Proceedings of the 4th ICAPS Workshop on Hierarchical Planning (HPlan 2021)

A Hierarchical Approach to Multi-Agent Path Finding

Han Zhang, Mingze Yao, Ziang Liu, Jiaoyang Li, Lucas Terr, Shao-Hung Chan, T. K. Satish Kumar, Sven Koenig
University of Southern California
{zhan645, mingzeya, ziangliu, jiaoyanl, terr, shaohung}@usc.edu, tkskwork@gmail.com, skoenig@usc.edu

Abstract

The Multi-Agent Path Finding (MAPF) problem arises in many real-world applications, ranging from automated warehousing to multi-drone delivery. Solving the MAPF problem optimally is NP-hard, and existing optimal and bounded-suboptimal MAPF solvers thus usually do not scale to large MAPF instances. Greedy MAPF solvers scale to large MAPF instances, but their solution qualities are often bad. In this paper, we therefore propose a novel MAPF solver, Hierarchical Multi-Agent Path Planner (HMAPP), which creates a spatial hierarchy by partitioning the environment into multiple regions and decomposes a MAPF instance into smaller MAPF sub-instances for each region. For each sub-instance, it uses a bounded-suboptimal MAPF solver to solve it with good solution quality. Our experimental results show that HMAPP solves as large MAPF instances as greedy MAPF solvers while achieving better solution qualities on various maps.

Introduction

The Multi-Agent Path Finding (MAPF) problem arises in many real-world applications, including automated warehousing (Wurman, D'Andrea, and Mountz 2008; Li et al. 2020) and multi-drone delivery (Choudhury et al. 2020). In the MAPF problem, each agent is required to move from a start vertex to a goal vertex on an undirected graph while avoiding conflicts with other agents. A conflict happens when two agents stay at the same vertex or traverse the same edge in opposite directions at the same time.

Two common objectives for the MAPF problem are minimizing the sum of the path costs and minimizing the makespan. Solving the MAPF problem optimally for either objective is NP-hard (Yu and LaValle 2013; Ma et al. 2016). Thus, existing optimal and bounded-suboptimal MAPF solvers (Sharon et al. 2015; Barer et al. 2014) usually do not scale to large MAPF instances. Greedy MAPF solvers (Silver 2005) are able to scale to large MAPF instances, but their solution qualities are often bad.

Although planning can find MAPF solutions of good quality for small MAPF instances, planning in small steps from one vertex to another has the disadvantage that its runtime can dramatically increase with the number of agents and the size of the environment. In this paper, we approach the MAPF problem from a rarely-pursued spatial-hierarchy perspective. We propose a novel MAPF solver, Hierarchical Multi-Agent Path Planner (HMAPP). In HMAPP, a high-level planner generates a high-level plan for each agent that moves the agent from one region to another, and each regional planner subsequently refines the high-level plan to a low-level path for the agent. Therefore, regional planners can use existing MAPF techniques to find solutions with good qualities while the total runtime of HMAPP is still reasonable for large MAPF instances.

Our experimental results show that HMAPP solves as large MAPF instances as greedy MAPF solvers while achieving better solution qualities on various maps. The solutions of HMAPP have makespans for large MAPF instances that are about 50% smaller than the ones of the spatial-hierarchical MAPF solver Ros-dmapf (Pianpak et al. 2019).

Related Work

Spatial hierarchies have been used for path planning (Botea, Müller, and Schaeffer 2004; Pelechano and Fuentes 2016) by partitioning a map into several regions, precomputing and caching the optimal sub-paths that connect adjacent regions and abstracting these sub-paths to edges of a smaller abstract graph, that is then searched. These approaches do not directly apply to MAPF since the cached sub-paths do not take conflicts between agents into account and are thus difficult to reuse for MAPF.

Hierarchies have also been used for multi-agent motion planning (Kapadia et al. 2013; Ma et al. 2017), but these approaches do not use spatial hierarchies but rather planning hierarchies that plan on different abstraction levels, such as path and motion planning. HMAPP can be used for path planning in such approaches.

Spatial hierarchies have not yet been used extensively for MAPF. The Spatially Distributed Multi-Agent Planner (SDP) (Wilt and Botea 2014) partitions a map into high- and low-contention regions and uses different MAPF solvers for regions of different types. Unlike HMAPP, SDP does not partition the map into several regions in the absence of high-contention regions and cannot solve MAPF instances unless all start or goal vertices are in low-contention regions. Ros-dmapf (Pianpak et al. 2019), like HMAPP, partitions a map into several regions. Unlike HMAPP, Ros-dmapf uses answer set programming for the regional planners and has to synchronize the execution of the high-level plans of

all agents, causing agents that reach their next regions earlier than other agents to wait unnecessarily for those other agents, which impacts the solution quality negatively.

Preliminaries

In this section, we provide background material on MAPF, the optimal MAPF solver Conflict-Based Search (CBS) and the bounded-suboptimal MAPF solver Enhanced Conflict-Based Search (ECBS).

MAPF

The MAPF problem is defined by an undirected graph $G = (V, E)$ and a set of m agents $\{a_1 \dots a_m\}$. Each agent has a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. In each timestep, an agent either moves to an adjacent vertex or waits at its current vertex. Both move and wait actions have unit cost unless the agent terminally waits at its goal vertex, which has zero cost. A *path* of an agent is a sequence of move and wait actions from its start vertex to its goal vertex. A *sub-path* of an agent is a sequence of actions from one vertex at a specific timestep to another vertex at a specific timestep. The *path cost* of a path is the accumulated cost of all actions in this path. A *vertex conflict* happens when two agents stay at the same vertex simultaneously, and an *edge conflict* happens when two agents traverse the same edge in opposite directions simultaneously. A *solution* is a set of conflict-free paths of all agents. The *Sum of path Costs* (SoC) is the sum of the path costs of the paths of all agents, and the *makespan* is the maximum path cost of the paths of all agents. In this paper, we consider only graphs that are four-neighbor grids (Stern et al. 2019). However, HMAPP can be applied to any graph as long as a graph-partitioning approach is provided for it.

CBS and ECBS

CBS (Sharon et al. 2015) is an optimal two-level MAPF solver. On the high level, CBS maintains a *Constraint Tree* (CT). Each CT node contains a set of constraints and a set of paths, one for each agent, that satisfies all these constraints. The cost of a CT node is the SoC or makespan of all these paths, depending on the objective of the MAPF problem. On the low level, for each CT node, CBS finds a path for each agent that has the smallest path cost while satisfying all constraints of the CT node (but might conflict with the other paths). When expanding a CT node, CBS returns a solution if its paths are conflict-free. Otherwise, CBS picks a conflict, splits the CT node into two child CT nodes and adds a constraint to each child CT node to prohibit either one or the other of the two conflicting agents from using the conflicting vertex or edge at the conflicting timestep. On the high level, CBS expands nodes in a best-first order. Therefore, the paths of the first expanded CT node with conflict-free paths form an optimal solution.

ECBS(w) (Barer et al. 2014) is a bounded-suboptimal MAPF solver based on CBS. Given suboptimality factor w, ECBS(w) finds a w-suboptimal solution. The high- and low-level search algorithms of ECBS are focal search (Pearl and Kim 1982) instead of best-first search. Unlike best-first search, focal search maintains a FOCAL list, which is a subset of the OPEN list of search nodes, and expands nodes from the FOCAL list based on a user-provided tie-breaking criterion. On the low-level, ECBS uses focal search to find paths that have fewer conflicts with the paths of other agents. On the high-level, ECBS uses focal search to expand CT nodes that more likely lead to a conflict-free bounded-suboptimal solution.

Algorithm 1: HMAPP.

```
   input: A MAPF instance.
1  initialize();
2  find_HL_plan();
3  T ← 0;
4  foreach region r ∈ R do
5  |  P_r.plan_initial_path();
6  end
7  while paths for all agents to their goal vertices have
     not yet been found do
8  |  T ← next timestep when an agent is ready to
   |    enter its next region;
9  |  foreach region r ∈ R do
10 |  |  A' ← agents that are ready to enter r at
   |  |    timestep T;
11 |  |  if A' is not empty then
12 |  |  |  P_r.replan(A');
13 |  |  end
14 |  end
15 |  foreach region r ∈ R do
16 |  |  if an agent is delayed to exit r at timestep T
   |  |    then
17 |  |  |  P_r.replan(∅);
18 |  |  |  if P_r.replan failed to find a solution then
19 |  |  |  |  return failure;
20 |  |  |  end
21 |  |  end
22 |  end
23 end
24 return extract_solution();
```

HMAPP

Algorithm 1 shows the pseudo-code of HMAPP. HMAPP first partitions the vertices into regions. Let R denote the set of all regions. For each pair of adjacent regions, HMAPP finds pairs of adjacent vertices (one from each region), called *boundary pairs*, and uses them to transfer agents between regions. To simplify the interaction between regions, agents are allowed to travel in only one direction through each boundary pair. A *high-level planner* generates a *high-level plan* for each agent (Line 2), which specifies the sequence of regions that the agent should visit to reach its goal vertex. When we describe HMAPP, we assume that the high-level plan of each agent does not include each region more than once so that each agent has at most one sub-path in each region. However, this assumption is only for the ease of pre-

sentation. HMAPP allows the high-level plan of an agent to include a region multiple times and maintains one sub-path for each visit of the agent to the region.

For an agent a_i that moves from region r to its next region r', the *regional planner* $\mathcal{P}_r$ plans a sub-path for a_i to a boundary vertex v that is part of a boundary pair $\langle v, v' \rangle$ to region r'. v (v') is the determined *exit (entry) vertex* of a_i from r (to r'). The *exit (entry) timestep* of a_i from r (to r') is the last timestep that a_i is in r. $\mathcal{P}_r$ initially assumes that a_i immediately exits r once it has followed its sub-path in r. However, the actual exit timestep is determined by the regional planner $\mathcal{P}_{r'}$ when $\mathcal{P}_{r'}$ determines the entry timestep and the sub-path for a_i in r', which must not be smaller than the timestep when a_i has followed its sub-path in r. Once determined, the entry and exit timesteps and the entry and exit vertices of agents can no longer be changed.

In the beginning of Algorithm 1, for each region r, $\mathcal{P}_r$ plans a set of conflict-free sub-paths for all agents in the region (Lines 4-6). We say that agent a_i is *ready to enter* (*exit*) region r' (r) at a timestep t iff (1) a_i has followed its sub-path in r at timestep t and (2) the exit timestep of a_i from r has not been determined yet. Inside the while loop (Lines 7-23), T is updated to the earliest timestep when an agent is ready to enter its next region. HMAPP iterates over each region r and invokes $\mathcal{P}_r$ to determine the entry timesteps for agents that are ready to enter r at timestep T (Lines 9-14). Let a_i be such an agent. We say that a_i is *delayed to exit* its region iff its determined entry timestep is larger than T. $\mathcal{P}_r$ might have to replan the sub-paths of all agents in r if such a delay happens. HMAPP iterates over each region r that has a delayed-to-exit agent and invokes $\mathcal{P}_r$ to replan the sub-paths of all agents (Lines 15-22). Except for the initial planning, each regional planner plans at most twice for each value of T, once to take the ready-to-enter agents into account and once to take the delayed-to-exit agents into account. When replanning, each regional planner is allowed to modify the entire sub-paths of its agents in the region (even the parts before timestep T) as long as they obey the determined entry and exit timesteps and the determined entry and exit vertices. HMAPP repeats this procedure until it has found paths for all agents to their goal vertices. Finally, HMAPP appends the sub-paths in different regions and returns the obtained paths as the solution (Line 24).

The resulting paths are conflict-free because (1) the sub-paths inside each region are conflict-free and (2) no edge conflict happens when an agent exits a region since the movements within each boundary pair are in one direction only. However, HMAPP is not a complete MAPF solver since the sub-instances for the regions can be unsolvable even if a solution for the MAPF instance exists. Limiting the number of agents in each region may make HMAPP complete, which we leave for future work.

HMAPP is a general algorithmic framework that can use different approaches for graph partitioning, high-level planning and regional planning. In the following sections, we describe how each of these components is implemented currently.

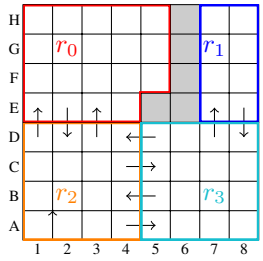


Figure 1: Shows an example of partitioning an 8×8 grid into four regions r_i (for $i = 0, \ldots 3$), each with a different color. Shaded areas are obstacles. Arrows between adjacent vertices indicate boundary pairs and their movement directions.

Graph Partitioning and High-Level Planning

In this paper, we consider only MAPF instances on four-neighbor grids (Stern et al. 2019), and HMAPP partitions the grids into rectangular regions of similar sizes, determined by parameters num_row and num_col, which specify the numbers of regions in the vertical and horizontal directions, respectively. If a region is not connected, then HMAPP partitions it further. HMAPP then iterates over each pair of adjacent regions, collects all pairs of adjacent vertices, one from each each region, that have not yet been used in any boundary pair and adds them to the set of boundary pairs. It assigns alternating directions to boundary pairs so that there are enough boundary pairs for agents to move from one region to another.

A naive partitioning approach can result in a bad partition and poor scalability of HMAPP on grids with obstacles. If HMAPP partitions the grid in Figure 1 into $2 \times 2 = 4$ regions, each of size 4×4, then it further partitions the top-right region into two regions since it is not connected. One of the resulting regions consists of cells $F5, G5$ and $H5$ and is corridor-shaped. For a corridor-shaped region, a solution might not exist for even only two agents. To improve the quality of the resulting partition, (1) if there is a corridor-shaped region, then HMAPP randomly picks one of its adjacent regions (if there is one) and merges these two regions to eliminate regions that contain only narrow corridors, and (2) if there is a pair of adjacent regions that share fewer than two boundary pairs, then HMAPP merges them to ensure that an agent can always reach an adjacent region from its current region. HMAPP repeats this procedure until no such cases exist any longer. Figure 1 shows the region r_0 obtained after merging the top-left region with the corridor-shaped region.

The high-level planner of HMAPP is responsible for finding high-level plans for all agents. For each agent, HMAPP randomly picks one of its shortest paths from its start vertex to its goal vertex that moves from region to region only at boundary pairs in their directions. HMAPP then generates the high-level plan that corresponds to the sequence of regions visited by this path. Due to Partitioning Rule (2) above, there always exists such a path for each agent.

Algorithm 2: *replan*() for regional planners.

input: Regional planner $\mathcal{P}_r$ and a set of agents A', which is the set of agents that are ready to enter r.

1 $\mathcal{P}_r.P \leftarrow ECBS(\mathcal{P}_r.A \cup A', \mathcal{P}_r.C)$;
2 **if** *ECBS failed to find a solution* **then**
3 **return** *failure*;
4 **end**
5 **foreach** *agent* $a_i \in A'$ **do**
6 $t \leftarrow$ entry timestep of a_i according to $\mathcal{P}_r.P$;
7 $\langle v', v \rangle \leftarrow$ the boundary pair a_i uses to enter r;
8 $r' \leftarrow$ the region v' is part of;
9 $\mathcal{P}_r.C.add(entry\langle a_i, v, t \rangle)$;
10 $\mathcal{P}_{r'}.C.add(exit\langle a_i, v', t \rangle)$;
11 **end**
12 $\mathcal{P}_r.A \leftarrow \mathcal{P}_r.A \cup A'$;
13 **return** *success*;

Regional Planning

The regional planners of HMAPP find sub-paths for the agents inside their regions. The regional planner $\mathcal{P}_r$ for region r maintains multiple data structures to keep track of the agents and their sub-paths. $\mathcal{P}_r.A$ is the set of agents that already have determined entry timesteps to r. $\mathcal{P}_r.C$ is the set of constraints of the agents in $\mathcal{P}_r.A$ that keep track of their entry timesteps to r and exit timesteps from r and the associated entry and exit vertices, respectively. Two types of constraints can be added to $\mathcal{P}_r.C$. The first one is an *entry-vertex-timestep constraint* $entry\langle a_i, v, t \rangle$, which enforces that agent a_i enters r from entry vertex v at entry timestep t. The second one is an *exit-vertex-timestep constraint* $exit\langle a_i, v, t \rangle$, which enforces that agent a exits region r from exit vertex v at exit timestep t. $\mathcal{P}_r.P$ is a set of conflict-free sub-paths of the agents in $\mathcal{P}_r.A$ that satisfy the constraints in $\mathcal{P}_r.C$.

HMAPP uses ECBS to solve the regional planning problems. Agent a_i is a *local agent* of region r if a_i does not have a next region in its high-level plan when it is in r; otherwise, agent a_i is a *migrating agent* of region r. For each migrating agent a_i of region r, let p_i denote the sub-path of a_i in r and $\langle v, v' \rangle$ denote the boundary pair to the next region of a_i that p_i leads to. The cost of a_i for $\mathcal{P}_r$ is $cost(p_i) + h(v') + 1$, where $h(v')$ is an admissible heuristic function for the distance from v' to g_i (if all other agents are ignored) and $cost(p_i)$ is the path cost of p_i. For each local agent a_i of r, the cost of a_i for $\mathcal{P}_r$ is the path cost of p_i.

On Line 12 of Algorithm 1, HMAPP invokes Algorithm 2 to plan the entry timestep and sub-path of each agent a_i in A' that is ready to enter region r from region r' via boundary pair $\langle v', v \rangle$ at timestep T and adds both an entry-vertex-timestep constraint to $\mathcal{P}_r.C$ so that a_i must enter region r at v at timestep t (Line 9 of Algorithm 2) and an exit-vertex-timestep constraint to $\mathcal{P}_{r'}.C$ so that a_i must exit from region r' at v' at timestep t (Line 10 of Algorithm 2).

On Line 17 of Algorithm 1, HMAPP invokes Algorithm 2 to replan the sub-paths of the agents in $P_r.A$ to ensure that the newly-added entry-vertex-timestep and exit-vertex-timestep constraints are satisfied. During this procedure, no new constraints are added.

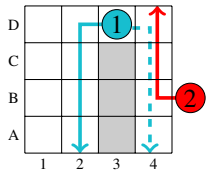


Figure 2: Shows an example where the regional planner replans the sub-path of agent a_1 when agent a_2 is ready to enter the region. Agent a_1 has its start vertex at $D2$, and agent a_2 is ready to enter the region from entry vertex $B4$ at timestep 2.

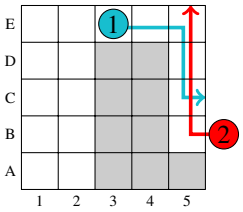


Figure 3: Shows an example where the regional planner is unable to find a solution. Agent a_1 has its start vertex at $E3$, and agent a_2 is ready to enter the region from entry vertex $B5$ at timestep 3.

The regional planning problem is similar to the online MAPF problem (Švancara et al. 2019), where agents move along their paths as T increases. However, agents do not move in the regional planning problem. Therefore, when $\mathcal{P}_r$ replans the sub-paths for the agents in $P_r.A$, it is allowed to modify their entire sub-paths in r.

Example 1. *Figure 2 shows an example where the regional planner replans the sub-path of agent a_1 when a new agent a_2 is ready to enter the region. Initially, only a_1 is in the region, and the regional planner finds a sub-path for a_1 to exit the region at $A4$ at timestep 4, which is shown by the dashed blue line. At timestep 2, a_1 is at $C4$, and a_2 is ready to enter the region. The regional planner then finds new sub-paths for a_1 and a_2. If a_1 follows its original sub-path, then a_2 must be delayed to exit its region. However, there is an alternative sub-path for a_1, which is shown by the solid blue line and has the same path cost as the current sub-path of a_1. The regional planner therefore replans the sub-paths so that a_1 uses the alternative sub-path and a_2 is not delayed to exit its region. In contrast, an online MAPF solver could not change the movement of the agents before timestep 2.*

Handling Regional Planning Failures

On Lines 9-10 of Algorithm 2, new constraints are added that determine the entry and exit timesteps of agents. Since the exit timestep of an agent from its current region is determined by the regional planner of the next region of the agent, this exit timestep may prevent the regional planner of

the current region of the agent from finding a solution.

Example 2. *Figure 3 shows an example where the regional planner is unable to find a solution. Initially, only agent a_1 is in the region, and the regional planner finds a sub-path for a_1 to exit the region at $C5$ at timestep 4, which is shown by the solid blue line. At timestep 3, agent a_2 is ready to enter the region. The regional planner finds the sub-paths for a_1 and a_2 shown in the figure, where neither agent needs to wait since the regional planner assumes that a_1 exits the region immediately when it is at $C5$ and a_2 exits the region immediately when it is at $E5$. At timestep 4, a_1 is at $C5$ and its exit timestep is determined. Assume that it is 10. The regional planner then finds new sub-paths for a_1 and a_2, for example, where a_1 waits in $E3$ for 6 timesteps and a_2 still does not wait since the regional planner assume that a_2 exits the region immediately when it is at $E5$. At timestep 7, agent a_2 is at $E5$ and its exit timestep is determined. Assume that it is 9. The regional planner then tries to find new sub-paths for a_1 and a_2 but fails since a_2 exiting the region at $E5$ at timestep 9 implies that a_1 cannot exit the region before timestep 12.*

When a regional planner is unable to find a solution (Lines 18-19 of Algorithm 1), HMAPP determines the vertices of all agents at timestep T, deletes all constraints and restarts HMAPP at timestep T. In Example 2, the regional planner fails to find a solution at timestep $T = 7$, and HMAPP restarts at timestep 7 with agent a_1 at $E4$ and agent a_2 at $E5$. Assume that the exit timestep of a_2 is again determined to be 9. The regional planner then finds new sub-paths for a_1 and a_2, for example, where a_1 waits at $E4$ (until a_2 exits the region) and then moves to $C5$ and exits the region immediately. Therefore, HMAPP is now able to find a solution.

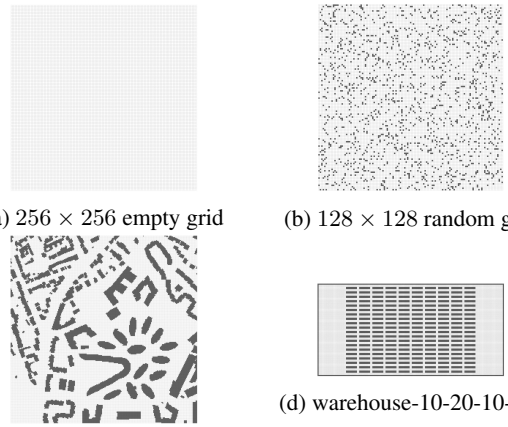
Experimental Evaluation

We compared HMAPP with Ros-dmapf (Pianpak et al. 2019), CA* (Silver 2005), WHCA* (Silver 2005) and ECBS on different grids. CA* is a greedy MAPF solver which plans for one agent at a time. WHCA* is a variant of CA* which interleaves moving agents and planning within a time window of a given length. The objectives for ECBS and the regional planners of HMAPP were all minimizing the SoC, and the suboptimality factors for ECBS and the regional planners of HMAPP were all set to 1.2. The length of the time window of WHCA* was set to 16, which we found to achieve a higher success rate than smaller window sizes while still achieving a small runtime. Except for Ros-dmapf, all MAPF solvers were implemented in C++ and share the same code base as much as possible. We ran all experiments on a laptop with an i7-8850H CPU and 32 GB of memory.

Experiment 1: Comparison with Ros-dmapf. We did not have a working implementation of Ros-dmapf available. Therefore, to compare HMAPP with Ros-dmapf, we ran HMAPP on the 60×60 empty grid MAPF instances used in (Pianpak et al. 2019) and compared the results of HMAPP with the results of Ros-dmapf in the paper. Table 1 shows the average makespans and numbers of moves of HMAPP and Ros-dmapf. The number of moves is the sum of the number of move actions of all agents. HMAPP used both the partition size 10×10 and runtime limit of 100s used in (Pianpak et al. 2019). HMAPP solved all MAPF instances within the time limit. Table 1 shows that the average makespan of the solutions of HMAPP was less than half of the average makespan of the ones of Ros-dmapf for MAPF instances with 576 agents or more.

Agents	Ros-dmapf		HMAPP	
144	149	5,996	99	6,043
288	185	12,934	118	12,487
432	230	21,627	118	19,307
576	264	30,704	116	25,520
720	310	43,740	120	32,951

Table 1: Shows the average makespans and numbers of moves of Ros-dmapf and HMAPP on the 60×60 empty grid for different numbers of agents.



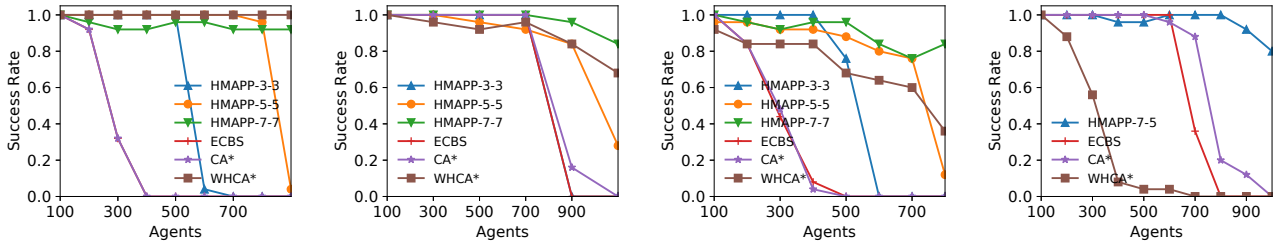
(a) 256×256 empty grid (b) 128×128 random grid

(c) Paris_1_256 (d) warehouse-10-20-10-2-2

Figure 4: Shows the grids of the MAPF instances used in Experiment 2.

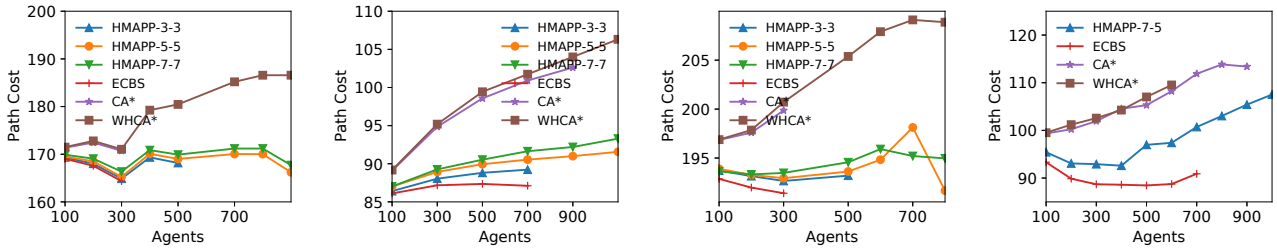
Experiment 2: Comparison with ECBS and greedy MAPF solvers. We evaluated all MAPF solvers on the four grids shown in Figure 4: (a) the 256×256 empty grid, (b) a 128×128 grid with 10% randomly blocked vertices, (c) Paris_1_256 and (d) warehouse-10-20-10-2-2. Grids (c) and (d) are from the MAPF benchmark (Stern et al. 2019). We did not use the empty and random grids from the MAPF benchmark since we were interested in large MAPF instances. For Grids (a)-(c), we used HMAPP with partition sizes $(num\_row, num\_col) = (3, 3)$, $(5, 5)$ and $(7, 7)$. For Grid (d), we used HMAPP with partition size $(7, 5)$ since the runtime of HMAPP turned out to be very sensitive to the size of the regions.

Figure 5 shows that, on most grids, the success rates of ECBS and CA* quickly dropped as the number of agents increased. WHCA* successfully solved all MAPF instances for up to 900 agents on Grid (a). However, on Grids (b)-(d), the success rate of WHCA* was lower than the ones of some versions of HMAPP since WHCA* plans only within a time window of limited length.



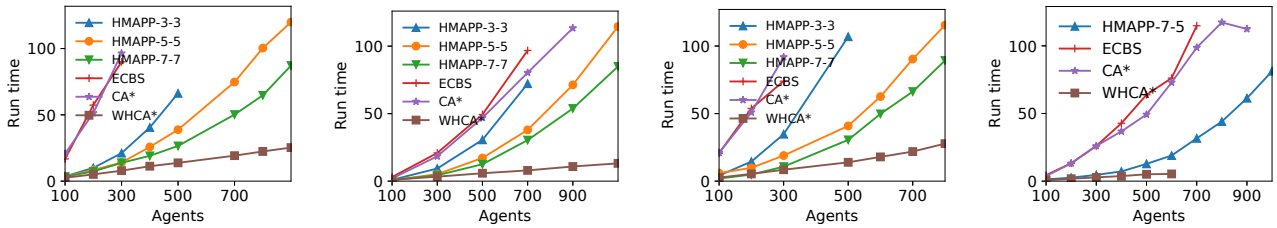
(a) 256 × 256 empty grid (b) 128 × 128 random grid (c) Paris_1_256 (d) warehouse-10-20-10-2-2

Figure 5: Shows the success rates (that is, the percentages of MAPF instances solved within a time limit of two minutes) of various MAPF solvers on each grid for different numbers of agents.



(a) 256 × 256 empty grid (b) 128 × 128 random grid (c) Paris_1_256 (d) warehouse-10-20-10-2-2

Figure 6: Shows the average path costs per agent (averaged over the MAPF instances solved by all MAPF solvers that successfully solved at least one MAPF instance) of various MAPF solvers on each grid for different numbers of agents.



(a) 256 × 256 empty grid (b) 128 × 128 random grid (c) Paris_1_256 (d) warehouse-10-20-10-2-2

Figure 7: Shows the average runtimes (in seconds, averaged over the MAPF instances solved by all MAPF solvers that successfully solved at least one MAPF instance) of various MAPF solvers on each grid for different numbers of agents.

Figure 6 shows that all versions of HMAPP had smaller average path costs on all grids than CA* and WHCA*. Except for Grid (c), the average path costs of HMAPP were more than 10% smaller than those of WHCA* for large numbers of agents. Except for Grid (d), which has many narrow corridors, the average path costs of HMAPP were close to the average path costs of ECBS.

Figure 7 shows that all versions of HMAPP had smaller average runtimes than CA* and ECBS on all grids because HMAPP does not plan paths across the entire grid. However HMAPP had larger average runtimes than WHCA* since WHCA* plans within smaller time windows.

Conclusions and Future Work

In this paper, we have proposed HMAPP which solves the MAPF problem by creating a spatial hierarchy that decomposes a MAPF instance into MAPF sub-instances. Our experimental results show that HMAPP solves as large MAPF instances as greedy MAPF solvers while achieving better solution qualities on various maps.

Our future work includes (1) making HMAPP complete by controlling the number of agents in each region; (2) automatically generating a good partition of the graph of a MAPF instance and (3) developing high-level planning approaches that take potential congestion into account.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779 and 1935712, as well as a gift from Amazon.

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 *International Symposium on Combinatorial Search (SoCS)*, 19–27.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1(1): 7–28.

Choudhury, S.; Solovey, K.; Kochenderfer, M. J.; and Pavone, M. 2020. Efficient large-scale multi-drone delivery using transit networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 4543–4550.

Kapadia, M.; Beacco, A.; Garcia, F.; Reddy, V.; Pelechano, N.; and Badler, N. I. 2013. Multi-domain real-time planning in dynamic environments. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 115–124.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2020. Lifelong multi-agent path finding in large-scale warehouses. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1898–1900.

Ma, H.; Hönig, W.; Cohen, L.; Uras, T.; Xu, H.; Kumar, T. S.; Ayanian, N.; and Koenig, S. 2017. Overview: A hierarchical framework for plan generation and execution in multirobot systems. *IEEE Intelligent Systems* 32(6): 6–12.

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 (AAAI)*, 3166–3173.

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

Pelechano, N.; and Fuentes, C. 2016. Hierarchical path-finding for navigation meshes (HNA*). *Computers & Graphics* 59: 68–78.

Pianpak, P.; Son, T. C.; Toups, Z. O.; and Yeoh, W. 2019. A distributed solver for multi-agent path finding problems. In *International Conference on Distributed Artificial Intelligence (DAI)*, 1–7.

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

Silver, D. 2005. Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 117–122.

Stern, R.; Sturtevant, N. R.; Atzmon, D.; Walker, T.; Li, J.; Cohen, L.; Ma, H.; Kumar, T. K. S.; Felner, A.; and Koenig, S. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *International Symposium on Combinatorial Search (SoCS)*, 151–158.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *AAAI Conference on Artificial Intelligence (AAAI)*, 7732–7739.

Wilt, C. M.; and Botea, A. 2014. Spatially distributed multiagent path planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 332–340.

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. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1443–1449.