Exploratory Path Planning for Mobile Robots in Dynamic Environments with Ant Colony Optimization

Valéria de C. Santos
Federal University of Ouro Preto
Ouro Preto, Brazil
valeriacs@ufop.edu.br

Fernando E. B. Otero
University of Kent
Chatham Maritime, United Kingdom
F.E.B.Otero@kent.ac.uk

Colin Johnson
University of Nottingham
Nottingham, United Kingdom
Colin.Johnson@nottingham.ac.uk

Fernando S. Osório
University of São Paulo
São Carlos, Brazil
fosorio@icmc.usp.br

Cláudio F. M. Toledo
University of São Paulo
São Carlos, Brazil
claudio@icmc.usp.br

ABSTRACT

In the path planning task for autonomous mobile robots, robots should be able to plan their trajectory to leave the start position and reach the goal, safely. There are several path planning approaches for mobile robots in the literature. Ant Colony Optimization algorithms have been investigated for this problem, giving promising results. In this paper, we propose the Max-Min Ant System for Dynamic Path Planning algorithm for the exploratory path planning task for autonomous mobile robots based on topological maps. A topological map is an environment representation whose focus is the main reference points of the environment and their connections. Based on this representation, the path can be composed by a sequence of state/actions pairs, which facilitates the navigability of the path, with no need to have the information of the complete map. The proposed algorithm was evaluated in static and dynamic environments, showing promising results in both of them. Experiments in dynamic environments show the adaptability of our proposal.

CCS CONCEPTS

• **Computing methodologies → Robotic planning**;

KEYWORDS

path planning, exploration, ant colony optimization, mobile robots

ACM Reference Format:
Valéria de C. Santos, Fernando E. B. Otero, Colin Johnson, Fernando S. Osório, and Cláudio F. M. Toledo. 2020. Exploratory Path Planning for Mobile Robots in Dynamic Environments with Ant Colony Optimization. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3390219

1 INTRODUCTION

There are two key competences required for mobile robot navigation, according to [18]. The first one is the path planning, that involves identifying a trajectory that will lead the robot to reach the goal location. The second competence is the obstacle avoidance, in which, given real-time sensor readings, the robot will follow the trajectory in order to avoid collisions. The path planning is considered exploratory since robots do not have information about the environment beforehand. They should plan their trajectory to the goal at the same time as they explore the environment.

The environment where mobile robots are deployed is represented by maps, which can be either provided or constructed during the exploration task. The major types of maps are metric and topological, which represent the lowest and the highest abstraction level, respectively [15]. The main characteristic of metric maps is the availability of detailed spacial information from the environment, such as coordinates and angles. In the case of topological maps, only the representation of the structure of the environment—i.e., the relative position among reference points—is available. Hence, topological maps contain less detailed information, taking into account regions of interest according to the robot task. A topological map can be represented by a graph, such that nodes are reference points of the environment and edges are the navigability among them. We have adopted this representation since precise information from the environment is not required, which is the case in many real-world scenarios (e.g., deployment of rescue robots). In this context, the exploratory path problem can be seen as an combinatorial optimization problem where the goal is to find an optimal combination of reference points to visit that lead to the goal.

Ant Colony Optimization (ACO) [4] has been successfully applied to combinatorial problems, starting with Ant System (AS) the first ACO algorithm applied to the traveling salesman problem. Since then, other improvements of the AS have been proposed, such as the Max-Min Ant System (MMAS) [4, 19]. MMAS provides four main modifications to AS: (i) it exploits the best tours found, in which just the ant that found the best global solution or the best solution of an iteration is allowed to deposit pheromone; (ii) it limits the possible range of pheromone trail values to an interval; (iii) the pheromone trails are initialized to the upper pheromone trail limit for increasing the exploration of tours at the start of the search;

(iv) pheromone trails are reinitialized once the system approaches stagnation.

In this paper we investigate the application of MMAS algorithm to the exploratory path planning task in dynamic environments, named Max-Min Ant System for Dynamic Path Planning (MMAS-DPP). The main contribution of this work is that robots do not have information about the environment beforehand. They collect information from the environment as they explore it. Furthermore, we analyze the cost to obtain the solution based on the distance traveled by ants, which represents the overall cost that ants are required to travel during the execution of the algorithm—e.g., the total distance traveled by all ants to explore the environment in order to find a solution. The current study extends the investigation made in [16], in which an application of MMAS algorithm was proposed to the exploratory path planning problem in static environments by considering a dynamic environment where routes become blocked or free over time and the goal position changes. The results show that the proposed MMAS-DPP algorithm is robust to cope with these dynamic situations.

2 RELATED WORK

According to Mac et al. [12], the widely known algorithms inspired by biology behaviors successfully applied in robot path planning are Genetic Algorithms [1, 9, 14], Particle Swarm Optimization [3, 22] and Ant Colony Optimization (ACO) [4] algorithms. In this section we focus on works using ACO since it is the meta-heuristic used by the proposed algorithm.

A heterogeneous ACO algorithm to solve the robot path planning problem was proposed in [21]. Two types of ants were defined, one dedicated to exploration and another dedicated to exploitation, and the number of each type of ant is managed to control the convergence rate of the algorithm. The authors do not mention how a robot will navigate the maps. Zhu et al. [23] presented a robot navigation algorithm for dynamic unknown environments based on an ant-based algorithm to find a local optimal static navigation path within the visual domain of the robot. Although authors state that the visual domain of robots is based on their sensors, they have calculated this region based on a grid map, so that no robotic simulator or realistic sensors were presented.

An approach based on ACO to solve the problem of path planning for mobile robots is presented by Garcia et al. [6]. The proposed algorithm uses the distance between the source and the target (goal) node to select the next node to move and a memory to the ants remember the visited nodes and avoid stagnation. The algorithm was evaluated in the ACO Test Center simulator, also proposed by the authors. The environment was represented as a matrix of interconnected nodes, in which each node can be marked as free or occupied, that is very similar to the grid representation.

There are works that combine ACO algorithms with other approaches for the path planning problem. Chaari et al. [2] have proposed an algorithm that combines ACO and Genetic Algorithms (GA) for robot path planning. The environment is static and robots have a prior knowledge about the environment. The environment map model is based on a Wireless Sensor Network, so that each sensor node is identified by its coordinate points. The authors state that their combined algorithm outperforms classical ACO and GA in this problem.

Ioannidis et al. [8] proposed a combination of ACO and Cellular Automata (CA) to create collision-free trajectories for every robot of a team while their formation is kept immutable. The CA is a grid structure which is updated by ACO to generate collision free paths. The method can be used in dynamic or unknown environments, with no prior knowledge of the space. The authors created a simulation environment to evaluate the algorithm and it was also implemented in Webots [13], a real world simulation environment. According to the authors, the proposed algorithm was effective at creating collision free paths.

Since most of works using ACO to the path planning task adopt the metric map (e.g., occupation grid) to represent the environment, they represent the path by a sequence of points that the robots should follow precisely to reach the goal. Therefore, a robust localization algorithm is required to execute the path safely in real environment, although in many cases, authors do not mention how the robots will execute the navigation of the path. The use of a topological map only requires an approximated representation of the environment and provides a simpler way to execute the navigation of the path.

3 METHODOLOGY

The proposed Max-Min Ant System for Dynamic Path Planning (MMAS-DPP) algorithm is run in two steps. First, the MMAS procedure is run to find the best solution in the static environment. Then, after the environment changes, the smoothing procedure is applied and the MMAS procedure is recalled to search for the new solution.

The MMAS-DPP algorithm is presented in Algorithm 1. In the *main procedure*, the initialization step is called in line 2, in which all ants are positioned at the start point. As proposed in the standard MMAS algorithm, pheromone trails are initialized with a large amount of pheromone on all edges. After the first iteration, this amount will be set as the upper pheromone trail limit. The *runMMAS procedure* is called in line 3, which will return the best solution $path_{gb}$. This procedure is the Max-Min Ant System for static environments presented in [16].

The *runMMAS procedure*, lines 13 − 19, shows the construction step, in which each ant k constructs the path $path_k$ from the start to the goal position. In each step of the construction phase, an ant k in the current node i calculates the probability to move to a neighbor node j, except the predecessor of node i. Figure 1 shows an ant (robot) in a crossing with all possible directions to move: *west, northwest, north, northeast, east, southeast, south and southwest.*

The probability p_{ij}^k of an ant k to move from node i to node j is given by [4]:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{if} j \in N_i^k \quad (1)$$

where τ_{ij} is the amount of pheromone in the edge that links nodes i and j; η_{ij} is the heuristic value to move from node i to j; α and β defines the influence of the pheromone trail and the heuristic information, respectively; N_i^k is the set of neighbors of node i. The heuristic function adopted in this study is the inverse of the

Algorithm 1 The proposed MMAS-DPP algorithm

```
 1: procedure MAIN
 2:     $initialization()$;
 3:     $path_{gb} = runMMAS()$;
 4:     while $changeIsDetected()$ do
 5:         $applySmoothing()$;
 6:         $path_{gb} = runMMAS()$;
 7:     end while
 8: end procedure
 9: procedure RUNMMAS
10:     $path_{gb} \leftarrow \emptyset$;
11:     while (stop criteria is not reached) do
12:         $path_{ib} \leftarrow \emptyset$;
13:         for $k \leftarrow 1$ to $numberOfAnts$ do
14:             $path_k \leftarrow constructPath()$;
15:             $localSearch(path_k)$;
16:             if $quality(path_k) > quality(path_{ib})$ then
17:                 $path_{ib} \leftarrow path_k$;
18:             end if
19:         end for
20:         $pheromoneEvaporation()$;
21:         $pheromoneUpdate()$;
22:         if $quality(path_{ib}) > quality(path_{gb})$ then
23:             $path_{gb} \leftarrow path_{ib}$;
24:         end if
25:     end while
26:     return $path_{gb}$;
27: end procedure
```

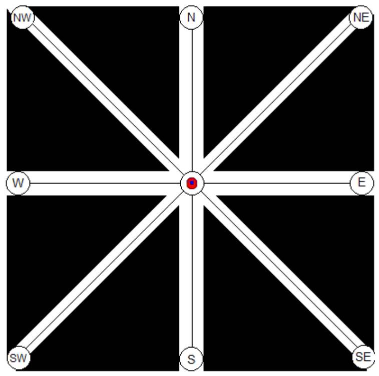

Figure 1: A robot in a crossing with all possible directions to move.

euclidean distance between the current and the goal node ($\eta_{ij} = 1/d_{ij}$). This calculation is based on the approximate coordinates of the robot and destination goal in the environment.

Since there is no control over nodes previously visited, apart from the current previous node, this approach can generate loops in the paths. Therefore, a local search procedure (line 15) is applied to remove loops after ants reach the goal. When the path is completed, the ants go back to the start position, so that only the ant that found the best solution deposits pheromone on the path. The other ants also follow this path. The best solution found after the construction phase is kept in $path_{ib}$ variable (lines 16 – 18).

After that, the pheromone evaporation step (line 20) is executed according to:

$$\tau_{ij} = (1 - \rho)\tau_{ij} \tag{2}$$

where ρ is the evaporation rate. Evaporation rate plays an important role in the convergence speed of the algorithm. The higher is this value, the quicker convergence is reached, but the larger the chance of the algorithm settling on a sub-optimal solution.

Then, the pheromone update step is applied (line 21). The standard MMAS algorithm proposes that just the ant that found the best path during an iteration or during the whole execution deposits pheromone. In the proposed MMAS-DPP algorithm, we use the best path during an iteration to explore more the space search. The pheromone update is defined as:

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^{best} \tag{3}$$

where $\Delta\tau_{ij}^{best} = 1/C^{best}$ and C^{best} is the length of the best solution of an iteration. If the best solution of an iteration is the best solution found so far, it is kept in the $path_{gb}$ variable (lines 22 – 24). At the end, all ants follow the same path in the backward step, whose distance is summed up to give the total traveled distance. Note that values of the pheromone on each edge in the trail are limited by a range to avoid stagnation. The maximum pheromone trail value is τ_{max}, defined as $1/\rho C^{bs}$, where C^{bs} is the length of the best solution found in the whole execution of the algorithm. The minimum pheromone trail value is set to $\tau_{min} = \tau_{max}/a$, where a is a user-defined parameter [4, 20].

This process is repeated until the stop criteria are satisfied. We adopt two stop criteria: (1) the best solution is not improved over a fixed number of iterations and (2) that a maximum number of iterations is reached. The *runMMASprocedure* returns the best solution $path_{gb}$ found (line 26) in the static environment.

In order to analyze the adaptability of the algorithm to deal with dynamic environments, after the best solution is found, we have applied the algorithm to a changing environment. This is an extension of the previous investigation presented in [16]. Based on the literature, we have applied two kinds of changes: removing an edge (equivalent to adding an obstacle) and moving the goal position. Before recalling the *runMMAS procedure* to search for the new solution, we have called the *applySmoothing procedure* (line 5). This is a procedure suggested by the MMAS algorithm to smooth the pheromone amount when the algorithm has converged [4]. This mechanism increases the pheromone trails proportionally to their difference to the maximum pheromone trail limit:

$$\tau_{ij}^*(t) = \tau_{ij}(t) + \delta(\tau_{max}(t) - \tau_{ij}(t)) \tag{4}$$

where $\tau_{ij}^*(t)$ and $\tau_{ij}(t)$ are the pheromone trails before and after the smoothing. The purpose of the smoothing is to facilitate the exploration by increasing the probability of selecting solution components with low pheromone trail. Furthermore, the information gathered during the run of the algorithm is not completely lost but weakened. For $\delta = 1$ this mechanism corresponds to a reinitialization of the pheromone trails, while for $\delta = 0$ it is disabled.

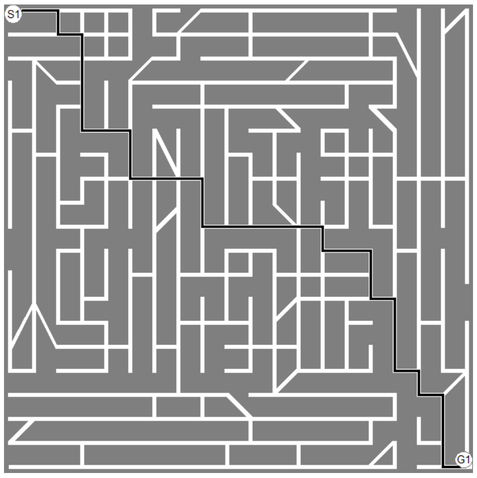

Figure 2: The map used to evaluate the MMAS-DPP performance, extracted from [16]. The highlighted black line shows the optimal path for this map manually defined and also discovered using the MMAS-DPP approach.

4 RESULTS AND DISCUSSION

MMAS-DPP is applied in two steps. Firstly, it finds the solution in the static environment. Then, the environment changes and the algorithm has to find the new solution. Two types of dynamic change is evaluated: removing one edge from the previous best solution and changing the goal point. First, we present the adjustment of MMAS-DPP parameters in the static environment. Then, we analyze how the algorithm behaves when the environment changes.

The map adopted in this paper is the same one of [16]. The map is illustrated in Figure 2, in which the best solution is highlighted. S_1 and G_1 are initial and goal points, respectively. The algorithm was run 100 times in this map. The results are presented in terms of average and standard deviation.

4.1 Static Environment

Initially, the MMAS algorithm is applied to a static environment. Here, the analysis of number of ants, evaporation rate and minimum pheromone trail parameters are considered.

Table 1 presents the results regarding the number of ants. Traveled distance is the distance spent by an ant during the whole executing of the algorithm. Each time an ant moves from one node to another, the distance between these nodes is summed up to the traveled distance of the correspondent ant. Column *percentage* refers to how many times the best solution was found among the 100 executions of the algorithm. Column *avg best path* shows the average and standard deviation of the best solution found by all executions. *Best travel dist* is the average of the traveled distance spent by the ant that found the best solution in all executions during the whole running of the ACO; *travel dist it* refers to the average of the traveled distance spent by each ant in one iteration.

The other parameters of the algorithm were defined as following. For stop criteria, fixed number of iterations with no update of the best solution is 100 and maximum number of iterations is 1000; $\alpha = 1$ and $\beta = 0.1$ are parameters of Equation 1; $\rho = 0.1$ is the evaporation rate from Equation 2; $a = 10$ is the parameter to define the minimum pheromone trail. In Table 1, we can observe that *best travel dist* decreases using more ants. Number of ants does not affects the traveled distance per iteration. The algorithm was able to find always the optimal solution when using 50 ants, therefore, we have defined this value for the next experiments.

Table 2 shows the analysis of the evaporation rate ρ. Although the results show that this parameter does not affect much the performance of the algorithm, the convergence is slow when this value is too low and the algorithm can converge quickly to a local minimum solution when this value is too high. Based on our results, we have adopted $\rho = 0.1$ to the next evaluations.

As explained in Section 3, MMAS-DPP defines a range for the pheromone trail to avoid stagnation, in which the minimum pheromone trail is set to $\tau_{min} = \tau_{max}/a$. The larger is the pheromone trail range, the slower the algorithm tends to converge, favoring exploration of the search space. On the other hand, the smaller is the pheromone trail range, the bigger is the chance the algorithm will exploit the best solutions found. The results showed in Table 3 reflect this, in which the first column is the parameter a.

The quality of the paths is better with small values of parameter a. This is expected because the search space is more restricted and the algorithm tend to exploit the best solutions. However, in these cases, the traveled distances are higher and the algorithm will tend to keep these solutions even when the environment changes. Therefore, we chosen the value 50 that is a median value and to make the algorithm more flexible to cope with dynamic environment.

4.2 D* Lite Algorithm

We have compared the MMAS-DPP results with D* Lite algorithm [10]. D* Lite is an algorithm used to find the optimal path in unknown and dynamic environments. It is based on LPA* algorithm [11], an incremental version of A* algorithm [7]. D* Lite maintains two estimates of costs to each node $s \in S$, in which S denotes the finite set of vertices of the graph: $g(s)$ and $rhs(s)$. The $g(s)$ is the estimate of the start distance of each node s. The rhs-values are one-step lookahead values that satisfy the following relationship:

$$rhs(s) = \begin{cases} 0 & \text{if s = } s_{start} \\ min_{s' \in Pred(s)}(g(s') + c(s', s)) & \text{otherwise} \end{cases} \quad (5)$$

where $Pred(s) \subset S$ denotes the set of predecessors of $s \in S$ in the graph; $0 < c(s, s') < \infty$ denotes the cost of moving from s to $s' \in Succ(s)$, $Succ(s) \subset S$ denotes the set of successors in the graph.

A node is considered consistent if its g-value is equal to its rhs-value, otherwise it is inconsistent. The priority queue maintains the inconsistent vertices, the ones that needs to update to become consistent. The priority of vertex s in the priority queue is:

$$k(s) = [k_1(s); k_2(s)] \quad (6)$$

$$k_1(s) = min(g(s), rhs(s)) + h(s, s_{goal}) \quad (7)$$

$$k_2(s) = min(g(s), rhs(s)) \quad (8)$$

Algorithm 2 shows *Main()* and *ComputeShortestPath* procedures of the basic version of D* Lite. At the beginning, the *Main()* procedure calls *Initialize()*, in which the g-values and rhs-values are initialized to infinity. The rhs-value of goal vertice has been set

Table 1: Analysis of number of ants.

Number of ants	Percentage	Avg Best path	Best Travel Dist	Travel Dist It
10	27%	1168.38 ± 58.52	556975.88 ± 414802.66	8597.80 ± 696.51
20	72%	1119.80 ± 48.28	605291.47 ± 472561.49	8656 ± 453.74
30	95%	1096.78 ± 16.83	495381.75 ± 344283.67	8696.57 ± 340.20
40	99%	1094.02 ± 7.41	401832.06 ± 279262.32	8674.64 ± 235.36
50	**100%**	**1093.27**	**275602.66 ± 167281.54**	**8741.87 ± 184.81**
100	100%	1093.27	176561.11 ± 83673.91	8676.45 ± 122.92

Table 2: Analysis of the evaporation rate.

Evaporation rate	Percentage	Avg Best Path	Best Travel Dist	Travel Dist It
0.01	99%	1094.02 ± 7.41	1354969.20 ± 475963.15	15479.11 ± 937.69
0.05	99%	1093.33 ± 0.60	413470.75 ± 238790.32	9900.13 ± 322.03
0.1	**100%**	**1093.27**	**275602.66 ± 167281.54**	**8741.87 ± 184.81**
0.3	100%	1093.27	188046.59 ± 116356.69	8023.37 ± 122.41
0.5	100%	1093.27	179750.01 ± 138014.02	7794.65 ± 110.69
0.6	99%	1094.16 ± 8.80	207929.42 ± 182693.14	7750.33 ± 111.34

Table 3: Analysis of the minimum pheromone trail.

Parameter a	Avg Best Path	Avg Best Travel Dist	Travel Dist It
10	1094.15 ± 8.80	288731.99 ± 173807.23	8727.10 ± 201.08
20	1094.75 ± 10.55	216364.05 ± 123843.57	5750.01 ± 168.68
30	1095.92 ± 15.09	235027.81 ± 141441.83	5729.05 ± 186.61
40	1109.64 ± 43.30	202750.63 ± 124771.63	4494.66 ± 166.49
50	**1111.47 ± 42.84**	**194960.87 ± 100896.87**	**4307.19 ± 174.98**
60	1112.00 ± 42.94	195965.80 ± 109504.77	4158.63 ± 185.81
70	1117.88 ± 51.59	166252.38 ± 105274.34	4098.74 ± 161.54
80	1112.15 ± 39.69	183546.99 ± 102434.58	4007.26 ± 186.89
90	1112.54 ± 43.04	183388.60 ± 85755.18	3999.54 ± 153.66
100	1115.38 ± 47.76	169666.89 ± 87023.35	3985.88 ± 174.22

to zero, that makes it inconsistent. Therefore, the goal vertice is inserted to priority queue.

Then, the algorithm computes the shortest path from the s_{start} node to the goal and the robot navigates following this path. If the robot has not reached the goal, it moves to the next node of the shortest path and updates s_{start} to the current node of the robot. If some change occurs in the environment, the algorithm updates the affected edge costs and calls *UpdateVertex()* to update the rhs-values of the vertices affected by the changed edge costs so that they satisfy again the equivalent of Equation 5. It also updates the priority queue to contain again the inconsistent vertices with priorities that satisfy the equivalent of Equation 6. Then, the algorithm updates the priorities of all vertices in the priority queue, recalculates a shortest path and iterates.

Table 4 shows the comparison between D* Lite and MMAS-DPP before the environment changes, in relation to the traveled distance. The best traveled distance and the average traveled distance by iteration of the MMAS-DPP is lower than the traveled distance achieved by the D* Lite algorithm, although the average of the best traveled distance is greater. Hence, the MMAS-DPP algorithm can find a suitable path travelling a shorter distance than the D* Lite algorithm.

Table 4: Results of D* Lite and MMAS-DPP algorithms in the static environment.

D* Lite	MMAS		
Travel Dist	Best Travel Dist	Avg Best Travel Dist	Avg Travel Dist It
36135.20	**6935.26**	194960.87 ± 100896.87	**4307.19 ± 174.98**

4.3 Dynamic Environment

After of defining the parameters in the static environment, we have analyzed how the algorithm copes with dynamic environments. The MMAS-DPP finds the best solution in the static environment. The environment changes and the MMAS-DPP finds the best solution for the new environment configuration. We compare our results with the ones obtained by D* Lite algorithm [10] in relation to the traveled distance spent during the search.

Firstly, we have adjusted the delta parameter for the smoothing procedure presented in Equation 4. Table 5 shows the results, in

Algorithm 2 D* Lite Algorithm

```
procedure ComputeShortestPath()
    while ($U.TopKey()$ < $CalculateKey(s_{start})$ OR $rhs(s_{start}) \neq g(s_{start})$) do
        $u = U.Pop()$;
        if ($g(u) > rhs(u)$) then
            $g(u) = rhs(u)$;
            for $all s \in Pred(u)$ do
                $UpdateVertex(s)$;
            end for
        else
            $g(u) = \infty$;
            for $all s \in Pred(u) U u$ do
                $UpdateVertex(s)$;
            end for
        end if
    end while
end procedure
procedure Main
    $Initialize()$;
    $ComputeShortestPath()$;
    while ($s_{start} \neq s_{goal}$) do
        /*$if(g(s_{start}) = \infty)$ then there is no known path*/
        $s_{start} = argmin_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
        Move to $s_{start}$;
        Scan graph for changed edge costs;
        if (any edge costs changed) then
            for (all directed edges $(u, v)$ with changed costs) do
                $Updatetheedgecostc(u, v)$;
                $UpdateVertex(u)$;
            end for
            for $all s \in U$ do
                $U.Update(s, CalculateKey(s))$;
            end for
            $ComputeShortestPath()$;
        end if
    end while
end procedure
```

which the first line of the table presents the results obtained before the environment modification. The following lines show the results after the environment modification, varying the delta value. Column *Avg Best Path* is the average of the best solutions found by the algorithm; *Avg Best Travel Dist* is the average of the traveled distance spent by the ant that found the best solution from all executions during the whole running of the ACO; *Iteration Best* is the iteration in which the best solution was reached; *Total Iterations* is the total number of iterations spent by the algorithm; *Avg Travel Dist It* refers to the average of the traveled distance spent by each ant in one iteration.

Table 5 shows the results analysis of changing the delta value in the algorithm to evaluate the adaptability to environment modifications. In this experiment, the dynamic modification consists of removing an edge from the middle of best solution found in the static environment. As the smoothing procedure is applied after

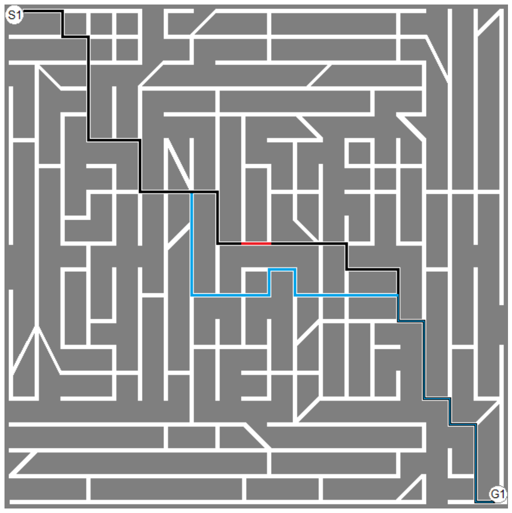

Figure 3: Edge removed from the middle of best solution are highlighted red line in this map. The blue line shows the new edges of the solution.

the algorithm finishes its execution in the static environment, the delta value has no influence in these results. The next best solution obtained by different delta values is approximately the same. However, we can observe that the lower the delta value, the lower the traveled distance and the iteration to find the best solution. The traveled distance per iteration is lower for $delta = 0.1$, as well. This suggests that keeping a little pheromone trail from the previous solution is useful, however, this small value allowed the algorithm to explore other possibilities and to escape from the previous solution. Therefore, 0.1 is fixed for delta value in the next evaluations.

Table 6 shows the results for a dynamic environment when the edges are removed from the beginning, middle and end of the previous best solution. When the edge of the beginning is removed, we can see from Table 6 that the MMAS-DPP algorithm could find the new solution quickly, as showed in Column *Iteration Best*, and the best solution was reached in all executions.

The algorithm spent more distance to find the new solution when the edge is removed from the middle, as we can see in the second line from Table 6. Nevertheless, the algorithm could find new good solutions in the first iterations, based on columns *Iteration Best* and *Total Iterations*. This result shows the adaptability of MMAS-DPP to cope with dynamic environment, due to its capability of explore the search space. Figure 3 shows the removed edge and the new optimal solution. For the edge removed from the end of the path, the algorithm could find new good solutions in few iterations.

Table 7 shows the comparison between D* Lite and the MMAS-DPP algorithm, in relation to the traveled distance spent to find the best solution. The first line shows the results before the modification in the environment. Column *Traveled Distance* is the traveled distance spent by D* Lite algorithm to find the optimal solution. Column *Best Traveled Distance* is the best traveled distance spent by MMAS-DPP to find the best solution. Column *Avg Best Travel*

Table 5: Analysis of the delta parameter of the smoothing procedure.

		Avg Best Path	Avg Best Travel Dist	Iteration Best	Total Iterations	Avg Travel Dist It
Static		1111.47 ± 42.84	194960.87 ± 100896.87	22.22 ± 25.57	123.22 ± 25.57	4307.19 ± 174.98
Delta	**0.1**	**1151.36 ± 15.68**	**94998.22 ± 60163.73**	**12.11 ± 13.32**	**113.11 ± 13.22**	**3931.63 ± 153.77**
	0.3	1151.71 ± 12.33	183621.44 ± 86213.99	15.35 ± 14.74	116.35 ± 14.74	4777.99 ± 222.09
	0.5	1151.24 ± 11.98	281552.84 ± 105949.99	21.12 ± 16.46	122.21 ± 16.45	5439.03 ± 287.03
	1.0	1161.05 ± 24.12	475914.37 ± 133456.30	29.03 ± 13.40	130.03 ± 13.40	6625.58 ± 324.98

Table 6: Analysis of the dynamic environment when edges are removed.

		Avg Best Path	Best Travel Dist	Iteration Best	Total Iterations	Avg Travel Dist It
Static		1111.47 ± 42.84	194960.87 ± 100896.87	22.22 ± 25.57	123.22 ± 25.57	4307.19 ± 174.98
Edge	Start	1208.93	12960.33 ± 21449.54	0.66 ± 3.31	101.66 ± 3.31	3843.20 ± 82.85
	Middle	1151.36 ± 15.68	94998.22 ± 60163.73	12.11 ± 13.32	113.11 ± 13.22	3931.63 ± 153.77
	End	1217.37 ± 17.41	44368.81 ± 30722.85	3.18 ± 3.35	104.18 ± 3.35	4033.26 ± 113.25

Distance is the average of the best traveled distance from all executions of MMAS-DPP and Column *Avg Travel Dist It* is the average of the traveled distance spent by each ant in one iteration.

The best traveled distance spent by MMAS-DPP is much better than the one spent by D* Lite, although in comparison to the average of this value, the result from D* Lite is lower. The average traveled distance per iteration means that each ant traveled less than the D* Lite to leave the start position and reach the goal. It shows that the collaborative work generates better results by comparison with to an individual working alone.

In the dynamic environment experiments, we have removed edges from the start, middle and end of the previous best solution. Table 7 shows the results of these cases considering D* Lite and MMAS-DPP algorithm. The best traveled distance by MMAS-DPP is much lower than the one spent by D* Lite to find the next best solution. In the three cases, the values obtained by MMAS-DPP are very similar.

In relation to average of the best traveled distance, results when edges from start and end are removed are similar. In these cases, the average of the best traveled distance is close to the distance obtained by D* Lite. In situation two, which is the most complex case, the average of the best traveled distance obtained by MMAS-DPP is worse than the one obtained by D* Lite. The average traveled distance per iteration is quite similar in the three cases, so that the results obtained by MMAS-DPP are the lowest.

We also have evaluated the MMAS-DPP algorithm in the dynamic environment, considering that the goal point has changed. Results of the environment before and after the change are presented in Table 8. Figure 4 shows the goal points, where G_1 is the initial goal, G_2, G_3, G_4, G_5 and G_6 are the new goal positions. We can observe in Column *Avg best path* that the new best solution is found in all executions from MMAS-DPP algorithm every time the goal changes. The best traveled distance is lower to find the new solutions. Column *iteration best* shows that the new solution is found in the first iterations of the algorithm. Including, when the goal changes to G_3 and G_4, it is found in the very first iteration. The traveled distance per iteration is very close to the value obtained in the static environment, so that for G_3 and G_4 it is lower. The authors of D* Lite algorithm [10] do not mention the application of their algorithm for this type of dynamic environment.

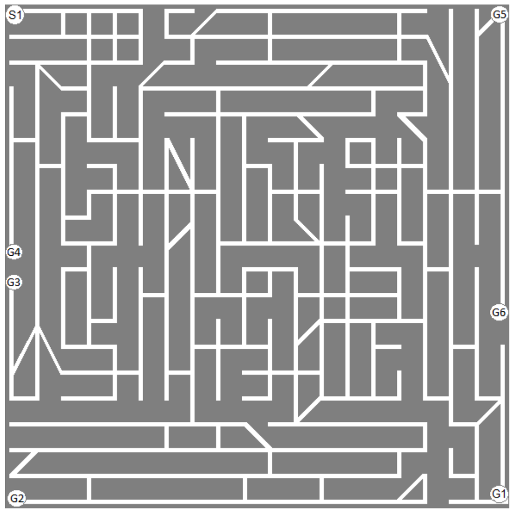

Figure 4: The map with the new goal points highlighted.

Although the D* Lite algorithm finds the optimal solution when the environment changes, it spends more traveled distance to reach the goal point.

The MMAS-DPP do not always find the optimal solution, but it can find a near optimal suitable solution (see Column *Avg Best Path* from Table 6) and can reach the goal quickly, considering the traveled distance. This characteristic can be very useful in an application in which the goal needs to be reached quickly, such as search and rescue tasks.

Moreover, the MMAS-DPP works in an approximated representation of the environment, considering roughly defined maps and imprecise robot and goal localization, which is enough to find a navigable path. Although we have applied the D* Lite in the same environment representation, it completely depends of this map to find the path. During the navigation, the robot has to follow

Table 7: Comparison between D* Lite and MMAS-DPP results.

		D* Lite	MMAS-DPP		
		Traveled Distance	**Best Travel Dist**	**Avg Best Travel Dist**	**Avg Travel Dist It**
Static		36135.20	7702.31	163580.26 ± 79431.09	4322.43 ± 159.77
Dynamic	Start	34217.4	1317.85	12960.33 ± 21449.54	3843.20 ± 82.85
	Middle	28519.2	1181.29	94998.22 ± 60163.73	3931.63 ± 153.77
	End	70289.8	1210.67	44368.81 ± 30722.85	4033.26 ± 113.25

Table 8: Analysis of the dynamic environment – change goal.

Goal	Avg Best Path	Best Travel Dist	Iteration Best	Total Iterations	Travel Dist It
G1	1111.47 ± 42.84	194960.87 ± 100896.87	22.22 ± 25.57	123.22 ± 25.57	4307.19 ± 174.98
G2	1108.36	71768.60 ± 69149.38	2.03 ± 4.29	103.03 ± 4.29	4500.83 ± 119.80
G3	686.30	26483.03 ± 27585.10	0	101	2862.75 ± 78.99
G4	434.20	32515.89 ± 30792.79	0	101	2183.90 ± 82.31
G5	1099.68	74883.89 ± 64981.25	2.49 ± 5.64	103.49 ± 5.64	4249.93 ± 109.85
G6	1067.91	75089.63 ± 57757.71	2.12 ± 3.74	103.12 ± 3.74	4332.75 ± 111.82

precisely a set of points and if something is different in the map representation, the algorithm can fail.

The navigation of the best path between points S_1 and G_1 obtained in the map illustrated in Figure 2 was executed in the robot simulator V-REP, a realistic robotic simulator [5], using a Pioneer P3DX robot with a 2D Laser Scanner sensor.[1] The path provided by our approach can be represented by a sequence of actions, therefore the robot does not have any prior information about the environment, it just knows the sequence of actions it should execute. In the navigation of this path, the robot starts walking in a corridor. When it detects a change of state, the next action is activated. The state recognizer proposed in [17] was applied to identify when the robot state changes based on the sensors of the robot.

5 CONCLUSION

In this paper, we have presented the MMAS-DPP algorithm applied to the exploratory path planning problem for autonomous mobile robots in dynamic environments. This problem considers that the robots do not know the environment previously. They just know their approximate start and goal positions.

We have analyzed the MMAS-DPP in static and dynamic environments, considering the quality of the obtained solutions and the traveled distance spent by the ants to find the solution. If we consider the total distance traveled by each ant individually, it spends a very high traveled distance. However, when the overall algorithm is observed, the result of all ants working in a collaborative way contributes to obtain good quality solutions, spending lower traveled distance. The dynamic change is evaluated after the algorithm finishes its execution in the static environment. According to the results, the algorithm can find the next solution quickly, showing that it is flexible to cope with dynamic environments.

The MMAS-DPP algorithm was compared to the D* Lite algorithm, an extension of A* algorithm to dynamic environments. D* Lite works with one agent to find the optimal solution that spends a very high traveled distance to reach the goal. The advantage of the MMAS-DPP is that using several agents exploring the environment, the goal can be reached quicker and the information about good paths is gathered through the pheromone deposit in the environment.

We also have analyzed the MMAS-DPP for dynamic environments in which the goal point changes. The new solution is found in the first iterations of the algorithm, showing the influence of the heuristic function and that the amount of pheromone deposited in the static environment was suitable to the algorithm to explore new paths when some modification occurs in the environment.

As future research directions, we will investigate an way to the MMAS-DPP to improve the quality of the solutions with no need of spending higher traveled distance. Also, we are studying methods to implement robots that reproduce the pheromone deposition in real environments.

[1]Available online at: https://youtu.be/RHDTbjUdGow

ACKNOWLEDGMENTS

The authors would like to thank CNPq (project ID 159573/2013-0), CAPES (project ID 99999.003520/2015-03) and UFOP (Federal University of Ouro Preto) for their financial support.

REFERENCES

[1] Azzeddine Bakdi, Abdelfetah Hentout, Hakim Boutami, Abderraouf Maoudj, Ouarda Hachour, and Brahim Bouzouia. 2017. Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control. *Robotics and Autonomous Systems* 89 (2017), 95 – 109. https://doi.org/10.1016/j.robot.2016.12.008

[2] Imen Chaari, Anis Koubaa, Hachemi Bennaceur, Sahar Trigui, and Khaled Al-Shalfan. 2012. smartPATH: A hybrid ACO-GA algorithm for robot path planning. *2012 IEEE Congress on Evolutionary Computation, CEC 2012*, 1–8. https://doi.org/10.1109/CEC.2012.6256142

[3] P.K. Das, H.S. Behera, and B.K. Panigrahi. 2016. A hybridization of an improved particle swarm optimization and gravitational search algorithm for multi-robot path planning. *Swarm and Evolutionary Computation* 28 (2016), 14 – 28. https://doi.org/10.1016/j.swevo.2015.10.011

[4] M. Dorigo and T. Stützle. 2004. *Ant Colony Optimization.* Bradford Company, Scituate, MA, USA.

[5] M. Freese E. Rohmer, S. P. N. Singh. 2013. V-REP: a Versatile and Scalable Robot Simulation Framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS).*

[6] M. A. P. Garcia, O. Montiel, O. Castillo, R. Sepúlveda, and P. Melin. 2009. Path planning for autonomous mobile robot navigation with ant colony optimization

and fuzzy cost function evaluation. *Applied Soft Computing* 9, 3 (2009), 1102 – 1110. https://doi.org/10.1016/j.asoc.2009.02.014

[7] P.E. Hart, N.J. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on* 4, 2 (July 1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[8] K. Ioannidis, G. Ch. Sirakoulis, and I. Andreadis. 2011. Cellular ants: A method to create collision free trajectories for a cooperative robot team. *Robotics and Autonomous Systems* 59, 2 (2011), 113 – 127. https://doi.org/10.1016/j.robot.2010.10.004

[9] Rahul Kala. 2012. Multi-robot path planning using co-evolutionary genetic programming. *Expert Systems with Applications* 39, 3 (2012), 3817 – 3831. https://doi.org/10.1016/j.eswa.2011.09.090

[10] S. Koenig and M. Likhachev. 2002. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, Vol. 1. 968–975 vol.1. https://doi.org/10.1109/ROBOT.2002.1013481

[11] Sven Koenig, Maxim Likhachev, and David Furcy. 2004. Lifelong Planning A*. *Artificial Intelligence* 155, 1 (2004), 93 – 146. https://doi.org/10.1016/j.artint.2003.12.001

[12] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. 2016. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems* 86 (2016), 13 – 28. https://doi.org/10.1016/j.robot.2016.08.001

[13] O. Michel. 2004. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems* 1, 1 (2004), 39–42.

[14] Hong Qu, Ke Xing, and Takacs Alexander. 2013. An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots. *Neurocomputing* 120 (2013), 509 – 517. https://doi.org/10.1016/j.neucom.2013.04.020 Image Feature Detection and Description.

[15] R. A. F. Romero, E. P Silva Jr, F. S. Osório, and D. F. Wolf. 2014. *Robótica Móvel*. LTC - Grupo Gen, Rio de Janeiro, Brasil.

[16] V. C. Santos, F. S. Osório, C. F. M. Toledo, F. E. B. Otero, and C. G. Johnson. 2016. Exploratory path planning using the Max-min ant system algorithm. In *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*. 4229–4235. https://doi.org/10.1109/CEC.2016.7744327

[17] V. C. Santos, C. F. M. Toledo, and F. S. Osório. 2014. A Hybrid Approach for Path Planning and Execution for Autonomous Mobile Robots. In *Proceedings of 2nd Brazilian Symposium on Robotics and 11th Latin American Robotics Symposium*. ACM, 124–129.

[18] R. Siegwart and I. R. Nourbakhsh. 2004. *Introduction to Autonomous Mobile Robots*. Bradford Company, USA.

[19] T. Stützle. 1999. *Local search algorithms for combinatorial problems - analysis, improvements, and new applications*. DISKI, Vol. 220. Infix. I–XI, 1–203 pages.

[20] T. Stützle and H. H. Hoos. 2000. MAX-MIN Ant System. *Future Gener. Comput. Syst.* 16, 9 (jun 2000), 889–914.

[21] Y. Yao, Q. Ni, Q. Lv, and K. Huang. 2015. A novel heterogeneous feature ant colony optimization and its application on robot path planning. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*. 522–528. https://doi.org/10.1109/CEC.2015.7256934

[22] Yong Zhang, Dun wei Gong, and Jian hua Zhang. 2013. Robot path planning in uncertain environment using multi-objective particle swarm optimization. *Neurocomputing* 103 (2013), 172 – 185. https://doi.org/10.1016/j.neucom.2012.09.019

[23] Q. Zhu, J. Hu, W. Cai, and L. Henschen. 2011. A new robot navigation algorithm for dynamic unknown environments based on dynamic path re-computation and an improved scout ant algorithm. *Applied Soft Computing* 11, 8 (2011), 4667 – 4676. https://doi.org/10.1016/j.asoc.2011.07.016