This is an accepted version of the article: Pereyra, E., Araguás, G., and Kulich, M. (2018). **Path Planning for a Formation of Mobile Robots with Split and Merge**. *Modelling and Simulation for Autonomous Systems (MESAS 2017)*, 10756, which has been published in a final form at https://doi.org/10.1007/978-3-319-76072-8_4.

Path Planning for a Formation of Mobile Robots with Split and Merge

Estefanía Pereyra[1], Gastón Araguás[1] and Miroslav Kulich[2]

[1]Research Centre in Informatics for Engineering,
National Technological University,
Córdoba Regional Faculty, Argentina.
{mepereyra,garaguas}@frc.utn.edu.ar
[2]Czech Institute of Informatics, Robotics, and Cybernetics,
Czech Technical University in Prague, Czech Republic.
kulich@cvut.cz

Abstract. A novel multi-robot path planning approach is presented in this paper. Based on the standard Dijkstra, the algorithm looks for the optimal paths for a formation of robots, taking into account the possibility of split and merge. The algorithm explores a graph representation of the environment, computing for each node the cost of moving a number of robots and their corresponding paths. In every node where the formation can split, all the new possible formation subdivisions are taken into account accordingly to their individual costs. In the same way, in every node where the formation can merge, the algorithm verifies whether the combination is possible and, if possible, computes the new cost. In order to manage split and merge situations, a set of constrains is applied. The proposed algorithm is thus deterministic, complete and finds an optimal solution from a source node to all other nodes in the graph. The presented solution is general enough to be incorporated into high-level tasks as well as it can benefit from state-of-the-art formation motion planning approaches, which can be used for evaluation of edges of an input graph. The presented experimental results demonstrate ability of the method to find the optimal solution for a formation of robots in environments with various complexity.

Keywords: MPP, multi-robot, planning, dijkstra, voronoi, graph

1 Introduction

Recent advances in mobile robotics and deployment of robotic systems in many practical applications lead to intensive research of multi-robot systems and robot formations as their special case. One of the most studied topic deals with planning trajectory/path of a formation in an environment with obstacles, i.e. the problem, how to find a continuous collision-free motion of the formation through a known environment from a current configuration to a given final configuration. Besides optimization of some parameters of the solution (e.g., path distance, energy consumption, mission time), a shape and size of the formation is constrained and violation of these constrains is penalized.

Approaches to motion and path planning for multi-robot systems and robot formations can be classified into several categories. Behavior-based algorithms [4, 20, 18, 23] are decentralized and reactive, i.e. each robot is controlled individually, using only local information about its neighborhood. Robot's behavior is typically composed of several simple behaviors (e.g., separation, alignment, and cohesion in [20]), which describe basic actions. These approaches are easy to implement and applicable to large swarms. They, on the other hand, fail in finding a plan in complex environments and do not guarantee precise formation control. To deal with the first problem, several heuristic search based algorithms were introduced based on particle swarm optimization [3], genetic algorithms [19] or ant colony optimization [15]. Nevertheless, precise formation shape can not be still maintained.

In contrast, centralized approaches consider a formation as a single body and plan trajectories in a high-dimensional composite configuration space (CCS). Exact solutions [2] are complete, but their complexity is exponential in the dimension of CCS and therefore, methods based on sampling CSS were introduced. For example, a probabilistic road map with sampling strategies designed especially for multi-robot systems that enable fast coverage of the configuration space was presented in [7], while a generalization of rapidly exploring random trees (RRT) to a graph structure is introduced in [11].

Another research stream considers a leader-follower architecture. Besides other approaches, which compute leader's trajectory and find trajectories of followers relative to this trajectory [5, 6] or coordinate motion of robots on a preplanned paths [17, 13], a big class of algorithms employs a concept of artificial potential fields. As classical potential fields [22] tend to find a local optimum, Garrido et al. [9] employed the Voronoi Fast Marching method, which propagates a wave over a viscosity map for a leader. Trajectories of followers are then dynamically computed to keep desired nominal inter-robot distances using Fast Marching (FM) with incorporated potentials reflecting leader's path, obstacles and positions of other robots. The method produces paths with minimal Euclidean lengths and avoids local minima, but generated paths are not smooth and go too close to obstacles. This can be solved by Fast Marching Square (FM^2) [10], which modifies wave expansion by incorporating velocity maps. Moreover, FM^2 manages uncertainties in robot's positions, sensor noise, and moving obstacles. This was recently applied for formations of unmanned surface vehicles (ships) allowing to model their dynamic behavior [14]. Application of the Frenet-Serret frame to control orientation of a formation enabled path planning for formations of unmanned aerial vehicles in 3D environments [1].

The works mentioned above do not explicitly address splitting and merging of a formation during movement, although this is possible with some approaches. While centralized exact methods are computationally demanding and thus practically inapplicable for larger formations, random sampling are more promising. For example, the authors in [21] present combination of RRT and particle swarm optimization for cooperative surveillance and demonstrate splitting of a formation in a simplified scenario with a single obstacle. A reactive obstacle avoidance

with added rules for split and merge are introduced in [16], while extension of flocking behavior [20] with game-theoretic based reconfiguration is presented in [8].

The proposed approach can be seen as a part of a general hierarchical planning algorithm, which consists of a global path planner and a local motion planner. While the the global planner searches for a topological path of a formation and thus generates primary movement directions, the motion planner determines (based on a path found by the global planner) motion for particular robots in a formation. This combination prevents the whole planner to be trapped at a local minima and enables to compute feasible trajectories fast. Similar approach is described in [12], where the global planner constructs a partial Voronoi diagram on the fly and a memetic evolution algorithm is employed for motion generation along this diagram.

The key contribution of the paper lies in design of a novel algorithm for global path planning which extents the well known Dijkstra's algorithm to be applicable for multi-robot systems and which considers possible split and merge of a formation. This is in contrast with [12], which is primarily focused on a solution of local motion planning, while the global planner is simplified to assume a formation as a single point robot. On the other hand, motion planning is not addressed in the presented paper as some of the aforementioned approaches can be directly used for it.

The rest of the paper is organized as follows. The general overview of the approach is described in Section 2, while the proposed extension of Dijkstra's algorithm for robot formations is introduced in Section 3. In Section 4 we present and discuss some experimental results. Finally, Section 5 is dedicated to concluding remarks and future directions of the research.

2 Framework

The planning problem for a fleet of mobile robots can be generally understood as search for a continuous sequence of feasible configurations of the fleet from a start configuration to a given goal configuration. A feasible configuration is such a configuration, in which a fleet does not collide with a surrounding environment as well as robots in the fleet do not collide with each other. In many scenarios we don't ask for arbitrary sequence of configurations (trajectory), an optimal trajectory with respect to some criteria (e.g., a mission time) is required instead. Moreover, additional constrains to fleet's geometry may be applied. For example, robots may be requested to form and keep a specified shape or lattice or to move close together in order to ensure visibility or communication to their neighbours.

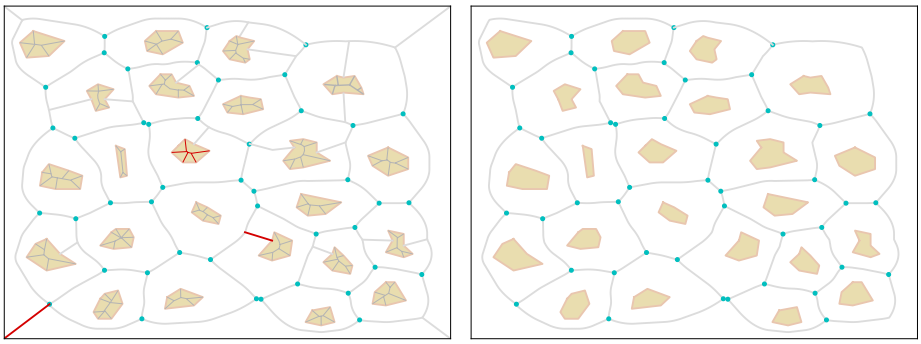
Given a polygonal representation of the environment, start and goal positions, and the number of robots R, the proposed solution works in several steps, as described in Algorithm 1. It starts with construction of a graph $G(V, E)$ (line 1), which is done in three steps. A Voronoi diagram of obstacles is generated first. As illustrated in Fig. 1(a), the original Voronoi diagram contains edges, which are inside obstacles or which are connected to some obstacle vertex. These edges

are thus removed together with nodes and edges forming tails, i.e. components of the graph, which can not be a part of any shortest path except paths originating or ending in these components (one of these components is highlighted red in Fig. 1(a)). The graph after this removal is shown in Fig. 1(b). If the goal or final positions are not in the graph, they are added into it finally.

Algorithm 1: The general planning framework

1 Construct a connected graph $G(V, E)$;
2 Evaluate each edge $e \in E$ accordingly to the number of robots;
3 For the given start node Q_{ini} compute a shortest path in $G(V, E)$ to each other node $v \in V$;
4 Generate motion along the found shortest path to the given goal node(s);

All edges of the graph are evaluated in the second step (line 2). A cost of a particular edges is a vector $\boldsymbol{c} = (c_1, c_2, \dots c_R)$, where c_k is the cost for a formation of k robots to traverse the edge. The cost of an edge can be either determined as the time needed to traverse the edge with some motion planning algorithm (e.g. [10]) in simulation or it can be approximated based on distance of the edge from the obstacle nearest to it. Some penalty can be also added to the cost for $k < R$ expressing that the formation is split. For simplicity the



(a) (b)

Fig. 1. Graph construction. (a) A Voronoi diagram (green) of obstacles (brown) and a rectangular border[2]. The red line represents a tail to be filtered. (b) The same after filtering of edges and vertices.

[2] We use implementation of a Voronoi Diagram (VD) from the Boost Polygon Library (`http://www.boost.org/doc/libs/1\_60\_0/libs/polygon/doc/voronoi\_diagram.htm`). The image was generated directly from the output of the library.

algorithm processes all nodes in the graph assuming that they have a degree less or equal to three, $\rho \leq 3$, that is, they have two or three connected edges. Nodes with $\rho > 3$ are substituted by an equivalent set of nodes-edges that meet this constraint. Nodes with $\rho = 4$ are substituted with two nodes, which are connected with an edge with cost equal to zero. Similarly, nodes with $\rho = 5$ are substituted with three nodes. Some examples of this substitution are shown in Fig. 4.

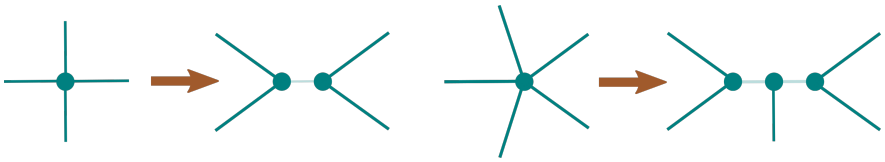


Fig. 2. Substitution of a node with a degree (ρ) higher than 3. A node with $\rho = 4$ is substituted with two nodes, which are connected with an edge with cost equal to zero (left). Similarly, node with $\rho = 5$ is substituted with three nodes (right). Edges with the zero cost are shown in the light color.

The proposed algorithm for formations, described later in Section 3, is run next (line 3), which for a given start computes shortest paths to all other nodes in $G(V, E)$ and all possible sizes of a formation. In other words, it will be possible to reconstruct an optimal path from the start to an arbitrary node Q_{goal} and the given number of robots $r \in \langle 1, R \rangle$ after application of this step. This path is a sequence of nodes $p = \{v_0 = v_{ini}, v_1, v_2, \ldots, v_n = Q_{goal}\}$ with the properties and constrains detailed in 2.1. A complete motion of a formation is generated finally, given the path p (line 4). Trajectories of robots in the formation are determined in this step so that relative positions of robots are computed with respect to geometrical constrains on the formation and to avoid nearby obstacles. Again, some motion planning algorithm can be employed to plan a motion between each two consecutive nodes of p.

2.1 Path properties and constrains

Given a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the working environment, with a set of vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_i\}$ and edges $\mathcal{E} = \{e_1, e_2, \ldots, e_j\}$, and a fleet of robots $\mathcal{R} = \{r_1, r_2, \ldots, r_R\}$, the corresponding collision-free paths for the fleet in the graph are denoted as $\mathcal{P} = \{p_1, p_2, \ldots, p_R\}$, with $p_i : \mathbb{Z}^+ \to \mathcal{V}$. Moreover, a path of the individual robot r_i is a sequence of vertices $p_i = \{v_{i_1}, v_{i_2}, \ldots v_{i_k}\}$ such that $(v_{i_j}, v_{i_{j+1}})$ is an edge of the graph. In order to compute the cost of the path another representation of the same path is used, where the number of robots passing through each edge is considered. In such representation, a path

Note that a Voronoi Diagram for polygon generally contains parabolic segments, which are approximated with line segments in the image.

of the individual robot r_i is a sequence of tuples, edges and a number of robots sharing each edge, $p_i = \{(e_{i1}, r_{e_{i1}}), (e_{i2}, r_{e_{i2}}), \dots (e_{ik}, r_{e_{ik}})\}$, where $(e_{ij}, e_{i(j+1)})$ are connected edges of the graph.

The feasibility of the path p_i is conditioned upon the following constrains: 1) initially, all the robots in the fleet are in the start position: $p_i(0) = Q_{\text{ini}}$, $\forall p_i \in \mathcal{P}$. 2) there exists a state in the path $k_{\min} \in \mathbb{Z}^+$ such that $p_i(k_{\min}) = Q_{\text{goal}}$, meaning that the robot r_i reaches the goal on the shortest possible path. 3) any two paths from $\mathcal{P}$ have not to be in a collision, i.e. given any pair of states $m, l \in \langle 0, k_{\min} \rangle$, two paths p_i, p_j are in collision if $(p_i(m), p_i(m+1)) = p_j(l+1), p_j(l))$. 4) given two paths p_i, p_j and two states $m, l \in \langle 0, k_{\min} \rangle$, if $p_i(m) = p_j(l)$ then $m = l \leftarrow max(m, l)$, i.e. the robot that arrives first to a vertex waits for the second one, in order to keep the formation joint as much time as possible.

3 Multi-robot path planning algorithm

Standard Dijkstra's algorithm finds the cheapest paths together with their costs from the given source node Q_{ini} to all other nodes in the graph $\mathcal{G}$ given costs of all graph edges. The algorithm stores for each node $v \in \mathcal{G}$ the minimum cost $C_{v_{\min}}$ to reach this node and the predecessor v_{prev} from where the node is reached. The shortest path for the given node v can be then easily determined by walking consecutively over predecessors starting in v. In the initialization stage, costs C_v of all the nodes are set to infinity and their predecessor v_{prev} to a fictive node $None$, what means that the shortest path has not been found yet. The only exception is the start node, whose cost value is set to 0. The start node is then put into a priority queue, which is sorted according to C_{v_i}. The algorithm then consecutively takes the nodes from the priority queue and for the current node u, their neighbors v_i are processed. For each neighbor v_i, the total cost C_{v_i} of arriving to it from u is computed. If v_i is reached for the first time by the algorithm, it is added into the priority queue with its total cost C_{v_i} corresponding of arrive to it from u, and u is assigned as its predecessor, $v_{\text{prev}} = u$. If it is not the case, the computed cost C_{v_i} is compared with the cost stored in the node and, if it is smaller, the staged cost is updated to C_{v_i}, v_{prev} is set to u, and v_i with its new cost is added to the priority queue.

For the multi-robot case, the cost of an edge can vary with the size of the formation traversing the edge, and thus, all possible combinations of formation sizes traveling through the node must be considered. This means that a node can not be processed at once, because it is not guaranteed that all needed information is available until all the robots are processed. The proposed algorithm for a formation is depicted in Algorithm 2. Here, a formation of R robots, a connected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, and a vector of costs $\boldsymbol{c}^e = (c_1^e, c_2^e, \dots, c_R^e)$ for each edge e, where c_r^e is the cost paid for traversing e with $r \in \langle 1, R \rangle$ robots are considered. Similarly to the standard Dijkstra's algorithm, the algorithm uses a priority queue $\mathcal{H}$ to select the next state to be processed. In our approach, a state is a data structure associated to a node v, given by $s_{\text{vr}} = \langle v, r_{\text{v}}, c_{\text{vr}}, p_{\text{vr}} \rangle$, where r_{v} is the number of robots arriving to the node, p_{vr} are the paths for each of this robots computed

Algorithm 2: Multi-robot path planning

Input: $\mathcal{G}(\mathcal{V}, \mathcal{E})$ - connected graph
R - number of robots
Q_{ini} - start node
Output: $\mathcal{P}$ - set of paths

```
1  foreach node v ∈ G(V, E) do
2      N.add(v)
3  foreach number of robots r ∈ {1, ..., R} do
4      foreach neighbour v of Q_ini do
5          c_vr ← compute_cost(v, r)
6          p_vr ← compute_path(v, r)
7          s_vr = ⟨v, r, c_vr, p_vr⟩
8          H.add(s_vr)
9          S.add(s_vr)
10 N.remove(Q_ini)
11 while N do
12     ⟨u, r_u, c_ur, p_ur⟩ ← s_ur = H.pop()
13     if p_ur ∉ P then
14         P.add(p_ur)
15     foreach number of robots r ∈ {1, ..., r_u} do
16         foreach neighbour v of u do
17             c_vr ← compute_cost(s_ur, v, r)
18             p_vr ← compute_path(s_ur, v, r)
19             s_vr = ⟨v, r, c_vr, p_vr⟩
20             if s_vr ∉ S then
21                 combine_states(s_vr, S)
22                 H.add(s_vr)
23                 S.add(s_vr)
24     if r_u = R then
25         N.remove(u)
```

from Q_{ini} to v, and c_{vr} is the highest cost corresponding to the worst paths of p_{vr}. A node can have a lot of associated states for the same number of robots, depending on the combination of paths used for each robot in the node. All of these states are stored in a state table $\mathcal{S}$ which will be used later to combine paths in the merge operation.

The algorithm starts with initialization of data structures (lines 1 – 10). First, all nodes in the graph are stored in a dynamic list $\mathcal{N}$. Then, the states associated to each neighbour of Q_{ini} considering a different number of robots $r \in \langle 1, R \rangle$ are generated (line 7) and stored in the queue (line 8) and in the state table (line 9). After that, the algorithm loops processing each of the states from the priority queue (line 11) until the optimal paths for the R robots from Q_{ini} to each node in the graph are computed.

The algorithm processes each state as follows. It takes the next state s_{ur} from the queue (line 12). If this state corresponds to an arrival of r robots to the node u from Q_{ini} for the first time, what means that it is the cheapest solution so far, it is stored in the set of optimal paths P (line 14). Then, considering the possibility of formation split, for each number of robots $r < r_{\text{u}}$, and for each neighbor v of u, a new state s_{vr} is generated (line 19). If this state does not exist in the state table $\mathcal{S}$, it is combined if possible with other states associated with the node v and stored in $\mathcal{S}$. In this step, the possibility of formation merge is considered by means of combination of states. Two states are combinable if they meet a set of rules, which will be described in section 3.1. Finally, if $r_{\text{u}} = R$, the node u is removed from the list $\mathcal{N}$ (line 25).

3.1 Merge of the formation

The process of states combination allows the algorithm to deal with formation merge. A state s_{vk} represents the arrival of k robots to the node v by means of their p_{vk} paths. The existence of a second state s_{vq} associated to the same node v, such that $k + q <= R$, allows the combination of both states which leads to generation of a new state $s_{\text{v(k+q)}}$. The new state represents the arrival of $k + q$ robots to the node v by means of a combination, that is a merge of $k+q$ robots of the formation. In order to evaluate whether a merge is possible, a set conditions must be satisfied. The fuction to combine two sattes is depicted in Algorithm 3. Given a state s_{vr}, the function tries to combine it with all other states in the state table S, ensuring that the number of robots after the combination remains less or equal to R (lines 2 - 3). Each possible combination is evaluated by three rules (lines 5 - 7), and if all of them are satisfied the new combined state is added to the queue $\mathcal{H}$ and state table $\mathcal{S}$ (lines 8 - 11).

Using the edge representation of paths,

$$p_i = \{(e_{i1}, r_{e_{i1}}), (e_{i2}, r_{e_{i2}}), \ldots, (e_{ik}, r_{e_{ik}})\},$$

two paths p_1 and p_2 must fulfill the following rules in order to be combined:
Shared edges rule This rule looks for edges common to both paths, and verifies whether those shared edges contain equal number of robots.

Algorithm 3: Function *combine_states()*

Input: s_{vr} - state
$\mathcal{S}$ - states table
R - number of robots
Output: New combined states

$p_{\mathrm{vr}}, r \leftarrow s_{\mathrm{vr}}$
foreach *number of robots* $i \in \{1, \ldots, R - r\}$ **do**
  foreach *state* $s_{\mathrm{vi}} \in \mathcal{S}$ **do**
    $p_{\mathrm{vi}} \leftarrow s_{\mathrm{vi}}$
    $passed = shared\_edges\_rule(p_{\mathrm{vi}}, p_{\mathrm{vr}})$
    $passed = shared\_nodes\_rule(p_{\mathrm{vi}}, p_{\mathrm{vr}})$
    $passed = cross\_edges\_rule(p_{\mathrm{vi}}, p_{\mathrm{vr}})$
    if *passed* **then**
      $s_{\mathrm{v(r+i)}} = \langle v, r + i, c_{\mathrm{v(r+i)}}, p_{\mathrm{v(r+i)}} \rangle$
      $\mathcal{H}.add(s_{\mathrm{v(r+i)}})$
      $\mathcal{S}.add(s_{\mathrm{v(r+i)}})$

Shared nodes rule This rule looks for nodes common to both paths, and verifies whether the number of robots arriving to the node is higher than or equal to the number of robots leaving the node.
Cross edges rule This rule avoids combinations where both paths contain the same edge but in opposite direction, given that these combinations don't meet the no collision constrain for paths (Section 2.1).

4 Experimental evaluation

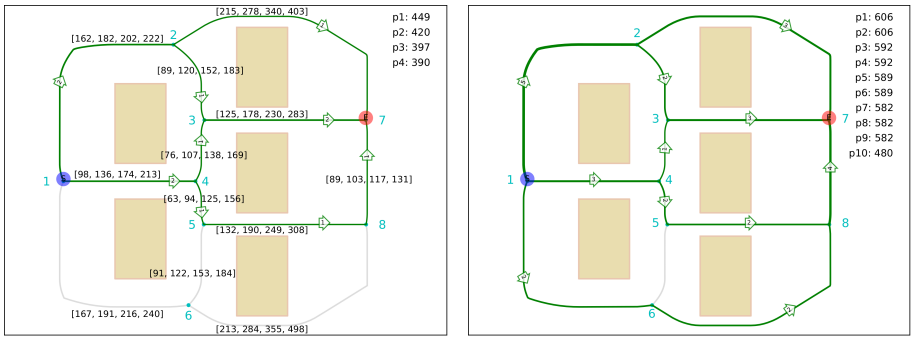
The proposed algorithm for multi-robot path planning has been implemented in Python 2.7. All the experiments were evaluated under the same conditions, using a notebook with an `Intel CORE i5` processor and 4GB of RAM, running a Debian GNU/Linux. Given that the algorithm is based in a Voronoi diagram of the map, it is complete; and since it is based on the standard Dijkstra, it is optimal too. In order to prove its optimality, the experiments were performed on a set of maps with various complexities from `http://imr.ciirc.cvut.cz/planning/maps.xml`. Nevertheless, because of the high complexity of the problem, the optimality of the solutions was possible to verify manually only for maps with a low number of nodes and with a low number of robots.

In the Fig. 3 an example of a path planning in a simple map with 8 vertices is shown. For each edge of the graph a vector of costs is determined by the

following equations,

$$
\begin{aligned}
e_{12} &= int(142.93 + r * 19.99) & e_{34} &= int(44.99 + r * 31.24) \\
e_{14} &= int(59.67 + r * 38.43) & e_{45} &= int(31.94 + r * 31.23) \\
e_{16} &= int(142.88 + r * 24.38) & e_{58} &= int(73.37 + r * 58.71) \\
e_{27} &= int(153.37 + r * 62.43) & e_{56} &= int(59.92 + r * 31.24) \\
e_{23} &= int(58.41 + r * 31.24) & e_{68} &= int(141.78 + r * 71.33) \\
e_{37} &= int(72.96 + r * 52.55) & e_{78} &= int(76.048 + r * 13.88)
\end{aligned}
$$

where $r \in 1, 2, \ldots, R$ is the number of robots travelling through the edge. All the robots of the formation are initially at the blue start node, the formation splits and merges at different nodes, following the green paths. The arrows in the graph represent the number of robots moving through each edge and the direction of the movement. Fig. 3a shows the solution for four robots. Note the



(a) (b)

Fig. 3. The optimal solutions computed by the algorithm for a formation consisting of four robots in (a), and ten robots formation in (b).

number in square brackets drawn near each edge which express vectors of costs computed by the equations presented before for $r = 1$ to $r = 4$. The optimal paths given by the algorithm are

$$
\begin{aligned}
p_1 &= \{1, 2, 3, 7\} & c_1 &= 449 \\
p_2 &= \{1, 4, 5, 8, 7\} & c_2 &= 420 \\
p_3 &= \{1, 2, 7\} & c_3 &= 397 \\
p_4 &= \{1, 4, 3, 7\} & c_4 &= 390,
\end{aligned}
$$

and the formation cost is then given by the worst cost of the set, that is the cost of the path p_1, $c_1 = 449$. In the Fig. 3b the solution for ten robots formation is

shown, in this case the solution found is

$$
\begin{array}{llll}
p_1 = \{1,6,8,7\} & c_1 = 606 & p_6 = \{1,4,5,8,7\} & c_6 = 589 \\
p_2 = \{1,6,8,7\} & c_2 = 606 & p_7 = \{1,2,7\} & c_7 = 582 \\
p_3 = \{1,2,3,7\} & c_3 = 592 & p_8 = \{1,2,7\} & c_8 = 582 \\
p_4 = \{1,2,3,7\} & c_4 = 592 & p_9 = \{1,2,7\} & c_9 = 582 \\
p_5 = \{1,4,5,8,7\} & c_5 = 589 & p_{10} = \{1,4,3,7\} & c_{10} = 480
\end{array}
$$

Although the complexity of this map is relatively low, the problem becomes quickly very complex as the number of robots grows and therefore it is not easy to prove the optimality of the generated solutions. In the same way, if the complexity of the map (i.e. the number of vertices) grows, the optimality of the solutions is very difficult to verify even with a low number of robots. For example, Fig. 4 shows a solution for a three robots formation in four complex maps, the map on 4a has a total of 77 nodes, the map on 4b has 62 nodes and the maps on 4c and 4d have 45 nodes.

5 Conclusion

A novel algorithm for multi-robot path planning was presented. The algorithm finds a path for each robot of a formation, from a given node to all other nodes in the graph, considering the possibility of split and merge of the formation. It is an extended version of the standard Dijkstra's algorithm, and it is therefore deterministic, complete and finds an optimal solution. The algorithm is computationally heavy, and finding solutions for more than ten robots in maps with more than twenty nodes is very time consuming.

In the future work, improvements in realization of the constrains for formation merge will be considered, because this is the most time consuming part of the algorithm. Naturally, some aforementioned motion planning approach for formations will be employed to form, together with the proposed solution, the integrated approach for formation planning. This will allow us to perform experiments in realistic setups in simulation and then with real robots.

ACKNOWLEDGMENT

The work of M. Estefanía Pereyra and R. Gastón Araguás has been supported by the "Multirrotores Autónomos para Aplicaciones en Ambientes Exteriores" project, U.T.N. PID UTI4534. The work of Miroslav Kulich has been supported by European Community's HORIZON 2020 Programme under grant agreement No. 688117 "SafeLog: Safe human-robot interaction in logistic applications for highly flexible warehouses".

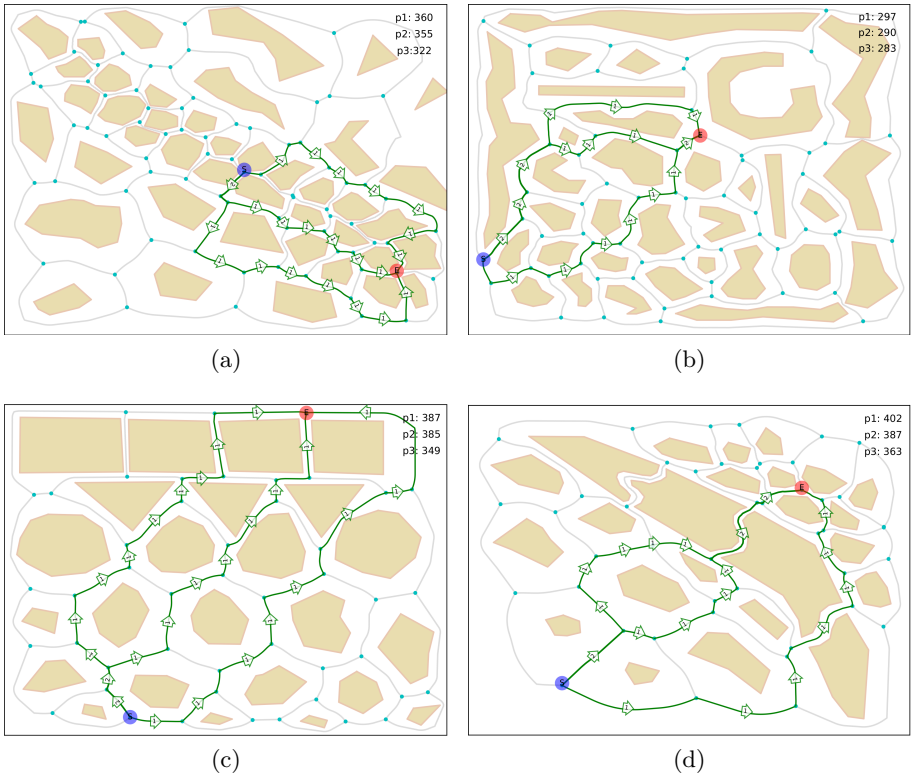


Fig. 4. The optimal solutions computed by the algorithm for a three robots formation in maps with various complexity.

References

1. Álvarez, D., Gómez, J.V., Garrido, S., Moreno, L.: 3D Robot Formations Path Planning with Fast Marching Square. Journal of Intelligent & Robotic Systems 80(3-4), 507–523 (Feb 2015)
2. Aronov, B., de Berg, M., van der Stappen, A.F., Švestka, P., Vleugels, J.: Motion planning for multiple robots. Discrete & Computational Geometry 22(4), 505–525 (1999)
3. Bai, C., Duan, H., Li, C., Zhang, Y.: Dynamic multi-uavs formation reconfiguration based on hybrid diversity-pso and time optimal control. In: Intelligent Vehicles Symposium, 2009 IEEE. pp. 775–779 (June 2009)
4. Balch, T., Arkin, R.: Behavior-based formation control for multirobot teams. IEEE Transactions on Robotics and Automation 14(6), 926–939 (1998)
5. Barfoot, T., Clark, C.: Motion planning for formations of mobile robots. Robotics and Autonomous Systems 46(2), 65–78 (Feb 2004)
6. Chen, J., Sun, D., Yang, J., Chen, H.: A leader-follower formation control of multiple nonholonomic mobile robots incorporating receding- horizon scheme. The International Journal of Robotics Research (2009)

7. Clark, C.M.: Probabilistic Road Map sampling strategies for multi-robot motion planning. Robotics and Autonomous Systems 53(3-4), 244–264 (Dec 2005)
8. Dasgupta, P., Cheng, K.: Robust multi-robot team formations using weighted voting games. In: Distributed Autonomous Robotic Systems, pp. 373–387. Springer (2013)
9. Garrido, S., Moreno, L., Lima, P.U.: Robot formation motion planning using Fast Marching. Robotics and Autonomous Systems 59(9), 675–683 (Sep 2011)
10. Gómez, J.V., Lumbier, A., Garrido, S., Moreno, L.: Planning robot formations with fast marching square including uncertainty conditions. Robotics and Autonomous Systems 61(2), 137–152 (Feb 2013)
11. Kala, R.: Rapidly exploring random graphs: motion planning of multiple mobile robots. Advanced Robotics 27(14), 1113–1122 (2013)
12. Lin, C.C., Chen, K.C., Chuang, W.J.: Motion Planning Using a Memetic Evolution Algorithm for Swarm Robots. International Journal of Advanced Robotic Systems p. 1 (May 2012)
13. Liu, S., Sun, D., Zhu, C.: Coordinated Motion Planning for Multiple Mobile Robots Along Designed Paths With Formation Requirement. IEEE/ASME Transactions on Mechatronics 16(6), 1021–1031 (Dec 2011)
14. Liu, Y., Bucknall, R.: Path planning algorithm for unmanned surface vehicle formations in a practical maritime environment. Ocean Engineering 97, 126–144 (Mar 2015)
15. Noormohammadi Asl, A., Menhaj, M.B., Sajedin, A.: Control of leader-follower formation and path planning of mobile robots using asexual reproduction optimization (aro). Appl. Soft Comput. 14, 563–576 (Jan 2014)
16. Ogren, P.: Split and join of vehicle formations doing obstacle avoidance. In: IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004. vol. 2, pp. 1951–1955 Vol.2. IEEE (2004)
17. Olmi, R., Secchi, C., Fantuzzi, C.: Coordination of multiple agvs in an industrial application. In: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on. pp. 1916–1921. IEEE (2008)
18. Pereira, G.A.S., Kumar, V., Campos, M.F.M.: Closed loop motion planning of cooperating mobile robots using graph connectivity. Robotics and Autonomous Systems 56(4), 373–384 (Apr 2008)
19. Qu, H., Xing, K., Alexander, T.: An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots. Neurocomputing 120, 509–517 (2013)
20. Reynolds, C.W.: Steering behaviors for autonomous characters. In: Game developers conference. vol. 1999, pp. 763–782 (1999)
21. Saska, M., Chudoba, J., Přeučil, L., Thomas, J., Loianno, G., Třešňák, A., Vonásek, V., Kumar, V.: Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance. In: Unmanned Aircraft Systems (ICUAS), 2014 International Conference on. pp. 584–595 (May 2014)
22. Zhang, M., Shen, Y., Wang, Q., Wang, Y.: Dynamic artificial potential field based multi-robot formation control. In: Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE. pp. 1530–1534 (May 2010)
23. Zhong, X., Zhong, X., Peng, X.: VCS-based motion planning for distributed mobile robots: collision avoidance and formation. Soft Computing (Mar 2015)