Proceedings of the International Conference of Control, Dynamic Systems, and Robotics
Ottawa, Ontario, Canada, May 15-16 2014
Paper No. 49

Dynamic Robot Path Planning Using Improved Max-Min Ant Colony Optimization

Nicholas Charabaruk, Mark Manning, Scott Nokleby
University of Ontario Institute of Technology
2000 Simcoe St. N., Oshawa, Ontario, L1H 7K4, Canada
nicholas.charabaruk@uoit.ca; mark.manning@uoit.ca, scott.nokleby@uoit.ca

Abstract - This paper presents a method of using an improved version of the Max-Min Ant Colony Optimization (ACO) algorithm for use in dynamic global robot path planning. A modified Bug2 algorithm was used to determine the initial best path on a map. After running the Max-Min ACO algorithm to find the optimal path, the path was checked to see if it crossed any previously unknown obstacles, and if so the route was recalculated from the obstacle to the goal. The previously found best path was saved to help subsequent runs find the optimal solution faster. This algorithm was tested using simulations, and it was determined that it performed well in finding the average shortest path. It also resulted in greatly reduced processing times.

***Keywords*:** Robot, path planning, Ant Colony Optimization (ACO), optimization.

1. Introduction

Global path planning is an important part of mobile robotic systems. For a robot to be autonomous it must be able to determine how to travel from point A to point B. However, there may be multiple paths to get from the start to the goal location. Path optimization must be used in order for the robot to determine the best path. While there are many aspects that may make a path the best, this paper will concentrate on finding the shortest route. Ant Colony Optimization (ACO) has been shown to have good results when used for shortest path planning. When reviewing the literature however, it was found that ACO algorithms were rarely used for dynamic path planning, i.e., environments that are changing. This is unusual as environments are rarely static in the real world. This paper, therefore, presents a method of using ACO in dynamic environments. Max-Min ACO was chosen for this problem as it results in shorter paths with faster convergence speeds than basic ACO. This algorithm was further adapted to achieve better paths by incorporating a modified version of the Bug2 algorithm presented by Lumelsky and Stepanov (1987) to determine the initial best path.

The remainder of this paper is laid out as follows: Section 2 outlines the basic ACO algorithm, Section 3 presents the state of the art in ACO path planning, Section 4 details the algorithm presented in this paper, Section 5 shows the results of the simulations, and Section 6 presents the conclusions.

2. Basics of ACO

Hsiao et al. (2004) stated that ACO is a graph based evolutionary meta-heuristic optimization technique. It has been used successfully to solve a variety of optimization problems. ACO is used to solve minimum cost-path searching in graphs. The algorithm works by using a large number of simulated ants walking through a grid or nodal map to find the shortest route between two points. Individual ants follow a very simplistic behaviour and therefore by themselves achieve very poor results. By cooperating with many simulated ants however, better paths can be found.

The behaviour of the simulated ants mimics the behaviour of biological ants searching for food. As ants search, they lay a pheromone trail behind them. The ants distribute the pheromones at a constant rate as they travel. This results in a higher concentration of pheromones being placed on shorter paths. This is because the ant following the shorter path can make more trips in the same amount of time as the ant following a longer route. This results in overlap of the pheromone on the shorter route. The pheromones

diminish over time, making the shorter path still more attractive as the pheromone on the longer paths will diminish faster. Other ants prefer to follow the path with the stronger pheromone trail as this indicates a better route.

The simulated ants in ACO differ in some ways than the biological ants. The main difference is in how they distribute the pheromones. Rather than distributing the pheromone at a constant rate, the pheromone is instead distributed only after the simulated ant reaches the goal node. In ACO, each ant has a set amount of pheromone when leaving the starting node. When the ant reaches the goal node, the pheromone is distributed evenly over the length of the ant's path. In this way, if the ant followed a short path then the pheromone density would be higher than if the ant took a longer path. When the pheromone densities are increased, that is also when the artificial evaporation takes place. The evaporation of the pheromone paths occurs in order to prevent simulated ants from getting caught in local optima.

Another difference between biological ants and the simulated ants is in how the simulated ants decide in what direction to travel. Simulated ants are digital, and therefore can only travel in discrete directions based on the connectivity of the nodes. What directions the ants decide to follow are based on a probability function. According to Hsiao et al. (2004), this probability is based on a heuristic factor and a pheromone factor. For the heuristic portion, there is a higher probability of an ant choosing to travel to nodes that are closer together. For the pheromone portion, the higher probability goes to the connection that has the highest amount of pheromones between the nodes. Weighting values for the heuristic and pheromone factors are set to determine each factor's relative influence on the ant's decision making. For the algorithm to work, a small constant amount of pheromone must be applied to every node when the algorithm is initialized. When all of the ants have travelled through the map, the path with the highest concentration of pheromone is determined to be the shortest.

The probability function used in the Improved Max-Min ACO algorithm, which is a variation on the function presented by Hsiao et al. (2004), is given by:

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{j \varepsilon S}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} \quad (1)$$

where P_{ij} is the probability of travelling from node i to node j, η_{ij} is the desirability of travelling from node i to node j based on the direction between the current node and the prospective node related to the direction between the current node and the goal node, τ_{ij} is the amount of pheromone between nodes i and j, β is the heuristic weighting value, α is the pheromone weighting value, and S is the population of allowable nodes.

3. State of the Art

Han et al. (2011) compared basic ACO with an improved algorithm for robot path planning. The improved algorithm was similar to the basic ACO algorithm with one major change. In the improved method, the path of the first ant was set as the best path of the iteration. Subsequent paths were compared to the best path, and if one was shorter, then it became the new best path. Once all of the ants made their run, the pheromone update was made using just the best path of the iteration. After the max number of function calls was reached, the best path from all of the iterations was determined. It was found that the improved algorithm achieved shorter paths and higher convergence speeds than the basic ACO algorithm, but had more trouble dealing with local optima. This method is now referred to as Max-Min ACO.

Hsiao et al. (2004) were some of the first to apply ACO to global path planning. They implemented the basic ACO technique to determine the best path between manually inputted start and goal positions on randomly generated maps. They showed that ACO was a viable algorithm to use in global path planning.

Brand et al. (2010) presented a method of using basic ACO for robot path planning in a dynamic environment. The authors allowed the ACO algorithm to find the best path from a start to goal locations on a map, and then blocked that path. This forced the ants to find a new path. Two methods of path planning were tested. The first was termed global re-initialization, which consisted of resetting the entire

map to its initial pheromone levels and then re-running the ACO algorithm using the blocked position as the new start position and leaving the goal position the same. The second method was the local re-initialization technique, in which only nodes close to the blocked location had their pheromone levels reset.

4. Path Planning Algorithm

The proposed algorithm in this work was based on the Max-Min ACO algorithm. Several modifications were made to this algorithm. The first change was that the heuristic factor was not based on the distance between nodes, but the direction of the node in relation to the direction of the goal position. The closer a direction is to the direction of the goal position, the higher the likelihood of the ant travelling in that direction. With each move of the ant's location being more likely to bring it closer to its goal, each ant will eventually reach its goal. When it does, the ant in the population with the shortest route will reinforce the pheromone level along its path. In addition, all pheromone levels will diminish over time. The pheromone will not be allowed to decrease below a minimum level. The Max-Min ACO iterates for another population of ants and will keep doing so until an optimal solution or the max number of ants is reached.

To allow the algorithm to be capable of dynamic path planning, the goal was to take advantage of the faster computation times of the Max-Min ACO method and make modifications to this method to allow for more optimal results. When using the basic Max-Min ACO method, it was found that the first ant in each generation typically found a very poor path. This resulted in increased computation time at the beginning of the algorithm, resulting in the algorithm requiring more time to generate quality solutions. To improve the processing time, a modified version of the Bug2 algorithm presented by Lumelsky and Stepanov (1987) was used to generate the initial best path. This algorithm works by planning a straight line between the start and goal locations. The ant tries to follow that line until it hits an obstacle. The ant then follows the edge of the obstacle until it intersects with the original straight line path, at which point it starts following the original path again. The ant continues this way until it reaches the goal. The Bug2 algorithm needed to be modified slightly to allow for this same behaviour to be modelled when using nodes to define the ant's path. In the method presented here, the ant followed the edge of any obstacles until there was a free node on the line between its current location and the goal. The modified Bug2 algorithm provided an excellent baseline best path for the rest of the Max-Min algorithm. The second modification was that the re-initialization method presented by Brand et al. (2010) was also implemented. When the best path intersected with a dynamically blocked node, the program ran again with the node before the blocked node set as the new initial node and the goal node remaining the same. The pheromone left by the previously found best path was left in the program's memory, so that if the new ants found their way back to the previous best path, they could follow that rather than having to calculate the entire route again.

The algorithm presented here worked in the following fashion: first, the simulated map was generated by MATLAB. The map contained random pathways which were known by the robot, and blocked pathways which were unknown to the robot. A random map was used to allow larger test maps to easily be created to test the algorithm, but the code could easily be modified to accept an occupancy grid instead. Next, the start and goal nodes were selected. The improved algorithm was then run to determine the best path. If the path found by the algorithm did not pass through any blocked corridors then the goal was reached. If the path did intersect with a blocked corridor, then the node before the blocked node was set as the new initial node and the map was updated to include the blocked node. The goal node remained the same, and the Improved Max-Min ACO algorithm was run again but with the pheromones from the previously determined path left to help guide the new ants. The best path from the robot's current location to the goal was then determined. This process repeated itself until the goal position was reached.

4. 1. Test Map Generation

In order to test the performance of the Improved ACO algorithm for path planning, random maps needed to be generated. The maps used for this experiment were path maps, which represented the

connections in a grid of nodes. The basis for this style of map is that all (x,y) nodes of the map are reachable, however thin walls in between these nodes block certain adjacent nodes from being reached. The path map is represented by a $x \times y \times 8$ matrix. The user declared the number of nodes in both of the x and y directions. For each node (x,y), there were 8 possible directions of travel to a neighbouring node. A connected node in the direction given by the direction index between 1 and 8 was represented by a 1, whereas a 0 represented a blocked node. The direction index was representative of the directions shown in Figure 1.

The number and directions of connected pathways were generated randomly for each node. Once the pathways were generated, they were displayed graphically to the user in a greyscale image. The user was then prompted to select a node for the robot's starting position, followed by selecting a node for the robot's desired goal position. Figure 2 shows an example of a map used for testing the algorithm.

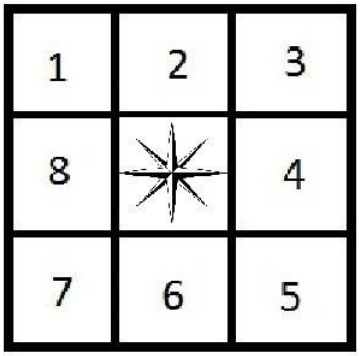

Fig. 1. Direction indices.

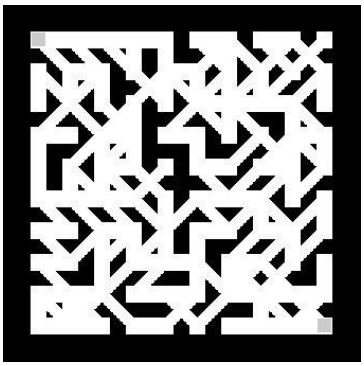
Fig. 2. Example test map.

4. 2. Modifications to the Max-Min ACO Algorithm

The Max-Min ACO algorithm was modified to achieve faster convergence speeds while still finding shorter paths. The main issue found with the Max-Min ACO algorithm was that although it converged on a solution quickly, the shortest absolute path took much longer to find. To compensate for this, a high weighting value was used for the heuristic factor of the first ant. This was done to replicate the Bug2 algorithm presented by Lumelsky and Stepanov (1987) as explained earlier. The issue implementing the Bug2 algorithm in this way was that it could get trapped in dead ends on the map. This was fixed by having the ant knowing its old positions and knowing not to follow the same path again. If the ant found its path again, it knew that following it again would result in falling into a loop, and therefore would go to a different node instead. This provided an excellent baseline best path for the following ants to work with. Once the first ant completed its journey the pheromone trail was updated. As the Max-Min ACO algorithm only saves paths if they are shorter than the best path, using the Bug2 algorithm allowed for faster convergence speeds as shorter paths were saved from the beginning. After the first ant made its way to the goal, the weighting value for the heuristic factor was lowered to allow for more exploration, and also so that the pheromone factor would be more effective as well. This was the only time adjustments were made to the weighting values, afterward the algorithm used a well-balanced probability function with the heuristic and pheromone factors weighted evenly.

Algorithm 1 shows the pseudocode for the Improved Max-Min ACO algorithm presented in this paper.

Algorithm 1. Improved Max-Min ACO Algorithm

Create randomly generated map
Allow user to select the robots start and goal position
Initialize pheromone for all paths to one
while $RobotPosition \neq GoalPosition$ **do**
  while $Ants \leq TotalAnts$ **do**
    for $i \leftarrow 1, Np$ **do**
      $Ants = Ants + 1$
      $AntPosition \leftarrow StartPosition$
        while $AntPosition \neq GoalPosition$ **do**
          if $Ants = 1$ **then**
            $hweight \leftarrow 5$
            Move ant using Bug2 Algorithm
            return $Position$ and $Direction$
          else
            $hweight \leftarrow 1$
            Move ant using standard ACO Algorithm
            return $Position$ and $Direction$
          end if
          $antPaths(i, move, :) \leftarrow [posx\ posy\ direction]$
          $dist(i) = dist(i) + distanceTravelled$
          $move = move + 1$
        end while
        if $ants = 1$ **then**
          $bestpath \leftarrow antpaths(i, :, :)$
          $bestdist \leftarrow dist(i)$
        else if $dist(i) \leq bestdist$ **then**
          $bestpath \leftarrow antpaths(i, :, :)$
          $bestdist \leftarrow dist(i)$
        end if
      end for
      Draw best path on the map for the user to see
      {Evaporate all pheromones}
      $pher = pher(1 - \rho)$
      if $pher < minpher$ **then**
        $pher \leftarrow minpher$
      end if
      {Reinforce pheromones on best path}
      $pher(bestpath) = pher(bestpath) + delta$
    end while
    Simulate robot moving… Check for blockages along desired path…
    if {*path is blocked*} **then**
      $startPosition \leftarrow robotPosition$
    else
      {Goal Reached!}
    end
end while

5. Results and Comparison

To quantify how well the Improved Max-Min ACO algorithm performed, solutions found using this method were compared to the results found using both the standard ACO and the unmodified Max-Min ACO algorithms. For proper comparison, each algorithm was performed on the same map using the same parameters. Each population consisted of 50 ants. Each algorithm would run until convergance occured or until the maximum allowed number of ants had been reached. Each algorithm was tested 10 times per map. Three map sizes were used for this experiment: 10×10, 20×20, and 40×25. Three performance factors were evaluated: the minimum length solution found, the average solution length, and the execution time. Table 1 presents the results of the simulation.

Table 1. Performance Comparison of ACO Methods

10×10 map	**ACO**	**Max-Min ACO**	**Improved Max-Min**
Min. Path Distance	351.1	351.1	289.7
Avg. Path Distance	352.3	352.3	297.9
Avg. Processing Time (s)	14.6	1.36	1.17
20×20 map	**ACO**	**Max-Min ACO**	**Improved Max-Min**
Min. Path Distance	737.4	737.4	636
Avg. Path Distance	823.9	753.7	670.1
Avg. Processing Time (s)	33.1	8.27	5.05
40×25 map	**ACO**	**Max-Min ACO**	**Improved Max-Min**
Min. Path Distance	1082.3	1127.1	1098.8
Avg. Path Distance	1161.7	1154.2	1153.7
Avg. Processing Time (s)	52.7	9.86	7.06

As can be seen in Table 1, for a small 10×10 map the Improved Max-Min algorithm found substantially shorter average and absolute distance paths than either the basic ACO or the unmodified Max-Min methods. It also had the fastest processing speed. In a similar vein, for the 20×20 map the improved algorithm found much shorter average and absolute paths than the other methods. Again, the processing time was much shorter than for ACO, and still shorter than for Max-Min. For the 40×25 map, the results were less encouraging. ACO found the absolute shortest path, followed by the Improved Max-Min method, with the basic Max-Min method finding the longest path. The improved method however did have a slightly shorter average path length compared to the other two methods. Again however, the Improved Max-Min algorithm showed significantly faster processing speeds.

It is believed that for the small and medium sized maps, the Bug2 portion of the algorithm was able to solve large portions of the optimal path at the beginning of the run. This allowed for rapid convergence with excellent path lengths. For longer, more complicated paths however, the Bug2 portion was less effective. While the improved algorithm still converged much more rapidly than either of the other methods, its results were only comparable to the other methods rather than significantly better as shown with the smaller maps. The faster processing speed makes the Improved Max-Min ACO algorithm presented here attractive for practical applications. Faster processing speeds are especially advantageous for dynamic problems when the algorithm may be executed multiple times.

Figures 3-5 show the path maps for each map size tested. The red lines show the paths found by the basic ACO algorithm, the green lines show the paths found using the Max-Min ACO algorithm, and the blue lines show the paths found using the Improved Max-Min ACO algorithm.

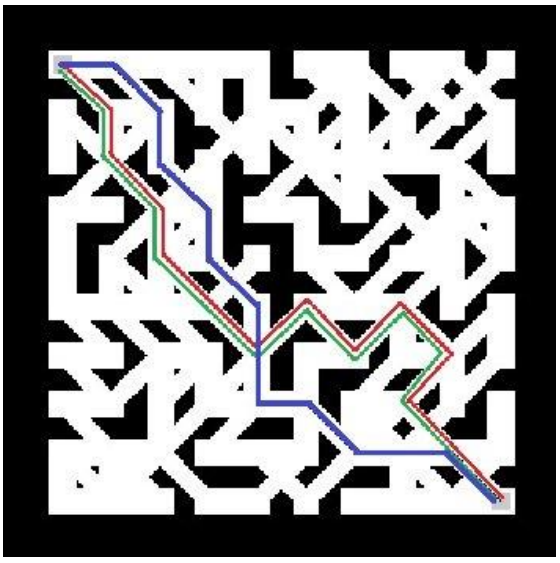

Fig. 3. Route Comparison - 10 × 10 map. The red line represents the basic ACO result, the green line is the unmodified Max-Min result, and the blue line is the Improved Max-Min result.

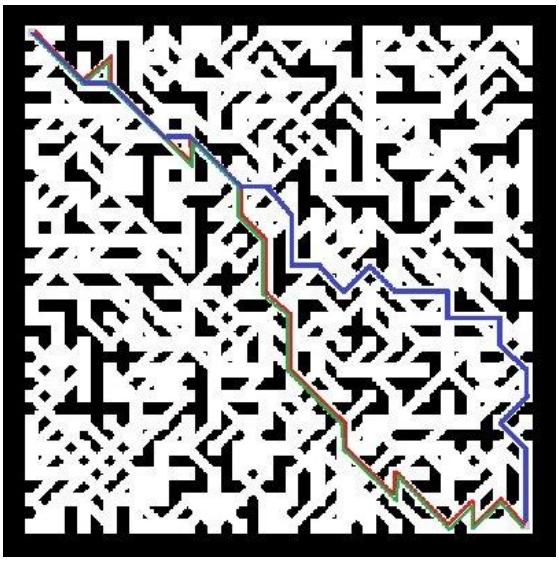

Fig. 4. Route Comparison - 20 × 20 map. The red line represents the basic ACO result, the green line is the unmodified Max-Min result, and the blue line is the Improved Max-Min result.

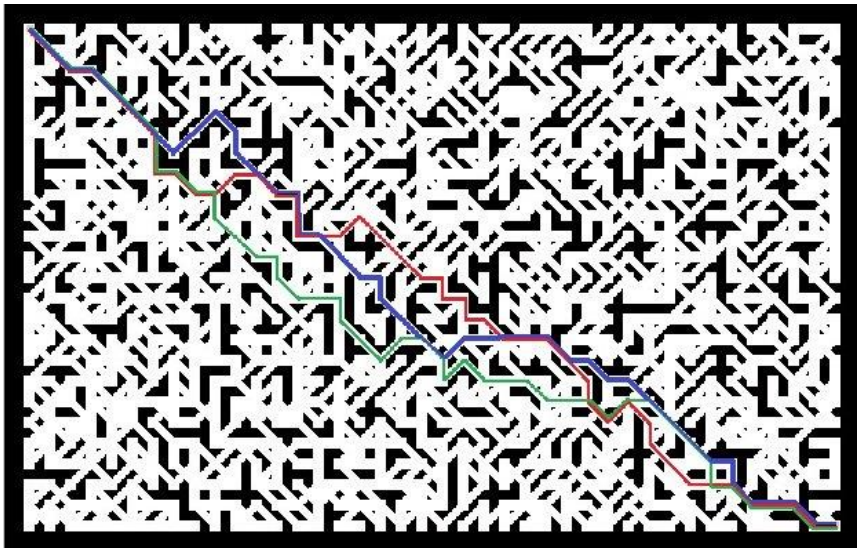

Fig. 5. Route Comparison - 40 × 25 map. The red line represents the basic ACO result, the green line is the unmodified Max-Min result, and the blue line is the Improved Max-Min result.

5. 1. Results of Dynamic Simulation

To determine if the Improved Max-Min ACO would work with a dynamic environment, tests were run with dynamic objects being added to the map. Figure 9 shows the path found using the Improved Max-Min algorithm on a dynamic map. The blue line shows the path that robot would take after being blocked by two obstacles. The red sections show where the paths were before the obstacles were added. As can be seen in the figure, the first obstacle resulted in an almost entirely new path being calculated. With the second obstacle however, only a short detour was required before the algorithm converged on the previous path again. The results show that the Improved Max-Min ACO algorithm works for dynamic environments.

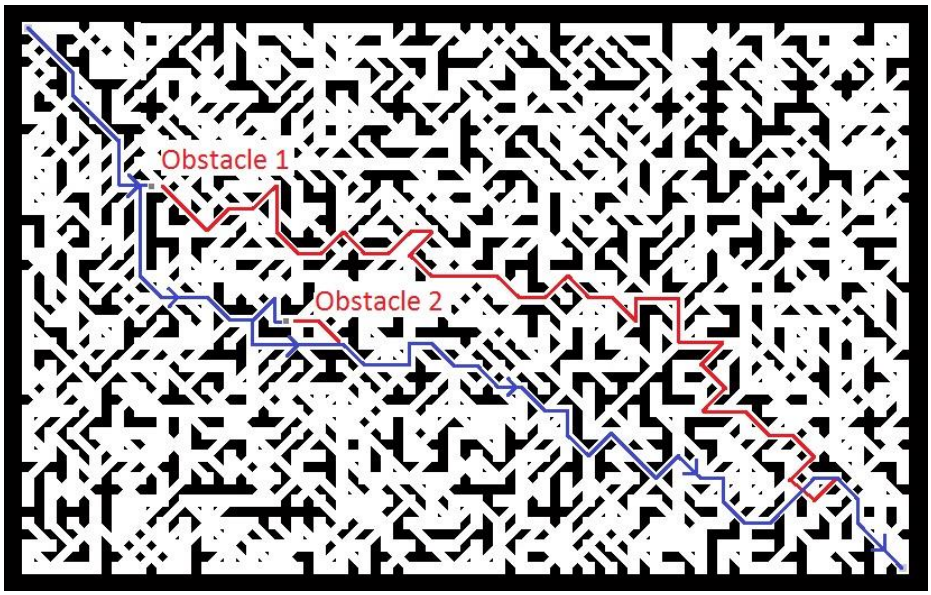

Fig. 6. Paths found on a map with obstacles

6. Conclusions

This paper presented an Improved Max-Min ACO algorithm for dynamic path planning. By adding the Bug2 algorithm to the Max-Min ACO algorithm, faster convergence speeds with better solutions were found. Leaving the old pheromone trails when re-initializing the system when the path was blocked also increased the processing speed. It is believed that this method would be viable for global path planning. The Improved Max-Min ACO method is superior to the standard ACO and Max-Min ACO algorithms, as it results in shorter average path lengths and vastly reduced processing times. Future work on this algorithm would include testing the algorithm with an actual robot in a known environment with moving obstacles.

References

Brand M., Masuda N., Wehner N., Yu X.-H. (2010). Ant colony optimization for robot path planning "Proc. of Int. Conf. Comp. Design Appl. Vol. 3," Qinhuangdao, China, Jun. 25-27, pp. 436-440.

Han Q., Wang Q., Zhu X., Xu J. (2011). Path planning of mobile robot based on improved ant colony optimization "Proc. of Int. Conf. Consumer Elec. Comm. Networks," Xianning, China, Apr. 16-18, pp. 531-533.

Hsiao Y.-T., Chuang C.-L., Chien C.-C. (2004). Ant colony optimization for best path planning "Proc. of Int. Symp. Comm. Info. Tech. Vol. 1," Sapporo, Japan, Oct. 26-29, pp. 109-113.

Lumelsky V. J., Stepanov A. A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2(1-4), 403-430.