Improvement on shortest path algorithm in tactical decision support systems

Petr Stodola, Jan Mazal

Abstract—This paper deals with an optimal path-finding algorithm being used in tactical decision support systems which has been developed at the University of Defence in Brno. The article presents improved versions of the original algorithm which are demonstrated while searching for an optimal path for a ground autonomous robot in a general environment. The article shows two different approaches for the algorithm improvement, along with their basic principles. The possibilities of the improvement are analyzed on a particular example and the results of the new versions are compared with the original algorithm.

Keywords—Optimal path-finding algorithm, autonomous vehicles, tactical decision support systems

I. INTRODUCTION

Tactical decision support systems have started being trends in combat management recently; all modern armies are interested in these new possibilities. Especially the US Army has been dealing with this issue deeply within its program Deep Green; it is a system for decision support intended for commanders of the US Army [1].

The purpose of those systems is to use computer technology for effective and accurate predictions of possible situation scenarios and to facilitate evaluation of potential results of commanders’ decisions. Predictions are based on thorough analysis of a present situation considering huge amount of input parameters. The systems are characteristic by their advanced prediction capabilities.

The University of Defence in Brno also deals with these issues within its research program. Presently, there are several projects on using autonomous vehicles for increasing the efficiency of commanders in operations. One of the key projects is the development of a ground autonomous vehicle for reconnaissance and combat purposes. Within the project, we developed an experimental autonomous robot to demonstrate and verify algorithms and principles of autonomous motion in a general environment [2], [3], [4].

The article considers an improvement of the optimal path-finding algorithm which the robot uses during its movement from its current position to the target position (the algorithm is used also for other tasks of tactical decision support). There are presented two different variants of the improvement in the article, along with deeper analysis and evaluation.

II. ORIGINAL OPTIMAL PATH-FINDING ALGORITHM

While moving, the robot uses the algorithm based on Floyd-Warshall principle with essential structural and optimization modifications which makes the algorithm computationally usable for vast data structures with more than $10^9$ nodes. The algorithm is designed for parallel processing and is implemented on a graphical processor in CUDA integrated development environment [5].

The basic prerequisite for the algorithm is evaluation of all graph edges by non-negative integers (costs). An example of the algorithm function in the real terrain is shown in Fig. 1. Costs were set according to influence of several factors: terrain relief, vegetation, waters, roads, and visibility of the threatening element.

![Fig. 1 Example of the optimal path-finding algorithm](image_url)
The algorithm is able to find all variants of the shortest paths when there are more of them. For illustration, Fig. 3 presents three variants of the shortest path from the initial point to the target point. All three paths are evaluated by the same cost. It is apparent that the Euclidean distance of the middle path is lesser than the distance of the remaining two. The main problem is the fact that the algorithm is not able to recognize the Euclidean shortest path what is a quite important criterion for optimal autonomous motion of unmanned vehicles.

There are designed two new versions of the algorithms in the following text dealing with the presented issue.

III. VERSION WITH 16 EDGES FROM EACH NODE

The version in this chapter is very easy to implement since the only change in comparison with the original version is extending the number of edges from each node of the grid from 8 to 16. The principle is shown in Fig. 4 on the left. Fig. 4 on the right presents integer costs which are based on the Euclidean distances of individual edges.

The principle of extending the number of edges does not solve the problem completely. It is only way how to reduce the problem slightly. Moreover, the double amount of edges results in the significant increase of the running time of the algorithm. Deeper evaluation of the method is presented in the following text.

IV. VERSION WITH ADDITIONAL ANALYSIS OF THE PATH

The second approach for the problem solution uses additional analysis of the path found by the original algorithm. When there are more paths (e.g. as in Fig. 3), then it is possible to use whichever one.

The principle consists in sequential processing of particular path sections and checking if the sections are possible to be shortened by lines with the shortest Euclidean distance. The algorithm is presented in Fig. 5 in pseudocode. The function `FindPath (path[], size, A, B)` searches for the shortest path from points A to B by the original algorithm. Points of this path are stored in array `path`; total number of points is in variable `size`.

```plaintext
1. FindPath (path[], size, A, B)
3. start = 0
4. newsize = 0
5. while ( start <= size ) do
6.   for ( i = start to size ) do
7.     if (TestPath(path[],start,i) == FALSE) do
8.       newsize += AddPath(newpath[],start,i-1)
9.     start = i
10.   break for
11. Store newpath, newsize
```

In the algorithm, there are two functions TestPath and AddPath. The function `TestPath (path[], start, i)` examines the path section stored in array `path` between points with indexes `start` and `i`. The function returns TRUE if cost of the path section is the same as cost of the path given by a line between both points, or FALSE otherwise. The principle is demonstrated in Fig. 3 where the middle path is given by a line between two points and it has the same cost as both adjacent paths.

The function `AddPath (newpath[], start, i-1)` stores a new path section given by a line between points of the original path with indexes `start` and `i-1`. It returns the number of points added to the new path.

As a result of the algorithm, we have a new path with the same cost but with lesser (or the same at worst) Euclidean distance. The big advantage of the principle is its linear running time $O (size)$ where `size` is the number of points of the path found by the original algorithm.

V. EVALUATION AND COMPARISON OF NEW VERSIONS

This chapter evaluates and compares the original algorithms with the new versions. Evaluation was taken place in our simulator of the experimental autonomous vehicle [7]. Fig. 6 shows the environment configuration where the analysis was conducted. The area is of size about 12 $\times$ 12 meters. Black squares represent obstacles in the area.
Fig. 7, 8, and 9 present paths of the autonomous vehicle acquired from the simulator. Green colour shows the path from the initial to the target position; costs of individual graph nodes are coded in gray shades. The path computed by the original algorithm is in Fig. 7; Fig. 8 presents the path from the algorithms with 16 edges and Fig. 9 presents the path from the version with additional analysis. We can see progressive improvement of the results in figures.

Table I shows several parameters obtained from the experiments. It is apparent that the algorithm with 16 edges provided better results than the original algorithm; nevertheless the average running time was more than twice longer which is caused by the double number of edges. The total distance covered was about 2.5 % shorter. Number of algorithm launches is smaller as the algorithm is able to move by two points in one direction (see Fig. 4).
The algorithm with additional analysis of the path found seems to be the best option. The average running time was affected only insignificantly (about 0.6 ms); it is caused by the linear running time of the additional analysis. The total distance covered was about 3.4 % shorter. Fig. 9 shows that there is the path with the Euclidean shortest distance.

VI. CONCLUSION

This paper deals with issues of improving the shortest path-finding algorithm in discrete state space. We designed two improving variants; both were compared on the particular example in our simulator of the experimental autonomous vehicle.

The obtained results are valid only for the selected example. Results can differ in other situations. Nonetheless, the both versions of the algorithm ensure the same or shorter path in comparison with the original algorithm. The best results are provided by the algorithm with additional analysis of the path.

An advantage of the algorithm with 16 edges from each node is its easy implementation. On the other hand the running time is more than double. The additional analysis affects the total running time negligibly as the running time of the analysis is linear $O(n)$.

REFERENCES


TABLE I

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Original algorithm</th>
<th>Algorithm with 16 edges</th>
<th>Algorithm with additional analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of graph edges</td>
<td>1,280,000</td>
<td>2,560,000</td>
<td>1,280,000</td>
</tr>
<tr>
<td>Number of algorithm launches</td>
<td>455</td>
<td>349</td>
<td>455</td>
</tr>
<tr>
<td>Average running time</td>
<td>57.6 ms</td>
<td>134.1 ms</td>
<td>58.2 ms</td>
</tr>
<tr>
<td>Total running time</td>
<td>26.2 s</td>
<td>46.8 s</td>
<td>26.5 s</td>
</tr>
<tr>
<td>Total distance covered by vehicle</td>
<td>15.70 m</td>
<td>15.31 m</td>
<td>15.17 m</td>
</tr>
<tr>
<td>Total distance in %</td>
<td>100 %</td>
<td>97.5 %</td>
<td>96.6 %</td>
</tr>
</tbody>
</table>