Overview: A Hierarchical Framework for Plan Generation and Execution in Multirobot Systems

Hang Ma, Wolfgang Hönig, Liron Cohen, Tansel Uras, Hong Xu, T.K. Satish Kumar, Nora Ayanian, and Sven Koenig, *University of Southern California*

A framework for coordinating task- and motion-level operations for many robots uses informed search to find causally feasible solutions and simple temporal networks to include kinematic constraints and react to dynamic changes.

The problem of coordinating task- and motion-level operations for multirobot systems arises in many real-world scenarios. A simple example is an automated warehouse system in which heavy robots move inventory pods in a space inhabited by humans. The robots may have to avoid close proximity to humans and each other, or they may have to compete for resources with each other, yet they have to work toward a common objective.[1] Another example is airport surface operations in which towing vehicles autonomously navigate to aircraft and tow them to their destinations.[2] This task-level coordination has to be done in conjunction with the motion-level coordination of action primitives so that each robot has a kinematically feasible plan.

The coordination of task- and motion-level operations for multirobot systems requires a large search space. Current technologies are inadequate for addressing the complexity of the problem, which becomes even worse since we have to take imperfections in plan execution into account. For example, exogenous events might not be included in the domain model. Even if they are, they can often be modeled only probabilistically.[3]

In this article, we present an overview of our hierarchical framework for the long-term autonomy of multirobot systems. Our framework combines techniques from automated artificial intelligence (AI) planning, temporal reasoning, and robotics. Figure 1 shows its architecture for a small example.

The plan-generation phase uses a state-of-the-art AI planner[4,5] for causal reasoning about the task-level actions of the robots, independent of their kinematic constraints to achieve scalability. It then identifies the dependencies between the preconditions and effects of the actions in the generated plan

1541-1672/17/$33.00 © 2017 IEEE
Published by the IEEE Computer Society

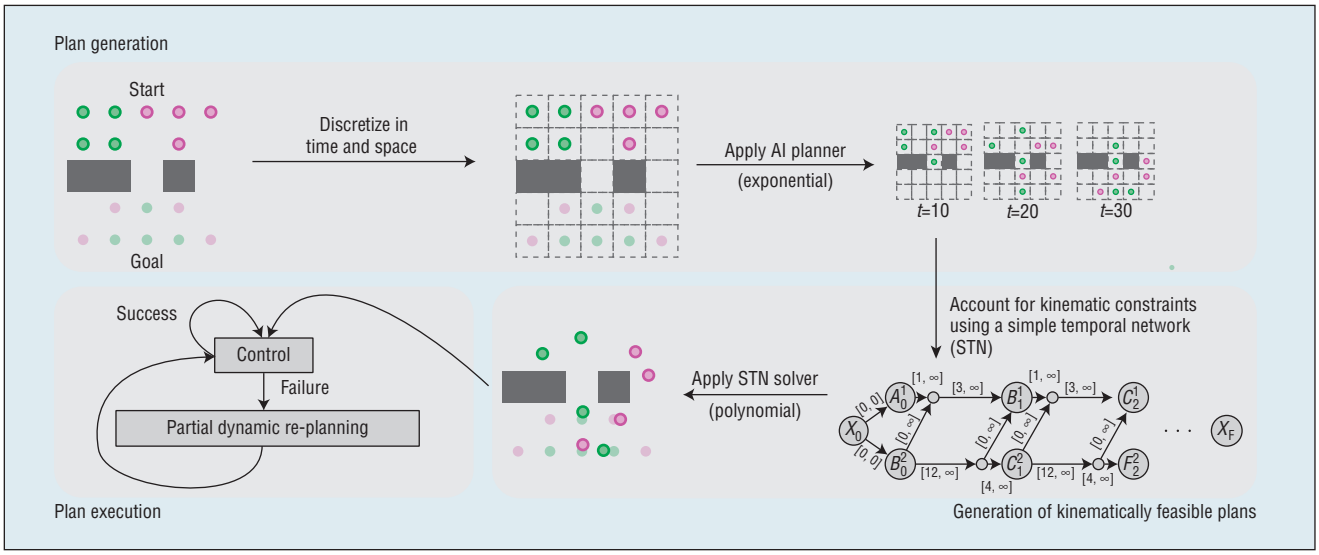

Figure 1. Architecture of our hierarchical framework. First, we discretize the continuous multiagent pathfinding (MAPF) problem in time and space and use an AI planner to solve the resulting NP-hard problem. Then, we solve the simple temporal network (STN) for the resulting discrete MAPF plan in polynomial time to generate a kinematically feasible plan that provides guaranteed safety distances among robots under the assumption of perfect plan execution. Control uses specialized robot controllers during plan execution to exploit the slack in the plan to try to absorb any imperfect plan execution. If this doesn't work, partial dynamic re-planning re-solves a suitably modified STN in polynomial time. Only if this doesn't work either, partial dynamic re-planning re-solves a suitably modified MAPF problem more slowly.

and compiles them into a temporal plan graph (TPG) that encodes their partial temporal order. Finally, it annotates the TPG with quantitative information that captures some kinematic constraints associated with executing the actions. This converts the TPG into a simple temporal network (STN) from which a plan (including its execution schedule) can be generated in polynomial time that takes some of the kinematic constraints of the robots into account (for simplicity called a kinematically feasible plan here), namely, by exploiting the slack in the STN. The term "slack" refers to the existence of an entire class of plans consistent with the STN, allowing us to narrow down the class of plans to a single kinematically feasible plan. A similar notion of slack is well studied for STNs in general in the temporal-reasoning community.

The plan-execution phase also exploits the slack in the STN, namely for absorbing any imperfect plan execution to avoid time-consuming re-planning in many cases.

We use a multi-robot path-planning problem as a case study to present the key ideas behind our framework and demonstrate it both in simulation and on real robots.

Plan Generation

We use a state-of-the-art AI planner for reasoning about the causal interactions among actions. In the multiagent pathfinding (MAPF) problem, which is well studied in AI, robotics and theoretical computer science, the causal interactions are studied oblivious to the kinematic constraints of the robots. We're given a graph with vertices (that correspond to locations) and unit-length edges between them. Each edge connects two different vertices and corresponds to a narrow passageway between the corresponding locations in which robots cannot pass each other. Given a set of robots with assigned start vertices and targets (goal vertices), we have to find collision-free paths for the robots from their start vertices to their targets (where the robots remain) that minimize the makespan (or some other measure of the cost, such as the flowtime). At each timestep, a robot can either wait at its current vertex or traverse a single edge. Two robots collide when they're at the same vertex at the same timestep or traverse the same edge at the same timestep in opposite directions.

The MAPF problem is NP-hard to solve optimally or bounded suboptimally since it's NP-hard to approximate within any constant factor less than 4/3, called the suboptimality guarantee.[6] Yet, powerful MAPF planners have recently been developed in the AI community that can find (optimal or bounded suboptimal) collision-free plans for hundreds of

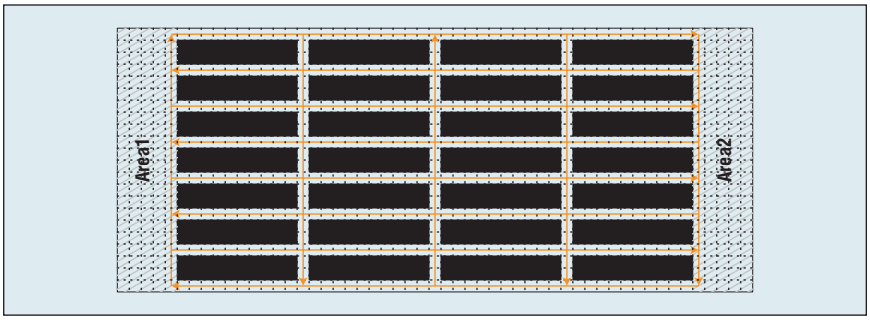

Figure 2. Environment of a simulated automated warehouse system where robots need to swap sides from Area1 to Area2 and vice versa. The red arrows show user-suggested edges to traverse (called highways). Highways make the resulting plan more predictable and speed up planning.

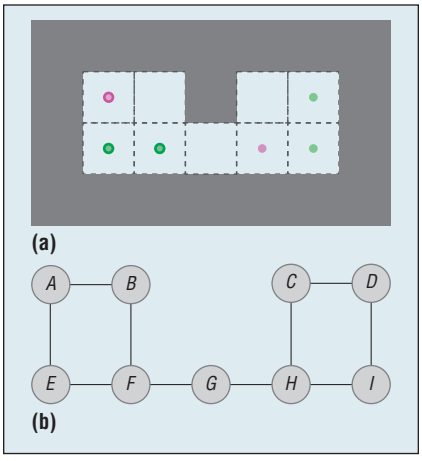

Figure 3. Target assignment and path-finding. (a) TAPF instance with two teams: Team 1 (in pink) and Team 2 (in green). The circles on the left are robots. The circles in light colors on the right are targets given to the team of the same color. (b) Graph representation of the TAPF instance. Team 1 consists of a single robot with start vertex A and target H. Team 2 consists of two robots with start vertices E and F, respectively, and targets D and I.

robots at the cost of ignoring the kinematic constraints of real robots.[3–5,7] We report on two of our own contributions to such MAPF planners below.

Consistency and Predictability of Motion

For many real-world multirobot systems, the consistency and predictability of robot motions is important (especially in work spaces shared by humans and robots), which isn't taken into account by existing MAPF planners. We've shown that we can adapt AI planners, such as the bounded suboptimal MAPF planner Enhanced Conflict-Based Search (ECBS),[8] to generate paths that include edges from a user-provided set of edges (called highways) whenever the suboptimality guarantee allows it, which makes the robot motions more consistent and thus predictable. The highways can be an arbitrary set of edges and thus be chosen to suit humans. For example, highways need to be created only in the part of the environment where the consistency of robot motions is important. Furthermore, highways provide suggestions, not restrictions. Poorly chosen highways don't make a MAPF instance unsolvable, although they can make the MAPF planner less efficient. On the other hand, well-chosen highways typically speed up the MAPF planner because they avoid front-to-front collisions between robots that travel in opposite directions.

Our version of the ECBS planner with highways either inflates the heuristic values or the edge costs nonuniformly in a way that encourages path finding to return paths that include the edges of the highways.[9] For example, we can place highways in an automated warehouse system along the narrow passageways between the storage locations as shown by the red arrows in Figure 2. We've also developed an approach for learning good highways automatically[4] that's based on the insight that solving the MAPF problem optimally is NP-hard but computing the minimum-cost paths for all robots independently is fast, by employing a graphical model that uses the information in these paths heuristically to generate good highways automatically.

Target Assignment and Path Finding

For the MAPF problem, the assignments of robots to targets are predetermined, and robots are thus not exchangeable. In practice, however, the assignments of robots to targets are often not predetermined. For example, consider two robots in an automated warehouse system that have to deliver two inventory pods to the same packing station. It doesn't matter which robot arrives first at the packing station, and their places in the arrival queue of the packing station are thus not predetermined. We therefore define the combined target assignment and path-finding (TAPF) problem for teams of robots as a combination of the target-assignment and path-finding problems. The TAPF problem is a generalization of the MAPF problem where the robots are partitioned into equivalence classes (called teams). Each team is given the same number of unique targets as there are robots in the team. We have to assign the robots to the targets and find collision-free paths for the robots from their start vertices to their targets in a way such that each robot moves to exactly one target given to its team, all targets are visited, and the makespan is

minimized. Any robot in a team can be assigned to any target of the team, and robots in the same team are thus exchangeable. However, robots in different teams aren't exchangeable. Figure 3 shows a TAPF instance with two teams of robots.

The TAPF problem is NP-hard to solve optimally or bounded suboptimally for more than one team.[6] TAPF planners have two advantages over MAPF planners: optimal TAPF plans often have smaller makespans than optimal MAPF plans for TAPF instances since optimal TAPF plans optimize the assignments of robots to targets, and state-of-the-art TAPF planners compute collision-free paths for all robots on a team very fast and thus often scale to a larger number of robots than state-of-the-art MAPF planners. We developed the optimal TAPF planner Conflict-Based Min-Cost Flow (CBM),[5] which combines heuristic search-based MAPF planners[10] and flow-based MAPF planners[11] and scales to TAPF instances with dozens of teams and hundreds of robots.

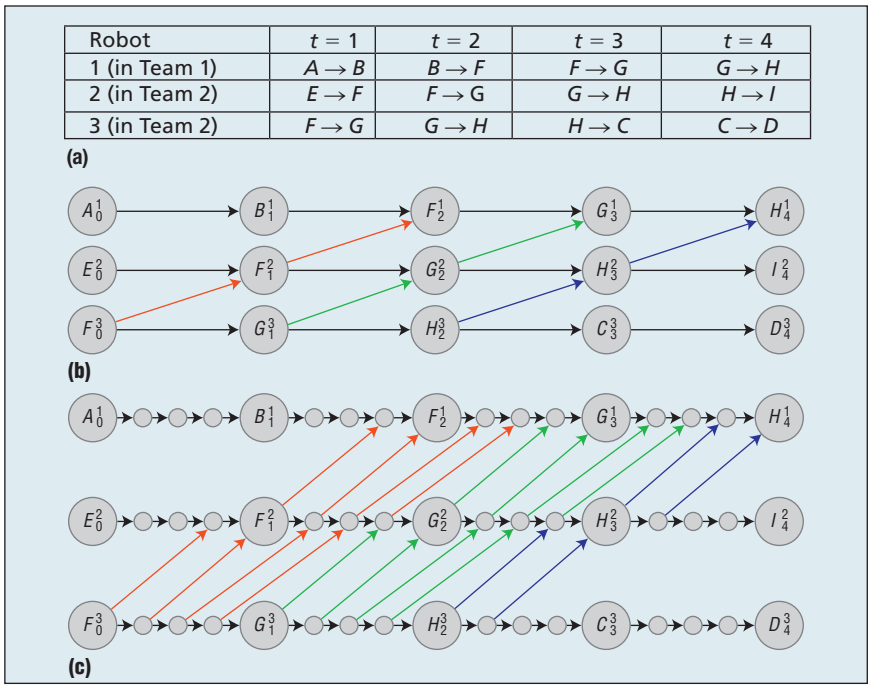

Figure 4. (a) TAPF plan produced by the optimal TAPF planner CBM for the TAPF instance in Figure 3. (b) Temporal plan graph (TPG) for the TAPF plan. Each node in the TPG represents the event "robot *j* arrives at vertex *l*" at timestep *i*. The arcs indicate temporal precedences between events. (c) Augmented TPG.

Generation of Kinematically Feasible Plans

MAPF/TAPF planners generate plans using idealized models that don't take the kinematic constraints of actual robots into account. For example, they gain efficiency by not taking velocity constraints into account and instead assuming that all robots always move with the same nominal speed in perfect synchronization with each other. However, it's communication-intensive for robots to remain perfectly synchronized as they follow their paths, and their individual progress will thus typically deviate from the plan. Two robots can collide, for example, if one robot already moves at large speed while another robot accelerates from standstill. Slowing down all robots results in large makespans and is thus undesirable.

We thus developed MAPF-POST, a novel approach that makes use of an STN[12] to postprocess a MAPF/TAPF plan in polynomial time and create a kinematically feasible plan.[13,14] MAPF-POST utilizes information about the edge lengths and maximum translational and rotational velocities of the robots to translate the plan into a temporal plan graph (TPG) and augment the TPG with additional nodes that guarantee safety distances among the robots. Figure 4 shows an example. Then, it translates the augmented TPG into an STN by associating bounds with arcs in the augmented TPG that express non-uniform edge lengths or velocity limits (due to kinematic constraints of the robots or safety concerns). It then obtains an execution schedule from the STN by minimizing the makespan or maximizing the safety distance via graph-based optimization or linear programming. The execution schedule specifies when each robot should arrive in each location of the plan (called arrival times). The kinematically feasible plan is a list of locations (that specify way-points for the robots) with their associated arrival times.[14]

Plan Execution

The robots will likely not be able to follow the execution schedule perfectly, resulting in plan deviations. For example, our planner takes velocity constraints into account but doesn't capture higher-order kinematic constraints, such as acceleration limits. Also, robots might be forced to slow down due to unforeseen exogenous events, such as floors becoming slippery due to water spills. In such cases, the plan has to be adjusted quickly during plan execution.

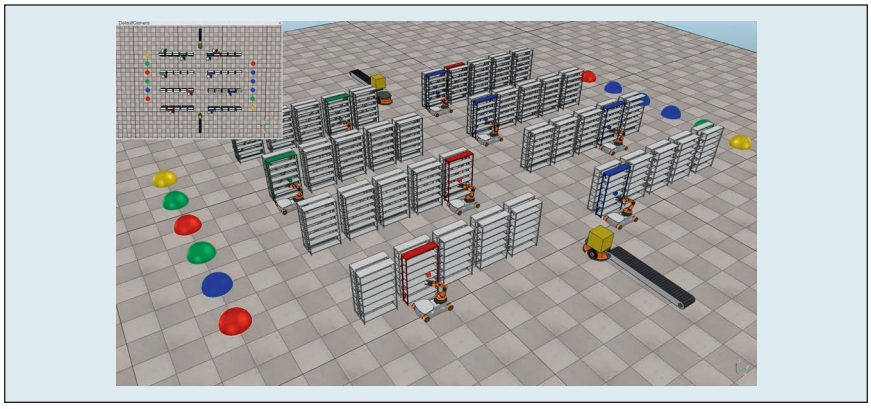

Figure 5. Simulated automated warehouse environment. The in-set in the top-left corner shows an overhead view. The robots are at different pickup locations and need to deliver the color-coded boxes to the left and right side, respectively.

Frequent re-planning could address these plan deviations, but it's time-consuming (and thus impractical) due to the NP-hardness of the MAPF/TAPF problem. Instead, control uses specialized robot controllers to exploit the slack in the plan to try to absorb any imperfect plan execution. If this doesn't work, partial dynamic re-planning re-solves a suitably modified STN in polynomial time. Only if this doesn't work either, partial dynamic re-planning re-solves a suitably modified MAPF problem more slowly.

Control

A robot controller takes the current state and goal as input and computes the motor output. For example, the state of a differential drive robot can be its position and heading, and the motor output is the velocities of the two wheels. The goal is the execution schedule, assuming a constant movement velocity between two consecutive way-points (called the constant velocity assumption). Robots can't execute such motion directly because they can't change their velocities instantaneously and might not be able to move sideways. The actual safety distance during plan execution is thus often smaller than the one predicted during planning, which is why we recommend to maximize the safety distance during planning rather than the makespan. We use robot controllers that try to minimize the effect of the above limitations. For differential drive robots, we use the fact that turning in place is often much faster than moving forward. Furthermore, we adjust the robot velocities dynamically based on the time-to-go to reach the next way-point. It's especially important to monitor progress toward locations that correspond to nodes whose slacks are small. Robots could be alerted of the importance of reaching these bottleneck locations in a timely manner. Similar control techniques can be used for other robots as well, such as drones, as long as no aggressive maneuvers are required.

Partial Dynamic Re-planning

If control is insufficient to achieve the arrival times given in the execution schedule, we adjust the arrival times by resolving a suitably modified STN, resulting in a new execution schedule. Only if this doesn't work either, we re-solve a suitably modified MAPF problem, resulting in a new kinematically feasible plan. If probabilistic models of delays and other deviations from the nominal velocities are available, they could be used to determine the probabilities that each location will be reached in a certain time interval and trigger re-planning only if one or more of these probabilities become small[13].

Experiments

We implemented our approach in C++ using the boost library for advanced data structures, such as graphs. Experiments can be executed on three abstraction levels, namely, agent simulation, robot simulation, and real robots:

- The agent simulation uses the constant velocity assumption and is fast. It can be used to verify the code and create useful statistics for the runtime, minimum distance between any two robots, and average time until any robot reaches its target, among others. It can also be used for scalability experiments with hundreds of robots in cluttered environments.
- The robot simulation adds realism because it uses a physics engine (instead of the constant velocity assumption) and realistic robot controllers for the simulated robots to follow the execution schedule. We use V-REP as robot simulation for differential drive robots, robots with omnidirectional wheels, flying robots, and spiderlike robots.
- Real robots are the ultimate testbed. We use a team of eight iRobot Create2 differential drive robots.[14]

In the following, we discuss two example use cases on a 2.1 GHz Intel Core i7-4600U laptop computer with 12 Gbytes RAM. Each example is solved within 10 seconds of computation time and also shown in our supplemental video at http://idm-lab.org/project-p.html.

Automated Warehouse

In the automated warehouse use case, we model two robot teams. The first team consists of 10 KUKA youBot robots, which are robots with omnidirectional wheels capable of carrying (only) small boxes. The second team consists of two Pioneer P3DX robots, which are differential-drive robots capable of carrying (only) large boxes. The robots have to pick up small and large color-coded boxes and bring them to a target of the same color. We split the task into two parts.

First, each robot has to move to an appropriately sized box and pick it up. Second, it has to move to a target of the same color. The first part is a TAPF instance with two teams, one for each robot type. The second part is a TAPF instance with four teams, one for each color.

We use the robot simulation on a 2D grid. Figure 5 shows a screenshot after the first part has already been executed and the robots are at different pickup locations. The KUKA robots use their grippers to pick small boxes from shelves while the Pioneer robots receive the large boxes from a conveyor belt. The robots then need to move to the targets on the left and right side of the warehouse, respectively.

Formation Changes

Formations are useful for convoys, surveillance operations, and artistic shows. The task of switching from one formation to another, perhaps in a cluttered environment, is a TAPF problem. In the formation-change use case, we model a team of 32 identical quadcopters that start in a building with five open doors. The robots have to spell the letters U – S – C outside the building, which is a special TAPF instance where all robots are exchangeable (also called an anonymous MAPF instance[11]). We use the robot simulation on a 3D grid. Figure 6 shows a screen-shot of the goal formation.

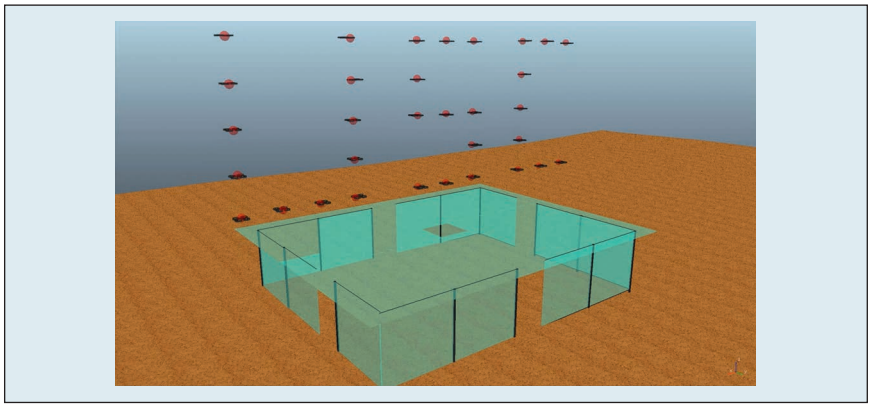

Figure 6. Simulated formation-change environment. The 32 quadcopters start inside the glass building at the bottom of the picture and need to coordinate the usage of the four exit doors in order to create the depicted goal formation spelling the letters U – S – C.

In the future, we plan to apply our framework to more realistic applications such as planning for real automated warehouses, where orders need to be fulfilled continuously. For more information on our research, see http://idm-lab.org/project-p.html. ■

Acknowledgments

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 the National Science Foundation under grant numbers 1409987 and 1319966. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the US government.

References

1. P.R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," *AI Magazine*, vol. 29, no. 1, 2008, pp. 9–20.
2. R. Morris et al., "Planning, Scheduling and Monitoring for Airport Surface Operations," *Proc. AAAI-16 Workshop Planning for Hybrid Systems*, 2016, pp. 608–614.
3. H. Ma, T.K.S. Kumar, and S. Koenig, "Multi-Agent Path Finding with Delay Probabilities," *Proc. AAAI Conf. Artificial Intelligence*, 2017, pp. 3605–3612.
4. L. Cohen et al., "Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding," *Proc. Int'l Joint Conf. Artificial Intelligence*, 2016, pp. 3067–3074.
5. H. Ma and S. Koenig, "Optimal Target Assignment and Path Finding for Teams of Agents," *Proc. Int'l Conf. Autonomous Agents and Multiagent Systems*, 2016, pp. 1144–1152.
6. H. Ma et al., "Multiagent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem," *Proc. AAAI Conf. Artificial Intelligence*, 2016, pp. 3166–3173.
7. H. Ma et al., "Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks," *Proc. Int'l Conf. Autonomous Agents and Multiagent Systems*, 2017, pp. 837–845.

THE AUTHORS

Hang Ma is a PhD student in the Department of Computer Science at the University of Southern California. Contact him at hangma@usc.edu.

Wolfgang Hönig is a PhD student in the Department of Computer Science at the University of Southern California. Contact him at whoenig@usc.edu.

Liron Cohen is a PhD student in the Department of Computer Science at the University of Southern California. Contact him at lironcoh@usc.edu.

Tansel Uras is a PhD student in the Department of Computer Science at the University of Southern California. Contact him at turas@usc.edu.

Hong Xu is a PhD student in the Department of Physics and Astronomy at the University of Southern California. Contact him at hongx@usc.edu.

T.K. Satish Kumar leads the Collaboratory for Algorithmic Techniques and Artificial Intelligence at USC's Information Sciences Institute. Contact him at tkskwork@gmail.com.

Nora Ayanian is an assistant professor in the Department of Computer Science at the University of Southern California. Contact her at ayanian@usc.edu.

Sven Koenig is a professor in the Department of Computer Science at the University of Southern California. Contact him at skoenig@usc.edu.

8. M. Barer et al., "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem," *Proc. Annual Symp. Combinatorial Search*, 2014, pp. 19–27.
9. L. Cohen, T. Uras, and S. Koenig, "Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding," *Proc. Annual Symp. Combinatorial Search*, 2015, pp. 2–8.
10. G. Sharon et al., "Conflict-Based Search for Optimal Multi-Agent Pathfinding," *Artificial Intelligence*, vol. 219, 2015, pp. 40–66.
11. J. Yu and S. M. LaValle, "Multi-Agent Path Planning and Network Flow," *Algorithmic Foundations of Robotics X, Springer Tracts in Advanced Robotics*, vol. 86, 2013, pp. 157–173.
12. R. Dechter, I. Meiri, and J. Pearl, "Temporal Constraint Networks," *Artificial Intelligence*, vol. 49, nos. 1–3, 1991, pp. 61–95.
13. W. Hönig et al., "Multi-Agent Path Finding with Kinematic Constraints," *Proc. Int'l Conf. Automated Planning and Scheduling*, 2016, pp. 477–485.
14. W. Hönig et al., "Formation Change for Robot Groups in Occluded Environments," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, 2016, pp. 4836–4842.