Optimized Solving Anonymous and Combinatorial Multi-Agent Path Finding Problems in Polynomial Time

Stefan Edelkamp

Department of Theoretical Computer Science and Mathematical Logic Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Computer Science Department, Artificial Intelligence Center Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic

Abstract

In this paper, we consider the planning missions for a fleet of robots in undirected graphs, such as grids. In contrast to regular multi-agent path-finding (MAPF), the solver has to find the assignment of goals to the agents on its own. With the same number of goals and agents, this is Anonymous MAPF (AMAPF), whereas the generalization for multiple goals is known as Combinatorial MAPF (CMAPF). The polynomial-time planner finds conflict-free optimized routes. First, shortest path tables from all goals to all states are stored, and a matrix of pairwise distances is computed. For AMAPF a fast solver for the assignment problem is used to compute the initial mapping of start to goal locations, and, for the initial assignment of partial tours to agents in CMAPF a solver based on Monte Carlo search is called. All solvable instances of the MoveAI MAPF benchmark suite interpreted as AMAPF and CMAPF problems have been solved. In these instances with up to several thousands of robots, the automated assignment strategies greatly reduce the number of potential conflicts, with the remaining ones easily being resolved by elaborating on the concepts of ants-on-the-stick and cuckoo'ing agents that have already arrived at their destinations.

Introduction

Multi-agent path finding (MAPF) is the problem of computing collision-free paths for a set of agents from their current locations to given destinations. Application examples include automated warehouse systems, office robots, and non-player moves in video games.

Regular MAPF (Stern et al. 2019a) with a fixed assignment of k agent locations to k goals usually executed in an undirected and unifomly weighted graph such as a grid has impacted considerable research progress. Anonymous multi-agent path finding (AMAPF) is MAPF where the goal assignment is left to the solver (see Figure 1). For multiple goals to be assigned to each agent, this is called combinatorial multi-agent path finding (CMAPF).

Multiple goals are a significant extension for current MAPF solvers. For example, the authors in (Ren, Rathinam, and Choset 2023) study an extension of the conflict-based algorithm of MAPF to CMAPF, which later was shown to be non-optimal (Mouratidis, Nebel, and Koenig 2024). The core issue was the greedy manner of the chained A* algorithm, clearing the hash table of nodes at different stages of the search. Mixing individual A* search trees into one retains optimality, but such a composite search is less efficient.

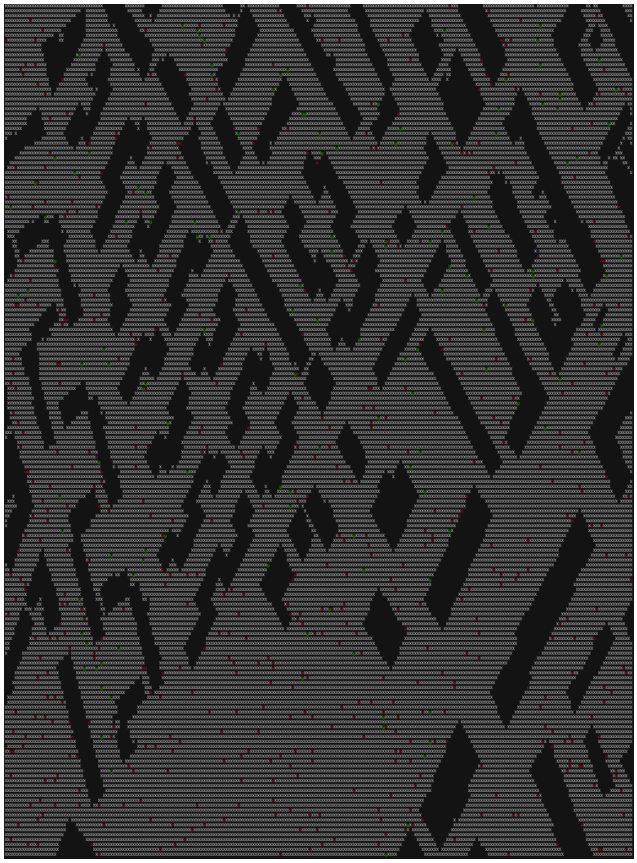


Figure 1: Simulation step in Anonymous MAPF benchmark in (255×255) map with obstacles and 950 agent current (red) and remaining goal positions (green, best seen magnified with colors on screen).

Although optimal polynomial-time algorithms for AMAPF are known, the computational complexities are high, so that the proposed network flow algorithm by Yu and LaValle (2013) has not shown practical impact in

experiments. The core reason is that the algorithm operates on the time-expanded graph, so that the input size for the network is already large, which together with the superlinear time complexity for network flow leads to considerable long running times in practice.

In contrast to this line of work, in our approach we work on the graph reduction that —up to computing, storing and simulating shortest paths— is independent of the size of the underlying graph. We derive efficient implementations for the single-source all-targets shortest path problem (linear-time in 4-and 8-connected grids) and for AMAPF to compute an optimal solution of the assignment problem. The Hungarian algorithm (Kuhn 1955) to solve the assignment problem initially took $O(k^4)$, with n for AMAPF being the number of agents in the graph, but later it was reduced to $O(k^3)$. Among these $O(k^3)$ algorithms, we selected the efficient shortest augmentation path algorithm (Jonker and Volgenant 1987). Instead, to solve the CMAPF we use the NRPA algorithm proposed by (Rosin 2011).

The paper is structured as follows. We briefly review related work on the problem and possible applications. Next, we introduce the general solution principle that we apply, which also helps to warrant polynomial-time solubility. We consider the computational limits and possibilities for anonmymous and combinatorial MAPFs, and introduce polynomial-time solutions to these problems. Experimentally, we study the reduction obtained in the sum of costs and makespan, as well as the reduction and elimination of potential conflicts on the set of 500 well-known MAPF benchmarks instances from the Moving AI repository (Stern et al. 2019b). Finally, we reflect on the results obtained and indicate future research avenues.

Related Work

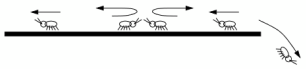
The entire body of MAPF research is too large to be presented here, so we selected some recent trends and applications. A tractable algorithm for MAPF on undirected graphs was proposed by (Wang and Botea 2014). More recently, the MAPF algorithm on directed graphs was shown to be NP-complete (Nebel 2023). Although the problem was known to be NP-hard, Nebel proved that the problem is also contained in NP, therefore, showing that *short solution hypothesis* for strongly connected digraphs holds. Another interesting recent MAPF variant is the option to disconnect trailers or containers from vehicle agents (Bachor, Bergdoll, and Nebel 2023).

From a logistics perspective with CMAPF we consider extensions of the vehicle routing problem (VRP) known from Operations Research (Dantzig and Ramser 1959) to indoor navigation, where the objective is to find optimized tours for a fleet of vehicles to service a given set of customer requests while avoiding collisions. Usually in logistics, the assigning of the goals to the vehicles has to be found by the solver. Dynamic VRPs were studied by (Bullo et al. 2011), and vehicle routing for robots with temporal constraints by (Kiesel et al. 2012), using a high-level task planner to statically assign waypoints to vehicles and a low-level motion planner that generates a feasible path that respects the vehicle motion model. While Edelkamp, Plaku, and Warsame (2019a) tackle pickup and delivery VRPs over a discrete abstraction using sampling-based motion planning to guide the search in an obstacle-rich environment, which produces paths that maximize the number of customers that could be visited, Zacharia and Xidias (2020) developed an approach that addresses the problem of vehicle route routing with uncertain customer demands and travel distances applying fuzzy theory.

Liangou and Dentsoras (2021) presented an approach that combines A* and a genetic algorithm to solve a variant of the TSP with energy and capacity constraints, where A* is used to calculate the shortest path for each combination of pair of waypoints. An approach to MAPF with precedence-constrained goal sequences was studied by (Zhang et al. 2022). Andreychuk et al. (2022) highlight that the simplifying discretized time limits the applicability of multiagent pathfinding algorithms in real-world applications and raise questions of how to discretize time effectively, proposing algorithms for finding optimal solutions that do not rely on any time discretization. (Perez et al. 2014) introduced the multi-objective physical TSP as a real-time game where the player controls a ship that must visit a series of waypoints in a maze while minimizing three opposing goals: time spent, fuel consumed and damage taken, and the proposed controller was based on single-objective Monte Carlo tree search, while the evaluation stressed multi-objective concepts. There is related work on mission planning in terrains with varying energy consumption (Herynek and Edelkamp 2024), but this work has a different focus, since the energy to be planned must not be depleted.

Ants on a stick

One hundred ants are dropped on a meter stick. Each ant is traveling either to the left or the right with constant speed 1 meter per minute. When two ants meet, they bounce off each other and reverse direction. When an ant reaches an end of the stick, it falls off.



At some point all the ants will have fallen off. The time at which this happens will depend on the initial configuration of the ants.

Questions:

(a) over ALL possible initial configurations, what is the longest amount of time that you would need to wait to guarantee that the stick has no more ants?

(b) a slightly more technical task: calculate the *average* time it takes for arbitrary initial number of ants N to fall off the stick.

Figure 2: Ants-on-a-stick problem (a known Math exercise).

Solution Principle: Ants-on-the-Stick

Our approach to solving the anonymous and combinatorial MAPF is based on the following intriguing mathematical puzzle (see Figure 2)[1].

Several ants are dropped on a $1m$ stick. Each ant travels to the left or right with a constant speed $1m$ / minute. When two ants meet, they bounce off each other and reverse direction. When an ant reaches an end of the stick, it falls off. At some point all the ants will have fallen. The time at

[1]discussed, e.g., by Su, Francis E., et al. in blog "Ants on a Stick." Math Fun Facts. See https://www.math.hmc.edu/funfacts

which this occurs will depend on the initial configuration of the ants. While ants bouncing off each other seems difficult to keep track of, we observe that two ants bouncing off each other is equivalent to two ants that pass through each other, instead of turning and exchanging their intended direction. In this way, all the ants fall after traversing the length of the stick and it will be empty after 1 minute.

Polynomial-Time Solutions

In an undirected graph with n goals and a set of k moving agents, we consider a multi-goal multi-agent path-finding problem. For a fixed assignment from k robots to k goals, we have the (nonoptimal) classical MAPF problem that has been solved in polynomial time for an undirected graph (Röger and Helmert 2012; Kornhauser, Miller, and Spirakis 1984). The work goes back to (Wilson 1974), who proved this for $k = n - 1$.

Anonymous MAPF

For the AMAPF problem with k agents and k goals, there are even optimal polynomial algorithms. The main idea is to replicate the graph of the underlying problem with the depth of the solution and look for an optimal flow of the network (Yu and LaValle 2013). However, the approach is hardly practical and, to the best of our knowledge, has not been implemented. It also does not easily carry over to multiple goals as even for one agent we have a TSP. We start our study with the following definition.

Definition 1 (Anonymous MAPF alias AMAPF) *Let $G = (V, E, w)$ be a weighted graph with $w : E \to \mathbb{N}$. We assume that the graph is undirected, so that for all $(u, v) \in E$ we have $(v, u) \in E$ and $w(u, v) = w(v, u)$. In the* Asynchronous MAPF problem*,* AMAPF *for short, for a set of agents R with $k = |R|$ and pairs start and goal locations $(s_i, t_j) \in V \times V$ for $i, j \in \{1, \ldots, r\}$, collision-free routes π_i, $i \in \{1, \ldots, k\}$, have to be found that for each robot starts in s_i, ends in some goal location and that together minimize accumulated travel weight $\sum_{i=0}^{k} \sum_{(u,v)\in\pi_i} w(u, v)$ (sum-of-cost) or maximum travel time $\max_{i=0}^{k} \sum_{(u,v)\in\pi_i} w(u, v)$ (makespan).*

A collision occurs during the simulation of the routes, if two robots are located at a node at the same time (node conflict), or on traversing an edge at the same time, so that additional to robot moving actions, wait actions might be needed. Grid graphs can be easily be compiled into a graph, with each cell representing a node.

As a first step, we compute a shortest-path reduction of the graph. Applying Dijkstra's algorithm for each agent computing shortest paths backwards from every goal, for $n = |V|$ and $e = |E|$ this takes time $O(e + n \log n)$ (with Fibonacci heaps). For simulation and path tracking, we still keep the shortest paths to the goals. In essence, for k robots, this precomputation step takes at most $O(k(e + n \log n)) = (kn^2)$ time and space $O(kn)$.

Faster structures like bucket structures and radix-heaps reduce the running time further to almost linear time (for non-negative integer costs bounded by C, a one-level form of the radix heap yields a time limit for Dijkstra's algorithm of $O(e + n \log C)$, and a two-level form of radix heap gives a limit of $O(e + n \log C / \log \log C)$, so that assuming 64-bit integers to encode the cost values, radix-heaps lead to an additional constant to the linear time complexity. Note that for undirected graphs and integer weights, Mikkel Thorup (1997) showed that a linear-time shortest path algorithm running in $O(e)$ exists but becomes more complicated to implement. For planar graphs, like grids, by Euler's formula, we directly obtain $e = O(n)$.

```
1  void sssp(int w, x, int y) {
2    q.push(x * SIZEY + y);
3    for (int i=0;i<SIZEX;i++) {
4      for (int j=0;j<SIZEY;j++) {
5        closed[i][j] = false; shortest[w][i][j] = -1;}}
6    while (!q.empty()) {
7      int n = q.front(); q.pop();
8      int x = n / SIZEY, y = n % SIZEY, succs = 0;
9      if (closed[x][y]) continue;
10     if (x > 0 && !map[x-1][y]) succ[succs++] = (x-1) * Y + y;
11     if (x < X-1 && !map[x+1][y]) succ[succs++] = (x+1) * Y + y;
12     if (y > 0 && !map[x][y-1]) succ[succs++] = x * Y + (y-1);
13     if (y < Y-1 && !map[x][y+1]) succ[succs++] = x * Y + (y+1);
14     if (x > 0 && y < (SIZEY-1) && !map[x-1][y+1])
15       succ[succs++] = (x-1) * SIZEY + (y+1);
16     if ((x < SIZEX-1) && (y < SIZEY-1) && !map[x+1][y+1])
17       succ[succs++] = (x+1) * SIZEY + (y+1);
18     if ((x < SIZEX-1) && y > 0 && !map[x+1][y-1])
19       succ[succs++] = (x+1) * SIZEY + (y-1);
20     if (x > 0 && y > 0 && !map[x-1][y-1])
21       succ[succs++] = (x-1) * SIZEY + (y-1);
22     if (!closed[x][y]) { shortest[x][y] = n; q.push(succ[s]))
         ;}}}}
```

Figure 3: Single-source all-targets shortest-paths implementation for undirected octile grids via (backward) BFS.

For uniformly weighted grid graphs, breadth-first search (BFS) suffices, so in grid graphs we end up with time $O(kn)$ for this step. This implementation is provided in Fig. 3. Using BFS instead of Dijkstra's shortest-path search in weighted undirected graphs would also yield solubility but worse solutions.

As a second step, we compute the matrix of pairwise distances between all start-to-goal locations, which —by chaining back associated shortest-path pointers— is available in time $O(n \times k^2)$ using the precomputed information on the shortest paths from Step 1. As the graph is undirected, only half of the distance matrix needs to be determined and the other half can be copied. If distances are precomputed with each node in the closed list, the running time for this step reduces to $O(k^2)$.

As a third step, we compute the mapping of an equal number of agents' starting locations and goals, weighted with shortest path distances. The assignment problem asks for such a mapping so that the total travel time is minimized. The first implementation with the Hungarian algorithm for solving the problem given the almost 1000 agents in the example of Fig. 1 was less than half a minute, but took about 15 minutes for examples with about 7000 agents. Therefore, we chose the faster Jonker/Vologenant algorithm to solve the AP instead. It has a cubic time in the number of agents

$O(k^3)$ and in practice leads to adequate running times (see experiments). While there are faster parallel implementations for solving the assignment problem on multi-core CPU and many-core GPUs we chose our implementation to run on a single-core. As stated in the Introduction, this step requires time $O(k^3)$.

Last but not least, in Step 4 we simulate the solution in the graph to handle the collision conflicts. We look at the precomputed shortest-path distance table to find the next location to go to for the agent. For each agent, we denote its position and orientation (that is, the direction of the next goal to visit). In this way, we can compute the agent-dependency graph.

We are using the above ant-on-the-stick metaphor to exchange the goals (and subsequent shortest-path tables) among the agents. The simulation time is the number of simulation steps multiplied by the efforts in each step. The number of simulation steps will (in a first approximation) be bounded by the length of the longest path and, therefore, the diameter of the graph, so that the simulation takes at most time quadratic to the size of the graph.

There are, however, two subtleties to this approach as the ant-on-the-stick problem is considered a continuous line, while for MAPF we have nodes and edges of the underlying graph and agents do not drop off.

While edge conflicts of agents can be handled immediately by swapping the goal information of the agents, for node conflicts of approaching agents, such conflict may incur a delay of one agent. This delay is propagated recursively among other agents along the agent-dependency graph, namely to all agents that have the location of the waiting agent as targets. For this an inverted action dependency tree has to be traversed. When assuming a maximum branching factor, this backward traversal is constant-time for each node in the tree, after which it is removed from further consideration. As all node conflicts are removed and the action dependency graphs are trees resolved in the same iteration, this will not increase the running time by at most a factor $O(k)$ per iteration. The total number of iterations for the simulation at remains bounded, as the node conflict in one iteration will be an edge conflict in the other. If we have a node conflict of more agents, e.g. by paths intersecting, we can do a one-step lookahead to resolve it. In case of 4- or 8-connected grids, there is a limited number of agents involved in a node conflict, so that we can break ties in alternating fashion. Although the maximum time complexity increases to $O(kn \cdot diam) = O(kn^2)$, after solving the assignment problem, conflicts are rather rare in practice, and the simulation is usually much faster than all the other steps.

Secondly, there is the additional problem that agents might have arrived at a destination and stopped their journey there. This might lead to the blocking of other agents. We resolve this problem with a concept that we call cuckoo'ing: the agents at the goal location are pushed out of their location with the agents arriving at it. This is assisted by exchanging the goal and shortest-path information among them. This step also has to be executed recursively on the shortest path of an agent to move until one agent leaves the sequence of goals or all agents arrive at their desired spot. If there is an agent pointing to the location at the end of the cuckooing process, it has to wait and so do the agents pointing to it, so that these goal conflicts are handled before the agent conflicts. As a positive side effect, cuckooing may reduce the makespan and thus the number of iterations for the simulation. Cuckoo'ing is a local update that in total for each iteration does not add more than $O(k)$ to the runtime and $O(kn \cdot diam) = O(kn^2)$ for Step 4.

Given $k \leq n$ as a result, we arrive at time $O(kn^2 + kn^2 + k^3 + kn^2) = O(k \cdot n^2)$ for the entire algorithm, supporting the following result on polynomial solubility.

Theorem 1 (Polynomial-Time Solubility of AMAPF)
Let k be the number of agents and n be the number of nodes in the graph. The AMAPF problem is solvable in polynomial time wrt. n and k. Optimization of solutions is available in time $O(k \cdot n^2)$. The space requirements are dominated by the tables that store the shortest path information, which is of size $O(k \cdot n)$.

As we first solve the assignment problem and then the resulting MAPF, we cannot expect optimal solutions with our polynomial-time algorithm, as the well-known (n^2-1)-puzzle can be cast as a multi-agent path-finding problem, which is polynomially solvable but computing the optimal solution is NP-hard.

Combinatorial MAPF

For CMAPF we start with some observations. The traveling salesman problem (TSP) is NP-complete even for uniformly weighted and undirected graphs, and thus, as generalizations, most multi-agent multi-goal path-finding problems are at least NP-hard (Applegate et al. 2006). If we do not insist on exclusive visits to the nodes, this complexity may decrease. For example, the Eulerian path problem, where each edge has to be visited exactly once, is polynomially solvable. Recall also that existing algorithms for the TSP are exponential in the number of goals to visit, not necessarily in the size of the input graph, as the above polynomial-time shortest-path reduction for the graph can be applied.

Definition 2 (Combinarorial MAPF alias CMAPF)
Let $G = (V, E, w)$ be a weighted graph with edge functions with cost function $w : E \to \mathbb{N}$ for travel distance. We assume that the graph is undirected, so that for all $(u, v) \in E$ we have $(v, u) \in E$ and $w(u, v) = w(v, u)$. In the combinatorial MAPF problem*, for a set of agents R with $k = |R|$ and starting locations $S \subseteq V$ and a set of goals $W \subseteq V$ with $m = |W|$ routes π_i have to be found for each of the robots $i \in \{1, \ldots, k\}$, that in total visit all the goals and minimizes a combination of accumulated travel time $\sum_{i=0}^{k} \sum_{(u,v)\in\pi_i} w(u, v)$ (sum-of-cost) and maximum travel time $\max_{i=0}^{k} \sum_{(u,v)\in\pi_i} w(u, v)$ (makespan).*

The algorithm we proposed is similar to that for AMAPF. In the experiments, for the sake of simplicity, we neglect the starting positions for the shortest path computations and assume that the agents start at some goal positions, which will be visited in the initial state. In terms of the algorithm's input, this approach is equivalent to extending the goal set of

size m by the agents' starting positions r to a new goal set with size $(m + r)$.

If the graph is disconnected, we first compute the connected components in linear time to the size of the graph and solve the problem in each connected component individually. So, w.l.o.g., we assume that the undirected graph is connected. In the benchmarks, this might not be the case, so some problems are quickly determined to be unsolvable.

It is also obvious that any given assignment of (still unvisited) goals to agents induces a partitioning $W_1 \ldots, W_k$ of the set of remaining goals W with $\cup_{i=1} W_i = W$ and $W_i \cap W_j = \emptyset$ for all $1 \leq i \neq j \leq k$.

We start with a(ny) greedy initial ordered assignments of the goals to the agents, which may be found by graph partitioning or weighted clustering approximation methods, large neighborhood searches, or, as in our approach, by a Monte Carlo search-based time-limited approximate vehicle routing solver.

Suppose that during the travel imposed by the shortest paths between subsequent goals, agent r_i has a goal g_j on top of its goal agenda and another agent $r_{i'}$ has goal $g_{j'}$ with $i \neq i'$ and $j \neq j'$ on top of its agenda. In case of a collision of the two, we again apply the argument from the ant-on-stick example. The two agents exchange their agendas, so that $r_{i'}$ next turns to the goal g_j with r_i next turns to goal $g_{j'}$. This is done for each of the agents at each collision, so that all goals are reached in polynomial time.

The actual implementation on graphs with nodes and edges aligns with the solution stages for AMAPF and looks as follows (we use m for the number of goals, n for the number of nodes in the graph, and $k \leq m$ for the number of robots).

As a first step, once more we compute the shortest-path reduction of the graph, starting from the goals' locations. With Dijkstra's algorithm for each goal in computing shortest paths, this step takes time $O(mn)$ in grid graphs. If we include the robots starting positions, this complexity increases to $((m+k)n)$ for grid graphs, and at most to $O((m+k)n^2)$ for general graphs.

As a second step, we again compute the matrix of pairwise distances between all destination locations, which is available in time $O((m + k)^2)$ using the precomputed information on the shortest paths, or $O(diam \cdot (m + k)^2) = O(n \cdot (m + k)^2)$ when using only the link information.

Next, we use a greedy vehicle routing solver for pairwise distances to find initial routes for each of the agents. Note that any assignment of goals to the agents will work, so this step can be obtained in linear time to the number of goals. However, for a minimized overall travel, we need an optimized assignment. Thus, we apply a time-limited Monte Carlo search, called a Nested Rollout with Policy Adaption (NRPA, see below).

With a level l search and iteration width i this step requires $O(i^l)$ rollouts and a (much) small number of policy adaptations. Each rollout runs in time $O((m + k)^2)$, with m in $(m + k)$ to traverse all potential successors and r in $(m + k)$ to detect the starting positions in each of the steps $m+k$. The adaptation procedure runs in $O((m+k)^2)$. Both procedures are independent of the initial size of the graph n. Although this step adds substantial running time for the computational complexity, we may consider the number of rollouts to be a constant controlled by the user.

When agents are assigned to tours, the actual simulation starts, which also resolved conflicts on-the-fly. The beauty of our approach is that it dynamically changes the goals for the agents. If a goal is reached, the corresponding agent continues with the next one, so that at each point in time, each agent has its goal agenda and the next goal to visit for which it can retrieve the shortest path for to navigate trough the graph.

Again, we cover the subtlety of agents having already arrived at their goal location, which we resolve via cuckoo'ing. Agent-agent interactions are even rarer, and are resolved with the same strategy of waiting on node conflicts, and exchanging agendas on edge conflicts.

Following the AMPF time complexity derivations, the simulation finishes in $O((m + k)n^2)$ time.

In summary, we obtain the following result, which adapts the goal agendas of the individual robots on-the-fly. With the NRPA algorithm, the constructed solution will have a quality optimized solution according to the stated optimization criterion. By the usually smaller number of agents than the number of goals to visit, the obtained CMAPF solutions have a much higher number of steps than for the corresponding AMAPF.

Theorem 2 (Polynomial-Time Solubility of CMAPF)
Let m be the number of goals, n be the number of nodes on the graph, and $k \leq m$ the number of agents. The CMAPF problem can be solved in polynomial time in n, k, and m. For computing an initial tour assignment and at most $O((k + m) \cdot n^2)$ computation for the rest to generate optimized solutions. The space requirements are dominated by the tables that store the shortest path information, which is of size $O((k + m) \cdot n)$.

The proposed algorithm includes a user-parameterized NRPA solver with time $O((m + k)^2 \cdot z)$, where z is the number of rollouts.

As optimal CMAPF generalizes a TSP, we cannot expect optimal solutions to be found in polynomial time.

We implemented the CMAP solver for grid graphs. While we utilize a queue for BFS which runs in linear time, it is not difficult to extend the algorithm with a priority queue to Dijkstra's algorithm. Note that we do not stop the backward exploration at any node and instead solve the single-source all-targets shortest-path problem. The result of the exploration is stored together with a goal and queried for the agents' current location.

As the CMAPF problem optimally is computationally hard, for a solvable instance, we will find an approximate solution without collision. For the initial assignments of goals to the agents we use the NRPA algorithm, which is an anytime solution and can be controlled by its parameters: the number of iterations (the width of the recursion) and the level of the search (the depth of the recursion tree). The implementation of the recursive search procedure for the shortest-path reduced CMAPF setting is given in Figure 4.

```
Pair search(int level) {
  Pair best; best.score = MAXVALUE;
  if (level == 0) {
    best.score = rollout();
    for (int j = 0; j < GOALS+AGENTS; j++)
      best.tour[j] = tour[j]; }
  else {
    for(int i = 0; i < GOALS; i++)
      for(int j = 0; j < GOALS; j++)
        backup[level][i][j] = global[i][j];
    for(int i=0; i<ITERATIONS; i++) {
      Pair r = search(level - 1);
      if (r.score < best.score) {
        best.score = r.score;
        for (int j = 0; j < GOALS+AGENTS; j++)
          best.tour[j] = r.tour[j]; }
        adapt(best.tour,level);}}
    for(int i = 0; i < GOALS; i++)
      for(int j = 0; j < GOALS; j++)
        global[i][j] = backup[level][i][j]; }
  return best; }
```

Figure 4: Main search procedure for computing initial CMAPF agendas in NRPA.

```
void adapt(int* tour, int level) {
  for (int j=0;j<GOALS;j++) visits[j] = false;
  visits[0] = 1;
  int successors, node = 0;
  for(int j=0; j<GOALS; j++)  {
    succs = 0;
    for(int i = 0; i < GOALS; i++)
      if (check(i)) moves[succs++] = i;
    double factor = 1.0;
    if (node == 0) factor /= AGENTS;
    double z = 0.0;
    for(int i=0; i<succs; i++)
      z += exp(global[node][moves[i]]);
    for (int i=0; i<succs; i++)
      backup[level][node][moves[i]] -= factor *
        exp(global[node][moves[i]])/z;
    node = tour[j]; visits[node] = true; }}
```

Figure 5: Policy adaptation in NRPA.

The implementation of the randomized simulation procedure or rollout that is executed on each leaf of the motion tree is given in Figure 6. It computes the sum-of-costs of the individual routes taken and returns it as an evaluation. The procedure that adapts the policy and steers the random simulation is shown in Figure 5. The core implementation trick is to generate one tour for all of the agents in common and split it at the next obtained starting locations. When generating the combined tour, the distance from the last location of the current agent to the next agent is set to zero, and the makespan is reset to zero.

Experiments

We compiled our program under the Linux (Ubuntu 13.2.0-23ubuntu4) subsystem of a Windows 11 Pro (laptop) computer using gcc version 13.2.0 and ran the above algorithms on 1 core of AMD Ryzen 7 PRO 7940U.

```
double rollout() {
  for (int j=0;j<GOALS;j++)
    visits[j] = 1;
  visits[0] = true;
  tour[0] = 0; tourSize = 1;
  int node = 0, prev = 0;
  double makespan = 0, cost = 0;
  while(tourSize < GOALS) {
    double sum = 0;
    int successors = 0;
    for(int i = 0; i < GOALS; i++) {
      if (check(i)) {
        moves[successors++] = i;
        for (int j = 0;j < GOALS;j++) {
          if (i != j) {
            if (check(j)) {
              successors--; break; }}}}}
    if (successors == 0) {
      for(int i = 0; i < GOALS; i++)
        if(check(i))
          moves[successors++] = i; }
    for(int i=0; i<successors; i++) {
      value[i] = exp(global[node][moves[i]]);
      sum += value[i]; }
    double mrand = ((double) rand()/RAND_MAX)*sum;
    int i=0; sum = value[0];
    while(sum<mrand) sum += value[++i];
    prev = node; node = moves[i];
    tour[tourSize++] = node;
    visits[node]--;
    cost += d[prev][node];
    makespan = makespan + d[prev][node];
    for (int r = 0; r < AGENTS; r++) {
      if (start[r] == node) {
        cost -= d[prev][node];
        ms = std::max(makespan,ms);
        makespan = 0.0;
        break; }}}
  ms = smax(makespan,ms);
  return ms + cost/ROBOTS;
}
```

Figure 6: Policy adaptation procedure in NRPA.

Anonymous MAPF

For the first experiment, we selected the Boston map with 950 agents (see Figure 1) from the MAPF benchmark suite in Nathan Sturtevant's Moving AI Lab. With *Classical MAPF* we denote the fixed assignment given in the benchmark. With *Anonymous MAPF* we denote the MAPF after reassignments of the mapping of agents to the goals. For the time being, we assume that the agents apply the ant-algorithm and exchange their goals on a potential conflict. The results are shown in Table 1.

Problem	Sum-of-Cost	Makspan	Pot. Conflicts
Classical MAPF	227473	593	26420
Anonymous MAPF	19267	267	552

Table 1: Reduction of key performance indicators in a reassignment of goals by solving the assignment problem in the running example of the Boston map from Figure 1.

We see that there the makespan reduces to less than 1/2 (45%), and the sum-of-cost reduces to less than 1/10 (8.4%).

But the most impressive impact is that the reassignment reduces the number of potential collisions in the solution simulation, where the robot exchanges their respective goals to less than 1/20 (1.97%). Note that a number of conflicts that is half of the number of robots means that on average there is less than one potential collision per agent that were resolved in the simulation with cuckoo'ing and propagated delay.

Across the 500 benchmark instances for the selected 20 domain in Fig-7 we first display the number of agents (used in AMAPF) alias the number of goals (used in CMAPF).

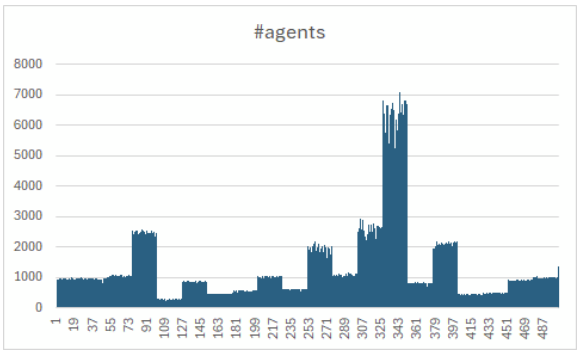


Figure 7: Total number of agents for computing and simulating AMAP in the 500 benchmark instances across 20 different domains.

Next, Figure 8 shows the reduction in sum-of-cost and Figure 9 the reduction in the makespan for AMAPF wrt. MAPF.

Figure 10 displays the potential conflicts observed during simulation. The gain in makespan is about 50%, while for the sum of cost we estimate a reduction of one order, and for the number of potential conflicts two orders of magnitudes.

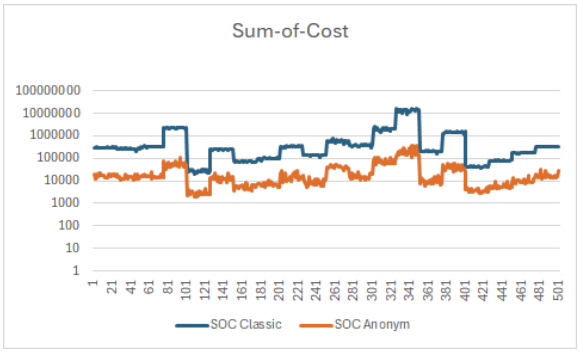


Figure 8: Comparing total travel distances of MAPF and AMAPF in the 500 benchmark instances on logarithmic scale.

The running time including the simulation was mostly below 1 second (see Figure 11) and around 15 seconds for the largest maps with several thousands of robots. If we assume that robots take more time for each simulation step, the investment of additional efforts to calculate a better goal assignment for the agents pays off.

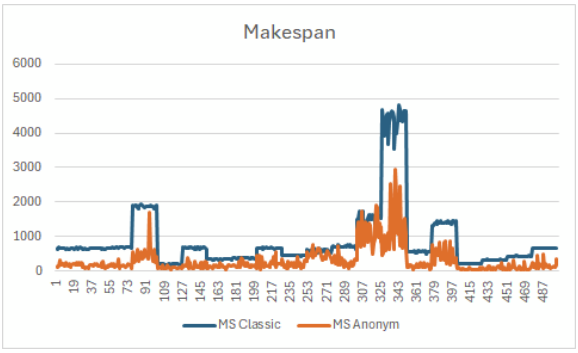


Figure 9: Comparing maximum travel distance of MAPF and AMAPF in the 500 benchmark instances.

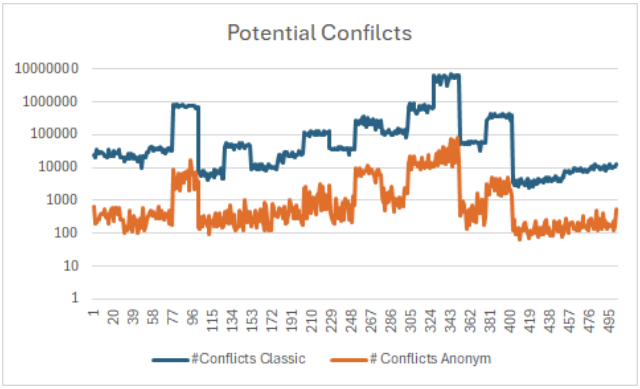


Figure 10: Comparing potential collisions of MAPF and AMAPF in the 500 benchmark instances on logarithmic scale. During simulation all conflicts are resolved on-the-fly.

Figure 12 shows the profile of the main steps in the algorithms over the set of reference instances. We see that the time for computing the pairwise shortest path matrix dominates the running time. In this case, we recomputed the length for each individual matrix entry. As described above and performed for the CMAPF experiment, it can be halved by storing the closed list for each agent with distances almost completely reduced. The main issue to recompute the distances on-the-fly was that keeping the closed list of the size of the grid, storing with the distances for each agent, on top of the shortest-path link information exceeded the memory of 32GB on our compute in the largest instances. As we were satisfied with the overall results of the experiments and the validation of conflict removal, we did not rerun it.

Combinatorial MAPF

It is possible to extend the above solution to the Combinatorial MAPF. The shortest path exploration and subsequent distance matrix between the goals and agents can be reused, so that individual tours for the agents are determined.

However, in this case, solutions to assignment problems are no longer sufficient, as they would lead to subtours that have to be eliminated. In fact, instead, we are solving the vehicle routing problem, which generalizes the salesman prob-

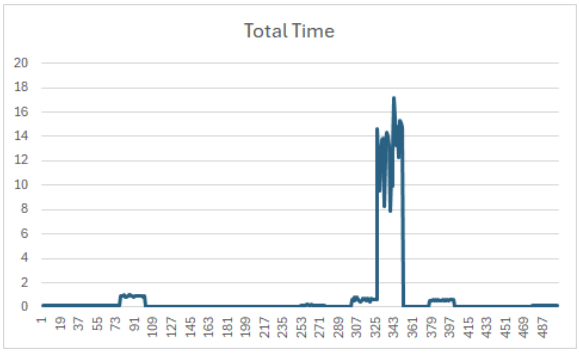


Figure 11: Total CPU Time for computing and simulating AMAP for the 500 benchmark instances [in s].

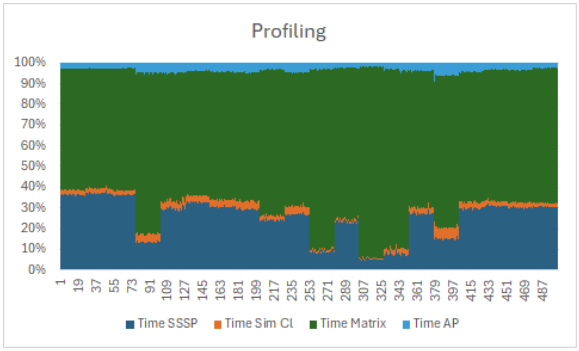


Figure 12: Running time profile of MAPF and AMAPF in the 500 benchmark instances.

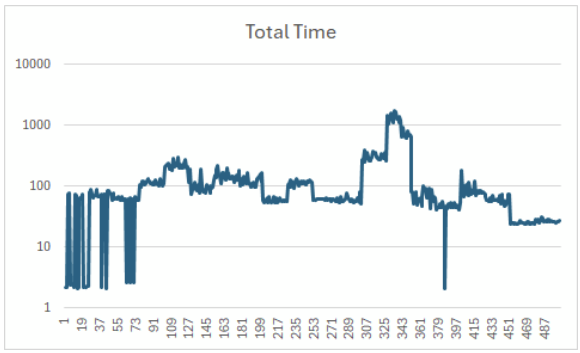


Figure 13: Total running time profile of CMAPF for the 500 benchmark instances [in s] (vertical axis on logscale!). Unsolvable instances are quickly found by the solver and manifest as spikes in plot.

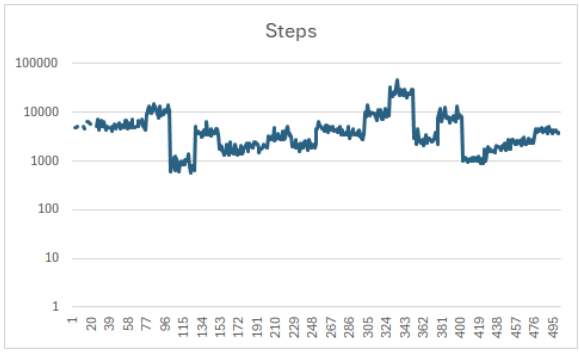


Figure 14: Number of agent steps needed to solve the problem (with unsolvable instances omitted from plot) in CMAPF for the 500 benchmark instances (vertical axis on logscale)

lem. Instead, we call NRPA with a fixed parameter setting to find an optimized solution to the tour assignment problem. In addition to the use for shortest-path navigation, the matrix for pairwise shortest paths of the goals was also used to bias the exploration of NRPA by presetting policy values.

We used a fixed number of k robots and decided to let them start at the first k target locations. Unfortunately, some of the goals in the domains are unreachable, so some of the problem instances become unsolvable. Using an NRPA with a limited number of rollouts, we solved each of the remaining benchmark instances in less than half an hour. The total running times are plotted in Figure 13.

There are different trade-offs between running time and solution quality. We chose a parameterization for NRPA to conduct the entire series of experiment in one day. We validated that given more time to carry out rollouts, the solutions improve. We also checked that, as expected, the quality of the solution is improving for a growing number of robots.

Conclusion

Anonymous MAPF and the Combinatorial MAPF are fascinating problems with several applications especially in indoor logistics. As we compute the assignments of goals to agents automatically, our approach also solves classical MAPFs and the regular MAPFs with multiple goals.

The approach is optimizing and not optimal, but the solutions based on an optimal shortest-path assignment for AMAPF are convincing, and for MAPF, the promising solutions can be further improved by investing more time in the NRPA optimization stage. This, in turn, can lead to an anytime algorithm behavior for CMAPF, while supposing that all precomputations have finished in start-up time, which even for a moderate number of robots and grid size is a matter of a few seconds.

As observed by (Mouratidis, Nebel, and Koenig 2024) even for the classical MAPF with multiple goals, proposed solutions like CBSS fail to compute optimal solutions for multi-goal MAPFs, as the underlying shortest-path search algorithm is not suited to chaining of the goals. There is a proposed fix, but the computational overhead for the combined search increases significantly.

This paper provides a polynomial-time collision-free solution for Anonymous and Combinatorial MAPF based on the ants-on-the-stick argument. While computing an optimal anonymous MAPF is polynomial, the computational complexity of the optimal combinatorial MAPF remains NP-hard.

We showed that a re-assignment of goals in the bench-

mark leads to a significant reduction in the makespan, and a reduction in the sum-of-cost in the order of about one magnitude. The number of potential conflicts shrinks by two orders of magnitude and can be resolved on-the-fly. The computational costs for the entire algorithm are moderate, and with the delegation of efforts to the initialization stage, allowing a fast and convincing solutions even for benchmark problems with thousands of agents on one core of a customary (laptop) computer.

In the future, we will include more realistic modeling aspects, such as robot rotation, as required for the 2024 League of Running Robots competition. With NRPA that generates solutions in a depth-first manner, our solution extends to other types of logistics settings, including pickup and delivery, as well as limited time and load (Edelkamp, Plaku, and Warsame 2019b). This work also shows that the above solutions can serve as a discrete abstraction for multi-robot multi-goal motion planning, where solutions are used as guidance for robot motions as implemented in the DROMOS framework (Plaku 2012; Edelkamp, Jabbar, and Lluch-Lafuente 2005; Plaku, Çela, and Plaku 2023). It is possible to extend the solution to distributed MAPF (Dergachev and Yakovlev 2024).

Acknowledgements

Thanks to Jáchym Herynek, Roman Barták, and Jiří Švancara for the discussion about this project. The presented work has been supported by the Czech Science Foundation (GAČR) under the research project number 22-30043S.

References

Andreychuk, A.; Yakovlev, K. S.; Surynek, P.; Atzmon, D.; and Stern, R. 2022. Multi-agent pathfinding with continuous time. *Artif. Intell.*, 305: 103662.

Applegate, D. L.; Bixby, R. E.; Chvatál, V.; and Cook, W. J. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press. ISBN 9780691129938.

Bachor, P.; Bergdoll, R.; and Nebel, B. 2023. The Multi-Agent Transportation Problem. In Williams, B.; Chen, Y.; and Neville, J., eds., *AAAI*, 11525–11532. AAAI Press.

Bullo, F.; Frazzoli, E.; Pavone, M.; Savla, K.; and Smith, S. L. 2011. Dynamic Vehicle Routing for Robotic Systems. *Proceedings of the IEEE*, 99(9): 1482–1504.

Dantzig, G. B.; and Ramser, J. H. 1959. The truck dispatching problem. *Management science*, 6(1): 80–91.

Dergachev, S.; and Yakovlev, K. S. 2024. Decentralized Unlabeled Multi-Agent Pathfinding Via Target And Priority Swapping. In Endriss, U.; Melo, F. S.; Bach, K.; Diz, A. J. B.; Alonso-Moral, J. M.; Barro, S.; and Heintz, F., eds., *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, 4344–4351. IOS Press.

Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2005. Cost-Algebraic Heuristic Search. In Veloso, M. M.; and Kambhampati, S., eds., *AAAI*, 1362–1367. AAAI Press / The MIT Press.

Edelkamp, S.; Plaku, E.; and Warsame, Y. 2019a. Monte-carlo search for prize-collecting robot motion planning with time windows, capacities, pickups, and deliveries. In *KI*, 154–167. Springer.

Edelkamp, S.; Plaku, E.; and Warsame, Y. 2019b. Monte-Carlo Search for Prize-Collecting Robot Motion Planning with Time Windows, Capacities, Pickups, and Deliveries. In *KI 2019: Advances in Artificial Intelligence*, volume 11793 of *Lecture Notes in Computer Science*, 154–167. Springer.

Herynek, J.; and Edelkamp, S. 2024. Multi-Robot Multi-Goal Mission Planning in Terrains of Varying Energy Consumption. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Jonker, R.; and Volgenant, A. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4): 325–340.

Kiesel, S.; Burns, E.; Wilt, C.; and Ruml, W. 2012. Integrating vehicle routing and motion planning. In *Twenty-Second International Conference on Automated Planning and Scheduling*.

Kornhauser, D.; Miller, G. L.; and Spirakis, P. G. 1984. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *25th Annual Symposium on Foundations of Computer Science*, 241–250. IEEE Computer Society.

Kuhn, H. W. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1–2): 83–97.

Liangou, T.; and Dentsoras, A. 2021. Optimization of motion and energy consumption of an industrial automated ground vehicle. In *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–7. IEEE.

Mouratidis, G.; Nebel, B.; and Koenig, S. 2024. Fools Rush in Where Angels Fear to Tread in Multi-Goal CBS. In Felner, A.; and Li, J., eds., *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, 243–251. AAAI Press.

Nebel, B. 2023. The Small Solution Hypothesis for MAPF on Strongly Connected Directed Graphs Is True. In Koenig, S.; Stern, R.; and Vallati, M., eds., *ICAPS*, 304–313. AAAI Press.

Perez, D.; Powley, E.; Whitehouse, D.; Samothrakis, S.; Lucas, S.; and Cowling, P. I. 2014. The 2013 Multi-objective Physical Travelling Salesman Problem Competition. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2314–2321.

Plaku, E. 2012. Motion Planning With Differential Constraints as Guided Search Over Continuous and Discrete Spaces. In *Symposium of Combinatorial Search (SOCS)*, 171–172. AAAI Press.

Plaku, E.; Çela, A.; and Plaku, E. 2023. Robot Path Planning with Safety Zones. In Gini, G.; Nijmeijer, H.; and Filev,

D. P., eds., *Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2023, Volume 1*, 405–412. SCITEPRESS.

Ren, Z.; Rathinam, S.; and Choset, H. 2023. CBSS: A new approach for multiagent combinatorial path finding. *IEEE Transactions on Robotics*.

Röger, G.; and Helmert, M. 2012. Non-Optimal Multi-Agent Pathfinding Is Solved (Since 1984). In *Multiagent Pathfinding, Papers from the 2012 AAAI Workshop*, volume WS-12-10 of *AAAI Technical Report*. AAAI Press.

Rosin, C. D. 2011. Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In *IJCAI*, 649–654. IJCAI/AAAI.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019a. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, (SOCS)*, 151–158. AAAI Press.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019b. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Thorup, M. 1997. Undirected Single Source Shortest Path in Linear Time. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, 12–21. IEEE Computer Society.

Wang, K. C.; and Botea, A. 2014. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *CoRR*, abs/1401.3905.

Wilson, R. M. 1974. Graph puzzles, homotopy, and the alternating group. *J. Combinatorial Theory Ser.*, B16: 86–96.

Yu, J.; and LaValle, S. M. 2013. Multi-agent Path Planning and Network Flow. In Frazzoli, E.; Lozano-Perez, T.; Roy, N.; and Rus, D., eds., *Algorithmic Foundations of Robotics X*, 157–173. Berlin, Heidelberg: Springer Berlin Heidelberg.

Zacharia, P. T.; and Xidias, E. K. 2020. AGV routing and motion planning in a flexible manufacturing system using a fuzzy-based genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 109(7): 1801–1813.

Zhang, H.; Chen, J.; Li, J.; Williams, B. C.; and Koenig, S. 2022. Multi-agent path finding for precedence-constrained goal sequences. In *International Conference on Autonomous Agents and Multiagent Systems*, 1464–1472.