Improving Path Planning Methods in 2D Grid Maps

Viet-Hung Dang[1], Hoang Huu Viet[2], Nguyen Duc Thang[3], Ngo Anh Vien[4], Le Anh Tuan[5]
[1] Faculty of Information Technology, Duy Tan University, Danang, Vietnam.
[2] Institute of Engineering and Technology, Vinh University, Vietnam.
[3] Biomedical Engineering Department, International University-Vietnam National University, Ho Chi Minh, Vietnam.
[4] School of EEECS, Queen's University Belfast, Belfast, UK.
[5] Department of Automotive Engineering, Vietnam Maritime University, Vietnam.

* Corresponding author. Tel.: +84905070709; email: dangviethungha@gmail.com
Manuscript submitted October 10, 2019; accepted December 6, 2019.
doi: 10.17706/jcp.15.1.1-9

Abstract: Path planning for 2D grid maps has been studied for a long time, however the substantial leaps have just been achieved with recent series of Theta* variations, which allow the found path more efficient and realistic. Since the search is on grid, optimality is not guaranteed by incremental searching methods including Dijkstra, A*, Theta* and their variations like A*-PS, Lazy-Theta*, etc. In this paper, we present several attempts of improving sub-optimal path planning techniques in 2D grid maps. The combination of these attempts has reached results that are asymptotic to the optimal solution in terms of path length. The runtime for this method is much lower compared to that of the brute-force A* on Visibility Graphs.

Key words: Path planning, grid map, 2D map, incremental search.

1. Introduction

Path planning in 2D maps plays a very important role in computer science when frequently encountered in robotics, video games, routing and navigation [1]-[6]. 2D maps are represented with different kinds of grids, such as square, triangle, trapezoid, etc. Square grid is often used thanks to its simplicity without loss of generality. For the path searching on 2D maps, there have been many methods even since the early days of computer science. Dijkstra can be considered as the earliest and most basic incremental method for finding a path connecting 2 points in a grid map. It is actually a variant of Breadth-First Search with the cost information. Then A* was proposed to reduce the explored space by adding the heuristic information. Both Dijkstra and A* give optimal shortest path through a list of consecutive neighbors. However, a sub-path constructed by connections between neighbors can be replaced by a direct path if there is no obstacle between its two ends. Hence, the path can be shortened but Dijkstra and A* no longer guarantee the optimality with the path composed of these straight lines.

In order to deal with paths constructed with straight segments that connect any two points on grid as long as they do not cross any obstacles, Theta* (consisting of Basic-Theta* and Angle Propagation [7]) has been proposed based on the architecture of A* where the search is also incremental. The key idea of Theta* for deciding the consecutive links is at the phase of choosing 'parent' node, which is the relation of nodes to construct the search tree. If there is a line-of-sight (LOS) between a node and its 'parent's parent' (grand-parent), then it assigns 'grand-parent' to be its 'parent'. In that case, a direct straight line is connected, and the cost function of a node is updated and therefore not monotone. As a result, admission

condition [8] is violated and the optimal solution for the shortest path is not assured. The only method for the optimal path length so far is the A* on Visibility Graphs (A*-VG). Where all the light-of-sight nodes of a checked node are considered as the neighbors and the search is conducted on the convention tree. This is a brute-force search where the full table of distances between all pairs of nodes at the corners of all obstacles, thus the search time increases exponentially with the size of the maps. The runtime for A*-VG can be reduced by constructing this table gradually during the search, in which unnecessary nodes and distances are ignored. This is called pruning A*-VG. In [9], Anya has been proposed with the claim of optimality but without experimental support. Moreover, the light-of-sight check must be done for all nodes at the ends of the line segments and thus it is, like A*VG, actually another pruning brute-force method. Study for improving the incremental search methods that avoid exploring brute-force have been conducted, including several Theta*'s variants such as Angle Propagation, Lazy Theta* and Batch-Theta*. Angle Propagation (AP Theta*) stores the angle ranges and propagates this information along the grid edges for the purpose of line-of-sight checking. Statically, AP Theta* gives longer path length than Basic Theta*. Meanwhile, Lazy Theta* skips calculating cost value for nodes in waiting list, thus compresses the runtime but lengthens the path length of Theta*. Batch-Theta* provides a modification in light-of-sight checking allowing detecting the blocking position when the light-of-sight is not true. Although the performance with Batch-Theta* is improved, the found paths are sometimes unpractical and even easy to recognize with human eyes that they are not the shortest path. This paper will investigate several techniques applying to the current Theta* framework in order to asymptotically reach the optimal path lengths and show how the trade-off running time is affected.

2. Theta* Weaknesses

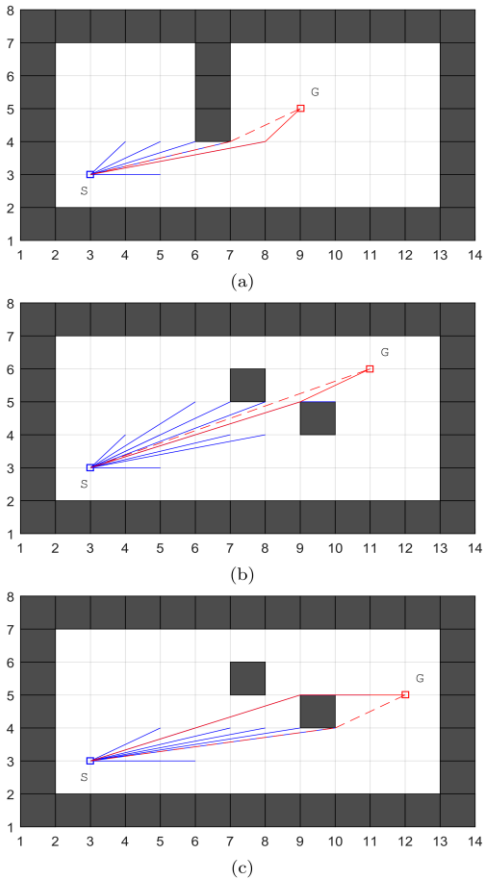


Fig. 1. Weaknesses of Theta*, where S: start node, G: goal node, blue line: parent-child link, dashed red line: true shortest path, red solid line: found path by Theta*.

The shared structure for static path planning methods, such as Dijkstra, A*, Theta* and their variants where $h(s)=0$ for Dijkstra, can be found in Algorithm 1 in [10]. This framework repeatedly checks every point in the waiting list open list and adds the neighbor points of the checked point to open list. Checked points are stored in closed list in order to avoid being checked again. The algorithms loop until the goal is reached or open list is empty. By tracing back along the points' parent-child relationships, the path is constructed when the goal is reached. The two functions $UpdateRelation(s?\ parent)$ and $UpdateCostEstimation(s?\ f)$ in Algorithm 1 in [10] choose a better parent for s' given the current explored information.

Nevertheless, this framework fails to find the optimal path through consecutive line-of-sight points as described in Fig. 1 [7]. Even when using coefficient w for the purpose of tuning the carefulness of the search [11], [7], these failures are not overcome. In other words, if the cost function is $f(s)=g(s)+w.h(s)$, no matter how w is tuned to make the search extremely careful $(w=0)$ or careless $(w=\infty)$, the optimal paths in these examples cannot be found. Therefore, this paper does not focus on this tuning technique, although it can be combined with any techniques we mention in the following Section.

3. Improving Theta*

4.1. Line-of-Sight Check

We proposed a new line-of-sight check in [10], called $Candit\_LOS(s_a,s_b)$. The main property of this checking function is illustrated in Fig. 2. $Candit\_LOS(s_a,s_b)$ locates the cell-edges that cut the segment between s_a and s_b. Along the checked segment, if an obstacle is detected, its specific cell-edge is identified and the two points of this edge (i.e., s_1 and s_2 in Fig. 2) will be considered if either of them can be a good middle point that links s_a and s_b. Meanwhile, when function $LOS(s_a,s_b)$ returns false, it does not utilize the information during the check while keeping the state of parent-child link unchanged, which makes s be the middle point between s_a and s_b.

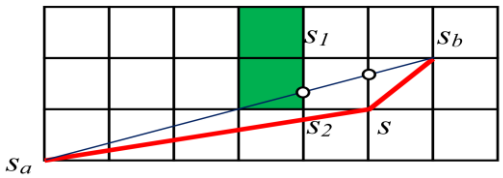


Fig. 2. Illustrations of how *Candit LOS()* works and overcomes the first Theta*'s weakness.

Thanks to the design of the algorithm in integers, $Candit\_LOS(s_a,s_b)$ does not increase much checking time, but allows Theta* to overcome the first weakness (see Fig. 1a). It can reduce the searching time in the map with low number of obstacles (see [10]) by assigning correct g-values to intermediate nodes and therefore compressing the expanded space. Nevertheless, this technique cannot help overcome the second and third weaknesses. The solutions for further improvement are presented in the next subsections

4.2. Line-of-Sight Check Skipping

Note that Theta* only updates the parent-child relations by the LOS detection from a node's neighbors to its parent. This causes the second weakness because the LOS from a node's neighbors to its grandparent are not examined. If LOS is applied to these additional tasks, the computational cost is increased considerably since it should be applied repeatedly to all of the current expanded node's ancestors until no LOS is found.

This paper takes these additional LOS tasks, but in a way that avoids unnecessary LOS checks. A fast pre-test is performed in order to determine if $Candit\_LOS()$ is needed.

The idea of the pre-test is explained via the example in Fig. 3. Node g4 is currently expanded with the 8 neighbors f3, f4, f5, g3, g5, h3, h4 and h5. At this state, d4 is g4's parent and b2 is d4's parent. Due to Basic Theta* algorithm, the neighbors will see if d3 could be their parent but do not check the parent candidate b2. To overcome this, each neighbor has to use function $LOS()$ to repeatedly check all nodes along its ancestor line until getting the result *False*. This surely increases the computing cost significantly because $LOS()$ takes much time compared to other processes of the search. We propose to reduce the number of using $LOS()$ by conducting a pre-test to rule out several non-line-of-sight connections from the neighbors (e.g., h4) to their grandparent (e.g., b2). This check needs the information of coordinates of the related nodes only without any loops. The idea is explained through examples in Fig. 3 as follows:

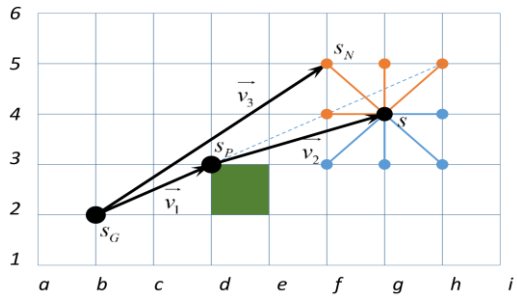


Fig. 3. Pre-test for skipping unnecessary LOS checking. Blue nodes do not have LOS to s_G and only orange nodes should be checked LOS to s_G.

Since g4 has LOS to d3 (g4's parent) and no LOS to b2 (g4's grand parent), there must be an obstacle with at least 1x1 of size. The straight line containing vector $\overrightarrow{b2d3}$ splits the search space into 2 half-plane. All g4's neighbors {f3, g4, h3, h4} in the same half-plane with the mentioned obstacle (green block in Fig. 3) will be in the shadow of the obstacle from the view of b2 (g4's grand parent). Note that if g4's grand parent has the direct LOS to g4's neighbors in the same half-plane, then Theta* will find another path to them via updating the parent-child relation based on g-values. This implies that these neighbors do not have LOS to b2 and it's not necessary to apply $LOS()$ for checking. In order to skip these LOS checks, the algorithm must be able to recognize the in-shadow neighbors based only on their coordinate information.

It can be seen that the neighbors the current expanded g4 is in the same half-plane with the obstacle since the obstacle blocks b2 and g4 but does not block either b2 and d3 or d3 and g4. As the result, if g4 and its neighbors are in the same half-plane split by $\overrightarrow{b2d3}$, they surely have no LOS to b2. In contrary, the neighbors in the other half-plane of g4 are the ones needs LOS check to their grandparent. For the special case of the neighbors on the splitting line, there certainly are the LOS between them and their grandparent.

Let s_G be grand parent of s, s_P be parent of s, s_N be neighbor of s, and $\vec{v_1} = \overrightarrow{s_G s_P}$, $\vec{v_2} = \overrightarrow{s_P s}$, $\vec{v_3} = \overrightarrow{s_G s_N}$. Then, due to the above analysis, the algorithm needs to recognize the neighbors that are not included by the half-plane containing the current expanded node. Hence, the directional angles $(\vec{v_1}, \vec{v_2})$ and $(\vec{v_1}, \vec{v_3})$ must have opposite signs ($\vec{v_1}$ indicates the splitting line). Because we do not have to calculate the exact angles but just the signs of these angles, we use vector cross product formula to recognize this feature:

$$p = (\vec{v_1} \times \vec{v_2})(\vec{v_1} \times \vec{v_3}),$$

where $\vec{v_i} \times \vec{v_j}$, is the cross product of two vectors $\vec{v_i}$ and $\vec{v_i}$, which can be calculated by

$$\overrightarrow{v_i} \times \overrightarrow{v_j} = \mathbf{DET}[\overrightarrow{v_i}, \overrightarrow{v_j}],$$

as the determinant of the matrix composed of 2 column vectors $\overrightarrow{v_i}$ and $\overrightarrow{v_j}$. If $p > 0$, there is no need to check LOS between s_N and s_G. If $p = 0$, there surely is LOS between them. Note that p in Equ.(1) is error-free from the floating point errors in computer system because all the calculations deal with integers only (vectors' components are integers).

4.3. Chances for Other Candidate Paths

From Fig. 1c, we can see that the shorter path is missed because the algorithm stops when a goal is reached. It means the other paths to the goal are not examined to see if they are shorter. We modify the algorithm to give the intermediate nodes that construct a longer path length a chance to be expanded because the results of additional exploration may lead to a better final path length. In other words, we allow the search to continue even when the goal is found. The arising problem is that if the goal is put into *closed_list*, what the conditions to stop the algorithm are.

Note also that the algorithm must update the information in *open_list* in terms of elements and their attributes (g-values, f-values and parent-child relations). Therefore, we can utilize the state of *open_list* to control the breaking condition based on the maximum f-values of the *open_list*'s elements. Specifically,

$$stop = (\max\{s_i.g\} > s_{goal}.g + d), s_i \in open\_list;$$

where $d > 0$ is the elastic parameter to widen the expanded space, called WES. Higher value of d leads to more expansion and vise versa, since $\max(s_i.g)$ grows gradually when *open_list* continuously pops out the element with lowest g-value.

This mechanism gives more opportunities to examine more candidate paths and overcome the weakness mentioned in Fig.1c and even the one in Fig. 1a, too. Nevertheless, it overlooks good paths if it decides bad parent-child relations of the intermediate nodes (see experiment in Section IV.*B* and Fig. 5 for more details). It is necessary to reduce the bad parent-child assignments as many as possible. We apply the simple technique of re-expanding, called RE, combining with the WES search to increase more optimal parent-child links. RE basically permits a node in *closed_list* the chances to be back into *open_list* for re-check if a better candidate path goes through it. In other words, RE updates nodes' information, in terms of cost-functions and their parent-child links.

Note that the RE only can not help the algorithm solve the weaknesses in Fig. 1a and Fig. 1c whereas WES only gives many bad intermediate links. Section IV.*B* discusses about the results of applying WES and the combination of WES and RE.

Algorithm 1: Theta* with Pre-test Ancestor *Candit LOS()*, WES and RE

```
1  Data: s_S, s_G
2  Result: found_path, found_path_length
3  Push(s_S, open_list)
4  while open_list ≠ ∅ do
5  |  s = arg min_{s_i ∈ open_list} (s_i.f);
6  |  open_list = open_list \ {s};
7  |  if max{s_j.g} > s_G.g + d, s_j ∈ open_list then
8  |  |  return [found_path, found_path_length];
9  |  else
10 |  |  foreach s' ∈ succ(s) do
11 |  |  |  while pretest(s', s) <= 0 && Candit_LOS(s', (s.parent).parent)
   |  |  |  do
12 |  |  |  |  Update(s'.g, s'.parent);
   |  |  |  end
13 |  |  |  if s' ∉ open_list then
14 |  |  |  |  Push(s', open_list);
   |  |  end
   |  end
15 return "No path found";
```

Algorithm 1 is the simplified pseudo-code of our proposed method which improved Theta* by adding pre-test ancestor *Candit_LOS()*, WES and RE. $s_i.f = s_i.g + s_i.h$, where $s_i.h$ is the Euclidean distance between s_i and s_G. Condition in line 7 is for adding WES whereas that in line 11 is for adding pre-test ancestor *Candit_LOS()*. The found path is constructed from s_G, via the links to its ancestors, to the s_S. We eliminate the use of *closed_list* because even when a node is expanded, it may return to *open_list* for re-expansion. Therefore a node now has only two states, *in* and *not in open_list* and *closed_list* become redundant.

4. Experiments

4.1. Overcoming Theta*'s Weaknesses

Fig. 4 illustrates an example of overcoming Theta*'s first weakness by using *Candit_LOS()* only. The function works properly just as analyzed in Section III.*A*. The link between node (8,4) and the goal is replaced by that between (7, 4) and the goal. Meanwhile, with extra LOS checking from the expanded node's neighbor to its grandparent node, the optimal path is found as demonstrated in Fig. 4b. For the third weakness, only WES which allows extra search without pushing goal node into *closed_list* just when goal node is found. Note that no individual technique mentioned above can overcome all the weaknesses. The details of their capabilities are presented in Table 1.

Table 1. The Effectiveness of Using Different Techniques. '+' Means being able to Solve whereas '-' Means being unable to Solve

Technique	Weakness 1	Weakness 2	Weakness 3
*Candit_LOS()**	+	-	-
Candit_LOS() with loops and pre-test	+	+	-
WES	+	-	+
RE	-	-	-

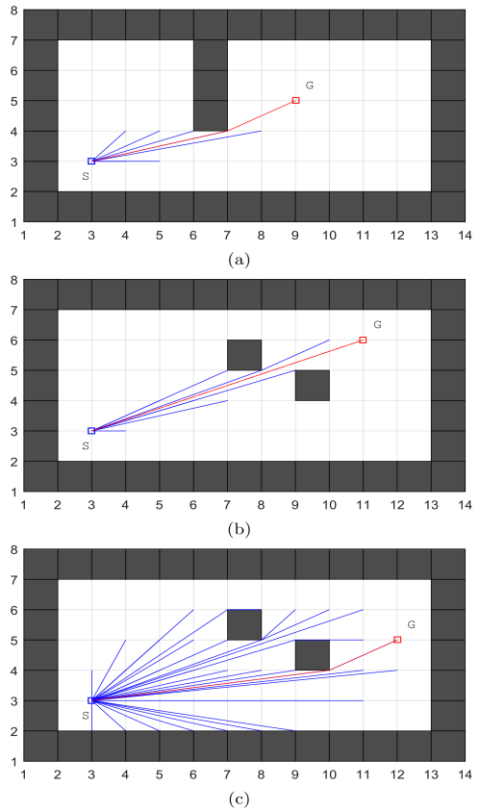


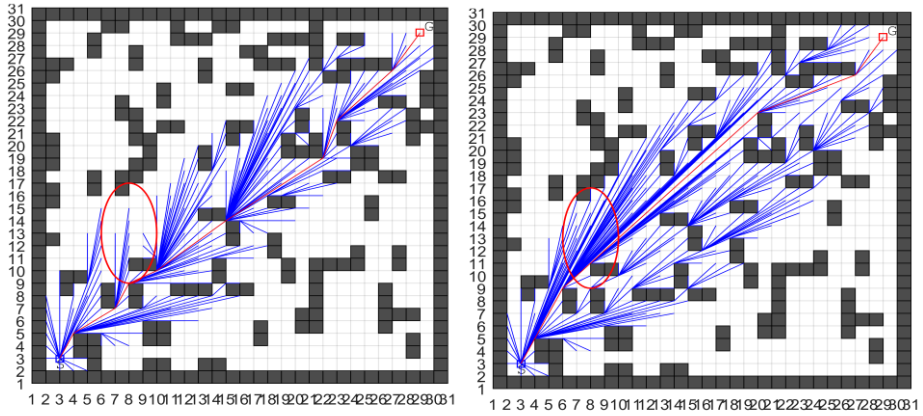
Fig. 4. Illustrations of how different techniques work and overcome Theta*'s weaknesses. Specific techniques are (a) *Candit LOS()* is used, (b) pre-test ancestor *Candit LOS()* and (c) WES.

Although the combination of WES and the function *Candit_LOS()* with loops and pre-test is proposed to improve Theta* in this paper, we find that the path length result can be further improved. This is because some intermediate nodes' relations are not assigned optimally, thus their g-values are not correct and the errors of their descendants' g-values are accumulated during the search. The red oval in Fig. 5a shows some of these wrong g-value assignments.

Once a node is added into *closed_list*, its g-value is fixed and cannot be updated if assigned wrong. Therefore, we propose to combine a simple re-expanding technique (RE) in order to give nodes in *closed_list* more chances to update their values and to be expanded again. Note that RE cannot help the algorithm overcome any of Theta*'s drawbacks as showed in Table 1. It is only useful when being combined with our WES as demonstrated in Fig. 5b. It can be seen that all the expanded nodes have the right g-values and optimal parent-child links and therefore the accumulated errors are eliminated, yielding the optimal path length solution (see nodes in the red oval of Fig. 5a and Fig. 5b for comparison).

4.2. Path Length Improvements and Trade-Off

The experiments are conducted in the computer with Intel Core i5-7200U CPU 2.5GHz and 2.7GHz and 12G of RAM. The aims of this experiment set are investigating the improvement of path lengths and the trade-off runtime. The chosen methods are (i) pruning A*-VG, (ii) Basic Theta* (iii) Theta* with *Candit_LOS()*, and (v) Theta* with the combination of pre-test ancestor *Candit_LOS()*, WES and RE. For the purpose of investigating statistical performance, we use a big map with the size of $100cell \times 100cell$. The ratio of obstacles over map size is 40% because if this ratio is lower, the differences of found path lengths by different methods are insignificant. Meanwhile, if this ratio is higher, when is more likely to exist no path between starting node and goal node. The experiment is conducted 1000 times and the result of path length and runtime are taken on average before normalized to compare with the true shortest path length found by pruning A*-VG.



(a) (b)

Fig. 5. Performance of assigning g-values and parent-child links of (a) Theta* with the combination of pre-test ancestor *Candit LOS*() and WES (d=1), and (b) Theta* with the combination of pre-test ancestor *Candit LOS*(), WES (d=1) and RE.

It is seen in Table 2 and Fig. 6 that Basic Theta* needs least time to construct a path, but its found paths are much longer compared to Theta* with *Candit_LOS()*. Basic Theta* gives 0.2% longer path length compare to the true shortest path, Basic Theta* with *Candit_LOS()* does 0.002% longer whereas Theta* with pre-test ancestor *Candit_LOS()*, WES ($d = 1$) and RE does only 0.2×10^{-7} longer. It takes pruning A*-VG about 20s to construct the best path whereas Basic Theta* needs 0.627s, Theta* with *Candit_LOS()* needs

0.61s. Overall, Theta* with pre-test ancestor *Candit_LOS()*, WES and RE needs more time than Theta* as a trade-off. However, it is designed to overcome most of Theta*'s weaknesses, thus the found paths are improved significantly and asymptotic to the true shortest paths.

Table 2. The Results on Average of Path-Length and Runtime (Seconds)

Method	Path-length ratio over true shortest path	Runtime (s)
Pruning A*-VG	1.000000000000000	19.943
Theta*	1.002250541747011	0.627
Theta* with $Candit\_LOS()$	1.000020495569852	0.613
Theta* with pre-test ancestor $Candit\_LOS()$, WES and RE	1.000000201024152	0.832

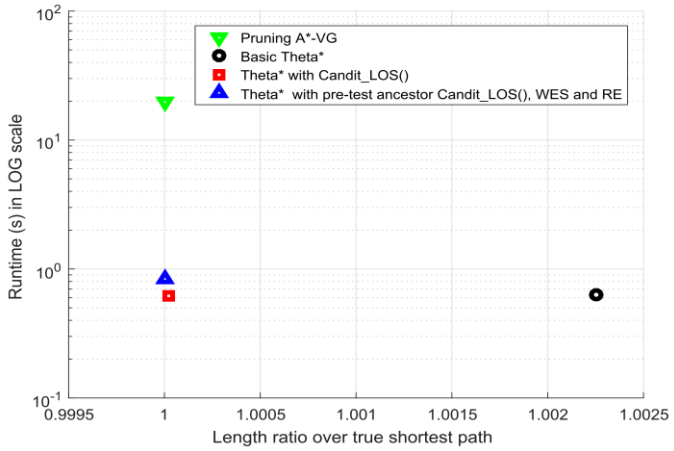


Fig. 6. Path length improvement and runtime tradeoff.

5. Conclusion

This paper discusses about Theta*'s weaknesses and points out how to overcome them. Several techniques are investigated for each mentioned weakness. An overall solution, called Theta* with pre-test ancestor *Candit_LOS()*, WES and RE, is proposed in order to improve the found path length of Theta*. The experiment results show that the proposed combined method gives the results asymptotic to that of the true shortest path and runtime trade-off is low for these improvement. It can also be inferred that the combined method could not guarantee the optimal path, but the probability for the method to miss the optimal solutions is very low. Therefore, the proposed Theta* with pre-test ancestor *Candit_LOS()*, WES and RE is a good consideration when used to construct path in 2D maps. Our future work will study on finding intermediate paths that cannot be constructed through any set of consecutive neighbors for further path improvement.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

VH Dang and HH Viet conducted the research, LA Tuan did the survey and wrote the related works; HH Viet and ND Thang did experiments to evaluate the results of both related work and the proposed idea for comparison; VH Dang and NA Vien wrote and finalized the paper; all authors had approved the final version.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.05-2016.18.

References

[1] Hwang, Y. K., & Ahuja N. (1992). Gross motion planning — A survey. *ACM Computing Surveys (CSUR), 24(3),* 219-291.

[2] Gabriely, Y., & Rimon, E. (2001). Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence, 31(4),* 77–98.

[3] Choset, H., & Pignon, P. (1997). Coverage path planning: The boustrophedon cellular decomposition. *Proceedings of the International Conference on Field and Service Robotics*. Canberra, Australia.

[4] Choset, H. (2001). Coverage for robotics — A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, *31,* 113–126.

[5] Viet, H. H., Dang, V., Laskar, M., & Chung, T. (2013). BA*: An online complete coverage algorithm for cleaning robots. *Applied Intelligence*, *39(2),* 217-235.

[6] Lopez, A. S., Zapata, R., & Lama, M. (2008). Sampling-based motion planning: A survey. *Computacion y Sistemas, 12(1),* 5-24.

[7] Daniel, K., Nash, A., & Koenig, S. (2010). Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research, 39,* 533–579.

[8] Russel, S. J., & Norvig, P. (2003). Artificial intelligence a modern approach. (2nd Ed). *Pearson Education*.

[9] Harabor, D., & Grastien, A. (2013). An optimal any-angle pathfinding algorithm. *Proceedings of 23th Conference on Automated Planning and Scheduling* (pp. 308-311).

[10] Dang, V., Thang, N., Viet, H., & Tuan, L. (2015). Batch-theta* for path planning to the best goal in a goal set. *Advanced Robotics*, *29,* 1537–1550.

[11] K¨oll, A., & Kaindl, H. (1992). A new approach to dynamic weighting. *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 16–17). ECAI '92, Vienna, Austria. New York, NY, USA: John Wiley & Sons, Inc.

Viet-Hung Dang is from Danang, Vietnam. He received his Ph.D degree at KyungHee University in 2012 under the major of computer engineering.

He has worked in Duy Tan University as a researcher from 2012 to 2016 and as the dean of Faculty of Technology since 2017. His research interests include localization in WSNs, planning computing, ubiquitous computing, and machine learning. His lately publications are mainly on 2D grid path planning and tracking for robot, reinforcement learning theory and applications.