Multiagent Pathfinding
AAAI Technical Report WS-12-10

Towards Using Discrete Multiagent Pathfinding to Address Continuous Problems

Athanasios Krontiris
University of Nevada, Reno
Reno, NV 89557 USA
tdk.krontir@gmail.com

Qandeel Sajid
University of Nevada, Reno
Reno, NV 89557 USA
sajid.qandeel@gmail.com

Kostas E. Bekris
University of Nevada, Reno
Reno, NV 89557 USA
bekris@cse.unr.edu

Abstract

Motivated by efficient algorithms for solving combinatorial and discrete instances of the multi-agent pathfinding problem, this report investigates ways to utilize such solutions to solve similar problems in the continuous domain. While a simple discretization of the space which allows the direct application of combinatorial algorithms seems like a straightforward solution, there are additional constraints that such a discretization needs to satisfy in order to be able to provide some form of completeness guarantees in general configuration spaces. This report reviews ideas on how to utilize combinatorial algorithms to solve continuous multi-agent pathfinding problems. It aims to collect feedback from the community regarding the importance and the complexity of this challenge, as well as the appropriateness of the solutions considered here.

Introduction

Multi-agent pathfinding requires the computation of paths for multiple agents so that they move from a set of start configurations to goal configurations while avoiding collisions. Efficient and scalable solutions to this problem are relevant in many applications, such as space exploration (Leitner 2009), warehouse management (Wurman, D'Andrea, and Mountz 2008), intelligent transportation (Dresner and Stone 2008), assembly or disassembly (Halperin, Latombe, and Wilson 1998); Sundaram, Remmler, and Amato (2001) and computer games [Silver (2005b); Nieuwen- huisen, Kamphuis, and Overmars (2007); Wang and Botea (2008a); Jansen and Sturtevant (2008b)] to name a few.

In many applications, the domain that the agents are operating in is continuous and the agents may have complicated geometry. In such setups, one needs to reason about the configuration space of the problem. Figure 1 provides an related example in a two dimensional workspace. This challenge is known in the literature as the multi-movers problem and has been shown to be a hard problem (Hopcroft, Schwartz, and Sharir 1984); Ratner and Warmuth (1986). A discrete version of the multi-agent pathfinding problem is often considered in the literature, similar to the example in Figure 2. This discrete variant involves agents operating in an abstraction,

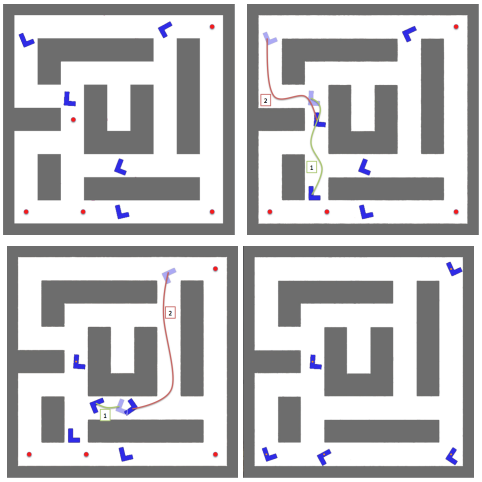

Figure 1: An example solution sequence of a multi-agent pathfinding instance in a continuous two-dimensional workspace. Planar rigid bodies need to be rearranged from a set of initial to a set of goal configurations. The red dots denote the goal locations of the rigid bodies. The configuration space is three dimensional.

such as a graph or a grid, and may be directly applicable in some applications. There are various approaches that have been proposed for this combinatorial version of multi-agent path finding with desirable properties. For instance, a theoretical study, which refers to the problem as the "pebble motion on a graph", outlines a decision algorithm and has shown that in the worst case a n^3 number of moves is sufficient for its solution, where n is the number of vertices in the underlying graph (Kornhauser, Miller, and Spirakis 1984). The same work has also provided examples of graphs where n^3 number of moves is required. More recently, concrete algorithms for the solution of the discrete challenge have been proposed, which return sub-optimal solutions in polynomial time complexity for important subcases of the general discrete challenge (Khorshid, Holte, and Sturtevant 2011), (Luna and Bekris 2011). Furthermore, there are recent complete search-based algorithms, which return optimal solutions and reduce the computational cost relative to naive A* search in the composite space of all agents (Sharon et al. 2011); (Standley and Korf 2011).

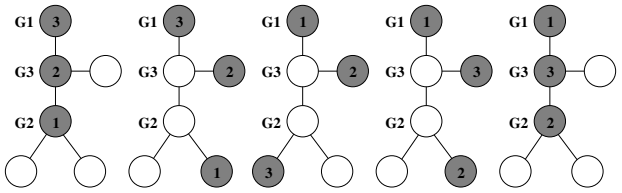

Figure 2: The solution to a discrete multi-agent pathfinding problem.

Given the desirable properties of efficient algorithms for discrete multi-agent pathfinding, this report investigates the following question: how straightforward is it to utilize discrete and combinatorial solutions for multi-agent pathfinding to solve challenges in continuous domains. It may initially appear that it is quite easy to impose a simple discretization of the continuous space, such as a grid or a graph, apply the combinatorial solution on the resulting discretization and then translate the result back to the continuous space. This straightforward approach may directly work in certain simple cases. Nevertheless, there are additional considerations that need to be addressed in the general case that complicate the challenge, even if all the agents have the same geometry, which is a simple variation of the continuous challenge and the starting point for the discussion in this report. These additional concerns are especially important if the objective is to provide some form of completeness guarantee in the continuous case, such as resolution or probabilistic completeness. The report describes various directions that could be considered towards utilizing efficient discrete multi-agent pathfinding algorithms to solve continuous problems.

Related Work

Coupled methods for multi-agent pathfinding consider the agents as a composite system with high dimensionality. The solution is found by searching the composite space with complete, optimal planners. For discrete challanges, this can be solved by involving A* (Hart, Nilsson, and Raphael 1968) or one of its variants (Korf 1985). Unfortunately, the straightforward search approach quickly becomes intractable due to its exponential complexity. This has led to methods that prune the search space while maintaining desirable guarantees. One recent technique searches for optimal paths in an iterative deepening manner, showing that a breadth-first search is feasible (Sharon et al. 2011). Another approach modifies state expansion using operator decomposition and segmenting instances into independent subproblems (Standley 2010), Standley and Korf (2011). There is also work on minimizing the maximum dimension of independent subproblems (van den Berg et al. 2009).

Many coupled planners cede optimality, but retain completeness, and operate on specific graph topologies. Specifically for the discrete challenge, a theoretical study of the problem refers to is as the "pebble motion on graphs" (Kornhauser, Miller, and Spirakis 1984), which can be seen as the general case of the famous 15-puzzle (Loyd 1959). In this setup pebbles occupy distinct vertices of a given graph and are moved from one placement to another. A move consists of transferring a pebble to an adjacent unoccupied vertex. It has been shown that a cubic number of moves as a function of the number of graph vertices is sufficient to solve this challenge and that the decision problem can be solved efficiently (Kornhauser, Miller, and Spirakis 1984).

Many recent methods provide concrete algorithms for specific instances of discrete multi-agent pathfinding. For trees, there is a linear time check for the solvability of an instance (Auletta et al. 1996), (Masehian and Nejad 2009). Based on this result, a tree-based technique has been proposed that employs single-agent primitives (Khorshid, Holte, and Sturtevant 2011). Similar single-agent primitives have been employed by a method that is complete on general graphs with two more vertices than agents and has polynomial complexity (Luna and Bekris 2011). Another method considers the minimum spanning tree of the graph and is complete if there are more leaves in the tree than agents (Peasgood, Clark, and McPhee 2008). There is also a complete approach specifically for bi-connected graphs with two more vertices than agents (Surynek 2009). For "slidable" grid-based problems, there is a polynomial time solution (Wang and Botea 2011). In another variant there is a single pebble and several movable obstacle-pebbles on the nodes of the graph (Papadimitriou et al. 1994). Another method segments the graph into subgraphs by planning high-level operations between subgraphs to reduce the branching factor (Ryan 2008). An unlabeled version of the pebble problem has also been considered (Calinescu, Dimitrescu, and Pach 2008).

In contrast, *decoupled techniques* compute individual paths and resolve collisions as they arise. They can solve problems much faster but they are not guaranteed to be optimal or complete. Certain decoupled methods are based on heuristic search. One creates a flow network within grid-worlds (Wang and Botea 2008), (Jansen and Sturtevant 2008b) and coordinates the actions where the flows intersect to reduce the number of paths and their intersections. WHCA* (Silver 2005a), (Sturtevant and Buro 2006) utilizes dynamic prioritization, windowed cooperative search and a perfect single-agent heuristic to compute scalable solutions. Prioritized planners compute paths sequentially for different agents in order of priority, where high priority agents are moving obstacles for the lower priority ones (Erdmann and Lozano-Perez 1986). The choice of priorities has a significant impact on solution quality (van den Berg and Overmars 2005), and searching the space of priorities can improve performance (Bennewitz, Burgard, and Thrun 2002). For trees, a hierarchical prioritization-based planner exists to coordinate sequential actions (Masehian and Nejad 2010). Other decoupled planners employ a plan-merging scheme to coordinate actions and detect deadlocks (Saha and Isto 2006).

Straightforward Approach

A straightforward approach to transfer solutions from the discrete to the continuous case, where all the agents have the same geometry, is the following two step procedure:

- Discretize the $\mathcal{C}$-space and acquire a graph-based repre-

sentation $G(V, E)$ of $\mathcal{C}_{\text{free}}$ for a single agent. This means that the nodes $v \in V$ correspond to an agent's collision-free configurations, i.e., $v \in \mathcal{C}_{\text{free}}$. Similarly for edges $e \in E$, where $e(s)$ represents the interpolated configuration between the initial and final configuration of the edge e for parameter s, it has to be that $e(s) \in \mathcal{C}_{\text{free}}, \forall s \in [0, 1]$. The approach needs to make sure that the initial and goal configurations of the agents are connected to the graph G.

- Compute a combinatorial solution for the discrete MAPF problem on G. If one exists, translate the discrete path on G to continuous paths in the $\mathcal{C}$-space for all the agents.

The discretization can be achieved either by employing a regular grid in the $\mathcal{C}$-space, implying that a solution can only be resolution complete, or by employing sampling, implying that a solution can only be probabilistically complete. Employing methods that would return asymptotically optimal or near-optimal sampling-based roadmaps would be appropriate in this context. They bring the hope of returning a relatively sparse representation that may still allow covering $\mathcal{C}_{\text{free}}$, reflecting its connectivity and returning good quality paths. If a solution is not found by the combinatorial algorithm, either a higher-resolution grid can be computed or additional sampling can be performed in order to hopefully return a graph G on which a solution can be found.

While the above approach seems relatively straightforward, it has to deal with an important challenge. The constructed roadmap G guarantees that its vertices and edges are collision free with the static geometry of the workspace but not among themselves. This means that if multiple agents are required to use the same roadmap G there is no guarantee that when agent A_1 occupies vertex v_1 and agent A_2 occupies vertex v_2, that the two agents will not be colliding. The same is true for an agent occupying a vertex and another agent traversing an edge of the roadmap or two agents traversing simultaneously two edges. These concerns can be addressed if the following constraints are imposed upon the construction of a roadmap:

a) $\forall v_1, v_2 \in V : A(v_1) \cap A(v_2) = 0$,

b) $\forall v \in V$ and $\forall e \in E$ where $v \neq e(0), e(1)$ and $\forall s \in [0, 1] : A(v) \cap A(e(s)) = 0$ and

c) $\forall e_1, e_2 \in E$ where $e_1(0), e_1(1) \neq e_2(0), e_2(1)$ and $\forall s_1, s_2 \in [0, 1] : A(v_1(s_1)) \cap A(v_2(s_2)) = 0$.

The above constraints guarantee that if a multi-agent path finding solution is computed on the graph $G(V, E)$ where no two agents occupy simultaneously the same vertex or edge, then this solution can be translated to continuous collision-free paths for the agents because a) two agents occupying two distinct vertices will never be colliding, b) an agent occupying a vertex will never collide with an agent traversing an edge and c) two agents traversing two distinct vertices will never collide. If the discrete multi-agent path finding algorithm computes a sequential solution, i.e., one where only one agent traverses an edge at a time, then the requirement c) is not necessary.

A sampling-based algorithm can be easily adapted to satisfy the above expressions. Nevertheless, these requirements are highly constraining. Very quickly an algorithm will not be able to add vertices and edges into the roadmap because they are going to be intersecting with existing features of the roadmap. Thus, the resulting approach cannot be easily shown to be probabilistically or resolution complete, as it is not easy to show that when a solution to the continuous space exists, a graph can be found that satisfies the above requirements and on which a discrete multi-agent path finding solution can be computed.

The following discussion will focus on describing related ideas that could be used in order to address this challenge in trying to transfer a multi-agent path finding solution from the discrete to the continuous case.

Ideas

Relocating Roadmap Nodes

As mentioned, some nodes $\mathcal{V}c \subseteq \mathcal{V}$ of the roadmap $\mathcal{G}$ can be in collision with each other ($A(v_1) \cap A(v_2) \neq 0$). One idea is to attempt to move nodes in the set $\mathcal{V}c$ to a different location, while trying to maintain the topological properties of the graph. By moving nodes, the graph's geometrical properties are changing. The new location of $\mathcal{V}c$ has to respect the following rules:

- The new configuration has to be collision free.
- It should not lead into collisions with other nodes.
- It should not lead into collisions with roadmap edges.

Unfortunately there is no guarantee that the algorithm can find a new location for all the nodes in $\mathcal{V}c$. It is, however, a post-processing step that can be applied after the construction of the roadmap.

Merging Nodes

An alternative solution is to merge two roadmap nodes into a single one so as to prevent more than one agent from attempting to occupy them simultaneously. As the number of intersecting nodes grows, merging these nodes becomes increasingly problematic. Instead of actually replacing the nodes by a single one, it is possible to define a super node, where only one agent is allowed to occupy a configuration in the super node at a time and there is a path between nodes. Converting clusters of nodes into one node significantly alters the original topology of the problem. It reduces the number of nodes on the graph, which can cause the discrete algorithm to fail to find a solution to a solvable problem. A simple solution to the loss of nodes can be to just add nodes to the graph. It is not obvious, however, howmany nodes should be added and where.

Addressing Node/Node and Node/Edge Interactions

The first case that has to be examined is the interaction between nodes $\mathcal{V}c \subseteq \mathcal{V}$ that are in collision. Assume that two robots r_A and r_B have to stop in adjacent nodes that will bring them into collision and that the robot r_B is the second one arriving. In this case r_A has to move out of its position and find a safe spot in a configuration $\mathcal{I}_A$, which can be considered as an intermediate parking spot for that vertex.

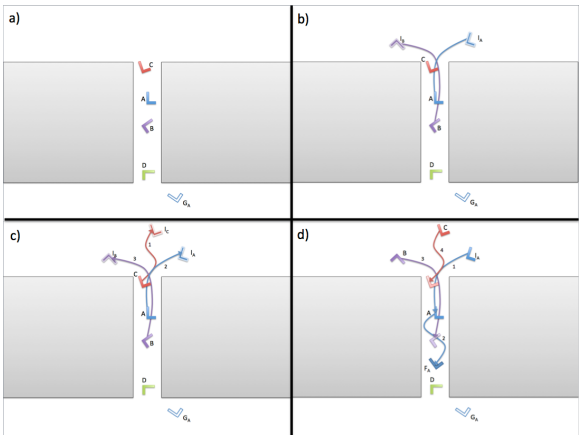

Figure 3: Example of how robot A can reach its goal G_A

Robot r_A has to be able to return to its node after r_B moves away. If r_A has to move before r_B, then r_A can first tries to find a path to its goal. If this fails, then r_B has to find an intermediate configuration $\mathcal{I}_B$ that will allow it to come back to its previous node and traverse the edge to reach its next destination.

In some cases a robot r_A is traversing an edge that is blocked by robot r_B occupying a node. The idea from the previous paragraph can be extended to this case. Robot r_B has to find an intermediate collision free configuration $\mathcal{I}_B$ such that r_A can traverse the edge. After r_A finishes its move and as long as the destination node of r_A in not in collision with the previous position of r_B, then r_B returns to its original node. In the case that the current position of r_A is in collision with the last position of r_B, r_B has to wait in $\mathcal{I}_B$ the situation is treated as interaction between two nodes.

In both cases (node/node and node/edge interaction) it is not guaranteed that the algorithm will find a valid space for the robot that has to move away from its position (e.g., r_A). Then the algorithm will claim that there is no solution for a problem, even if one exists.

It is also possible to consider a solution that operates directly in the continuous space. Consider that a graph $\mathcal{G}$ is computed initially and the discrete algorithm is executed on this graph. Each robot $r_A$is trying to find a path to the next node according to the solution specified by the discrete algorithm, using a sampling-based algorithm (e.g., RRT*). For the example in Fig. 3 the robot $r_A$tries to reach goal G_A, but there are other robots blocking its path. Following the idea from the discretized version of the algorithm the following steps can be executed:

- Robot r_A tries to reach the next node of its trajectory F_A.
- If r_B is blocking r_A.
- Both of the robots are expanding a tree using operations in the continuous space (e.g,. using RRT*).
- Try to find a combination of vertices on the RRT* trees so that:
  - Robot r_A can go to an intermediate position I_A that is collision free with respect to the current position (C_B) of robot r_B.
  - Robot r_B can go to an intermediate position I_B that is collision free with respect to the intermediate position (I_A) of robot r_A.
  - Robot r_A can return to the previous current position (C_A) from the intermediate position I_A that is collision free with respect to the intermediate position (I_B) of robot r_B.
  - Robot r_A can go to the next step of the trajectory that was following F_A that is collision free with respect to the intermediate position (I_B) of robot r_B.
  - Robot r_B can return to the previous current position (C_B) from the intermediate position $I_B$that is collision free with respect to the final position (F_i) of robot r_A.
- When both robots come up with an intermediate position (I_A, I_B) they are trying to move there.
- Robot r_C, though, is blocking r_A's way to its intermediate position I_A.
- Robot r_A and r_C have to find new intermediate positions so as $r_C$will free the road to robot r_A.
- In Fig.3 (c), r_C is moving to its intermediate position I_C, so that robot r_A can move to its intermediate position I_A. Final the robot r_B is moving to its intermediate position I_B.
- Now that the robot r_B moved away of the position that was blocking robot r_A, robot r_A can return to its C_A position and then move to its next position F_A. Then the previous steps have to happen in the opposite direction. Robot r_B is moving back to its C_B position and robot r_C is moving back to its C_C position (Fig .3 (d)).

Unfortunately, as before, this version cannot guarantee that it will find a solution in the general case, if one exists. The problem arises when more than two robots have to be pushed away from their position so that robot $r_A$can find a path to its goal. The algorithm has to be called recursively, which may result in an exhaustive search for the problem.

Transferring Primitives to the Continuous Space

Rather than use the discrete algorithm to search, in some cases it is easier to search directly in the continuous space. The idea behind this approach is to decompose each agent's path into segments defined by critical points. A critical point occurs at the following locations:

- Initial location of the agent
- Final location of the agent
- Before the intersection with another agent
- Right after the intersection with another agent.

The objective is to sequentially move each agent (i.e., "pushing agent") along the critical points until it reaches its goal. Blocking agents (all agents except the pushing agent) that occupy a segment between the critical points are forced to leave. Each agent has its own roadmap. When the pushing agent reaches a critical point, it clears the segment corresponding to the region between the current critical point

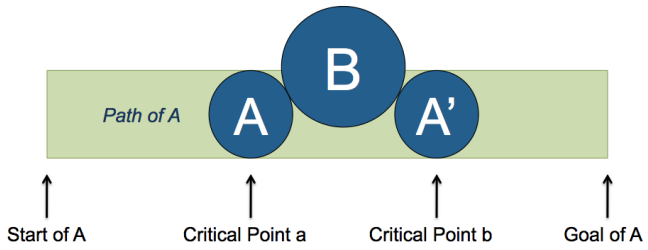

Figure 4: Agent A executes swap by moving from critical point a to b.

to the next by forcing any agent(s) on that segment to use their own roadmaps. Any other agent blocking the problematic agent(s) will be also pushed away. When the segment is cleared, the pushing agent can make progress from one critical point to the next. If the new critical point does not allow previously pushed agents to return to their original position, the pushing agent continues to move from its current critical point to next while agents that can return do so. Many of these pushed agents could have been on their goals in the original configuration or their original configurations were of importance to other agents that need to return to their goals. By returning all pushed agents to their original configurations, the algorithm attempts to find the solution in finite time.

An example is shown in Figure 4, in which agent A is attempting to move from its initial location to its goal while dealing with intersection with agent B. Agent A has four critical points along its goal: A's start/goal, and the point before and after intersection with agent B. When agent A is at critical point a, to get to point b it needs to push agent B from the path segment. Agent B uses its own roadmap to push away. When agent A reach critical point b, agent B is able to return to its original location.

In this algorithm, the roadmaps are constructed using sampling. Each agent keeps track of its own roadmap so as to allow the algorithm to be utilized with agents of various sizes. This is the alternative we are currently pursuing as we consider that it is the most hopeful in providing efficient solutions for the most general setups.

Acknowledgments

The work of the authors is supported by NSF CNS 0932423. Qandeel Sajid was also supported by a Nevada NASA EPSCOR scholarship award. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

References

Auletta, V.; Monti, A.; Parente, M.; and Persiano, P. 1996. A linear time algorithm for the feasibility of pebble motion on trees. In *Proc. of the 5th Scandinavian Workshop on Algorithm Theory*, 259–270.

Bennewitz, M.; Burgard, W.; and Thrun, S. 2002. Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning for Mobile Robots. *Robotics and Autonomous Systems* 41(2):89–99.

Calinescu, G. and Dimitrescu, A. and Pach, J. 2008. Graphs, R., and Grids. *SIAM Journal of Discrete Math* 22(1):124–138.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research* 31:591–656.

Erdmann, M., and Lozano-Perez, T. 1986. On Multiple Moving Objects. In *ICRA*, 1419–1424.

Halperin, D.; Latombe, J.; and Wilson, R. 1998. A general framework for assembly planning: The motion space approach. *Algorithmica* 26(3-4):577–601.

Hart, E. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 2:100–107.

Hopcroft, J. E.; Schwartz, J. T.; and Sharir, M. 1984. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the warehouseman's problem. *Int'l Journal of Robotics Research (IJRR)* 3(4):76–88.

Jansen, M. R., and Sturtevant, N. 2008a. Direction Maps for Cooperative Pathfinding. In *AIIDE.*

Jansen, M. R., and Sturtevant, N. R. 2008b. Direction maps for cooperative pathfinding. In *The Fourth Artificial Intelligence and Interactive Digital Entertainment Conference* (AIIDE08).

Khorshid, M. M.; Holte, R. C.; and Sturtevant, N. 2011. A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding. In *SOCS*, 76–83.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Kornhauser, D.; Miller, G.; and Spirakis, P. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups and applications. In *Proc. of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, 241–250.

Leitner, J. 2009. Multi-robot cooperation in space: A survey. *Advanced Technologies for Enhanced Quality of Life* 144–151.

Loyd, S. 1959. *Mathematical Puzzles of Sam Loyd*. Dover.

Luna, R., and Bekris, K. E. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *IJCAI*, 294–300.

Masehian, E., and Nejad, A. H. 2009. Solvability of Multi Robot Motion Planning Problems on Trees. In *IROS,* 59365941.

Masehian, E., and Nejad, A. H. 2010. A Hierarchical Decoupled Approach for Multi Robot Motion Planning on Trees. In *ICRA*, 3604 – 3609.

Papadimitriou, C. H.; Raghavan, P.; Sudan, M.; and Tamaki, H. 1994. Motion planning on a graph. In *Proc. of the Annual Symposium on Foundations of Computer Science (FOCS)*, 511–520.

Peasgood, M.; Clark, C.; and McPhee, J. 2008. A Complete and Scalable Strategy for Coordinating Multiple Robots within Roadmaps. *IEEE Transactions on Robotics* 24(2):282–292.

Ryan, M. R. K. 2008. Exploiting Subgraph Structure in Multi-Robot Path Planning. *Journal of AI Research* 31:497–542.

Saha, M., and Isto, P. 2006. Multi-Robot Motion Planning by Incremental Coordination. In *IROS*, 5960–5963.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In *IJCAI*, 662–667.

Silver, D. 2005a. Cooperative Pathfinding. In *AIIDE*, 23–28.

Silver, D. 2005b. Cooperative pathfinding. In *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'05)*, 23–28.

Standley, T., and Korf, R. 2011. Complete Algorithms for Cooperative Pathnding Problems. In *IJCAI,* 668673.

Standley, T. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*, 173–178.

Sturtevant, N., and Buro, M. 2006. Improving Collaborative Pathfinding Using Map Abstraction. In *AIIDE,* 8085.

Surynek, P. 2009. A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs. In *ICRA*, 3613–3619.

van den Berg, J., and Overmars, M. 2005. Prioritized Motion Planning for Multiple Robots. In *IROS*, 2217–2222.

van den Berg, J.; Snoeyink, J.; Lin, M.; and Manocha, D. 2009. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and Systems*.

Wang, K.-H. C., and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

Wang, K.-H. C., and Botea, A. 2011. MAPP: A Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of AI Research* 42:55–90.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1):9–19.