Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

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 finite maximum velocity limits) and suffer from imperfect plan-execution capabilities. We therefore introduce MAPF-POST, a novel approach that makes use of a simple temporal network 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, takes their maximum translational and rotational velocities into account, provides a guaranteed safety distance between them, and exploits slack 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.

Introduction

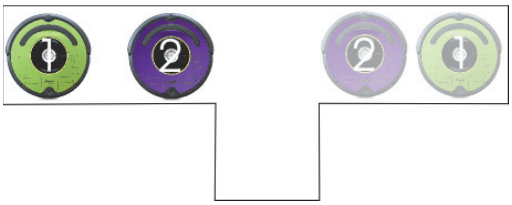
The Multi-Agent Path Finding (MAPF) problem is the following NP-hard combinatorial optimization problem. Given an environment and agents with assigned start and goal locations, find collision-free paths for the agents from their start to their goal locations that minimize the makespan. Solving the MAPF problem has many applications, including improving traffic at intersections, search and rescue, formation control, warehouse applications, and assembly planning. A comprehensive list of applications with references can be found in (Yu and LaValle 2015) and (LaValle 2006). We are motivated by designing autonomous aircraft towing vehicles that tow aircraft all the way from the runways to their gates (and vice versa), thereby reducing pollution, energy consumption, congestion and human workload (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 but 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, we can use them. However, 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 need to 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 translational and rotational 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.

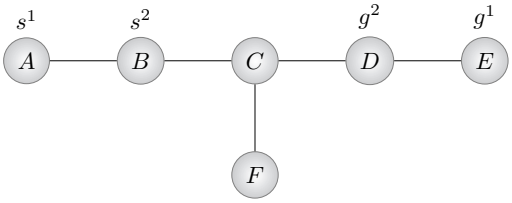
A Motivating Example

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 1a. The maximum velocity limit of Agent 1 is $1/4\,\mathrm{m/s}$, and the maximum velocity limit of Agent 2 is $1/16\,\mathrm{m/s}$. Agent 1 needs to pass Agent 2 to reach its goal location, which requires Agent 2 to move into an alcove temporarily, no matter what the maximum velocities and plan-execution capabilities of the agents are. We can thus 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 plan-execution capabilities into account. In Step 1, we formulate a given navigation task as

*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.
Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.



(a) Example environment with two agents. The transparent agent images on the right mark the goal locations of the agents with corresponding colors.



(b) Graph representation of the example environment.

Agent	$t=1$	$t=2$	$t=3$	$t=4$
1	$A \to B$	$B \to C$	$C \to D$	$D \to E$
2	$B \to C$	$C \to F$	$F \to C$	$C \to D$

(c) Optimal MAPF plan with makespan four.

Figure 1: Running example.

a MAPF problem on a graph of the environment, see Figure 1b. In Step 2, 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, see Figure 1c. 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 robots to remain perfectly synchronized as they follow their paths and 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 robots 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 3, we therefore use MAPF-POST to create a plan-execution schedule that takes information about the edge lengths and velocity limits into account to provide a guaranteed safety distance between the agents and to exploit slack to absorb imperfect plan executions and avoid time-intensive replanning in many cases.

MAPF Problem

We assume from now on that the agents are holonomic and thus able to move in all directions. We later relax this assumption. 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) and a set of agents with assigned start and goal vertices, 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^j_t be the vertex of agent j at timestep t. A path $p^j = [s^j_0, \ldots, s^j_{T^j}, s^j_{T^j+1}, \ldots]$ for agent j is *feasible* if and only if the following conditions hold.

1. Agent j starts at its start vertex, that is, $s^j_0 = s^j$;
2. Agent j ends at its goal vertex and remains there, that is, there exists a minimum finite T^j such that, for each $t \geq T^j$, $s^j_t = g^j$; and
3. Every action is either a move action along an edge or a wait action, that is, for all $t \in \{0, 1, \ldots, T^j - 1\}$, $(s^j_t, s^j_{t+1}) \in E$ or $s^j_t = s^j_{t+1}$.

A *MAPF plan* consists of feasible paths for all agents that are collision-free. A *collision* between agents j and k is either a vertex collision (where $s^j_t = s^k_t$ for some timestep t) or an edge collision (where $s^j_t = s^k_{t+1}$ and $s^j_{t+1} = s^k_t$ for some timestep t). The $T = \max_j T^j$ 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*.

Figure 1c shows the optimal MAPF plan for our running example. 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, Miller, and Spirakis 1984). Bounded w-suboptimal MAPF solvers from AI can approximate optimal MAPF plans with a factor of w for hundreds of agents in a reasonable amount of time (Barer et al. 2014), although they often run faster when trying to minimizing the flow time rather than the makespan. If desired, our post-processing step could use a different optimization criterion and different assumptions about the lengths of the edges than the MAPF solver, which is why we allow every edge in the following to have a finite positive length different from one.

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

Algorithm 1: Algorithm for constructing the TPG.

Data: A MAPF plan with a makespan of T for K agents consisting of path $p^j = [s_0^j, \ldots, s_T^j]$ for each agent j
Result: The temporal plan graph $\mathcal{G}$ for the MAPF plan

```
/* create vertices and Type 1 edges */
for j ← 1 to K do
    Add vertex v_0^j to G
    v ← v_0^j
    for t ← 1 to T do
        if s_t^j ≠ s_{t-1}^j then
            Add vertex v_t^j to G
            Add edge (v, v_t^j) to G
            v ← v_t^j
/* create Type 2 edges */
for j ← 1 to K do
    for t_j ← 0 to T do
        if v_{t_j}^j in G then
            for k ← 1 to K do
                if k ≠ j then
                    for t_k ← t_j + 1 to T do
                        if v_{t_k}^k in G and s_{t_j}^j = s_{t_k}^k then
                            Add edge (v_{t_j}^j, v_{t_k}^k) to G
                            break
```

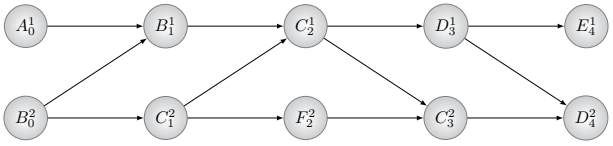


Figure 2: TPG for our running example.

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. We prove later 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 flexibility to take into account both maximum velocities and imperfect plan-execution capabilities of actual robots.

Constructing the Temporal Plan Graph

The TPG can be constructed as follows for a given MAPF plan consisting of the path $p^j = [s_0^j, \ldots, s_T^j, s_T^j, \ldots]$ or, for short, $p^j = [s_0^j, \ldots, s_T^j]$ for each agent j, see Algorithm 1.

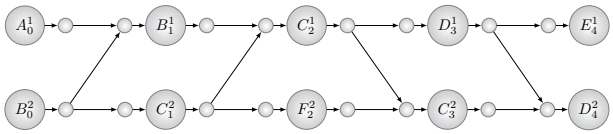


Figure 3: Augmented TPG for our running example. The small circles represent the safety markers.

For each agent j, we extract a route r^j from path p^j by keeping only the first of several consecutive identical locations on the path (which corresponds to removing the wait actions but keeping the move actions). For each (kept) location s_t^j on route r^t, we create a location vertex $v_t^j \in \mathcal{V}$ (with associated agent j and associated location s_t^j). For each two consecutive (kept) locations s_t^j and $s_{t'}^j$ on the same route, we create a Type 1 edge $(v_t^j, v_{t'}^j) \in \mathcal{E}$ (with associated agent j and associated length equal to the length of edge $(s_t^j, s_{t'}^j) \in E$). This edge corresponds to a precedence of Type 1, indicating that agent j enters location s_t^j directly before location $s_{t'}^j$ and thus enters locations in the order given by its path in the MAPF plan. Type 1 edges thus correspond to actions. For each two identical (kept) locations $s_t^j = s_{t'}^k = s$ on different routes (and thus $j \neq k$) with $t < t'$, we create a Type 2 edge $(v_t^j, v_{t'}^k) \in \mathcal{E}$. This edge corresponds to a precedence of Type 2, indicating that agent j enters location s before a different agent k enters the same location. 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. Algorithm 1 constructs it in $O(K^2T^2)$ time, avoiding to add some Type 2 edges that are implied by transitivity.

Augmenting the Temporal Plan Graph

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 $\delta > 0$ needs to 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.

Encoding Kinematic Constraints

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. 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. One can calculate a schedule (that assigns a time $t(v)$ to each event v with the convention that $t(X_S) = 0$) that satisfies all simple temporal constraints in polynomial time using minimum-cost path computations on the directed distance graph of the STN, typically done with the Bellman-Ford algorithm. Every vertex of the STN is translated into a vertex of the distance graph. Every edge of the STN $e = (v, v') \in \mathcal{E}'$ annotated with the STN bounds $[LB(e), UB(e)]$ is translated into two edges of the distance graph, namely one edge (v, v') of cost $UB(e)$ and one edge (v', v) of cost $-LB(e)$. The absence of negative cost cycles in the distance graph is equivalent to the existence of a schedule that satisfies all simple temporal constraints (Dechter, Meiri, and Pearl 1991).

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.

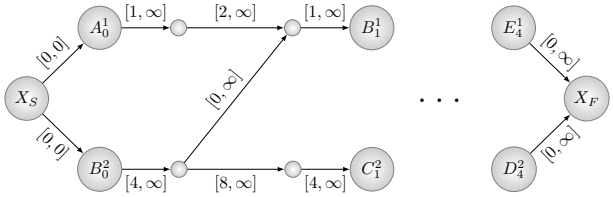


Figure 4: STN for our running example.

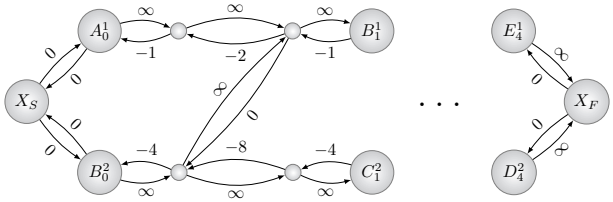


Figure 5: Distance graph for our running example.

Figure 4 shows a part of the STN for our running example, and Figure 5 shows its distance graph. Remember that 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 $\delta = 0.25\,\mathrm{m}$. 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.

Calculating a Plan-Execution Schedule

We now discuss how to calculate a plan-execution schedule for a given MAPF plan depending on the desired optimization criterion. If necessary, the optimization criterion used for the plan-execution schedule could be different from the one of the MAPF solver to adapt the plan-execution schedule of the resulting MAPF plan to the desired optimization criterion in case no MAPF solver is available for it.

Minimizing the Flow Time and the Makespan

A plan-execution schedule that is consistent with the simple temporal constraints and minimizes the flow time (the sum of the earliest times when each agent has reached its goal vertex and remains there) can be calculated in polynomial time with both graph-based optimization and linear programming approaches. Such a plan-execution schedule also minimizes the makespan (the earliest time when all agents have reached their goal vertices and remain there) and assigns finite plan-executions times to all vertices in the augmented TPG.

- **Graph-Based Optimization:** We can obtain a plan-execution schedule that is consistent with the simple temporal constraints and minimizes the flow time by calculating the earliest plan-execution time of any vertex $v \in \mathcal{V}$' that is consistent with the simple temporal constraints as the negative of the cost of a shortest path from v to X_S in the distance graph.

- **Linear Programming:** We can also solve the following linear program (LP) in polynomial time, whose objective function corresponds to minimizing the flow time and whose constraints correspond to the simple temporal constraints, where v^j is the location vertex in the STN with associated agent j (and associated goal vertex of the agent) that is connected to vertex X_F. (We could simply use "Minimize $t(X_F)$" to minimize the makespan.)

$$\begin{aligned}&\text{Minimize } \textstyle\sum_{j=1}^{K} t(v^j)\\ &\text{such that } t(X_S) = 0\\ &\text{and, for all } e = (v, v') \in \mathcal{E}',\\ &t(v') - t(v) \geq LB(e)\\ &t(v') - t(v) \leq UB(e)\end{aligned}$$

The optimization yields a plan-execution schedule that assigns a time $t(v)$ to each event v, corresponding to the time when the associated agent should enter the associated location. In the following, we make an assumption (called the uniform velocity model), namely that the velocity of an agent is constant for 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 and thus needs infinite acceleration capabilities at these locations. It is the task of the agent controllers to approximate this not quite realistic assumption. The velocity of the agent while traversing a Type 1 edge $e = (v, v') \in \mathcal{E}'$ with associated length $l(e)$ then needs to be set to $l(e)/(t(v') - t(v))$ to implement the plan-execution schedule, which implies that the agent always moves (albeit perhaps with small velocity) until it reaches its goal vertex since the traversal of an edge starts when the agent enters one of its vertices and ends when it enters the other vertex.

Maximizing the Global Minimum Velocity Limit

For a given plan-execution schedule, assume that the minimum and maximum velocities of any agent when traversing any Type 1 edge are $v_{\min}$ and $v_{\max}$, respectively, under the uniform velocity model. The following theorem establishes the existence of a plan-execution schedule that is consistent with the simple temporal constraints and moves all agents to their goal vertices. Every plan-execution schedule with these properties can be executed under the minimum velocity model and results in $v_{\min} > 0$ since otherwise some agent would not reach its goal vertex.

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.*

Proof. The distance graph of the STN does not have negative cost cycles (since the STN is a directed acyclic graph and all upper STN bounds are infinity), which guarantees that there exists a plan-execution schedule that satisfies all simple temporal constraint (Dechter, Meiri, and Pearl 1991). The earliest plan-execution time of any vertex $v \in \mathcal{V}'$ in the augmented TPG is the negative of the cost of a shortest path from v to X_S in the distance graph and thus finite (since all lower STN bounds are finite). □

The following theorem establishes a lower bound on the distance between any two agents at any time (called safety distance) with respect to graph G (and *not* with respect to the continuous environment). However, a weaker lower bound for the safety distance in the continuous environment can be obtained in a similar way for regular graph structures, such as grids (not shown here). Having a large safety distance is important because actual robots are not point agents and suffer from imperfect plan-execution capabilities.

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.*

We prove Theorem 2 in the appendix. The lower bound on the safety distance provided by the theorem is not necessarily tight because the minimum and maximum velocities can occur in different parts of the graph. However, the theorem allows us to use "maximize $v^*_{\min}$" for a global minimum velocity limit $v^*_{\min}$ (that bounds the minimum velocity $v_{\min}$ from below) or, equivalently, "minimize $(v^*_{\min})^{-1}$" for the reciprocal of $v^*_{\min}$ as objective function in the following LP to maximize the bound on the safety distance provided by Theorem 2 in polynomial time. Consider any Type 1 edge $e = (v, v') \in \mathcal{E}'$ with associated agent j and associated length $l(e)$. The upper STN bound corresponds to the maximum time needed by agent j for moving from the location associated with vertex v to the location associated with vertex v' and is thus reduced from infinity to $l(e)/v^*_{\min} = l(e)(v^*_{\min})^{-1}$.

$$\begin{aligned}&\text{Minimize } (v^*_{\min})^{-1}\\ &\text{such that } t(X_S) = 0\\ &\text{and, for all } e = (v, v') \in \mathcal{E}',\\ &t(v') - t(v) \geq LB(e)\\ &t(v') - t(v) \leq UB(e)\\ &t(v') - t(v) \leq l(e)(v^*_{\min})^{-1} \text{ if } e \text{ is a Type 1 edge}\end{aligned}$$

Executing the Plan

The plan-execution schedule determined by the chosen LP, if it is solvable, is consistent with the simple temporal constraints of the STN for the MAPF plan. Thus, 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 vertex $v \in \mathcal{V}'$ at entry time $t(v)$. Unfortunately, this is unlikely going to happen due to the imperfect plan-execution capabilities of the agents. However, it is unnecessary to replan each time in these cases. Rather, one can construct another STN for the remainder of the MAPF plan and calculate a new plan-execution schedule and replan only if no such plan-execution schedule exists, which can significantly reduce the number of times replanning is needed.

Non-Holonomic Agents

So far, we have assumed that the agents are holonomic and thus able to move in all directions. However, many

robots, such as cars or differential-drive robots, are non-holonomic. We make the following changes to accommodate differential-drive robots that operate on grid-worlds. First, we change the definition of the MAPF problem and adapt the MAPF solver appropriately. Vertices now are pairs of locations (cells) and orientations (discretized into the four compass directions) and actions either wait, move forward to the next location or rotate in place 90 degrees clockwise or counter-clockwise. Edges and collisions change accordingly. Second, when we determine the routes from a MAPF plan we remove the wait actions but keep the move and rotate actions. Then, we merge several consecutive rotate actions into one rotate action whose rotation angle is the sum of the merged rotations (and delete the rotate action if its rotation angle is zero). Third, we adapt the placement of Type 2 edges in the TPG since two consecutive vertices on a route can now correspond to the same location (with different orientations). Type 2 edges connect the last such vertex on a route to the first such vertex on another route. Fourth, we split those Type 1 edges (and introduce safety markers) that correspond to move actions but not those Type 1 edges that correspond to rotate actions. Fifth, we associate the STN bounds $[L(e)/V^*_{\max}(e), \infty]$ with a Type 1 edge $e \in \mathcal{E}'$ that corresponds to a rotate action, where $L(e)$ is the absolute value of the rotation angle and $V^*_{\max}(e)$ is the maximum rotational velocity limit. Sixth, the LP that maximizes the global minimum velocity limit has constraints only for Type 1 edges that correspond to move actions. Seventh, Theorem 2—as stated in this paper—requires a non-zero minimum translational velocity $v_{\min}$ to guarantee a positive safety distance but we can simply assume for the purpose of the theorem that non-holonomic agents do not rotate in place in the locations associated with location vertices but move slowly toward the locations of the next safety markers on their paths while rotating.

Experimental Validation

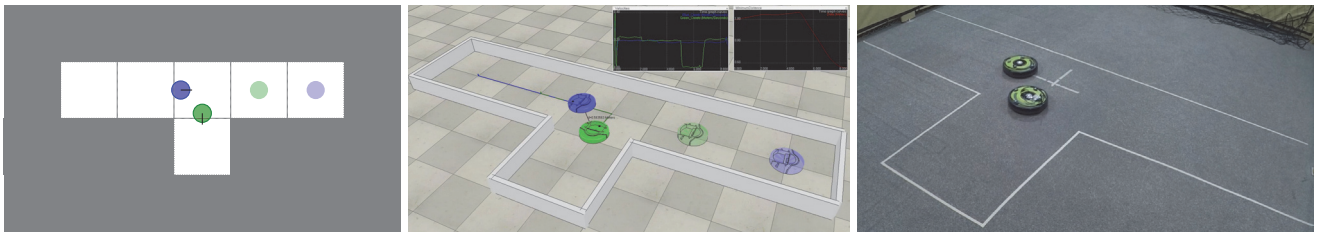
We used the ECBS+HWY solver (Cohen, Uras, and Koenig 2015) as state-of-the-art bounded suboptimal MAPF solver from AI, extended to support non-holonomic agents. We implemented MAPF-POST in C++ using the `boost graph` library (Siek, Lee, and Lumsdaine 2001) for the STN creation and GUROBI (Gurobi Optimization, Inc. 2015) 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 grid-worlds 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 6 shows screenshots of our running example for all three settings.

To ensure the similarity of the robot simulation and implementation on actual robots, we used the robot operation system ROS as middleware by implementing a robot controller directly in ROS that drives either virtual robots in V-REP or actual robots. The controller controls the state $[x, y, \theta]^T$ and tries to meet the deadline specified by the plan-execution schedule of MAPF-POST by setting the translational (or rotational) velocity of a robot to the ratio of the remaining time to reach the next location (or orientation) and the remaining translational (or rotational) distance, which approximates the uniform velocity model well. A PID-controller corrects for heading error and drift by driving the two motors independently and using the known location of the robot from either the simulation or motion capture system.

The implementation on actual robots used five differential-drive (and thus non-holonomic) Create2 robots from iRobot (iRobot 2015), which are refurbished Roomba vacuum cleaners designed specifically for STEM education and research. They have a cylindrical shape with diameter $0.35\,\mathrm{m}$, height $0.1\,\mathrm{m}$, and weight $3.5\,\mathrm{kg}$ and can reach a translational velocity of $0.5\,\mathrm{m/s}$ and a rotational velocity of $4.2\,\mathrm{rad/s}$. A central PC used `roscore` to run MAPF-POST and connected via WiFi to the robots. The robots were equipped with single-board computers, such as ODROID-XU4 and ODROID-C1+ (www.hardkernel.com). These on-board computers used Ubuntu 14.04 with ROS Jade to run all other software, and interfaced to their robot via its serial port. 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 (www.vicon.com). The robot simulation used a model of the Create2 robots added to the robotics simulator V-REP (Rohmer, Singh, and Freese 2013). The cell sizes are sufficiently large so that we do 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 *youtu.be/mV3BqnelqDU*. In the following, we report two experimental results for the robot simulation.

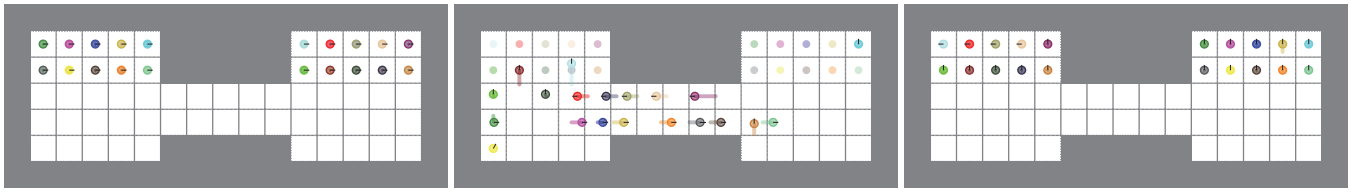
Experiment 1: 10 robots had to move from the left room to the right room, and 10 robots had to move from the right room to the left room in the environment from Figure 7. A narrow corridor connected the two rooms, potentially causing many robot collisions. The maximum translational and rotational velocities of half of the robots in each room were $0.2\,\mathrm{m/s}$ and $1\,\mathrm{rad/s}$, and the maximum velocities of the other half were $0.4\,\mathrm{m/s}$ and $2\,\mathrm{rad/s}$. We used highways for the ECBS+HWY solver with ECBS suboptimality bound 1.5 and highway suboptimality bound 1.5. Whether we maximized the minimum velocity (and thus the safety distance guaranteed by Theorem 2) or minimized the flow time, computing a MAPF plan and running MAPF-POST took only a few seconds and resulted in a makespan of $106\,\mathrm{s}$. When we minimized the flow time, the average flow time per robot was $87\,\mathrm{s}$. When we maximized the minimum velocity, our simulation showed that the actual minimum safety distance was $0.53\,\mathrm{m}$ (with respect to the continuous environment) and thus much higher than the safety distance of $0.4\,\mathrm{m}$ guaranteed by Theorem 2 (with respect to the graph on which the robots move). When we did not maximize the minimum velocity, the actual minimum safety distance was smaller than the diameter of the robots and thus insufficient.

Experiment 2: 100 robots had to navigate in a warehouse-like environment similar to the one in (Wurman, D'Andrea, and Mountz 2008). The maximum transla-



(a) Agent simulation. (b) V-REP simulation. (c) Implementation on Create2 robots.

Figure 6: Screenshots of the three validation settings for our running example.



(a) Start ($t = 0\,\mathrm{s}$). (b) Middle ($t = 53\,\mathrm{s}$). (c) End ($t = 106\,\mathrm{s}$).

Figure 7: Environment where 10 robots had to move from the left room to the right room, and 10 robots had to move from the right room to the left room. The tail of each robot in 7b shows its current direction of movement.

tional 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. Whether we maximized the minimum velocity or minimized the flow time, 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 LP due to the large suboptimality bounds of ECBS+HWY. We intend to decrease it in the future by using a graph-based optimization approach rather than an LP. When we minimized the flow time, the average flow time per robot was $69\,\mathrm{s}$. When we maximized the minimum velocity, the actual minimum safety distance again was $0.53\,\mathrm{m}$ and thus much higher than the safety distance of $0.4\,\mathrm{m}$ guaranteed by Theorem 2.

Related Work

The MAPF problem has been studied in artificial intelligence, robotics, and theoretical computer science, see (Wagner 2015) for an extensive overview. It can, for example, be solved by reductions to other well-studied problems, including satisfiability (Surynek 2015), integer linear programming (Yu and LaValle 2013), and answer set programming (Erdem et al. 2013). Optimal dedicated MAPF solvers include Independence Detection with Operator Decomposition (Standley and Korf 2011), Enhanced Partial Expansion A* (Goldenberg et al. 2014), Increasing Cost Tree Search (Sharon et al. 2013), Conflict-Based Search (Sharon et al. 2015), M* (Wagner 2015), and their variants. Dedicated suboptimal MAPF solvers include Push and Swap/Rotate (Sajid, Luna, and Bekris 2012; de Wilde, ter Mors, and Witteveen 2013), TASS (Khorshid, Holte, and Sturtevant 2011), BIBOX (Surynek 2009), and their variants. Other approaches, such as Windowed-Hierarchical Cooperative A* (Silver 2005; Sturtevant and Buro 2006), Flow Annotation Replanning (Wang and Botea 2008) and MAPP (Wang and Botea 2011), combine paths of individual agents.

The research reported in (Wagner 2015) and (Cirillo et al. 2014) uses a different approach than ours but shares many of our objectives. Approaches that share individual properties with our approach are, for example, the following ones.

- Probabilistic approaches, such as Dec-SIMDP (Melo and Veloso 2011) and UM* (Wagner 2015), take into account that robots have imperfect plan-execution capabilities but do this during planning.
- An approach such as (Cirillo, Uras, and Koenig 2014) uses lattice-based planning (Pivtoraiko and Kelly 2011) to take the kinematic constraints of robots into account but does this during planning.
- An approach such as (Peng and Akella 2005) is a post-processing approach that determines velocity profiles for given paths that obey the kinematic constraints of robots and avoid collisions while minimizing makespan.
- 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.

Conclusions

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. Many interesting extensions of our approach are possible. For example, we intend to make it work for

user-provided safety distances. 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 (which has not been implemented yet) and creating a hybrid between online and offline planning. For example, it is especially important to monitor progress toward locations that are associated with vertices in the simple temporal network whose slacks are small. Robots could be alerted of the importance of reaching these bottleneck locations in a timely manner. If probabilistic models of delays and other deviations from the nominal velocities are available, they can be used to determine the probability that each location will be reached between its earliest and latest entry time and trigger replanning only if one of these probabilities become small.

Proof of Theorem 2

All distances in the following are with respect to graph G. When we discuss the distances between vertices in the augmented TPG, we mean the distances between the locations associated with them. We define the protected cloud of vertex $s \in S$ to contain all points on edges adjacent to it in graph G at distance less than δ from it. We use the following properties that follow from the fact that δ is chosen so that the lengths of all edges are larger than 2δ. Property 1: The protected clouds of different vertices in S do not intersect. Property 2: The distance between any two safety markers at a positive distance of each other that are connected to the same location vertex is 2δ.

Lemma 3. *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, no two point agents can be in the same protected cloud at the same time if they execute the plan-execution schedule under the uniform velocity model.*

Proof. The proof is by contradiction. Consider any event where two different agents j and k are in the protected cloud of some vertex $s \in S$ at the same time. Consider the timestep t when agent j visits vertex s and the timestep t' when agent k visits vertex s in the given MAPF plan (that both correspond to the event). Since the MAPF plan does not have vertex collisions, timesteps t and t' must be different. Without loss of generality, let $t < t'$. The Type 2 edges of the augmented TPG then enforce that agent j exits the protected cloud of vertex s before agent k enters it, which is a contradiction. □

Lemma 4. *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, no two point agents can traverse the same graph edge at the same time in opposite directions if they execute the plan-execution schedule under the uniform velocity model.*

Proof. The proof is by contradiction. Consider any event where agent j moves from vertex $s \in S$ to vertex $s' \in S$ while agent k moves from vertex s' to vertex s. Consider the timestep t when agent j moves from vertex s to vertex s' and the timestep t' when agent k moves from vertex s' to vertex s in the given MAPF plan (that both correspond to the event). Since the MAPF plan does not have edge collisions, timesteps t and t' must be different. Without loss of generality, let $t < t'$. Thus, agent j is at vertex s' at timestep $t + 1$, and agent k is at vertex s' at timestep t'. Since the MAPF plan does not have vertex collisions, it must be that $t + 1 < t'$. The Type 2 edges of the augmented TPG then enforce that agent j exits the protected cloud of vertex s' (which means that it finished moving from vertex s to vertex s') before agent k enters it (which means that it has not yet started to move from vertex s' to vertex s), which is a contradiction. □

To prove Theorem 2, we need to consider only two cases where the distance of two different agents might not satisfy the theorem. The other cases satisfy the theorem, result in no smaller distances than one of the two cases, or are impossible due to Lemmata 3 and 4. Theorem 2 then follows since the plan-execution schedule from Theorem 2 can be executed under the minimum velocity model and results in $v_{\min} > 0$.

Case 1: Agents j and k move from location vertex v via safety markers m and m' to location vertex v', and agent j starts to leave the protected cloud of location vertex v first. When both agents move from safety marker m to safety marker m', they both move at uniform velocities due to the uniform velocity model. Thus, their distance is smallest when agent k is at safety marker m (Subcase a) or agent j is at safety marker m' (Subcase b). Both of these subcases are also of independent interest. Subcase a: Agent k can enter the protected cloud of location vertex v only once agent j leaves it. So, at the time when agent j is at safety marker m, their distance is at least 2δ and thus satisfies the theorem. Agent k now needs at least $2\delta/v_{\max}$ time units to be at safety marker m. At that time, agent j is already a distance of at least $2\delta v_{\min}/v_{\max}$ away. So, at the time when agent k is at safety marker m, their distance satisfies the theorem. Subcase b: When agent j is at safety marker m', it needs at least $2\delta/v_{\max}$ time units to leave the protected cloud of location vertex v'. During that time, agent k must move a distance of at least $2\delta v_{\min}/v_{\max}$ but cannot enter the protected cloud of location vertex v'. So, at the time when agent j is at safety marker m', their distance satisfies the theorem.

Case 2: Agent j moves via safety marker m to location vertex v and is still outside of the protected cloud of location vertex v, and agent k moves from a safety marker different from m via location vertex v to another safety marker different from m. Due to the uniform velocity model, the distance of the agents is smallest when agent k enters the protected cloud of location vertex v (Subcase a), is at location vertex v (Subcase b), or leaves the protected cloud of location vertex v (Subcase c). Subcases a+c: When agent k enters or leaves the protected cloud, then their distance is at least 2δ and thus satisfies the theorem. Subcase b: When agent k is at location vertex v, it needs at least $\delta/v_{\max}$ time units to leave the protected cloud of location vertex v. During that time, agent

j must move a distance of at least $\delta v_{\min}/v_{\max}$ but cannot enter the protected cloud of location vertex v. So, at the time when agent k is at location vertex v, their distance is at least $\delta v_{\min}/v_{\max} + \delta \geq 2\delta v_{\min}/v_{\max}$ and thus satisfies the theorem.

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 *Annual Symposium on Combinatorial Search*, 19–27.

Cirillo, M.; Pecora, F.; Andreasson, H.; Uras, T.; and Koenig, S. 2014. Integrated motion planning and coordination for industrial vehicles. In *International Conference on Automated Planning and Scheduling*.

Cirillo, M.; Uras, T.; and Koenig, S. 2014. A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *International Conference on Intelligent Robots and Systems*, 232–239.

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *International Symposium on Combinatorial Search*, 2–8.

de Wilde, B.; ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: Cooperative multi-agent path planning. In *International Conference on Autonomous Agents and Multi-agent Systems*, 87–94.

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

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

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R.; and Schaeffer, J. 2014. Enhanced partial expansion A. *Journal of Artificial Intelligence Research* 50:141–187.

Gurobi Optimization, Inc. 2015. Gurobi optimizer reference manual.

iRobot. 2015. *iRobot Create2 Open Interface (OI) Specification based on the iRobot Roomba 600*.

Khorshid, M. M.; Holte, R. C.; and Sturtevant, N. 2011. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *International Symposium on Combinatorial Search*.

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

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

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

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

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

Morris, R.; Păsăreanu, C. S.; Luckow, K.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*.

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

Pivtoraiko, M., and Kelly, A. 2011. Kinodynamic motion planning with state lattice motion primitives. In *International Conference on Intelligent Robots and Systems*, 2172–2179.

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

Rohmer, E.; Singh, S. P. N.; and Freese, M. 2013. V-REP: A versatile and scalable robot simulation framework. In *International Conference on Intelligent Robots and Systems*, 1321–1326.

Sajid, Q.; Luna, R.; and Bekris, K. E. 2012. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *Annual Symposium on Combinatorial Search*, 88–96.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

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

Siek, J. G.; Lee, L.-Q.; and Lumsdaine, A. 2001. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley.

Silver, D. 2005. Cooperative pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment*, 117–122.

Standley, T., and Korf, R. 2011. Complete algorithms for cooperative pathfinding problems. In *International Joint Conference on Artificial Intelligence*, 668–673.

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

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

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

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

Wang, K.-H. C., and Botea, A. 2008. Fast and memory-efficient multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling*, 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 Artificial Intelligence 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):9–20.

Yu, J., and LaValle, S. M. 2013. Planning optimal paths for multiple robots on graphs. In *International Conference on Robotics and Automation*, 3612–3617.

Yu, J., and LaValle, S. M. 2015. Optimal multi-robot path planning on graphs: Complete algorithms and effective heuristics. *arXiv:1507.03290*.