of the maximum low-level lower bound of agents along the branch of CT during the search. Supposed that the parent CT node of CT node N is $\hat{N}$ and the low-level lower bound of agent a_i in CT node $\hat{N}$ is $\hat{N}.lb[i]$. While replanning agent a_i in CT node N, if the sum of the flex values from other agents is large enough, there may be a case where $f_1^i(best^i)$ is less than the low-level lower bound of agent a_i in CT node $\hat{N}$, while the low-level upper bound of agent a_i in CT node N is greater than the low-level upper bound of agent a_i in its parent CT node $\hat{N}$, that is,

$$f_1^i(best^i) < \hat{N}.lb[i] \quad (6)$$

$$w \cdot f_1^i(best^i) - |N.paths[i]| > w \cdot \hat{N}.lb[i] - |\hat{N}.paths[i]| \quad (7)$$

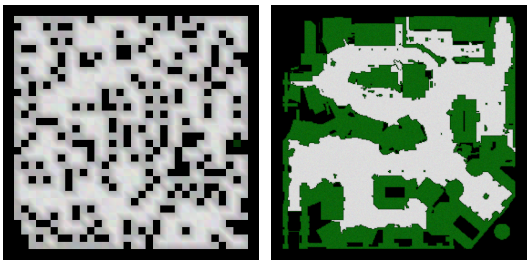
In this case, the low-level focal search of FECBS may find a path of agent a_i that satisfies the constraints while $f_1^i(best^i)$ is less than the lower bound of agent a_i in the parent CT node $\hat{N}$. Thus, if we set $f_1^i(best^i)$ to $N.lb[i]$ directly as what ECBS does, the lower-bound value of CT node N is less than the lower-bound value of its parent CT node $\hat{N}$, meaning that the sum of the low-level lower bounds of agents decreases. This results in decreasing of $w \cdot N.lb[i]$, and flex that have been used to replan any agents before branching to CT node N may "disappear", which causes FECBS to be invalid. Thus, given a branch of CT B and N being its tail, we define the maximum low-level lower bound of agents along branch B as $B.lb[i]$, and the initial minimum $f_1^i(n)$ value of agent a_i in CT node N should be

$$\max(f_1^i(n_i^0), B.lb[i]). \quad (8)$$

While replanning agent a_i on the low-level, the sum of flex values from other agents may be negative, meaning that there are other agents that have been replanned previously by using flex value of agent a_i. Still, the low-level upper bound of agent a_i should be at least the low-level lower bound of agent a_i, meaning that,

$$N.lb[i] \leq N.ub[i]. \quad (9)$$

If $N.lb[i] = N.ub[i]$, then this means that other agents have use up all the flex value of agent a_i. Thus, FECBS will run the low-level focal search exactly the same as the low-level focal search of ECBS with the initial minimum value of agent a_i in CT node N being as Equation (8). FECBS potentially distributes the flex values to agents whose constraints are difficult to satisfy. Also, for agent whose flex has been used up for other agents, FECBS tends to increases its low-level lower bound to find a path that satisfies the constraints, resulting in the increment of both high-level and low-level upper bounds and even generates more flex for other agents. Since the flex values are obtained from the range between the high-level upper bound and SoC, FECBS still guarantees to be bounded suboptimal with user-specified suboptimal factor w.



(a) `Grid` 32×32 (b) `Den520d` 257×256

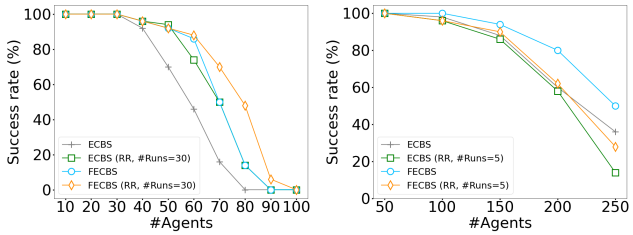
Figure 1: Maps used in the experiments with their size given in the form $height \times width$.

5 Empirical Evaluation

As shown in Figure 1, the scenarios for these experiments are all 4-neighbor grids from the MAPF benchmark suite (Stern et al. 2019), including a 32×32 grid map with 20% blocked cells (`Grid`), and a 257×256 grid map from the video game Dragon Age: Origin (DAO) (`Den520d`). We use both the "even" and "random" scenarios from the benchmark, with each containing 25 instances for each map and each number of agents. The start and goal vertices are distributed evenly in the instances of the "even" scenario. The start and goal vertices are distributed randomly in the instances of the "random" scenario. Our main comparison metric is the *success rate*, which is the percentage of the instances solved within a runtime limit of 5 minutes. For `Grid` map, which has small size but high obstacle density, we set

Maps	#Runs	ECBS	FECBS	ECBS (RR)	FECBS (RR)
Grid	5	52.4	63.8	59.2	67.2
	20			59.8	68.2
	30			**62.8**	**70**
	40			62.6	69.8
Den520d	5	76.4	86.8	60.4	58.4
	20			42	48.4
	30			54	57.2
	40			**62**	**76.8**

Table 1: The success rate (in percentage) of ECBS variants with #Runs $\in \{5, 20, 30, 40\}$.



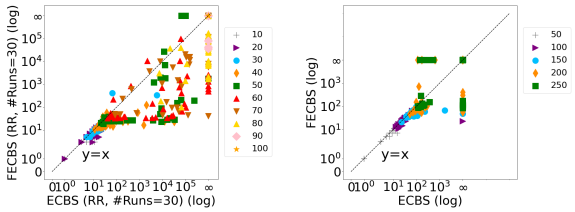
(a) Grid (b) Den520d

Figure 2: Success rate (%) of ECBS, ECBS (RR) with its best #Runs, FECBS, and FECBS (RR) with its best #Runs on different maps.

the suboptimality bound to $w = 1.05$. For Den520d map, $w = 1.05$ is too large, resulting in all ECBS variants having high success rates and preventing us from distinguishing among them. Thus, we set $w = 1.01$. We implement our ECBS variants in C++ and run them on servers with 2.80 GHz Intel Xeon Processors E5-2640 v4 and 2 GB RAM.

We compare our FECBS with Rapid Randomized Restart ECBS (Cohen et al. 2018), known as ECBS (RR), which is the state-of-the-art ECBS variant. ECBS (RR) uses a user-specified *number of runs* (or #Runs) to determine the times for restarting the search. Once the time limit T and #Runs are set, ECBS (RR) will restart every T/#Runs, each time randomly shuffling the order of agents before the restart. We evaluate ECBS (RR) on the maps with #Runs $\in \{5, 20, 30, 40\}$. We define the *best #Runs* as the value of #Runs in the set that leads to the highest number of solved instances. Since rapid randomized restart scheme only focuses on determining when to restart the search, it can also be used in FECBS, resulting in FECBS (RR). Table 1 shows the number of solved instances for different values of #Runs.

Figure 2 shows success rate versus the number of agents (or #Agents) of ECBS, ECBS (RR), FECBS, and FECBS (RR). We only show ECBS (RR) and FECBS (RR) with their individual best #Runs. In Grid map, as the number of agent increases, FECBS has higher success rate than ECBS and is compatible with ECBS (RR) of the best #Runs. Also, the combination of the flex distribution and the rapid random restart schemes even improves the success rates. In Den520d map, if #Runs is too high, then ECBS (RR) shows its weakness running in the large map and large number of agents. Still, our FECBS shows advantage among all the ECBS variants, which is also shown in Table 1.



(a) Grid (b) Den520d

Figure 3: Success rate (%) of ECBS, ECBS (RR) with its best #Runs, FECBS, and FECBS (RR) with its best #Runs on different maps.

	ECBS	FECBS	ECBS (RR)	FECBS (RR)
Grid	1.027	1.028	1.031	1.029
Den520d	1.006	1.0062	1.0057	1.0058

Table 2: The average suboptimalities of MAPF solver in different maps.

We also analyze the generated CT nodes of ECBS variants to show the efficiency while solving MAPF instances. If a MAPF solver is able to solve a MAPF instance with fewer CT nodes than another MAPF solver, then it has higher efficiency. In Grid map, we compare FECBS (RR) and ECBS (RR) with their individual best #Runs. In Den520d map, we compare FECBS and ECBS, since the rapid random restart scheme does not show improvements. Figure 3 shows the comparison of generated CT node between two ECBS variants. We use a logarithmic scale for both axes. For instances that not solved within the runtime limit, we set the number of CT nodes generated by the MAPF solver to ∞. Instances on the right hand side of the dashed line are the ones that can be solved by FECBS (RR) (or FECBS) with fewer generated CT nodes than by ECBS (RR) (or ECBS). Figure 3 shows that ECBS variants with the flex distribution scheme can reduce the number of generated CT nodes and thus improve the efficiency.

To analyze how close between the SoC of the solution and the high-level upper bound, we use the *suboptimality* α, which is the ratio between the SoC and the high-level lower bound of a MAPF instance. Thus, if the suboptimality is close to the user-specified suboptimality factor w, then the SoC of the solution is close its high-level upper bound. We also define *average suboptimality* $\bar{\alpha}$ as the average of the suboptimalities over the MAPF instances been solved. Table 2 shows the average suboptimality of the ECBS variants over the instances in different maps. Although flex distribution seems to push the SoC to the high-level upper bound, the average suboptimalities of FECBS and FECBS (RR) are still close to ECBS and ECBS (RR). Thus, the flex distribution scheme does not affect the solution qualities much.

6 Conclusions

In this paper, we introduce a new scheme for ECBS to use the range between its high-level upper bound and SoC, called the flex distribution scheme, and propose FECBS and FECBS (RR). Both ECBS variants gather flex values from other agents to increase the low-level upper bound of the agent that is currently being replanned. Potentially, constraints of the agents may be satisfied with the increment of the low-level lower bounds. The empirical evaluations show that using our approach can lead to 7.2% higher success rates than the state-of-the-art ECBS variant (ECBS (RR)) in `Grid` map, and 10.4% higher in `Den520d` map. The future work for this paper may be designing heuristics for flex distribution, instead of assigning all the flex values to a single agent, to improve the efficiency.

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. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 19–27.

Cohen, L.; Koenig, S.; Kumar, T. K. S.; Wagner, G.; Choset, H.; Chan, D.; and Sturtevant, N. 2018. Rapid Randomized Restarts for Multi-Agent Path Finding: Preliminary Results. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 1909–1911.

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.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2020. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 1898–1900.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 270–272.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions 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.

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.

Veloso, M. M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 4423–4429.

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*, 1443–1449.