Summary: Multi-Agent Path Finding with Kinematic Constraints[*]

Wolfgang Hönig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu,
Nora Ayanian and **Sven Koenig**
Department of Computer Science
University of Southern California
whoenig@usc.edu, tkskwork@gmail.com, {lironcoh, hangma, hongx, ayanian, skoenig}@usc.edu

Abstract

Multi-Agent Path Finding (MAPF) is well studied in both AI and robotics. Given a discretized environment and agents with assigned start and goal locations, MAPF solvers from AI find collision-free paths for hundreds of agents with user-provided sub-optimality guarantees. However, they ignore that actual robots are subject to kinematic constraints (such as velocity limits) and suffer from imperfect plan-execution capabilities. We therefore introduce MAPF-POST to postprocess the output of a MAPF solver in polynomial time to create a plan-execution schedule that can be executed on robots. This schedule works on non-holonomic robots, considers kinematic constraints, provides a guaranteed safety distance between robots, and exploits slack to avoid time-intensive replanning in many cases. We evaluate MAPF-POST in simulation and on differential-drive robots, showcasing the practicality of our approach.

1 Introduction

The Multi-Agent Path Finding (MAPF) problem is the following NP-hard combinatorial optimization problem. Given an environment and agents with assigned start and goal locations, find collision-free paths for the agents from their start to their goal locations that minimize the makespan. Solving the MAPF problem has many applications, including improving traffic at intersections, search and rescue, formation control, warehouse applications, assembly planning, and autonomous aircraft towing vehicles [LaValle, 2006; Yu and LaValle, 2016; Morris *et al.*, 2016].

Our objective is to develop a MAPF solver that combines the advantages of MAPF solvers from AI and robotics. MAPF solvers from AI typically work for agents without kinematic constraints in discretized environments and perform well even in cluttered and tight environments. On the other hand, MAPF solvers from robotics typically work for robots with kinematic constraints in continuous environments but do not perform well in cluttered and tight environments. We base our approach on MAPF solvers from AI because they solve MAPF problems for hundreds of agents in a reasonable amount of time. If even faster MAPF solvers become available, they can be used without modification.

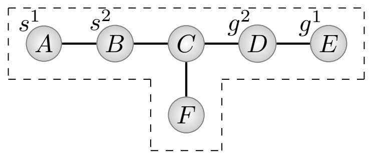


Figure 1: Example where two agents need to reach assigned goal locations in a known environment.

Using the resulting MAPF plans naively on actual robots has limitations. First, robots are subject to kinematic constraints (such as finite maximum velocity limits), that must be taken into account to allow them to execute a MAPF plan. Furthermore, robots suffer from imperfect plan-execution capabilities that need to be taken into account to avoid robot-robot collisions or repeated replanning, which can be slow due to the NP-hardness of the MAPF problem.

We therefore introduce MAPF-POST, a novel approach that makes use of a simple temporal network to postprocess a MAPF plan in polynomial time to create a plan-execution schedule that works on non-holonomic robots, takes their maximum velocities into account, provides a guaranteed safety distance between them, and exploits slack (defined as the difference of the latest and earliest entry times of locations) to absorb imperfect plan executions and avoid time-intensive replanning in many cases. We evaluate MAPF-POST in simulation and on differential-drive robots, showcasing the practicality of our approach.

2 Our Approach: MAPF-POST

We demonstrate our ideas with a simple running example of two agents in a narrow corridor of a grid-world with $1\,\mathrm{m} \times 1\,\mathrm{m}$ cells, see Figure 1. Agent 1 (which has to move from A to E) needs to pass Agent 2 (which has to move from B to D) to reach its goal location, which requires Agent 2 to move into an alcove temporarily, no matter what the

[*]This work is an abridged version of our ICAPS 2016 paper [Hönig *et al.*, 2016a]. Our research was supported by ARL under grant number W911NF-14-D-0005, ONR under grant numbers N00014-14-1-0734 and N00014-09-1-1031, NASA via Stinger Ghaffarian Technologies, and NSF under grant numbers 1409987 and 1319966.

maximum velocities and plan-execution capabilities of the agents are. We can use a MAPF solver from AI to discover such critical intermediate configurations and then use a post-processing step to create a plan-execution schedule that takes their maximum velocities and imperfect plan-execution capabilities into account.

In Step 1, we use a MAPF solver from AI to find a MAPF plan consisting of collision-free paths for all agents, assuming uniform edge lengths and synchronized agent movement from vertex to vertex at discrete timesteps. However, this MAPF plan is nearly impossible to execute safely on actual robots due to their imperfect plan-execution capabilities. It is communication-intensive for the agents to remain perfectly synchronized as they follow their paths. Their individual progress will thus deviate from the MAPF plan, for example, because the edges have non-uniform lengths or velocity limits (due to kinematic constraints or safety concerns) or because the agents cannot move at a uniform velocity (due to kinematic constraints, slip and other robot and environmental limitations). For example, if Agent 2 is slightly slower than Agent 1 in our running example, the agents can collide while they execute their first action. Controlling the velocities of the agents and enforcing safety distances between them can avoid collisions. In Step 2, we use MAPF-POST to create a plan-execution schedule that takes edge lengths and velocity limits into account to provide a guaranteed safety distance between the agents. In Step 3, during plan execution, we can exploit slack to absorb imperfect plan executions and avoid time-intensive replanning in many cases.

2.1 MAPF

The MAPF problem can be stated as follows. Given a graph with vertices (that correspond to locations) and unit-length edges connecting two different vertices each (that correspond to passages between locations in which agents cannot pass each other) as well as a set of agents with assigned start and goal vertices each, find collision-free paths for the agents from their start to their goal vertices (where the agents remain) that minimize the makespan. At each timestep, an agent can either wait at its current vertex or traverse a single edge. Two agents collide when they are at the same vertex at the same timestep or traverse the same edge at the same timestep in opposite directions.

We use the following definitions to formalize the MAPF problem. The graph is $G = (S, E)$, the set of K agents is $1, \ldots, K$, the start vertex of agent j is $s^j \in S$, and its goal vertex is $g^j \in S$. Let s_t^j be the vertex of agent j at timestep t. A path $p^j = [s_0^j, \ldots, s_{T^j}^j]$ for agent j is *feasible* if and only if a) agent j starts at its start vertex s^j and ends at its goal vertex g^j and remains there; and b) every action is either a move action along an edge or a wait action. A *MAPF plan* consists of feasible paths for all agents that do not collide with each other. The makespan of a MAPF plan is the earliest timestep when all agents have reached their goal vertices and remain there. A MAPF plan with the minimum makespan is called *optimal*.

The optimal MAPF plan for our running example is $p^1 = [A, B, C, D, E]$, $p^2 = [B, C, F, C, D]$ and has makespan 4.

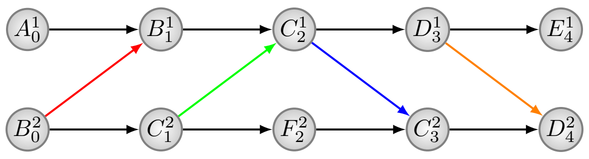


Figure 2: TPG for our running example.

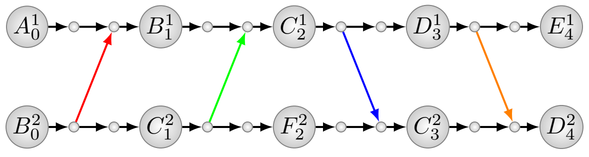


Figure 3: Augmented TPG for our running example. The small circles represent the safety markers.

Approximating optimal MAPF plans within any constant factor less than 4/3 is NP-hard [Ma *et al.*, 2016] but suboptimal MAPF plans (if they exist) can be found in polynomial time [Röger and Helmert, 2012; Kornhauser *et al.*, 1984]. Bounded w-suboptimal MAPF solvers from AI can approximate optimal MAPF plans within a factor of w for hundreds of agents in a reasonable amount of time [Barer *et al.*, 2014], although they often run faster when minimizing the flow time rather than the makespan. Our post-processing step can use a different optimization criterion and different assumptions about the lengths of the edges than the MAPF solver.

2.2 Temporal Plan Graph

We now present an algorithm for converting a MAPF plan to a data structure called the Temporal Plan Graph (TPG). A TPG is a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each vertex $v \in \mathcal{V}$ represents an event, which corresponds to an agent entering a location. Each edge $(v, v') \in \mathcal{E}$ is a temporal precedence between events v and v' indicating that event v must be scheduled before event v'. Basically, the TPG imposes two types of temporal precedences between events as dictated by the MAPF plan. Type 1: For each agent, precedences enforce that it enters locations in the order given by its path in the MAPF plan. Type 2: For each pair of agents and each location that they both enter, precedences enforce the order in which the two agents enter the location in the MAPF plan. A plan-execution schedule assigns a time to each event, corresponding to an entry time for each location. Agents that execute a plan-execution schedule enter all locations at these entry times. It is possible to prove that the agents do not collide if they execute a plan-execution schedule that is consistent with these precedences. The MAPF plan discretizes time and specifies a total order among the events. The TPG, however, does not discretize time and specifies only a partial order among the events, which provides it with the flexibility to take into account both maximum velocities and imperfect plan-execution capabilities of actual robots.

Figure 2 shows the TPG for our running example. In general, the TPG for a MAPF plan with a makespan of T has $O(KT)$ location vertices and $O(K^2T)$ edges and can be constructed in $O(K^2T^2)$ time.

We now add additional vertices (called safety markers) to the TPG to provide a guaranteed safety distance between agents. The safety markers correspond to new locations and allow us to relax the meaning of the edges in the TPG. Each edge $(v, v') \in \mathcal{E}$ can now be a precedence indicating that event v must be scheduled no later than (rather than before) event v'. Each Type 1 edge $e = (v, v') \in \mathcal{E}$ between location vertices v and v' of the TPG (with associated agent j and associated length $l(e)$) is split into three Type 1 edges (all associated with agent j), namely from the first location vertex to a new safety marker (with associated length δ), from there to another new safety marker (with associated length $l(e) - 2\delta$), and from there to the second location vertex (with associated length δ). The user-provided parameter $\delta > 0$ must be chosen so that the length of every edge in E is greater than 2δ. Each Type 2 edge between two location vertices is now changed to a Type 2 edge from the safety marker directly after the first location vertex to the safety marker directly before the second location vertex. We refer to the resulting TPG as the augmented TPG $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$. (The location vertices could easily be removed from an augmented TPG but we keep them for ease of exposition.) Figure 3 shows the augmented TPG for our running example.

We now associate quantitative information with the edges of the augmented TPG, transforming it into a Simple Temporal Network (STN). STNs are widely used for temporal reasoning in AI [Dechter *et al.*, 1991]. An STN is a directed acyclic graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$. Each vertex $v \in \mathcal{V}'$ represents an event. Each edge $e = (v, v') \in \mathcal{E}'$, annotated with the STN bounds $[LB(e), UB(e)]$, is a simple temporal constraint between events v and v' indicating that event v must be scheduled between $LB(e)$ and $UB(e)$ time units before event v'. We add two additional vertices. X_S represents the start event and therefore has edges annotated with the STN bounds $[0, 0]$ to all vertices without incoming edges. Similarly, X_F represents the finish event and therefore has edges annotated with the STN bounds $[0, \infty]$ to all vertices without outgoing edges.

The STN bounds allow us to express non-uniform edge lengths or velocity limits (due to kinematic constraints or safety concerns). We now explain which STN bounds to associate with the edges of the augmented TPG to transform it into an STN. Each edge $(v, v') \in \mathcal{E}'$ is a precedence indicating that event v must be scheduled no later than event v'. Thus, we have to associate the STN bounds $[0, \infty]$ with all edges. However, we can assign tighter STN bounds to Type 1 edges. Consider any Type 1 edge $e = (v, v')$ with associated agent j and associated length $l(e)$. The lower STN bound corresponds to the minimum time needed by agent j for moving from the location associated with vertex v to the location associated with vertex v', and the upper STN bound corresponds to the maximum time. From now on, we assume that agent j has a finite maximum velocity limit $v^*_{\max}(e)$ for the move, for example, due to the kinematic constraints of agent j or safety concerns about traversing edge e with high velocity. Then, agent j needs at least $l(e)/v^*_{\max}(e)$ time units to complete the move, meaning that it enters the location associated with vertex v' at least $l(e)/v^*_{\max}(e)$ time units after it enters the location associated with vertex v, resulting in a tighter lower STN bound than 0. Thus, we associate the STN bounds $[l(e)/v^*_{\max}(e), \infty]$ with the edge. The upper STN bound remains infinity due to the absence of a minimum velocity limit, although we can easily impose one if necessary, for example, for robotic planes or boats.

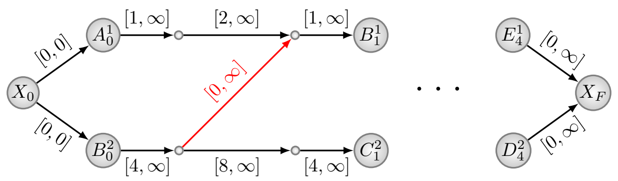


Figure 4: STN for our running example.

Figure 4 shows a part of the STN for our running example. The length of all edges in E is $1\,\mathrm{m}$, the maximum velocity limit of Agent 1 is $1/4\,\mathrm{m/s}$, and the maximum velocity limit of Agent 2 is $1/16\,\mathrm{m/s}$. We use $\delta = 1/4\,\mathrm{m}$. The simple temporal constraint between the safety marker after location vertex B_0^2 and the safety marker before location vertex B_1^1 enforces that Agent 1 cannot move at maximum velocity until it enters location B since it needs to let the slower Agent 2 exit location B before it enters the location.

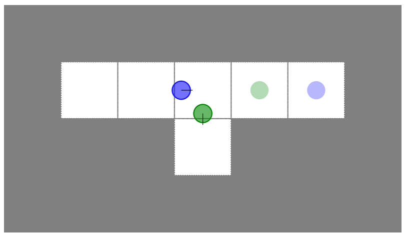
We compute a plan-execution schedule that is consistent with the simple temporal constraints of the STN using graph-based optimization or linear programming. We make an assumption (called the uniform velocity model) that the velocity of an agent is constant during the traversal of each Type 1 edge in the STN, which means that the agent might have to change its velocity instantaneously at location vertices or safety markers. The following two theorems about the existence of a plan-execution schedule and a guaranteed safety distance can be proved.

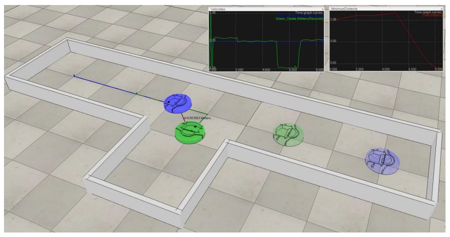
Theorem 1. *There always exists a plan-execution schedule that is consistent with the simple temporal constraints of the STN for a MAPF plan and assigns finite plan-execution times to all vertices in the augmented TPG.*

Theorem 2. *Consider a plan-execution schedule that is consistent with the simple temporal constraints of the STN for a MAPF plan and assigns finite plan-execution times to all vertices in the augmented TPG. Then, point agents always maintain a safety distance of at least $2\delta v_{\min}/v_{\max} > 0$ with respect to graph G (and thus do not collide) if they execute the plan-execution schedule under the uniform velocity model, where $v_{\min}$ and $v_{\max}$ are the minimum and maximum velocities of any agent when traversing any Type 1 edge.*

2.3 Plan Execution

Point agents can execute the plan-execution schedule under the uniform velocity model without colliding if they make sure that they enter the location associated with a vertex at the entry time given by the plan-execution schedule. Unfortunately, this is not likely to happen due to the imperfect plan-execution capabilities of the agents. However, it is unnecessary to replan (which is slow) each time in these cases. Rather, one can construct another STN for the remainder of the MAPF plan, calculate a new plan-execution schedule (which is fast) and replan only if no such plan-execution schedule exists, which can significantly reduce the number of times replanning is needed.


(a) Agent simulation.


(b) V-REP simulation.

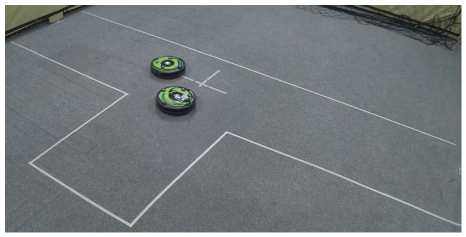

(c) Implementation on Create2 robots.

Figure 5: Screenshots of the three validation settings for our running example.

3 Experimental Validation

We used the ECBS+HWY solver [Cohen *et al.*, 2015] as a bounded suboptimal MAPF solver, extended to support non-holonomic agents. We implemented MAPF-POST in C++ using the `boost graph` library for the STN creation, and GUROBI as the LP solver, all running on a PC with an i7-4600U 2.1 GHz processor and 12 GB RAM. We validated our approach experimentally in three different settings, namely using an agent simulation (which implements the uniform velocity model perfectly using holonomic agents), a robot simulation and an implementation on actual robots, which all used a grid-world with $1\,\mathrm{m} \times 1\,\mathrm{m}$ cells and $\delta = 0.4\,\mathrm{m}$. We varied the size of the grid-world, placement of blocked cells, number of agents, and the maximum translational and rotational velocities of the agents. Figure 5 shows screenshots of our running example for all three settings.

We verified the schedule on actual robots using five differential-drive (and thus non-holonomic) Create2 robots from iRobot. A central PC used `roscore` to run MAPF-POST and connected via WiFi to the robots. The robots were equipped with a single-board computer using Ubuntu 14.04 with ROS Jade to run all other software. We ran the experiments in a space of approximately $5\,\mathrm{m} \times 4\,\mathrm{m}$ equipped with a 12-camera VICON MX motion capture system. The robot simulation used a model of the Create2 robots added to the robotics simulator V-REP. The cell sizes are sufficiently large so that we did not need to take the kinematic constraints of the robots other than the maximum velocity limits into account. Videos of sample experiments can be found at `http://idm-lab.org/bib/abstracts/Koen16g.html`.

In one simulation experiment, we had 100 robots navigate in a warehouse-like environment similar to the one in [Wurman *et al.*, 2008]. The maximum translational and rotational velocities of all robots were $1\,\mathrm{m/s}$ and $2\,\mathrm{rad/s}$. We used highways for the ECBS+HWY solver with ECBS suboptimality bound 1.5 and highway suboptimality bound 2.1. Computing a MAPF plan and running MAPF-POST took about $6\,\mathrm{min}$ (despite the large environment and many robots) and resulted in a makespan of $88\,\mathrm{s}$. The majority of the runtime was spent on solving the linear program due to the large suboptimality bounds of ECBS+HWY. When we maximized the minimum velocity, the actual minimum safety distance was $0.53\,\mathrm{m}$ and thus much higher than the safety distance of $0.4\,\mathrm{m}$ guaranteed by Theorem 2.

4 Related Work

The MAPF problem has been studied in AI, robotics, and theoretical computer science, see [Wagner, 2015] for an extensive overview. It can be solved by reduction to other well-studied problems (e.g. [Surynek, 2015; Yu and LaValle, 2013; Erdem *et al.*, 2013]). Optimal dedicated MAPF solvers include Enhanced Partial Expansion A* [Goldenberg *et al.*, 2014], Conflict-Based Search [Sharon *et al.*, 2015], and M* [Wagner, 2015]. Dedicated suboptimal MAPF solvers include Push and Rotate [de Wilde *et al.*, 2013], TASS [Khorshid *et al.*, 2011], and BIBOX [Surynek, 2009]. Other approaches combine paths of individual agents (e.g. [Sturtevant and Buro, 2006; Wang and Botea, 2011]).

The research reported in [Wagner, 2015] and [Cirillo *et al.*, 2014a] uses a different approach than ours but shares many of our objectives. Lattice-based planning takes kinematic constraints into account during planning [Cirillo *et al.*, 2014b]. Probabilistic approaches, such as Dec-SIMDP [Melo and Veloso, 2011] and UM* [Wagner, 2015], take into account the imperfect plan-execution capabilities of robots but do this during planning. It is possible to plan for each agent individually and use a postprocessing approach that determines velocity profiles for given paths that obey the kinematic constraints of robots and avoid collisions while minimizing makespan [Peng and Akella, 2005]. An approach such as [LaValle and Hutchinson, 1998] avoids replanning in case of changes to the optimization criterion under the assumption that robots are able to change their velocities instantaneously.

5 Conclusion

In this paper, we introduced MAPF-POST, a novel approach that makes use of a simple temporal network to postprocess the output of a MAPF solver from AI in polynomial time to create a plan-execution schedule that can be executed on actual robots. It takes kinematic constraints of the robots into account, provides a guaranteed safety distance between robots, and exploits slack to avoid time-intensive replanning in many cases. Various extensions of our approach are possible. For example, we have extended it to accommodate user-provided safety distances [Hönig *et al.*, 2016b]. We also intend to extend it to take additional kinematic constraints into account, such as the maximum accelerations important for heavy robots. Finally, we intend to exploit slack for replanning and creating a hybrid between online and offline planning.

References

[Barer *et al.*, 2014] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Intl. Symp. on Combinatorial Search*, pages 19–27, 2014.

[Cirillo *et al.*, 2014a] M. Cirillo, F. Pecora, H. Andreasson, T. Uras, and S. Koenig. Integrated motion planning and coordination for industrial vehicles. In *Intl. Conf. on Automated Planning and Scheduling*, 2014.

[Cirillo *et al.*, 2014b] M. Cirillo, T. Uras, and S. Koenig. A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *Intl. Conf. on Intelligent Robots and Systems*, pages 232–239, 2014.

[Cohen *et al.*, 2015] L. Cohen, T. Uras, and S. Koenig. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Intl. Symp. on Combinatorial Search*, pages 2–8, 2015.

[de Wilde *et al.*, 2013] B. de Wilde, A. W. ter Mors, and C. Witteveen. Push and rotate: Cooperative multi-agent path planning. In *Intl. Conf. on Autonomous Agents and Multi-agent Systems*, pages 87–94, 2013.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[Erdem *et al.*, 2013] E. Erdem, D. G. Kisa, U. Oztok, and P. Schüller. A general formal framework for pathfinding problems with multiple agents. In *AAAI Conf. on Artificial Intelligence*, pages 290–296, 2013.

[Goldenberg *et al.*, 2014] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. Sturtevant, R. Holte, and J. Schaeffer. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research*, 50:141–187, 2014.

[Hönig *et al.*, 2016a] W. Hönig, T. K. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig. Multi-agent path finding with kinematic constraints. In *Intl. Conf. on Automated Planning and Scheduling*, pages 477–485, 2016.

[Hönig *et al.*, 2016b] W. Hönig, T. K. S. Kumar, H. Ma, S. Koenig, and N. Ayanian. Formation change for robot groups in occluded environments. In *Intl. Conf. on Intelligent Robots and Systems*, pages 4836–4842, 2016.

[Khorshid *et al.*, 2011] M. M. Khorshid, R. C. Holte, and N. Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Intl. Symp. on Combinatorial Search*, 2011.

[Kornhauser *et al.*, 1984] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Annual Symp. on Foundations of Computer Science*, pages 241–250, 1984.

[LaValle and Hutchinson, 1998] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *Transactions on Robotics and Automation*, 14(6):912–925, 1998.

[LaValle, 2006] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[Ma *et al.*, 2016] H. Ma, C. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI Conf. on Artificial Intelligence*, 2016.

[Melo and Veloso, 2011] F. S. Melo and M. Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789, 2011.

[Morris *et al.*, 2016] R. Morris, C. S. Păsăreanu, K. Luckow, W. Malik, H. Ma, T. K. S. Kumar, and S. Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop on Planning for Hybrid Systems*, 2016.

[Peng and Akella, 2005] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *Intl. Journal of Robotics Research*, 24(4):295–310, 2005.

[Röger and Helmert, 2012] G. Röger and M. Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In *Intl. Symp. on Combinatorial Search*, pages 40–41, 2012.

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

[Sturtevant and Buro, 2006] N. Sturtevant and M. Buro. Improving collaborative pathfinding using map abstraction. In *Artificial Intelligence and Interactive Digital Entertainment*, pages 80–85, 2006.

[Surynek, 2009] P. Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *Intl. Conf. on Robotics and Automation*, pages 3613–3619, 2009.

[Surynek, 2015] P. Surynek. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *Intl. Joint Conf. on Artificial Intelligence*, pages 1916–1922, 2015.

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

[Wang and Botea, 2011] K. C. Wang and A. Botea. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[Wurman *et al.*, 2008] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008.

[Yu and LaValle, 2013] J. Yu and S. M. LaValle. Planning optimal paths for multiple robots on graphs. In *Intl. Conf. on Robotics and Automation*, pages 3612–3617, 2013.

[Yu and LaValle, 2016] J. Yu and S. M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.