





Towards a Complete Multi-agent Pathfinding Algorithm for Large Agents

Stepan Dergachev^{1,2}(✉)  and Konstantin Yakovlev^{1,2} 

¹ National Research University Higher School of Economics, Moscow, Russia
sadergachev@edu.hse.ru

² Federal Research Center for Computer Science and Control of Russian Academy of Sciences, Moscow, Russia
yakovlev@isa.ru

Abstract. Multi-agent pathfinding (MAPF) is a challenging problem which is hard to solve optimally even when simplifying assumptions are adopted, e.g. planar graphs (typically – grids), discretized time, uniform duration of move and wait actions etc. On the other hand, MAPF under such restrictive assumptions (also known as the Classical MAPF) is equivalent to the so-called pebble motion problem for which non-optimal polynomial time algorithms do exist. Recently, a body of works emerged that investigated MAPF beyond the basic setting and, in particular, considered agents of arbitrary size and shape. Still, to the best of our knowledge no complete algorithms for such MAPF variant exists. In this work we attempt to narrow this gap by considering MAPF for large agents and suggesting how this problem can be reduced to pebble motion on (general) graphs. The crux of this reduction is the procedure that moves away the agents away from the edge which is needed to perform a move action of the current agent. We consider different variants of how this procedure can be implemented and present a variant of the pebble motion algorithm which incorporates this procedure. Unfortunately, the algorithm is still incomplete, but empirically we show that it is able to solve much more MAPF instances (under the strict time limit) with large agents on arbitrary non-planar graphs (roadmaps) compared to the state-of-the-art MAPF solver – Continuous Conflict-Based Search (CCBS).

Keywords: Multi-agent systems · Coordination of multiple vehicle systems · Multi-agent path finding · Pebble motion · Large agents

1 Introduction

Multi-agent pathfinding (MAPF) is a challenging problem with topical applications in robotics, video games etc. There exist different ways to define the MAPF problem [8] and approaches to solve it. On the one hand, optimal and bounded sub-optimal solvers exist for what is known as Classical MAPF, like

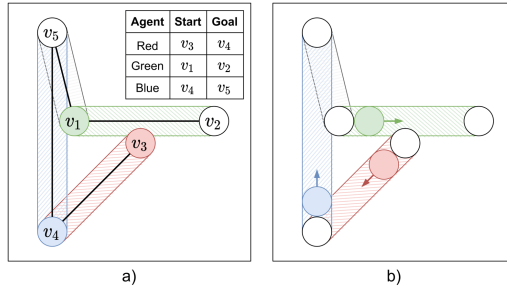


Fig. 1. An example of the multi-agent path finding for large agents (MAPF-LA) problem instance. It can only be solved if the synchronous moves of the agents are allowed (as shown on the right).

the ones presented in [2, 3, 6, 7]. On the other hand fast prioritized planners without any completeness/optimality guarantees are widespread [4, 11]. Finally, complete, non-optimal algorithms exist, such as the ones described in [5, 9], which borrow the solving techniques from the so-called Pebble Motion on Graph (PMG) problem. Still, all these algorithms do not consider the size/shape of the agents. Indeed, attempts to lift this restricting assumption are known [1, 10], however the algorithms known so far do not guarantee completeness. This work aims at drawing an attention to this gap and try narrowing it by adopting the PMG algorithms to the setting with the large agents, dubbed as MAPF-LA further on.

More specifically, we focus on one of the routines, regularly needed, in solving MAPF-LA instances – the one that makes it possible for one agent to traverse an edge without colliding to the other agents that prevent the transition due to their large bodies. We elaborate on how these agents can be safely moved away so the transition becomes valid. The suggested procedure was implemented and incorporated to the well-known PUSH AND ROTATE algorithm [5]. Unfortunately, the current variant of this algorithm is still incomplete for MAPF-LA. However, as we show in our empirical evaluation, it outperforms the state-of-the-art competitors, i.e. CCBS algorithms [1], in terms of number of solved MAPF-LA instances under the strict time limit.

The rest of this paper is organized as follows. We consider the definitions of MAPF-LA in Sect. 2. Section 3 describes a possible implementation of the procedure for moving away interfering agents, and also discuss cases when the proposed procedure is not sufficient to solve the planning problem. We report the results of an experimental evaluation in Sect. 4 and conclude in Sect. 5.

2 Problem Statement

Consider a tuple $(\mathcal{W}, G, r, K, start, goal)$, where $\mathcal{W} \subset \mathcal{R}^2$ is a metric workspace where K agents are operating. Each agent is modeled as a disk of radius $r > 0$. $G = (V, E)$ is a graph embedded in the workspace, i.e. each vertex $v \in V$ is associated with a point in \mathcal{W} . Edges correspond to the transitions between the

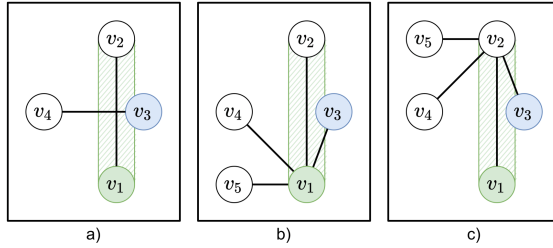


Fig. 2. An illustration of three cases that can occur when agents should be moved away from the edge along which the current agent needs to transit

locations. It is assumed that when moving along the edge the agent follows a straight-line segment connecting the corresponding vertices. $start : K \rightarrow V$ is a function that specifies the initial locations of the agents, $goal$ is the similar function specifying the target locations.

A set of K distinct graph vertices $S = (s_1, \dots, s_K)$, $s_i \in V, \forall i, j : s_i \neq s_j$ forms a *state*. Here s_i is a position of the i th agent. A state is valid iff $dist(s_i, s_j) \geq 2r$, where $dist$ stands for the Euclidean distance. In other words the state is valid if the bodies of the agents do not overlap. In this work we adopt an assumption that the distance between any two vertices of G is greater than $2r$. This infers that any state, as defined above, is valid.

A *transition* is formally a function $\pi(S, S') \rightarrow \{0, 1\}$, where $\pi = 1$ stands for the valid transition and $\pi = 0$ for the invalid one. Informally, transition corresponds to the movement of (some) agents between the locations in the workspace via the edges of the given graph. Conceptually, two possible assumptions regarding the transitions can be made:

- general: synchronous moves of the agents are allowed, i.e. more than one agent can change its location as a result of the transition
- restrictive: only one agent can change its location, while the rest stay at the same vertices

In this work we follow the second assumption. In such case for a transition $\pi(S, S')$ to be valid the following conditions should be met. First, the moving agent i may move only using one of the outgoing edges, i.e. $(s_i, s'_i) \in E$. Second, the move should be collision-free, i.e. $\forall j \neq i : segdist(e, s_j) > 2r$, where $segdist$ is the distance between the segment, defined by the edge e , and the vertex s_j .

The problem of multi-agent pathfinding for large agents (MAPF-LA) is now formulated as follows. Given $(\mathcal{W}, G, r, K, start, goal)$ find the sequence of states (S_0, S_1, \dots, S_n) s.t. $\forall s_i \in S_0 : s_i = start(i), \forall s_i \in S_n : s_i = goal(i), \forall i = 0, \dots, n-1$ a valid transition $\pi(S_i, S_{i+1})$ exists. In other words, the problem is to find the sequence of moves for the agents that transfer them from their start locations to their goal locations, while avoiding the collisions.

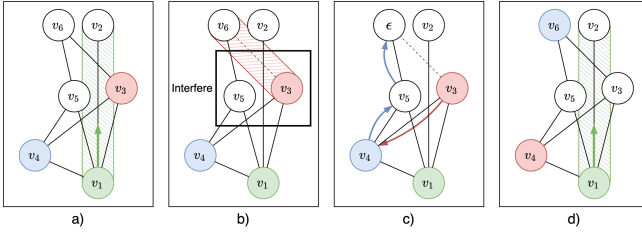


Fig. 3. Illustration of execution of PUSHTOEMPTY procedure. (a) The green agent must move along the edge (v_1, v_2) , but the red agent interferes with it. (b) First of all, we mark all edges that are interfered with by the vertex v_2 as untraversable and find all interfering with (v_1, v_2) vertices. (c) Single empty and non-interfering with (v_1, v_2) vertex v_6 to which there is a path not through v_1 or v_2 remain on the graph. (d) After all, pushing the red and blue agents along the path between v_3 and v_6 . Thus, the green agent can move along the edge (v_1, v_2) (Color figure online)

Example. An illustrative example of the considered problem is depicted in Fig. 1. Here the graph G consists of 5 vertices and 4 edges. The start and goal locations of the 3 agents are specified on the left part of the figure. Basically, each agent has to transfer to the adjacent vertex. However, under the considered assumption that only one agent moves at a time, the instance is not solvable. Meanwhile, if synchronous moves are allowed the problem is trivially solved via a single transition in which the agents simultaneously move to the adjacent vertices (as shown on the right). This highlights how defining the transition influence the possibility to find a solution. This is similar to the pebble motion on graphs problem, in which different assumptions regarding cycle-moves and chains-moves can be adopted, see [12] for details.

3 Suggested Approach

3.1 Preliminaries

The problem considered in this work, multi-agent pathfinding for large agents (MAPF-LA), is similar to the pebble motion on graphs (PMG) problem [12]. However the crucial differences exist, that prevent the straightforward application of the known methods to solve PMG for the considered formulation. Generally, two core differences between the MAPF-LA and PMG are as follows. First, in PMG every placement of pebbles (agents) on distinct graph vertices by definition form a valid state. In MAPF-LA this is not the case, however we adopted a (restrictive) assumption that the vertices of the given graph are located $2r$ distance units away from each other. Thus every placement of (large) agents on disjoint vertices also result in the valid state.

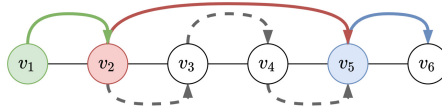


Fig. 4. Illustration of execution of PUSHALONGPATH procedure

Second, in PMG for a transition of one agent to be valid it is only required that the source vertex is free. In MAPF-LA this is not enough as the target vertex of the move can be free, but the moving agent, say a , may collide with the other agents while traversing an edge, as these agents stay too close to the edge itself. Such agents can be referred to as the *interfering* agents w.r.t. to the given edge $e = (v_{from}, v_{to})$. Thus, in order to reduce (our formulation of) MAPF-LA to PMG the following problem should be solved. Given a valid state S and an agent a that needs to traverse the edge $e = (v_{from}, v_{to})$ find a sequence of the valid transitions $(\pi_0, \dots, \pi_k, \dots, \pi_m)$ that results in the state S' , where the agent a is located at v_{to} and all other agents occupy the same vertices as in S . Here the transition π_k corresponds to the move of the agent a , while the other transitions are moves of the other agents which are made, first, in order to remove the interfering agents so the move through e is possible, and, second, move all agents (except a) back their vertices.

Lets denote the procedure that solves the described problem as MOVE-LA. Next we elaborate on how this procedure can be constructed. Generally, after MOVE-LA is defined, one can use one of the PMG algorithms to solve MAPF-LA. E.g., in this work we use PUSH AND ROTATE ALGORITHM [5], however other choices are possible.

3.2 Moving Along an Edge

A possible implementation of the MOVE-LA procedure consist of the three following steps. The first step is the sequential removal of every agent a' from every vertex v' that interfere with the move along e . This can be done via finding a path (for each interfering agent) to an unoccupied vertex and, then, sequentially moving the agent along this path (PUSH operation). The second step is to move the a along the e . The third step is to return every a' to its initial vertex. However, this should be done in such a way that agent a remains at v_{to} . We identify three different cases that may arise while removing an interfering agent a' (Fig. 2).

In the first case, a path to some empty vertex can be found for agent a' that does not go through the vertices v_{from} and v_{to} , as well as through the edges for which vertices v_{to} are interfering. Thus, agent a' can be pushed along such a path, and the resulting sequence of moves can be reversed at the end of the MOVE-LA operation to return agent a' to the initial position. An example of such a case is shown in Fig. 2a, where the green agent needs to move along the (v_1, v_2) . To do this, it is enough to move the blue agent to vertex v_5 , and after passing the green agent, return the blue one to v_3

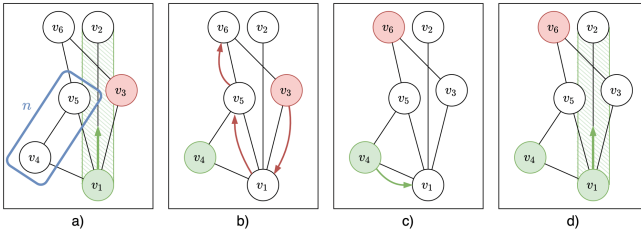


Fig. 5. Illustration of execution of PUSHTHROUGHVFROM procedure. (a) The green agent must move along the edge (v_1, v_2) , but the red agent on the v_5 vertex interferes with it. There are no paths between v_5 and non-interfering vertices, that do not pass through v_1 . So, it is necessary to move the green agent at one of the neighboring vertices so that the red agent can pass (b) After that, it is necessary to move the interfering agent from its position using the PUSHALONGPATH procedure (c) Next, the green agent can return to its original position v_1 (d) Thus, the green agent can move along the edge (v_1, v_2) (Color figure online)

In the second case, the path of a' can go through the vertex v_{from} , but not through the edges for which vertices v_{to} are interfering. However, for this, it is necessary to first push agent a from vertex v_{from} . Then we need to push a' to some non-interfering vertex. After the pushing of agent a' completes, agent a must be returned. The sequence of moves obtained in this way can be also reversed to return all agents to their positions, except for some actions that must be ignored when carrying out the reverse operation.

For the green agent on Fig. 2b, which illustrates the second case, to make the required move, it must first go to vertex v_5 . After that, the blue agent will be able to go to vertex v_4 , and the green agent can return to v_1 and move to v_2 . Finally, the resulting solution can be reversed after deleting the moves of the green agent.

In the third case, to remove the agent from the interfering vertex, it is necessary to find a path through v_{to} (and optional through v_{from} , but not through the edge e itself), or through the edges, for which vertices v_{to} (and optional v_{from}) are interfering. In this case, it is impossible to return the agents by a reversal, since by the time this operation is performed, the vertex v_{to} will be occupied by agent a , which will lead to a collision.

In the case on Fig. 2c, the blue agent can be moved to a non-interfering vertex only when passing through v_2 . Thus, after the green agent moves to v_2 , the blue agent can return to its original position v_3 only if the green agent misses it (e.g., when the blue agent leaves to v_4 , the green agent must go to v_5 for the blue one to return to v_3).

We propose PUSHTOEMPTY and PUSHTHROUGHVFROM procedures that can be used to solve the first and the second case respectively. Let's consider these procedures in more detail.

Algorithm 1 REVERSABLEEDGECLEANING procedure

Input: G – graph, S – current state, $e = (v_{from}, v_{to})$ – edge to clear,
 U – blocked vertices, r – agents radius

Output:

π – resulting solution, π_{rev} – the sequence of valid moves, which returns interfering agents back

```

1:  $\pi \leftarrow \square$ ,  $a \leftarrow$  agent, that need to move by edge  $e$ ,  $NotCleared \leftarrow \{\}$ 
2:  $I \leftarrow$  interfere vertices of  $e$ ,  $I_f \leftarrow$  unoccupied interfere vertices of  $e$ 
3: for all  $v' \in I \setminus I_{free}$  do
4:    $\pi' \leftarrow \square$ ,  $S' \leftarrow S$ 
5:   if PushToEmpty( $G$ ,  $S'$ ,  $\pi'$ ,  $v'$ ,  $U \cup \{v_{to}\}$ ,  $e$ ,  $I_f$ ,  $r$ ) then
6:      $\pi \leftarrow \pi + \pi'$ ,  $\pi_{rev} \leftarrow \pi_{rev} + \pi'$  except moves of  $a$ 
7:      $S \leftarrow S'$ ,  $I_f \leftarrow I_f \cup \{v'\}$ 
8:   else
9:      $NotCleared \leftarrow NotCleared \cup \{v'\}$ 
10: for all  $v' \in NotCleared$  do
11:    $a_e \leftarrow \{\}$ ,  $\pi' \leftarrow \square$ ,  $S' \leftarrow S$ 
12:    $\pi'' \leftarrow$  PushThroughVfrom( $G$ ,  $S'$ ,  $\pi'$ ,  $v'$ ,  $U$ ,  $e$ ,  $I_f$ ,  $r$ )
13:   if  $\pi'' \neq false$  then
14:      $\pi \leftarrow \pi + \pi''$ ,  $\pi_{rev} \leftarrow \pi_{rev} + \pi''$  except moves of  $a$ 
15:      $S \leftarrow S'$ ,  $I_f \leftarrow I_f \cup \{v'\}$ ,  $NotCleared \leftarrow NotCleared \setminus \{v'\}$ 
16:  $U \leftarrow U \setminus \{v_{to}\}$ ,  $\pi_{rev} \leftarrow$  Reverse( $\pi_{rev}$ )
17: if  $NotCleared$  not empty then
18:   return false
19: return  $\pi$ ,  $\pi_{rev}$ 

```

3.3 PushToEmpty

The procedure PUSHTOEMPTY is designed to remove interfering agent a' without affecting the vertex v_{to} and without pushing agent a' through v_{from} (Fig. 3a). First of all, mark all edges that are interfered by the vertex v_{to} as untraversable (Fig. 3b). This is necessary so that when the obtained solution is reversed, there are no conflicts with agent a who passed using the considered edge e . After that, all possible options are considered to move agent a' from the interfering vertex v' . For this, a set of empty, non-interfering with e vertices is formed. For each selected empty vertex ϵ , an attempt is made to find a path, avoiding using v_{from} and v_{to} (Fig. 3c) and pushing the agent a' along this path (using PUSHALONGPATH procedure). As a result, the agent a' from the interfering vertex is moved so that agent a can move along the edge e (Fig. 3d).

If the path to the vertex was found, but the PUSHALONGPATH operation failed, then the edge, through which the operation MOVE-LA inside PUSHALONGPATH could not be performed, is temporarily marked as untraversable and the path to ϵ has searched again. If the path to ϵ cannot be found, then the next empty vertex from the list is taken. A more detailed description of the procedure is provided in pseudocode in Appendix A.

PushAlongPath. Lets consider PUSHALONGPATH procedure. It consists of the sequential moving of agents along path to empty vertex starting from the last agent in the path. An important feature of this operation is that when passing a path through empty vertices, they will remain free after the end of the operation.

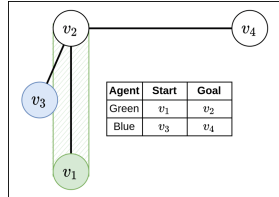


Fig. 6. An example of a case where it is necessary to correctly determine the planning order, even if there is a complete procedure for moving an agent along an edge (Color figure online)

An illustration of this operation is shown in Fig. 4. Agents (green, red and blue circles) must be pushed along path $v_1 - v_6$. In addition to the last vertex v_6 , the path also contains intermediate empty vertices v_3 and v_4 . The operation starts by moving the blue agent. He goes to vertex v_6 . All subsequent agents move to those vertices that were occupied by the previous ones (the red agent moves to the vertex v_5 passing through empty vertices, and the green agent moves to the vertex v_2). At the end of the operation, the first vertex of the path is freed because the last vertex of the path becomes occupied.

3.4 PushThroughVFrom

To solve the second case of removing agent a' from interfering vertex v' we suggest the PUSHTHROUGHVFROM procedure. It consists of two stages. At the first stage, agent a moves away to one of the neighbouring vertices n . If n is occupied, the operation of clearing the vertex is performed, similar to PUSHTOEMPTY.

If the operation of clearing of n was performed successfully, or if n was initially not occupied, then the operation MOVE-LA from v_{from} to n is performed. Note that if the edge e is marked as untraversable in the MOVE-LA operation, vertex v_{to} can be involved in it, since the obtained actions π'' will not be included in the REVERSE operation when the interfering agents return to their vertices. In the case when at least one of the operations described above fails, then an attempt is made to move agent a to another vertex n .

After the passage through v_{from} is cleared, an attempt is made to move agent a' away from v' also similarly to PUSHTOEMPTY. It is important to note that when creating the path of agent a' , it is necessary to block the vertices that are the current positions of the π'' participants. This is necessary to guarantee the return of agent a to the vertex v_{from} using the REVERSE operation.

If it was possible to find a path to an empty vertex for agent a' , after which the PUSHALONGPATH was successful, then the obtained solution is saved and

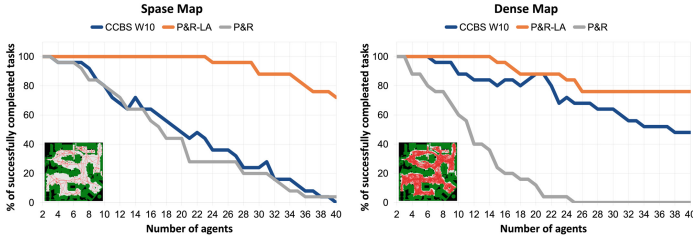


Fig. 7. Success rates of the suboptimal version of CCBS (CCBS W10), the suggested approach (P&R-LA) and “naive” version of the PUSH AND ROTATE for large agents (P&R) on two different roadmaps with varied number of agents

supplemented with reverse actions π'' to return agent a to vertex v_{from} . In addition, the solution is saved with the exclusion of action π'' , which is necessary for the further return of agent a' to vertex v' . A more detailed description of the procedure is also provided in pseudocode in Appendix A.

3.5 ReversibleEdgeCleaning

The sequential resolving algorithm of the first two cases can be combined into a single REVERSIBLEEDGECLEANING procedure (Algorithm 1), the result of which is the removal of interfering agents possible for considered cases, as well as a sequence of moves that will return all pushed agents to their positions after agent a passes along edge e .

The proposed algorithm consists of two stages. At the first stage, the procedure PUSHTOEMPTY is launched for each occupied vertex that interferes with the edge e (lines 6–12). If any of these vertices cannot be freed using procedure PUSHTOEMPTY, then they are stored in the *NotCleared* set (lines 12). After that, the procedure PUSHTHROUGHVFROM is executed for each element of the *NotCleared* set (lines 13–19).

3.6 Discussion

It should be noted that at the moment the proposed algorithm is not complete for MAPF for large agents problem, since it does not take into account two significant points.

The first point refers to the third case inside MOVE-LA procedure previously mentioned in the text. In this case, the path from interfering to the non-interfering vertex lies through the vertex v_{to} or the edges that this vertex is interfering with. Then there is no possibility of returning the interfering agent to its original position by reversing its actions. Thus, agent a must let agent a' go to its initial position v' , or another path must be found that leads agent a' to v' .

The second point is that may be the case that the interfering agent a' cannot return to the vertex v' at the end of MOVE-LA procedure (and MOVE-LA procedure fails), but still there exists a solution in which a' can be moved from vertex v' and not returned there. A trivial example of such a case is shown in Fig. 6. If the green agent has a higher priority during planning, then the blue agent cannot be pushed from vertex v_3 and returned there after moving the green agent. However, if planning starts with a blue agent, then a solution will be found. Thus, to construct a complete algorithm for MAPF-LA it is not enough to obtain a complete procedure for moving an agent along an edge. One of the possible solution to this problem is the choice of the correct planning order, as it is done in the PUSH AND ROTATE algorithm for solving not bi-connected instances.

4 Experimental Evaluation

We incorporated the suggested procedures to the well-known MAPF algorithm PUSH AND ROTATE, so it can be used to solve MAPF-LA instances. We denote it as P&R-LA. As one of the baselines we have also implemented a “naive” version of the PUSH AND ROTATE for large agents that simply halts when trying to move an agent along an edge with other interfering agents present. This version is denoted as, simply, P&R. Finally, the main baseline we were comparing with was the well-known CCBS algorithm [1] (we used the official implementation available on Github).

We evaluated planners on two different graphs (roadmaps) which were used in the CCBS paper: sparse and dense. The sparse roadmap contains 158 vertices and 349 edges, while the dense one – 878 vertices and 7341 edges. Originally, these roadmaps were automatically generated based on the `den520d` map from the MovingAI MAPF benchmark set [8]. An illustration of the roadmaps is shown in Fig. 7.

For each roadmap 25 scenarios were created, each one involving with 40 non-overlapping start and goal vertices. In each scenario, the first n start-goal pairs were selected, and then the evaluated algorithm was launched with a time limit of 30 s. In the experiment, the value of n varied from 2 to 40. We also set the sub-optimality factor for CCBS to 10, which notably speeds up the search (this feature is not described in the original paper, however is supported in the authors’ code).

The resultant success rates, i.e. the fractions of the solved instances per fixed number of agents, are presented in Fig. 7 (the higher - the better). As one can see, our modification of PUSH AND ROTATE outperforms the competitors, especially when the number of agents goes up. E.g. our algorithm managed to solve 80% of tasks with 40 agents on the sparse roadmap, while the success rate of both competitors was close to zero. The reason why CCBS failed almost always is that the density of the agents, i.e. the ratio of agents’ number to the number of graph vertices/edges is high and no easy solution is possible. This explains why CCBS copes better with the same number of agents on the dense roadmap – here there are much more graph vertices/edges that the algorithm can use

to find non-conflicting plans. Notably, the reasons why our algorithm failed to solve instances involving large number of agents differ for different maps. In sparse roadmap it terminated without providing a solution, while in the dense one it was not able to finish within the time limit. Thus, we infer, that a more efficient implementation of our method can actually provide a better success rate on the dense map.

5 Conclusion

In this work, we have considered the problem of designing a complete algorithm for a challenging variant of the multi-agent pathfinding problem when the size of the agents has to be taken into account (MAPF-LA). We elaborate on one of the core procedures such algorithm should incorporate, i.e. the procedure that clears an edge to allow for one agent to use this edge for a safe transition. We embed our implementation of this procedure to the well-known MAPF algorithm PUSH AND ROTATE enabling it to solve MAPF-LA problems. The results of the empirical evaluation provided us with the clear evidence that the resultant algorithm is able to find solutions for non-trivial MAPF-LA instances involving dozens of agents under the strict time limits (while its competitors often fail to do so).

Indeed, the main direction of future research is to develop a provably complete algorithm for solving MAPF-LA, as the method proposed in this work is only a step towards this goal (as it does not guarantee completeness).

A PushToEmpty and PushThroughVfrom procedures

Algorithm 2 PUSHTOEMPTY procedure

Input: G – roadmap, S – current state, π – solution, U – blocked vertices, v' – interfering vertex, $e = (v_{from}, v_{to})$ – edge to clear, r – agents radius, I_f – unoccupied vertices interfering to e .

```

1:  $E \leftarrow$  edges for which vertex  $v_{to}$  is interfering,  $G' \leftarrow$  remove  $E$  from  $G$ 
2: for all  $\epsilon$  in  $\text{EmptyVertices}(G, S) \setminus I_f$  do
3:   while true do
4:      $p \leftarrow$  path from  $v'$  to  $\epsilon$  in  $G' \setminus U \cup \{v_{from}\}$ 
5:     if  $p = \text{false}$  then
6:       break while
7:      $\pi' \leftarrow []$ ,  $S' \leftarrow S$ ,  $e_f \leftarrow \text{PushAlongPath}(G', S', \pi', p, U)$ 
8:     if  $e_f = \text{false}$  then
9:        $\pi \leftarrow \pi + \pi'$ ,  $S \leftarrow S'$ , return true
10:    else
11:      Remove edge  $e_f$  from  $G'$ 
12: return false

```

Algorithm 3 PUSHTHROUGHVFROM procedure

Input: G – roadmap, S – current state, π – solution, U – blocked vertices, v' – interfering vertex, a' – agent from v' , $e = (v_{from}, v_{to})$ – edge to clear, r – agents radius, I_f – unoccupied vertices interfering to e .

```

1:  $E \leftarrow$  edges for which vertex  $v_{to}$  is interfering
2: for all  $n \in \text{Neighbours}(G, v_{from}) \setminus U \cup \{v'\}$  do
3:    $G' \leftarrow$  remove edges  $E$  from  $G$ ,  $e_f \leftarrow$  false,  $p \leftarrow$  true
4:   if  $n$  is occupied then
5:     for all  $\epsilon \in \text{EmptyVertices}(S) \setminus (I_f \cup \{v_{to}\})$  do
6:        $e_f \leftarrow$  false
7:       while true do
8:          $S' \leftarrow S$ ,  $\pi' \leftarrow []$ ,  $p \leftarrow$  path from  $n$  to  $\epsilon$  in  $G' \setminus U \cup \{v_{from}, v_{to}, v'\}$ 
9:         if  $p =$  false then
10:          break while
11:          $e_f \leftarrow \text{PushAlongPath}(G', S', \pi', p, U \cup \{v_{to}\})$ 
12:         if  $e_f =$  false then
13:          break for
14:          Remove edge  $e_f$  from  $G'$ 
15:         if  $p =$  false or  $e_f \neq$  false then
16:          continue with next  $n$ 
17:          $G'' \leftarrow$  remove edge  $e$  from  $G$ ,  $\pi'' \leftarrow []$ 
18:         if not move-la( $G'', S', \pi'', v_{from}, n, U, r$ ) then
19:          continue with next  $n$ 
20:          $P \leftarrow I_f \cup$  vertices of  $\pi'' \cup$  interfere vertices of  $(v_{from}, n)$ 
21:          $U' \leftarrow U \cup \{v_{to}\} \cup$  current positions in  $S'$  of agents from  $\pi''$ 
22:         for all  $\epsilon \in \text{EmptyVertices}(S) \setminus P$  do
23:            $e_f \leftarrow$  false
24:           while True do
25:              $S'' \leftarrow S'$ ,  $\pi''' \leftarrow []$ ,  $p \leftarrow$  path from  $v'$  to  $\epsilon$  in  $G' \setminus U'$ 
26:             if  $p =$  false then
27:              break while
28:              $e_f \leftarrow \text{PushAlongPath}(G', S'', \pi''', p, U \cup \{v_{to}\})$ 
29:             if  $e_f =$  false then
30:               $\pi \leftarrow \pi + \pi' + \pi'' + \pi'''$ , remove all moves of  $a'$  from  $\pi''$ 
31:               $\pi \leftarrow \pi + \text{reverse}(\pi''')$ , return  $\pi' + \pi'''$ 
32:             else
33:              Remove edge  $e_f$  from  $G'$ 
34: return false

```

References

1. Andreychuk, A., Yakovlev, K., Boyarski, E., Stern, R.: Improving continuous-time conflict based search. In: AAAI Conference on Artificial Intelligence, vol. 35, pp. 11220–11227 (2021)
2. Barer, M., Sharon, G., Stern, R., Felner, A.: Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In: SoCS 2014, vol. 2014-January, pp. 19–27 (2014)
3. Boyarski, E., et al.: ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In: IJCAI 2015, pp. 740–746 (2015)
4. Čáp, M., Novák, P., Kleiner, A., Selecký, M.: Prioritized planning algorithms for trajectory coordination of multiple mobile robots. IEEE Trans. Autom. Sci. Eng. **12**(3), 835–849 (2015)
5. De Wilde, B., Ter Mors, A.W., Witteveen, C.: Push and rotate: Cooperative multi-agent path planning. In: AAMAS 2013, vol. 1, pp. 87–94 (2013)
6. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multiagent path finding. Artif. Intell. **218**, 40–66 (2015)
7. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Meta-agent conflict-based search for optimal multi-agent path finding. SoCS **2012**(1), 97–104 (2012)

8. Stern, R., et al.: Multi-agent pathfinding: definitions, variants, and benchmarks. In: SoCS 2019, pp. 151–158 (2019)
9. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: ICRA 2009, pp. 3613–3619 (2009)
10. Walker, T.T., Sturtevant, N.R., Felner, A.: Extended increasing cost tree search for non-unit cost domains. In: IJCAI 2018, vol. 2018-July, pp. 534–540 (2018)
11. Yakovlev, K., Andreychuk, A., Vorobyev, V.: Prioritized multi-agent path finding for differential drive robots. In: ECMR 2019, pp. 1–6 (2019)
12. Yu, J., Rus, D.: Pebble motion on graphs with rotations: efficient feasibility tests and planning algorithms. In: Akin, H.L., Amato, N.M., Isler, V., van der Stappen, A.F. (eds.) Algorithmic Foundations of Robotics XI. STAR, vol. 107, pp. 729–746. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16595-0_42