Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

Experience-based Multi-Agent Path Finding with Narrow Corridors

Rachel A. Moan, Courtney McBeth, Marco Morales, Nancy M. Amato, Kris Hauser
University of Illinois at Urbana-Champaign
Champaign, Illinois, USA
{rmoan2, cmcbeth2, moralesa, namato, kkhauser}@illinois.edu

***Abstract*—Multi-agent path finding is a computationally challenging problem that is relevant to many areas in robotics. Experience-based planning methods have been shown to significantly reduce the planning time of this problem, but the type of problem in which experience can be used has so far been limited to warehouse-like environments with ample open space. We present an experience-based multi-agent path finding algorithm that specifically addresses narrow corridors of width 1 (also known as doorways). This expands the domain of experience-based problems to include environments such as most houses, office spaces, retail spaces, and hospitals. We also present novel techniques for conflict resolution strategies that result in up to a 94% decrease in waiting steps per robot and final paths closer to the optimal decoupled path by up to 71% than the strategies used in current experience-based methods. We demonstrate our planner solving problems with hundreds of robots in congested environments in seconds, finding solutions in an allotted time more often than existing state of the art optimal methods.**

I. Introduction

Multi-agent path finding (MAPF) is a well-studied computational problem with applications to warehouse robots, factory automation, railway and roadway traffic management, and manipulation of multiple objects [1]–[4]. This problem is commonly formulated with agents moving in a grid and the objective is to find a trajectory for each agent to reach a designated goal. Although in the worst case the problem is NP-hard even to obtain approximately optimal solutions [5], in practice there are efficient heuristics that can solve MAPF problems of large size [6]. Decoupled planning methods can solve large MAPF problems that are relatively unconstrained, but are generally not complete [7]. Conflict-based search (CBS) methods are another popular class of methods that are complete and can often drastically outperform coupled search algorithms, which operate within the joint space of all agents, by judiciously exploring alternatives only when conflicts arise [8]. However, in congested scenarios, CBS methods can still become mired in exhaustive search causing computation time to explode.

A common strategy to address challenging scenarios is to identify subproblem structures, such as highways [9] or corridors [10] and address them using specialized strategies. Recently, there has been growing interest in the general approach of using *experience* to speed up planning using a database of stored solutions to prior planning problems [11], [12]. For MAPF, optimal solutions to especially congested subproblems could be cached offline and reused when such subproblems are encountered online, greatly speeding up the most challenging portions of planning. A recent example is the DDM planner [11], which uses an exhaustive database of 2×3 or 3×3 obstacle-free subproblems to greatly accelerate planning in problems with hundreds of agents. A major limitation of DDM, however, is that it assumes that the environment possesses a “low-resolution” structure in which there exist corridors of width 2 everywhere, which allow robots to pass by one another without encountering local deadlock. This prohibits environments with congestion in narrow corridors or doorways, and hence severely limits the applicability of the DDM approach to common real-world floorplans.

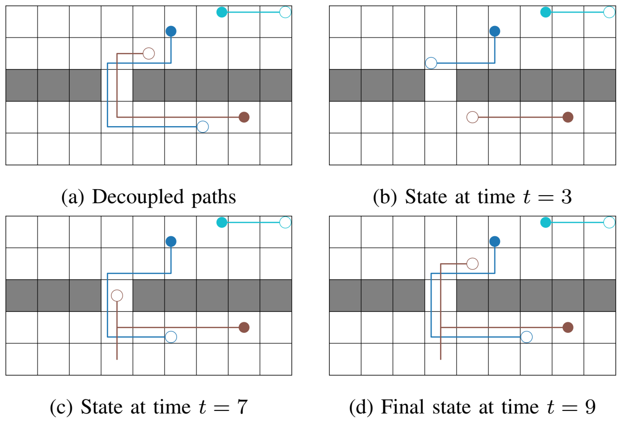
(a) Decoupled paths
(b) State at time $t=3$
(c) State at time $t=7$
(d) Final state at time $t=9$

Fig. 1: Our planner solves for three robots navigating in an environment with a doorway. The starts of the robots are drawn as solid circles and current positions are drawn as empty circles. Every robot has a decoupled path that they follow until a conflict occurs. (a) The robots' originally planned decoupled paths. (b) The robots advance along their paths. The light blue robot has already reached its final goal. (c) The dark blue and brown robots move around each other to avoid a conflict in the doorway. Conflict is avoided by having brown move down for a time step to get out of blue's way. This solution is given by a 5×2 doorway subproblem in our database.

In this paper, we extend experience-based MAPF to a broader domain of environments that contain doorways or corridors of width 1 (Fig. 1). This domain excludes corridors of width 1 that branch within the corridor, but it does cover most environments designed for humans such as homes, office buildings, hospitals, and airports. Narrow corridors are bottlenecks for traditional MAPF algorithms and our algorithm exhibits major speedups by caching an exhaustive set of solutions to corridor-navigation subproblems. Our proposed planner is a variant of DDM that stores a library of subproblem templates that can be instantiated to solve conflicts that arise along agents' desired paths. In particular, we use 2×3 free space, 3×3 free space, and 5×2 doorway subproblem templates (see Fig 2) to cover the doorway problem domain. Our contributions include:

- A framework for experience-based MAPF that accommodates multiple subproblem types including obstacle layouts.
- The 5×2 subproblem template, which we use to resolve conflicts in doorways via querying a database of exhaustive solutions. We introduce a notion of subproblem capacity constraints in the experience-based framework to handle doorways, which are limited to 6 robots total.
- Conflict resolution strategies that result in fewer waiting steps per robot and final paths closer to the optimal decoupled path than the strategies presented in DDM [11].

We evaluate our planner on many benchmark problems, demonstrating that it can scale to hundreds of agents and dozens of doorways. Moreover, we demonstrate that the conflict resolution strategies we present outperform DDM in congested environments even in the absence of doorways.

II. Related Work

A. Multi-robot Path Planning

Multi-robot path planning consists of finding valid, collision-free paths for a set of robots over a graph. Several prior works have proposed hybrid methods that initially plan decoupled paths for each robot and then reconcile inter-robot collisions.

Conflict-based Search (CBS) [8] performs a low-level decoupled search to find optimal paths for each individual robot. Inter-robot conflicts are resolved by searching over a constraint tree where each node represents a set of constraints on the motions of each robot based on the observed conflicts. Although CBS can find optimal paths, its performance is highly dependent on the number of conflicts it encounters. Several extensions to CBS have been proposed to improve its performance in scenarios that feature frequent conflicts while maintaining various levels of optimality guarantees [13], [14].

Prioritized planning algorithms such as Priority-based Search (PBS) [15] assign a priority value to each robot and plan a decoupled path for each robot that does not collide with the already-planned paths for the higher-priority robots. Rather than establishing a fixed priority ordering over all robots, PBS explores the set of permutations of priority orderings to find near-optimal paths.

M* [16] is an A*-like [17] search method that only considers the joint planning space of a group of robots if their optimal individual paths have been shown to interact. Its strategy of subdimensional expansion allows for the dynamic generation of low dimensional search spaces embedded in the full composite space that are only expanded when necessary.

Expanding A* (X*) [18] is an anytime MAPF algorithm that first plans decoupled paths for each robot and then resolves conflicts by replanning within a local repair window, reducing replanning effort. Over time, the size of the window is expanded, allowing higher quality solutions to be found. In the limit of time, X* finds optimal paths.

Although these methods perform well in uncongested environments, they struggle in scenarios where conflicts arise frequently. This includes environments with narrow passages and problems with high robot-vertex ratios.

B. Experience-based Planning

In the field of motion planning, which considers finding a path for a robot in a continuous state space, prior work has explored relying on experience to generate paths. Lightning [19], for example, builds a path library that stores entire paths between different starts and goals under various environment configurations. For each new single-robot query, the library is searched to find a path for a similar previous query, which is then fitted into the current environment. Thunder [20], in contrast to Lightning, aggregates experiences into a sparse roadmap graph, reducing memory usage and query time. Chamzas et al. [21] propose SPARK and FLAME, two variants of an experience-based approach to learning sampling distributions for sampling-based motion planning. These use workspace decompositions at different levels of fidelity to associate local sampling distributions with environment features.

Recent work has explored using deep learning for motion planning, a different form of experience-based guidance. For example, the Motion Planning Networks (MPNet) [22] approach involves two deep networks, one that encodes an environment representation and a planning network that generates the path waypoints. The planning network can be trained using output from another planner or expert demonstrations.

In the multi-robot path planning domain, the Diversified-path Database-driven Multi-robot path planning (DDM) [11] algorithm uses an exhaustive database of solutions for two local graph subproblems to accelerate conflict resolution. It considers subproblems in the form of a 2×3 and 3×3 subgraph. When conflicts arise between individual robot paths, the database is queried to find an appropriate conflict-free path segment, which is then placed into the affected robots' paths. DDM shows significant speedups over other state of the art methods without sacrificing much optimality. However, DDM assumes that there are no narrow (width 1) corridors in the environment. Building off of DDM, the Database-accelerated Enhanced Conflict-based Search (DCBS) algorithm [12] uses the database conflict resolution heuristic to resolve all 2×3 and 3×3 subproblems, reducing the size of the constraint tree it searches. However, in severe congestion DCBS will still create an intractably large constraint tree. Our approach also extends DDM but proposes an expanded set of subproblems intended to capture conflicts in non-branching narrow corridors, and is guaranteed to run in polynomial time.

III. Method

Our planner, which we refer to as EMP (Experience-based Multi-agent Path finding), is an online, coupled algorithm in the vein of DDM which moves agents along *desired paths* and then resolves conflicts as they are encountered. The desired paths are determined semi-independently at the start of the process using a congestion-avoiding heuristic. A *conflict* occurs when two or more agents want to move to the same cell or swap cells. Our planner addresses conflicts by creating small *subproblems* and looking up their solutions in an exhaustive *database* computed offline. Specifically, whenever conflicts arise, the algorithm:

1) Creates a set of subproblems that cover as many conflicts as possible while occupying the least amount of space (Section III-B)
2) For all agents in a subproblem, temporary goals are assigned within the subproblem. These typically advance along their desired paths, but may deviate in case of conflicting goals (Section III-B).
3) For all subproblems we perform a database query to find paths for all agents to their temporary goals (Section IV).
4) All agents advance or wait. Agents inside a subproblem advance along the queried path. Agents outside of subproblems advance along their desired paths, unless their next node enters the region of an existing subproblem. These agents are marked as *waiting* (Section III-C).
5) Any agent that deviates from its desired path will receive a new desired path.

We describe the details of the implementation in the following sections.

A. Problem definition

As in the classical MAPF problem, we are given the 4-connected grid $G = (V, E)$ with obstacle vertices removed, the set of N robots have known start locations $x_1^0, \dots, x_N^0$ and wish to reach goal locations $x_1^g, \dots, x_N^g$. The objective is to compute paths $x_r^0, \dots, x_r^T$ for each robot $r = 1, \dots, N$, such that only edges in G are traversed, the goal is reached $x_r^T = x_r^G$, and no two robots occupy the same vertex or cross the same edge at any time. It is necessary for each start and goal location to be distinct. As an objective function, we use the *makespan* T, which is the maximum number of steps for a robot to reach its goal.

Our algorithm operates in *semi-low resolution* grid environments as shown in Fig. 11. This is an expanded class of problems compared to the class of *low resolution* problems (as shown in Fig. 9) assumed by DDM, which ensures there is a 2×2 block of empty cells containing each empty cell. Semi-low resolution environments do not have long narrow passages with branches, but they do contain doorways. Specifically, define a "wide" cell as an empty cell for which there exists a 2×2 block of empty cells containing it, and a "narrow" cell as an empty cell that does not meet this condition. We define a semi-low resolution environment as one in which the graph of narrow cells consists of disjoint paths, i.e., the narrow corridors do not branch. We posit that most human environments are constructed with this strategy so that any branching corridors allow humans to pass by one another.

With such assumptions we can resolve conflicts locally using certain subproblem templates (Fig. 2), which are required to be solvable regardless of the robot layout. 2×3 and 3×3 free-space subproblems were originally introduced in [11] and

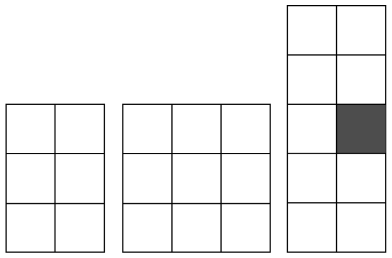
Fig. 2: The three subproblem templates used in our implementation. The 2×3 and 3×3 are both obstacle free. The 5×2 *doorway* subproblem has exactly one obstacle in the middle.

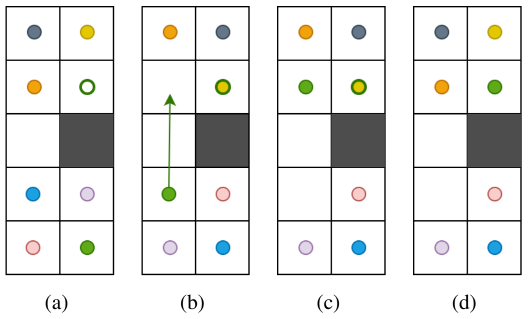
Fig. 3: Doorways containing up to 7 robots can be solved by a complete, suboptimal strategy using rotations and crossings. (a) The green robot has a goal (empty green circle) on the opposite side of the doorway. (b) First we rotate the source block CCW two times so that the green robot is in a position to enter the doorway. We also rotate the target block so that there is a free cell for opposite the doorway. (c) The green robot crosses to the other side. (d) The target block is rotated counterclockwise so the green robot reaches its goal.

can be solved optimally for any set of robots assuming that goals are distinct. The 5×2 doorway subproblem that we introduce in this paper has a narrow passage of length 1 between two 2×2 free blocks. Note that a longer narrow corridor can be shrunk in a topological sense to a length-1 path. A doorway can be solved for any 7 robots with distinct goals. Our planner handles this limit with a capacity constraint that is enforced at each step of planning.

To see why the doorway admits a solution for any 7-robot configuration, we construct a suboptimal solution strategy via "rotation" and "crossing" operations (Fig. 3). First, if any robot is in the doorway, we move it so that all robots are located in one of the 2×2 upper or lower blocks. This may require rotating robots in one of the blocks that has 3 or fewer robots occupying it. Then, we choose any robot whose goal is on the opposite side of the doorway, such that the opposite block is not filled with 4 robots. Call the robot to move the moving robot, the block that it occupies the source block, and the opposite block the target block. The robots in the source block will rotate until the moving robot is adjacent to the doorway (Fig. 3b). The robots in the target block will rotate until an empty space is adjacent to the doorway (Fig. 3b). Then, the moving robot will cross the doorway into the empty space (Fig. 3b). This continues until all robots are in the same block as their goal. (If the goal is the doorway itself, either block suffices). Next, the robots in each block will use the doorway as a temporary location and reorder themselves so that all robots are in the counterclockwise (CCW) order that matches the CCW order of their goals. Then, they will rotate until all robots are in their goal locations.

B. Subproblems

Let R be the set of all robots. At time t, suppose that each robot $r \in R$ moves from x_r^t toward the next position $\hat{x}_r^{t+1}$ on its desired path. A robot is in *conflict* at time t if it would arrive on the same node as another robot at time $t+1$ or if it crosses an edge being traversed by another robot. So for any $r, s \in R$, if $\hat{x}_r^{t+1} = \hat{x}_s^{t+1}$, then r and s have a node conflict. If $x_r^t = \hat{x}_s^{t+1}$ and $x_s^t = \hat{x}_r^{t+1}$, then r and s have an edge conflict. Up to four robots may be involved in a single conflict.

To resolve conflicts, we create a set of subproblems Q. Each subproblem $q \in Q$ must match one of the three subproblem templates allowing for rotational symmetries. Let $Cells(q)$ be the cells in the environment that are contained within the subproblem. We say that a subproblem *covers* a conflict if all of the robots involved in the conflict are located in $Cells(q)$ at time t. (This also implies that the robots are also located in the subproblem at time $t+1$.)

The requirements for subproblems are as follows:

1) No subproblem may overlap with another existing subproblem.
2) The obstacle pattern of the template must match the obstacles in the map accounting for symmetries.
3) The capacity constraint of the template cannot be violated. Only the 5×2 has a capacity constraint and is described in detail in Section III-C.
4) The subproblem must cover at least one conflict.

1) Selecting subproblems: We use a greedy approach to grow the set of subproblems to cover as many conflicts as possible. We begin with a set of conflicts, C, and an empty list of subproblems, Q. We address each conflict sequentially. For each $c \in C$ that has not already been covered by a subproblem, let Q_c be the set of possible subproblems that covers the conflict and adheres to the 3 subproblem requirements described in Sec. III-B. If Q_c is empty, we skip c and induce waiting (Sec. III-C). Otherwise, we select the best subproblem $q \in Q_c$ according to three prioritized criteria, listed in highest to lowest priority:

1) Most conflicts covered. This allows us to resolve as many conflicts as possible with a single subproblem, resulting in fewer queries to the database.
2) Fewest waiting robots. We minimize the number of robots with $x_r^t \notin Cells(q)$ and $\hat{x}_r^{t+1} \in Cells(q)$, which would need to wait if q were used.
3) Minimum subproblem area. This ensures robots outside of subproblems are disrupted as little as possible, and leaves more room to place subproblems later.

This strategy lets our planner choose better placements amongst multiple applicable subproblem types. For example, when it was originally introduced in [11], the 3×3 template

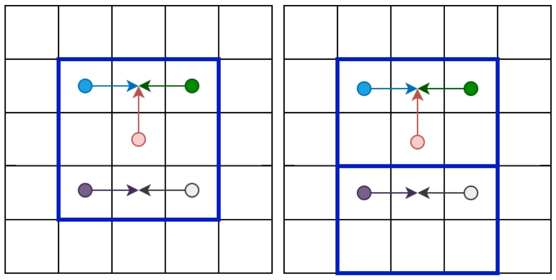
Fig. 4: When two conflicts are next to each other, our planner prefers covering both of them with a single 3×3 subproblem, because two 2×3 templates would be needed to cover the conflicts while taking up more space.

was shown to be unnecessary since it takes up more space than the 2×3, and 4-way conflicts can be resolved by letting one robot wait. In our strategy, however, the 3×3 is able to cover the same amount of conflicts (or more) than it would take for multiple 2×3 templates (see Fig. 4).

It is a natural question whether a greedy subproblem placement approach is reasonable, or whether we should seek an even more optimized approach. We note that the subproblem placement problem is similar to a weighted maximum independent set (WMIS) problem [23], [24]. In geometric MIS problems the goal is to find the largest subset of disjoint geometric shapes from a given set, and in the weighted case the sum of shape weights is maximized. MIS on graphs is an NP-hard problem, but leveraging the regularity of shapes geometric MIS is generally easier to approximate [25]. In our problem, every possible valid subproblem covering at least one conflict is the set of shapes, and the weight of a region could be the number of covered conflicts. Our method is especially fast, does not require a full arrangement computation [23] or LP relaxation [25], and it produces adequate results in quadratic time. We have tested the greedy method against a brute force approach that finds the best set of subproblems via extensive sampling of hundreds of thousands of valid subproblem sets. In a small, congested environment, our method was able to cover 89.7% of conflicts, while the brute force method covered 90.7%. This suggests that the sacrifice of using a greedy method is minimal, while the cost savings are tremendous.

2) Assigning robots to a subproblem: Robots that are not directly involved in a conflict may still be included in a subproblem. This avoids introducing new conflicts with other surrounding robots after resolving one conflict. A robot $r \in R$ is considered *involved* in the subproblem q if both $x_r^t, \hat{x}_r^{t+1} \in Cells(q)$. Denote this set of robots $Involved(q)$. Note that all robots in covered conflicts are involved.

3) Temporary goal assignments: Each subproblem will be solved by querying the database. In order to perform this query, we first need to assign temporary goals for each $r \in Involved(q)$. Each goal must lie within the boundary of the subproblem, no two robots may have the same goal assignment, and these temporary assignments should disrupt the robots as little as possible from their desired paths.

The ideal temporary goal assignment is the last node lying

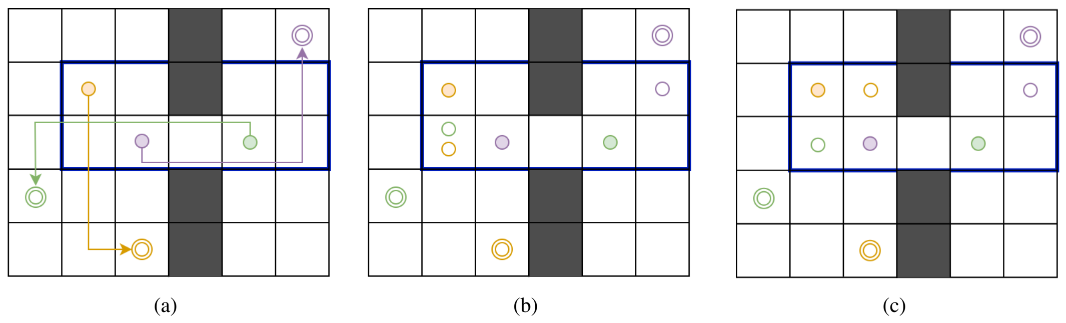

Fig. 5: An illustration of our goal assignment strategy. The robot's current position is denoted by a solid circle and its final goal is denoted by a double circle. First we examine the cost to go for each robot as shown in (a). The cost to go is $5, 5, 4$ for the purple, green, and orange robot respectively. We then find the desired goal, which is indicated by a single empty circle (b). Green and orange have the same desired goal, but green has a higher cost to go, so green is assigned its desired temporary goal while orange gets a random goal assignment (c).

on the robot's desired path that is in $Cells(q)$. This assignment strategy, however, can result in robots being assigned the same temporary goal so, instead, we employ a different strategy that utilizes randomness. We first order the robots according to cost-to-go, greatest to lowest. This gives us a prioritized list of the robots. Then for each robot in this list, we attempt to assign its ideal goal. If this node has already been assigned, we choose a random node in $Cells(q)$ that has not already been assigned a goal (see Fig 5). We elect to use randomness, rather than a deterministic strategy, to avoid deadlock. Our testing has determined that deterministic strategies, such as assigning the subgoal closest to the desired goal, result in a substantial risk of deadlock and infinite loops.

4) *Transformation to matching template:* In order to query the database, we must transform q to match its corresponding template exactly. If q is a 2×3 or 5×2, we first rotate q if necessary to ensure that it has the correct orientation. (See Fig. 6). Lastly, we follow from [11] and order the robots so that the starts and goals in the query match the order of the starts and goals in the database, which eliminates the need to store every permutation of robots in the database (See Fig. 6). We call the policy that performs this reordering π. After applying the necessary rotations and π to q, we then have a templated subproblem that we use to query the database. Fig. 8 illustrates an example of a solution that the database would return. After querying and getting a solution from the database, s, we simply apply the inverse of the policy, π^{-1}, to s to get the solution in the correct order (See Fig. 6).

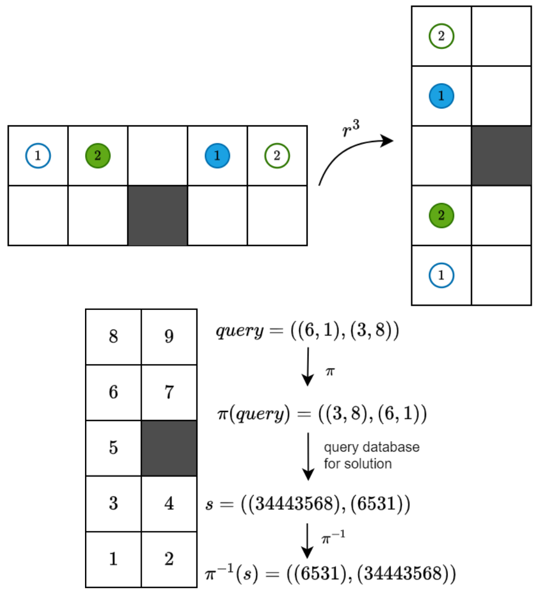
r^3
$query = ((6,1),(3,8))$
π
$\pi(query) = ((3,8),(6,1))$
query database for solution
$s = ((34443568),(6531))$
π^{-1}
$\pi^{-1}(s) = ((6531),(34443568))$

Fig. 6: A subproblem is placed to cover a conflict between robots 1 and 2 (top left). The temporary goals for robots 1 and 2 are denoted by empty circles. The subproblem is rotated three times clockwise to match the 5×2 template (top right). The starts and goals are then ordered for the query to the database (bottom). The query is $((6, 1), (3, 8))$ because robot 1 starts at cell 6 and has a temporary goal at cell 1. Similarly, robot 2 starts at 3 and has a temporary goal at 8. Next, we order the robots in the query by start cell according to the values of the cells 1–9 shown here. After reordering, we query the database for a solution. Lastly, we apply π^{-1} to the solution to get the paths in the same order as our original query.

C. *Waiting policies*

The selected subproblems may fail to resolve all conflicts and prevent the introduction of new conflicts. To cover these cases, we employ a waiting policy. A waiting robot r will alter its desired next step such that $\hat{x}_r^{t+1} = x_r^t$. There are four scenarios in which one or more robots would need to wait:

1) **A conflict cannot be addressed by a subproblem.** This can happen in two cases. First, the conflict has an obstacle pattern that does not match known subproblem templates (Fig 7a).
Second, there may not be space to place a subproblem due to other subproblems being in the way (Fig 7b). In this case, we let one robot proceed and the others wait. Let $c_R \subseteq R$ be the robots involved in a conflict. Then take an arbitrary $r \in c_R$ and make every robot in $c_R \setminus \{r\}$ wait at time t. Robot r will be allowed to advance. The choice of moving robot does not seem to affect performance much, so our algorithm chooses r at random. Comparing against a deterministic strategy, specifically choosing r as the robot furthest from its goal, showed an insignificant performance change on all problems tested.
2) **Robot would enter an active subproblem.** If a robot

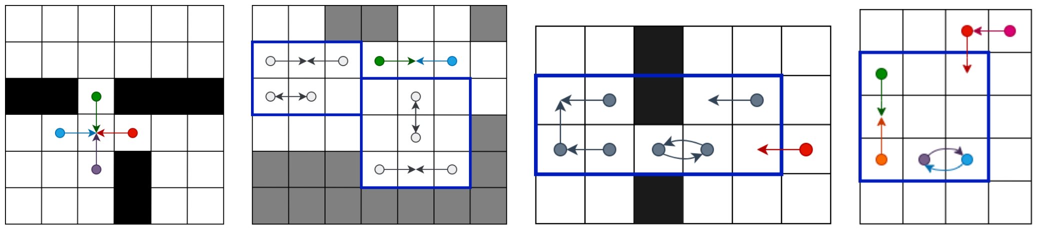
(a) (b) (c) (d)

Fig. 7: Examples of each case in which a robot would be marked as waiting. In Fig. (a) and (b), no valid subproblem exists. In Fig. (c), the red robot's next desired cell is inside a door with 7 robots. To prevent the doorway from becoming unsolvable, the red robot is marked as waiting. In Fig. (d), an active subproblem is shown in blue resolving two conflicts between 4 robots. The red robot's next desired cell is inside of the subproblem. This robot is marked as waiting so that no new conflicts are introduced. The pink robot's next desired cell is the red robot's current cell, so pink also gets marked as waiting.

is not in any subproblem at time t and its desired step at $t+1$ would cause it to enter a subproblem, then it must wait (Fig. 7d).

3) **Advancement would violate a capacity constraint.** The 2×3 and 3×3 subproblems are solvable even when they are fully packed, but the 5×2 doorway is only solvable with a maximum of 7 robots. If this capacity constraint would be violated on the robot's next step (Fig. 7c), the robot outside the doorway must wait.
4) **Next desired cell already contains a waiting robot.** When we mark a robot r in cell v as waiting at time t, we then have to mark any robot i such that $x_i^{t+1}=v$ as waiting at time t. This process of marking cells as waiting at time t creates a flood-fill style recursive algorithm that marks any robot wanting to enter a waiting cell as waiting as well (Fig. 7d).

A subtle remark is that it is beneficial to perform the waiting logic before assigning robots to subproblems, since a robot r that intends to leave a subproblem q can be omitted from $Involved(q)$ if its intended next cell will remain free. This greatly reduces the risk of unnecessary waiting for robots inadvertently finding themselves at the boundary of subproblems and unable to leave. On the other hand, if the next cell hits another subproblem or would cause r to hit a waiting robot, then the robot should be included.

D. Full algorithm

The algorithm takes as input a map of the environment, E, start nodes $x^0_{1:N}$, and goal nodes $x^g_{1:N}$. It then computes the decoupled path for each robot using A* with a congestion-avoiding heuristic (line 1). Then, it advances robots one step at a time.

At the start of the loop we read each robot's next desired position $\hat{x}_r^{t+1}$ in its decoupled path (line 4). The set of waiting robots W and the set of subproblems Q are initialized to empty, and we get a list of all conflicts that will occur if the robots advance (lines 5–7). Then we address each conflict sequentially. If it has not already been covered by a subproblem, we attempt to place a subproblem that covers it as we described in Section III-B. If no valid subproblem can be found, we mark all robots in conflict except one as waiting (lines 8–15). We then create a waiting map from the initial list of waiting robots and freeze waiting robots (lines 16–18). Then for each subproblem, we assign temporary goals and query the database for a solution (lines 20–22). Next, we update the next desired node of each robot involved in the subproblem (lines 23–24). Finally the robots advance and we replan each robot's decoupled path (line 25–26). This could be done only if that robot deviated from its intended path, i.e., $x_r^{t+1}\neq P_r[t+1]$, but for simplicity we simply replan all robots. Note that unlike DDM, we do not preserve subproblems until their solutions are fully executed, and in Sec. V-C we show this can prevent unnecessary waiting by allowing robots to leave subproblems. We continue this loop until every robot is at its goal node.

Algorithm 1: Experience-Based Multi-agent Planning (EMP)

```
1: $P_{1:N} \leftarrow PlanDecoupledPaths(x^0_{1:N}, x^g_{1:N}, G)$
2: $t \leftarrow 0$
3: while $x^t_{1:n} \neq x^g_{1:n}$
4:     $\hat{x}_r^{t+1} \leftarrow P_r[t+1]$ for $r = 1,\ldots,N$
5:     $W \leftarrow \emptyset$
6:     $Q \leftarrow \emptyset$
7:     $C \leftarrow AllConflicts(x^t_{1:N}, x^{t+1}_{1:N})$
8:     for $c$ in $C$
9:         If $c \in Cells(q)$ for any $q \in Q$, skip it.
10:        $Q_c \leftarrow ValidSubproblems(c, G, Q)$
11:        if $Q_c \neq \emptyset$
12:            $q \leftarrow BestSubproblem(Q_c, C, x^t_{1:N})$
13:            Add $q$ to $Q$
14:        else
15:            Add all but one robot in $Robots(c)$ to $W$
16:    $R_{wait} = CreateWaitingMap(x^t_{1:N}, \hat{x}^t_{1:N}, Q, W)$
17:    for $r \in R_{wait}$
18:        $\hat{x}_r^{t+1} \leftarrow x_r^t$
19:    for $q$ in $Q$
20:        $R_q \leftarrow Involved(q, x^t_{1:N}, \hat{x}^t_{1:N})$
21:        $q \leftarrow AssignTempGoals(q, P_{R_q})$
22:        $P^q_{R_q} \leftarrow SolveSubproblem(q)$
23:        for $r \in R_q$
24:            $\hat{x}_r^{t+1} \leftarrow P^q_r[2]$
25:    $x^{t+1}_{1:N} \leftarrow \hat{x}_r^{t+1}$
26:    $P_{1:N} \leftarrow ReplanPaths(x^{t+1}_{1:N}, P_{1:N}, G)$
27:    $t \leftarrow t+1$
```

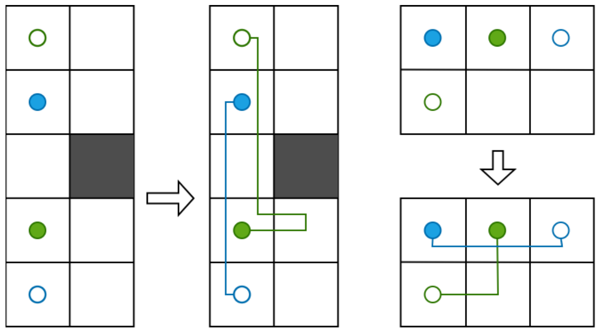

Fig. 8: A query to the database provides starts (shown in sold circles) and goals (shown in empty circles). The database then gives us a collision free path for each robot in the query. In the 5×2 example here, the solution is for the green robot to get out of the way and wait for blue to pass by before continuing to its goal. Note that our planner does not execute the entire solution to a subproblem. Instead, we take only the first step of the paths.

The asymptotic running time of this algorithm is dominated by the subproblem selection step because there are $O(n)$ conflicts, each conflict has over a dozen possible valid subproblems, and our selection metrics take $O(n)$ time to compute. The database query only uses an $O(1)$ hash lookup. Overall, a single iteration of the main loop is $O(n^2)$ leading to an $O(hn^2)$ algorithm where h is the computed makespan.

IV. Database

We generate an exhaustive set of queries for each of the 2×3, 3×3, and 5×2 environments by considering every collision-free set of start and goal points. We choose to use the integer linear programming-based method proposed in [6] due to its speed in solving problems in congested environments with up to 100% robot vertex occupancy. This is also the method that DDM [11] uses to generate their database. Following from [11], we transform each set of paths that we compute to generate several other solutions. The total number of problem scenarios for each subproblem environment is given by

$$\sum_{i=2}^{n_{rob}} \left(\frac{n_{vert}!}{(n_{vert} - i)!} \right)^2$$

where n_{vert} is the number of vertices in the subproblem environment and n_{rob} is the maximum number of robots we consider for the subproblem. For the 2×3, 3×3, and 5×2 subproblems, the number of scenarios is 1.2×10^6, 3.0×10^{11}, and 3.7×10^{10} respectively. It is intractable to solve this many scenarios; however, the use of path transformations significantly reduces the number we must solve.

First, we sort the individual paths in a scenario by the start position. This allows us to group equivalent scenarios together, and reduce the permutation giving the possible start positions to a combination. Then, each local environment that we consider has a different set of valid transformations. In every environment, the paths can be reversed to generate a set of paths from the goals to the starts.

In the 2×3 environment, the paths can also be rotated 180°, and mirrored across a vertical line through the center of the environment. Thus, each set of paths we compute represents a class of up to 8 path sets in the database. The number of scenarios we must solve is then approximately 1.7×10^3. The 2×3 database took 14.2 seconds to generate and requires about .09 MB of storage.

In the 3×3 environment, the paths can be rotated 90°, 180°, and 270° and mirrored across a vertical line through the center of the environment. Accounting for duplicates, each set of paths we compute corresponds to a class of up to 16 path sets in the database. The number of scenarios required to be solved is then approximately 1.1×10^6. The 3×3 database took 5 hours and 55 minutes to generate and requires about 2 GB of storage space. The average time to query the 3×3 databases is 2.43×10^{-6} seconds.

In the 5×2 doorway environment, the paths can be mirrored across a horizontal line through the center of the environment. Thus, each set of paths that we generated can only correspond to up to 4 path sets in the database. This, in addition to the more difficult scenarios present in the hallway environment, which features a narrow passage, led to a much longer database generation time. The number of 5×2 scenarios required to be solved is approximately 3.5×10^6. It took 13.5 days to generate the path sets for 2-6 robots. Due to the larger time frame required to solve the more difficult and larger set of 7-robot scenarios, we omitted this from our experiments and set the capacity limit for the doorway subproblem to a maximum of 6 robots. The 2×5 requires 1.37 GB of storage space. On average, it took 2.26×10^{-7} seconds to query the 2×5 database.

V. Simulated Results

A. Comparison with existing baselines

To highlight the differences of our EMP planner from DDM, Fig. 10 compares the two directly on the low resolution environment shown in Fig. 9. Public C++ code provided by the DDM authors [11] is used in this comparison. Each data point is the result of 20 trials ran with random start and goal configurations for each robot. We use a metric of *path deviation* which measures how much a robot's solved path deviates from the optimal decoupled path solved by A*. Specifically, for robot r, this metric is given by $dev = ||path^r_{final} - path^r_{decoupled}||$. We also compare the number of steps that each robot spends waiting.

The results show that EMP reduces both the amount of waiting and the deviation from the optimal path. This is because, in our planner, the solutions to subproblems are not executed to completion, meaning that there are fewer robots waiting for subproblems to finish before advancing. The deviation from the originally planned path is also low, because EMP always chooses temporary goals on the robot's original decoupled path when possible, so the robot is less likely to deviate from its optimal path. This indicates that our planner is better suited for settings of continuous robot task assignment, in which once a robot reaches its goal, it receives a new goal without waiting for the other robots to reach their goals. Overall we observed no statistically significant difference in

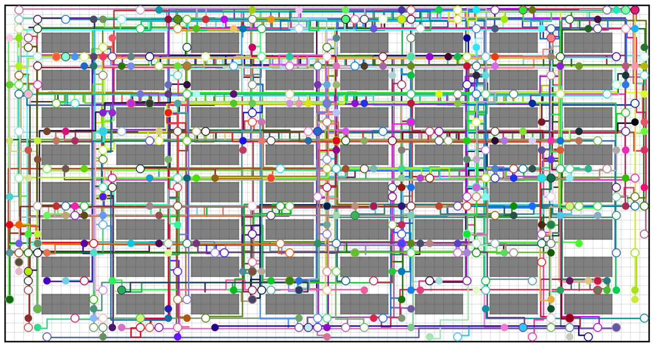

Fig. 9: Planned paths for 300 robots in a warehouse-like environment. Starts (solid circles) and goals (empty circles) were chosen randomly.

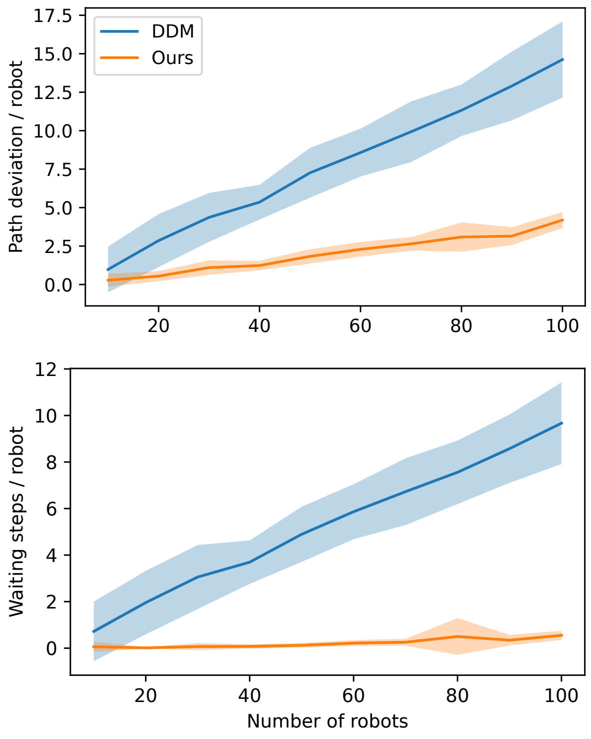

Fig. 10: Comparing EMP and DDM on the environment in Fig. 9. Mean and standard deviation over 20 randomized runs are shown.

makespan between EMP and DDM, since makespan is limited primarily by the robot that is farthest from its goal.

In especially crowded environments ($> 70\%$), we found that DDM's strategy of executing the entire subproblem solution proved beneficial. In the empty 10×10 environment at 80% occupancy, our makespan explodes to anywhere between 158-400, but DDM maintains reasonable makespan in the same scenario at around 81-144. This is because when we only execute one step of the subproblem, the conflict is likely to occur again in the next time step due to the high volume of robots. In these high congestion scenarios, executing the entire subproblem appears to be the better strategy.

We also ran CBS on both the warehouse environment in Fig 9 and the rooms environment in Fig 11. CBS solved the warehouse environment fairly quickly in around 0.1–10 s for 30 robots. However, in the rooms environment CBS failed to find a solution with a time limit of 10 minutes in 17/20 trials, and some runs took over 6 minutes to solve. Our planner by comparison takes anywhere from 2–12 s to solve the same scenario. Note that our algorithm is implemented in non-optimized Python and CBS is a publicly available C++ implementation [2]. The reason for the long running time of CBS is that its conflict tree becomes unmanageable in large environments with bottlenecks.

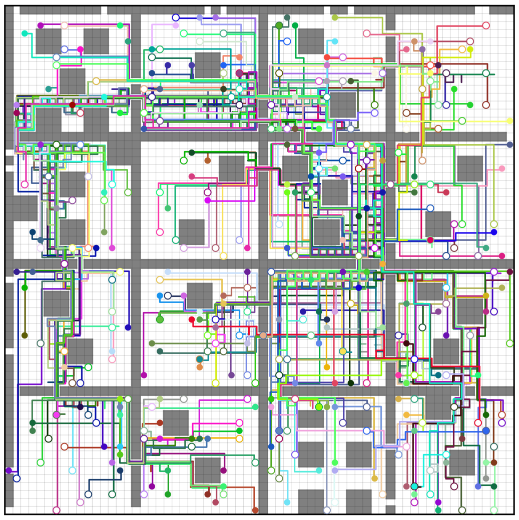

Fig. 11: Planned paths for 200 robots on a floorplan with rooms with width-1 doorways and random obstacles. Starts (solid circles) and goals (empty circles) were chosen randomly.

B. *Environments with doorways*

We also demonstrate that EMP generates solutions for hundreds of robots on environments containing doorways, even on a more challenging 64×64 environment that contain both doorways and obstacles scattered throughout rooms randomly (Fig. 11). This environment was created by augmenting a map from the MAPF benchmark website with obstacles inside the rooms [2]. Scaling results are shown in Fig. 12, showing moderate levels of waiting even at high levels of congestion.

C. *Ablation studies*

To evaluate the effects of each of EMP's contributions, we examine how several design choices affect performance:

1) **Executing all steps vs one step of a subproblem solution**. EMP uses one step of a subproblem's solution, then discards it. In contrast, DDM retains subproblems between steps until all robots have reached their goals.
2) **Prioritized vs random goal assignment**. EMP assigns temporary goals on the robot's intended path and uses prioritized assignment in the case of conflicts. DDM assigns random goals in the case of a conflict.
3) **First vs best subproblem placement**. EMP optimizes each subproblem according to various heuristics, while DDM picks the first valid subproblem.

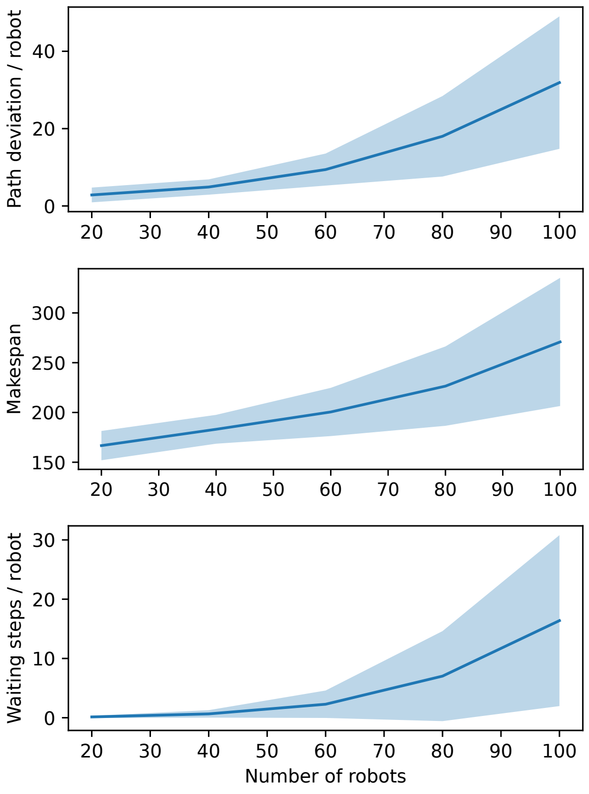

Fig. 12: Results of our planner from the environment in Fig. 11.

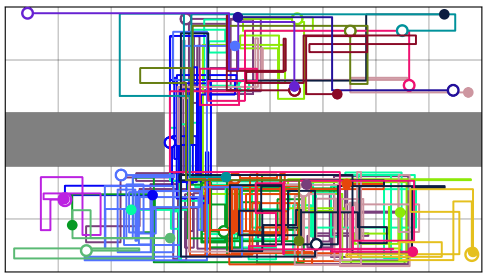
Fig. 13: Solution paths for a 5×9 doorway environment with 18 robots. Path segments are shifted within each cell to more clearly illustrate the complexity of robot coordination in and around the doorway.

4) **Number of subproblem templates.** Including the 3×3 subproblem does not change feasibility, but has the potential to improve efficiency.

Table I shows the results of running the planner on an empty 10×10 environment with 40 robots under various settings. EMP is our planner, which uses prioritized goal assignment, finds the best subproblem, and executes only the first step of the subproblem solution. "EMP first" is our planner that uses the first valid subproblem, rather than searching for the best subproblem. "EMP random" is our planner, but uses purely random subproblem goal assignment instead of prioritized. "EMP first + random" finds the first valid subproblem, uses random subproblem goal assignment, and executes only one step of the subproblem. We also include results from DDM

	Path deviation	Makespan	Waiting steps
EMP	13.5	45.2	11.0
EMP w/ first subproblem	17.4	52.7	20.7
EMP w/ random goal	14.5	48.2	10.5
EMP w / first + random	19.5	60.8	21.9
DDM	22.0	45.5	22.5

TABLE I: Ablation study of planner settings. Tested on 40 robots in the empty 10×10 environment, averaged over 20 random starts and goals.

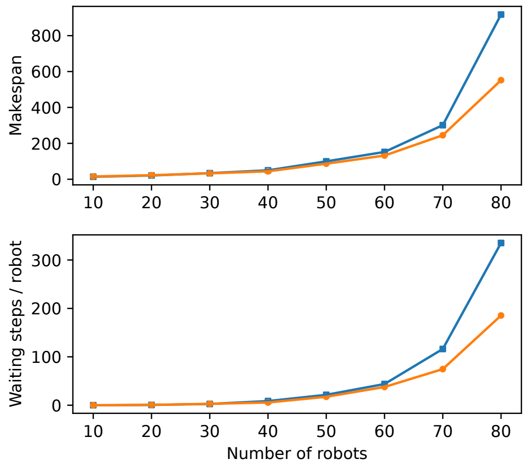

Fig. 14: Excluding the 3×3 subproblem template leads to longer makespans and more waiting in crowded scenarios. Results averaged over 20 random trials in an empty 10×10 environment.

which is the same style as our planner, but uses random goal assignment, finds the first valid subproblem, and executes the entire subproblem solution. These results show that finding the best subproblem (EMP and EMP w/ random goal assignment) cuts the number of waiting steps per robot nearly in half. The path deviation is smaller using both prioritized goal assignment and the best subproblem. This is because the prioritized goal assignment keeps robots closer to their optimal path, and the best subproblem reduces unnecessary waiting.

Finally, we found that including the 3×3 template is helpful in crowded environments. Fig. 14 shows EMP's performance in the same empty 10×10 environment with varying levels of occupancy. Observe that the 3×3 is particularly useful in crowded environments ($> 60\%$ occupancy) because it can cover more conflicts than the 2×3. However, at lower congestion levels there is no significant improvement. Since it takes much longer to generate an exhaustive solution database for larger subproblems, an EMP implementer should be careful to choose subproblem templates that address bottlenecks likely to be observed in practice.

VI. Conclusion and Future Work

In this paper, we present an experience-based MAPF solver that extends a prior framework to a broader domain of environments containing doorways. By leveraging an exhaustive database of solutions for three subproblem templates, our method can solve congested problems with hundreds of robots in seconds. We developed new strategies for selecting subproblems and assigning temporary goals that allow our method to handling the new doorway subproblem template efficiently and with fewer robots required to wait for subproblem resolution. We also examine the question of whether providing more subproblem templates are likely to boost experience-based MAPF performance, and we conclude that improvements are likely only in narrowly construed problem domains.

In the near future we are interested in speeding up our runtime by optimizing the implementation of our code. More broadly, our strategy to solve doorways is a first step toward experience-based conflict resolution in long and branching narrow passages. Subproblems to negotiate passages with complex topology (e.g., T junctions, four-way junctions, corridors with cutouts) could potentially be incorporated in our framework, but would require more work in studying how to topologically deform map geometry into canonical templates and to deform solutions back. Finally, we are interested in studying how experience can be leveraged for solving MAPF problems with continuous-space or continuous-time dynamics, such as in congested urban driving scenarios or warehouses that do not impose a strict grid layout.

Acknowledgements

We would like to thank Simon Kato and Joao Marques for their valuable feedback. Moan was supported by the National Science Foundation Graduate Research Fellowship (NSF GRFP). McBeth was supported by Foxconn Interconnect Technology (FIT) and the Center for Networked Intelligent Components and Environments (C-NICE) at UIUC.

References

[1] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, “Lifelong multi-agent path finding in large-scale warehouses,” in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS ’20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 1898–1900.

[2] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, “Multi-agent pathfinding: Definitions, variants, and benchmarks,” *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.

[3] D. Atzmon, A. Diei, and D. Rave, “Multi-train path finding,” in *Twelfth Annual Symposium on Combinatorial Search*, 2019, pp. 125–129.

[4] D. M. Saxena and M. Likhachev, “Planning for complex non-prehensile manipulation among movable objects by interleaving multi-agent pathfinding and physics-based simulation,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 8141–8147.

[5] H. Ma, C. Tovey, G. Sharon, T. K. Kumar, and S. Koenig, “Multi-agent path finding with payload transfers and the package-exchange robot-routing problem,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10409

[6] J. Yu and S. M. LaValle, “Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics,” *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[7] Q. Sajid, R. Luna, and K. E. Bekris, “Multi-agent pathfinding with simultaneous execution of single-agent primitives,” in *Fifth Symposium on Combinatorial Search (SoCS)*, 2012, pp. 19–21.

[8] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, “Conflict-based search for optimal multi-agent pathfinding,” *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[9] L. Cohen, T. Uras, and S. Koenig, “Feasibility study: Using highways for bounded-suboptimal multi-agent path finding,” in *Proceedings of the International Symposium on Combinatorial Search*, vol. 6, no. 1, 2015, pp. 2–8.

[10] J. Li, G. Gange, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, “New techniques for pairwise symmetry breaking in multi-agent path finding,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 193–201.

[11] S. D. Han and J. Yu, “Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1350–1357, 2020.

[12] T. Guo and J. Yu, “Efficient heuristics for multi-robot path planning in crowded environments,” in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 6749–6756.

[13] M. Barer, G. Sharon, R. Stern, and A. Felner, “Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem,” in *Symposium on Combinatorial Search*, 2014.

[14] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and S. E. Shimony, “Icbs: The improved conflict-based search algorithm for multi-agent pathfinding,” in *Symposium on Combinatorial Search*, 2015.

[15] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, “Searching with consistent prioritization for multi-agent path finding,” in *The Thirty-Third AAAI Conference on Artificial Intelligence*, ser. AAAI’19. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33017643

[16] G. Wagner and H. Choset, “M*: A complete multirobot path planning algorithm with performance bounds,” in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2011, pp. 3260–3267.

[17] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[18] K. Vedder and J. Biswas, “X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs,” *Artificial Intelligence*, vol. 291, p. 103417, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370220301661

[19] D. Berenson, P. Abbeel, and K. Goldberg, “A robot path planning framework that learns from experience,” in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Saint Paul, Minnesota: IEEE, 2012, pp. 3671–3678.

[20] D. Coleman, I. A. Şucan, M. Moll, K. Okada, and N. Correll, “Experience-based planning with sparse roadmap spanners,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 900–905.

[21] C. Chamzas, Z. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki, “Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1283–1289.

[22] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, “Motion planning networks,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 2118–2124.

[23] W. Gálvez, A. Khan, M. Mari, T. Mömke, M. R. Pittu, and A. Wiese, “A 3-approximation algorithm for maximum independent set of rectangles,” in *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022, pp. 894–905.

[24] A. Adamaszek and A. Wiese, “Approximation schemes for maximum weight independent set of rectangles,” in *2013 IEEE 54th annual symposium on foundations of computer science*. IEEE, 2013, pp. 400–409.

[25] T. M. Chan and S. Har-Peled, “Approximation algorithms for maximum independent set of pseudo-disks,” in *Proceedings of the twenty-fifth annual symposium on Computational geometry*, 2009, pp. 333–340.