Rapid Randomized Restarts for Multi-Agent Path Finding Solvers

Liron Cohen[1], **Glenn Wagner**[2], **David Chan**[3],
Howie Choset[2], **Nathan Sturtevant**[3], **Sven Koenig**[1] **and T. K. Satish Kumar**[1]
[1]University of Southern California
[2]Carnegie Mellon University
[3]University of Denver

Abstract

Multi-Agent Path Finding (MAPF) is an NP-hard problem that has been well studied in artificial intelligence and robotics. Recently, randomized MAPF solvers have been shown to exhibit heavy-tailed distributions of runtimes, which can be exploited to boost their success rates for given runtime limits. In this paper, we discuss different ways of randomizing MAPF solvers and evaluate simple rapid randomized restart strategies for state-of-the-art MAPF solvers such as iECBS, M* and CBS-CL.

Introduction and Background

Given a graph and a set of agents with unique start and goal vertices each, the Multi-Agent Path Finding (MAPF) problem is to find collision-free paths for all agents from their respective start vertices to their respective goal vertices. The agents traverse edges in discrete time steps with the possibility of waiting at vertices. Minimizing the solution cost given by the sum of the travel times of the agents along their paths is NP-hard (Yu and LaValle 2013b).

Different techniques have been used to develop a variety of MAPF solvers. These include reductions to problems from satisfiability (Surynek 2012), integer linear programming (Yu and LaValle 2013a) and answer set programming (Erdem et al. 2013).[1] In this paper, we focus on state-of-the-art MAPF solvers that are based on graph search. Search-based MAPF solvers exploit opportunities for decoupling agents when no coordination between them is required (for example, when they operate in different regions). We focus on two families of search-based MAPF solvers, namely, M* (Wagner 2015) and CBS (Sharon et al. 2015).

M* uses subdimensional expansion to initially create a one-dimensional search space embedded in the joint configuration space of the multi-agent system. When the search reaches a dead-end due to an agent-agent collision, the dimensionality of the search space is locally increased to ensure that an alternative path can be found. Like M*, CBS also tries to avoid operating in the joint configuration space. However, it does so using a two-level search. On the high level, it performs a search on a conflict tree that represents conflicts[2] between agents. Each high-level node represents a set of constraints imposed on the paths of agents. On the low level, it performs single-agent searches that generate paths for the agents that respect the constraints imposed by the relevant high-level nodes. The basic versions of both M* and CBS generate optimal solutions.

Since the MAPF problem is NP-hard, even M* and CBS fail to solve MAPF instances in which many agents interfere with each other in a small region. However, suboptimal versions of M* and CBS are capable of solving harder MAPF instances by trading off runtime with solution cost. w-suboptimal MAPF solvers generate solutions whose costs are at most a factor of w larger than the minimal solution costs, where w is a user-specified suboptimality bound. Suboptimal MAPF solvers, on the other hand, provide no such guarantees.

CBS with highways (CBS+HWY(w)) (Cohen, Uras, and Koenig 2015) is a w-suboptimal variant of CBS. Its suboptimality stems from inflating heuristic values non-uniformly using highways. Highways are a set of "useful" directed edges in the graph that are either human-generated or automatically generated (Cohen et al. 2016), first used in the context of Experience Graphs (Phillips et al. 2012). Given highways, CBS+HWY(w) inflates the costs of move actions along edges that do not belong to the highways by a factor of w. This biases the low-level searches to find paths that use the highways edges, which reduces the number of head-on conflicts between agents. M* can also use highways with a similar inflation mechanism and suboptimality guarantee. Enhanced CBS (ECBS(w)) (Barer et al. 2014) is also a w-suboptimal variant of CBS. Its suboptimality stems from its use of focal search (Pearl and Kim 1982) with parameter w. A focal search, like A*, uses an OPEN list whose nodes n are sorted in increasing order of their f-values $f(n) = g(n) + h(n)$. Unlike A*, a focal search also uses a FOCAL list of all nodes currently in the OPEN list whose f-values are no larger than w times the current smallest f-value in the OPEN list. The nodes in the FOCAL list are sorted in increasing order according to (possibly inadmissible) secondary heuristic values. A* expands a node

[1]The reader can find a list of references in Felner et al. 2017.

[2]We say that a *conflict* between two agents occurs if, at any discrete time step, both agents are at the same vertex or traverse the same edge in opposite directions.

in the OPEN list with the smallest f-value, but a focal search instead expands a node in the FOCAL list with the smallest secondary heuristic value. The high-level and low-level searches of ECBS(w) are focal searches, which use measures related to the number of conflicts as secondary heuristic values. Improved ECBS(w) (iECBS(w)) (Cohen et al. 2016) is a variant of ECBS(w) that uses the highways heuristic values to break ties among nodes with the same number of conflicts in its secondary heuristic. Finally, CBS with Constraint Layering (CBS-CL) (Walker, Chan, and Sturtevant 2017) is a suboptimal variant of CBS that plans in a hierarchy of graphs with different granularities of abstraction. Conflicts between agents are resolved by both introducing constraints (as in CBS) as well as changing the abstract graph in which an agent searches. This changes the length of the shortest path for each agent in the conflict and reduces the likelihood that the conflict will occur upon re-planning.

Suboptimal or w-suboptimal MAPF solvers based on M* and CBS are capable of solving harder MAPF instances than optimal MAPF solvers. However, their runtimes still increase exponentially with increased coupling among the agents. In part, the MAPF solvers' inability to cope with such MAPF instances comes from the deterministic nature of their searches. This weakness of deterministic search has also been observed for other combinatorial tasks in search (Valenzano et al. 2010), satisfiability and constraint satisfaction (Gomes et al. 2000). Given an instance, deterministic search builds the same search tree in every run. This means that "bad" decisions, especially higher up in the search tree, are not only expensive but also cannot be avoided in subsequent runs.

Recently, deterministic search-based MAPF solvers have been enhanced with randomization, and their runtimes have been observed to exhibit heavy-tailed distributions (Cohen et al. 2018). Heavy-tailed distributions in runtimes can be exploited with rapid randomized restart (RRR) strategies (Gomes et al. 2000) because multiple short runs have a better chance of solving an instance than one long run. In this paper, we provide a more detailed experimental evaluation of the efficiency of parallel runs and RRR strategies for randomized search-based MAPF solvers, such as iECBS(w) with and without highways, M* with and without highways, and CBS-CL.

Randomized MAPF Solvers

In this section, we describe ways in which we can incorporate randomness into the M* and CBS frameworks.

In the M* framework, randomization can be incorporated into the neighbor-generation process. Both Operator Decomposition M* (ODM*) and Enhanced Partial Expansion M* (EPEM*) (Wagner 2015) construct the possible neighbors of a vertex in an agent-by-agent fashion. M* takes each partial neighbor specifying the action of the first $N-1$ agents and creates a separate copy of it. M* appends to this copy each of the actions of agent N in the collision set of the vertex being expanded. Otherwise, it generates a single copy of the partial neighbor by appending the action dictated by the individual policy. The neighbors are then inserted into the OPEN list by a deterministic process that orders them by their g-values. As a result, the relative orders of neighbors with equal g-values depends on the order that they appear in the neighbor list, and thus on the labeling of the agents. Randomizing the labeling of the agents effectively changes how tie-breaking is done.

In the CBS framework, applicable to ECBS(w), iECBS(w) and CBS-CL, randomization can be incorporated in multiple ways. First, each run can use a random permutation of the labels of the agents. Given such a permutation, the solver performs a low-level focal search for each agent in that order to determine the paths of all agents in the high-level root node. Because the low-level focal search tries to avoid conflicts with the paths found for agents earlier in the order, the order has a significant effect not only on the paths found in the high-level root node but also on which conflicts will be resolved as the search proceeds.

Second, randomization can be used for choosing which high-level node to *expand* next. The deterministic version of iECBS(w) uses a FOCAL list for the high-level search that contains all high-level nodes with f-values of up to w times the current smallest f-value in the OPEN list. It then picks a high-level node for expansion from this list based on a deterministic secondary heuristic, namely, the number of conflicts among the paths represented by this high-level node. This choice can be randomized. Since the secondary heuristic provides useful information, the probability of expanding a high-level node with $\hat{K}$ conflicts can thus be proportional to $1/(\hat{K}+1)$.

Third, randomization can be used for choosing which high-level nodes to *generate* next. Generating a pair of high-level nodes in iECBS(w) is the process of identifying a conflict and adding two successor high-level nodes that resolve it. The deterministic version of iECBS(w) chooses the earliest conflict. This choice can be randomized. We have observed that choosing the earliest conflict works well in practice. The probability of choosing a conflict that occurs at time step $\hat{T}$ can thus be proportional to $1/\hat{T}$.

Fourth, randomization can be used for choosing which low-level node to expand next. The deterministic version of iECBS(w) uses a FOCAL list for the low-level search that maintains all low-level nodes with f-values of up to w times the current smallest f-value in the OPEN list. It then picks a low-level node for expansion from this list based on a deterministic secondary heuristic, namely, the number of conflicts of the partial path (from the start vertex to the vertex represented in this node) with the paths of all other agents. This choice can be randomized. Since the secondary heuristic provides useful information, the probability of expanding a low-level node with $\hat{K}$ conflicts can thus be proportional to $1/(\hat{K}+1)$.

Finally, CBS-CL can also incorporate randomization into its abstraction hierarchy. For example, each run can use a random permutation of the order of abstractions in the hierarchy. Furthermore, randomization can be used when building an abstraction by randomizing the set of edges that are removed from the initial graph.

Incorporating randomness into MAPF solvers has the additional benefit of parallelizability. Different instantiations

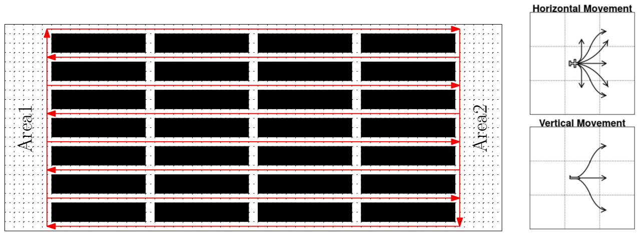


Figure 1: (a) shows the Kiva-like domain and the highways used to guide agents in our experiments. (b) shows the horizontal (top) and vertical (bottom) aircraft movements in the aviation domain.

of a randomized MAPF solver can run in parallel on the same MAPF instance, with each run having the same runtime limit. The first instantiation that solves the MAPF instance terminates all other instantiations. Moreover, randomized MAPF solvers have been shown to exhibit heavy-tailed distributions of runtimes (Cohen et al. 2018). RRR strategies can exploit the runtime variance of randomized MAPF solvers because shorter runs are almost as likely to solve a MAPF instance as longer runs. Therefore, multiple short runs are more likely to solve a MAPF instance than one long run of equivalent accumulated runtime. In the following section, we evaluate the efficiency of parallel runs and RRR strategies for one way of randomizing the MAPF solvers that is applicable to all of them, namely, by randomizing the labeling of the agents.

Experimental Results and Analysis

We conduct experiments in two domains inspired by real-world applications and show that parallel runs and our RRR strategy are beneficial in both the M* and CBS frameworks.

The first domain, illustrated in Figure 1(a), is a Kiva[3]-like domain. It is generally considered to be a hard domain for search-based MAPF solvers because significant coordination among agents is required as many of them need to travel through narrow passageways to reach the opposite sides of the map (Cohen, Uras, and Koenig 2015). For each number of agents in increments of 10, we use 100 randomly generated MAPF instances. Half of the agents are assigned a random start vertex in the left open space and a random goal vertex in the right open space, and vice-versa for the other half of the agents.

The second domain, illustrated in Figure 1(b), is the aviation domain in Walker et al. 2017. It is characterized by the following set of actions available at each cell in a 3D grid world: The change in heading is one of $\{$no change, $+45°$, $-45°$, $+90°$, $-90°$, left shift, right shift$\}$, the change in height is one of $\{$no change, climb, descend$\}$ and the change in speed is one of $\{$no change, speed up, slow down$\}$. Hence, the branching factor is 63. The costs of the actions are based on their fuel consumption.

The experiments for the Kiva-like domain were run on a cluster of 38 Amazon EC2 c4.xlarge instances running on an Intel Xeon E5-2666 processor with 4 vcpu (2 physical cores) and 7.5GB RAM per instance. iECBS(w) used 2 workers per MAPF instance with each run using $w = 2$ and a 10 minute runtime limit. M* used 1 worker per MAPF instance with each run using a 1 minute runtime limit.[4] All variants with highways used an inflation factor of 2. The experiments for the aviation domain were run in 16 parallel threads on an Intel Xeon E5-2650 processor with 16 vcpu (8 physical cores) and 132GB overall RAM.

Figures 2 and 3 show the general trend in the difficulty of solving Kiva-like and aviation domain instances with increasing numbers of agents. The difficulty is measured by the success rate, i.e., the percentage of MAPF instances solved within a given runtime limit. A sharp decline in the success rate is characteristic of a phase transition. MAPF instances on the left side of the phase transition do not require much coordination, which means that MAPF solvers can easily recover from bad decisions in the search process and likely solve them within the runtime limit. MAPF instances on the right side of the phase transition require significant coordination, which means that the MAPF solvers can solve only a small fraction of them within the runtime limit. MAPF instances near the phase transition require a critical amount of coordination, and thus they can serve as good test cases for distinguishing among MAPF solvers.

In our experiments, M* could run for a maximum of 1 minute before exhausting the available memory. Figures 2(a) and (b) report the success rates of M* with increasing numbers of parallel runs. Here, each run is given a runtime limit of 1 minute. As expected, the success rate increases monotonically with the number of parallel runs. For example, on instances with 70 agents, one run of M* without highways has a success rate of 16% compared to the 57% success rate of 10 parallel runs. On instances with 100 agents, one run of M* with highways has a success rate of 52% compared to the 94% success rate of 10 parallel runs. Figures 2(c) and (d) report the success rates of M* with our RRR strategy. Here, the runtime limit of 1 minute is divided evenly among the runs. For example, the red line represents 4 runs with 15 seconds allocated to each run. We notice that our RRR strategy generally boosts the success rate of M* with and without highways. For example, on instances with 60 agents, M* has a success rate of 47% compared to the 80% success rate of 10 runs. On instances with 90 agents, one run of M* with highways has a success rate of 79% compared to the 95% success rate of 3 runs. However, we also notice that increasing the number of runs (for example, to 10) does not always yield a higher success rate because solutions cannot be found arbitrarily quickly. The harder a MAPF instance is, the higher the runtime limit of a run needs to be.

Figures 2(e) and (f) report the success rates of iECBS(2) with increasing numbers of parallel runs. The MAPF instances have higher numbers of agents compared to the ones used for M*. Unlike M*, our implementation of iECBS(w) is not memory-intensive and can therefore run for a longer time. Thus, we set the runtime limit of each run to 10 min-

[3]now called Amazon Robotics

[4]This is due to the high memory consumption of our M* implementation.

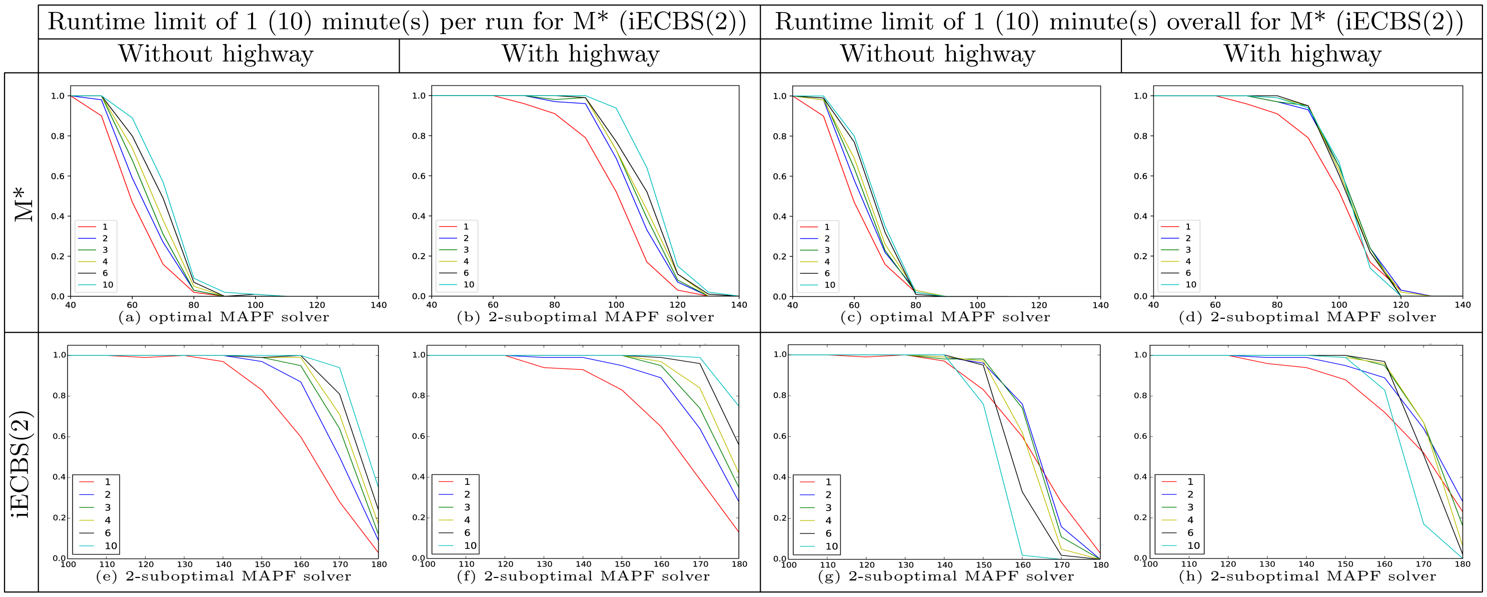


Figure 2: Illustrates the benefits of an RRR strategy in the M* and CBS frameworks for the Kiva-like domain, with and without highways. Each colored curve corresponds to a different number of runs. The x-axis shows the number of agents. The y-axis shows the percentage of MAPF instances solved within a certain runtime limit. For (a) and (b), the runtime limit of M* is 1 minute per run. For (c) and (d), the runtime limit of M* is 1 minute divided by the number of runs. For (e) and (f), the runtime limit of iECBS(w) is 10 minutes per run. For (g) and (h), the runtime limit of iECBS(w) is 10 minutes divided by the number of runs.

utes. As expected, the success rate of iECBS(2) increases monotonically with the number of parallel runs. For example, on instances with 170 agents, one run of iECBS(2) without highways has a success rate of 38% compared to the 100% success rate of 10 parallel runs. On instances with 170 agents, one run of iECBS(2) with highways has a success rate of 52% compared to the 100% success rate of 10 parallel runs. Figures 2(g) and (h) report the success rates of iECBS(2) with our RRR strategy. Here, the runtime limit of 10 minutes is divided evenly among the runs. Highways have a less significant influence on the success rate of iECBS(2) because they are used only in the secondary heuristic to order nodes in the `FOCAL` list. In fact, even in the secondary heuristic, highways are used only to break ties among nodes with the same number of conflicts, see Figure 1 of (Cohen et al. 2016). Nevertheless, our RRR strategy boosts the success rate of iECBS(w) with and without highways. For example, on instances with 160 agents, iECBS(2) without highways has a success rate of 60% compared to the 77% success rate of 3 runs. On instances with 160 agents, one run of iECBS(2) with highways has a success rate of 73% compared to the 96% success rate of 6 runs. As in M*, here too, we notice that simply increasing the number of runs does not always yield a higher success rate. In fact, the optimal number of runs depends on the hardness of the MAPF instance characterized by the number of agents. For example, 2, 3 and 4 runs dominate the success rate for up to about 170 agents. Beyond this point, the MAPF instances are even harder and require longer runs.

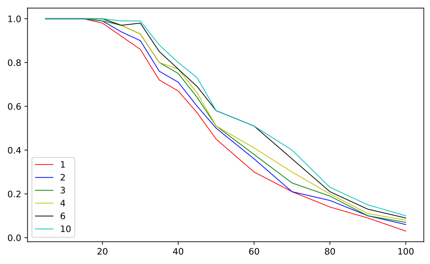
Figure 3(a) reports the success rates of CBS-CL with increasing numbers of parallel runs. As expected, the success rate increases monotonically with the number of parallel runs. For example, on instances with 60 agents, one run of CBS-CL has a success rate of 21% compared to the 40% success rate of 10 parallel runs. Figure 3(b) reports the success rates of CBS-CL with our RRR strategy. For example, on instances with 40 agents, CBS-CL has a success rate of 67% compared to the 76% success rate of 2 runs. Here, the increase in success rate is not as significant as for M* and iECBS(w). Our RRR strategy tends to be less beneficial when the heuristic guidance is poor, intuitively because short runs are less likely to solve a MAPF instance. While the low-level searches of M* and iECBS(w) use informed heuristic values (namely, the distances when ignoring other agents), the heuristic values of CBS-CL are based on the octile distance and hence are far from perfect.

Conclusions

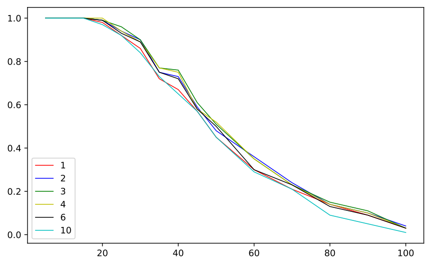
We presented different ways of randomizing deterministic MAPF solvers in the M* and CBS frameworks. In our experiments with these randomized MAPF solvers, we observed a boost in their success rates when increasing the number of parallel runs in two domains inspired by real-world applications. We also sometimes observed a boost in their success rates when using a simple RRR strategy, although the hardness of the MAPF instance, as reflected in our domains by the number of agents, affects the number of restarts for which this boost is most pronounced.

Acknowledgements

The research at the University of Southern California was supported by the National Science Foundation (NSF) under



(a) suboptimal MAPF solver



(b) suboptimal MAPF solver

Figure 3: Illustrates the benefits of an RRR strategy in CBS-CL for the aviation domain. Each colored curve corresponds to a different number of runs. The x-axis shows the number of agents. The y-axis shows the percentage of MAPF instances solved within a certain runtime limit. For (a), the runtime limit of CBS-CL is 10 minutes per run. For (b), the runtime limit of CBS-CL is 10 minutes divided by the number of runs.

grant numbers 1724392, 1409987 and 1319966 as well as a gift from Amazon. The research at Carnegie Mellon University was supported by a gift from Foxconn. The research at the University of Denver was supported in part by NSF IUCRC grant number 1439693.

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 7th Annual Symposium on Combinatorial Search*.

Cohen, L.; Uras, T.; Kumar, T. K. S.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Improved bounded-suboptimal multi-agent path finding solvers. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*.

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*.

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Proceedings of the 8th Annual Symposium on Combinatorial Search*.

Erdem, E.; Kisa, D.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Proceedings of the 10th International Symposium on Combinatorial Search*.

Gomes, C. P.; Selman, B.; Crato, N.; and Kautz, H. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning* 24(1-2):67–100.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4:392 –399.

Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Proceedings of the 8th Robotics: Science and Systems Conference*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *Proceedings of the 12th Pacific Rim International Conference on Artificial Intelligence*.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*.

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

Walker, T. T.; Chan, D.; and Sturtevant, N. R. 2017. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling*.

Yu, J., and LaValle, S. M. 2013a. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*.