Information Sciences 582 (2022) 618–632

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A multiple surrounding point set approach using Theta* algorithm on eight-neighbor grid graphs

Jihee Han *, Sven Koenig

Computer Science Department, University of Southern California, Los Angeles, CA 90089, United States

ARTICLE INFO

Article history:
Received 17 April 2020
Received in revised form 18 March 2021
Accepted 3 October 2021
Available online 7 October 2021

Keywords:
Multi-SPS (Surrounding Points Set)
Path planning
Grid graph
Theta* algorithm

ABSTRACT

Path planning on grid graphs has been studied for decades in several applications such as robotics and computer games. However, the process for determining short paths can still be inefficient on large grid graphs. The SPS (Surrounding Point Set) method has made it efficient by considering only a few unblocked cells. Since eight-neighbor grid graphs do not provide sufficient connectivity for SPS, it found a path on a graph where two cells are connected with an edge within a given density bound. However, verifying whether the resulting edges are blocked is time-consuming; further, it is unclear which density bound to use and restrictive to allow only edges within the bound. To address these limitations, we propose a multi-SPS with Theta* that uses eight-neighbor grid graphs, allows distant connectivity by Theta*, and improves the SPS by considering more obstacles between the start and goal cells. Our experimental results demonstrate that compared with the existing path planning algorithms, multi-SPS with Theta* and variants of Theta* can run faster and obtain similar path lengths. The results also demonstrate the beneficial effect for Theta* in that the expensive runtime of Theta* can be reduced by the SPS approach.

1. Introduction

Path planning on grid graphs has been studied for decades and is widely applied in areas such as robotics and computer games [1–3]. The grid is a simple approach that discretizes continuous spaces and represents maps as graphs. Further, grids are used in complex obstacle fields owing to their simplicity and certainty in terms of the location of nodes and edges [4,5]. In general, free space and obstacles are represented as unblocked and blocked cells, respectively, in grid graphs. Each cell serves as a node in a graph, and edges are generally restricted from each cell to its adjacent neighbors (typically, eight neighbors). It aims to find a short unblocked path from a given start cell and to a given goal cell where an agent can always move from its current cell to any one of its unblocked neighboring cells [6]. Graph search methods are generally applied to find a path on a grid graph such as A* or Dijkstra algorithm for a static and known environment [7], and LPA* (Lifelong Planning A*) or D* algorithm for a dynamic and unknown environment [8,9]. These graph search methods have recently been studied for enhancing their efficiency during searches or when practically applied by hybridizing them with other methods [5,10–12].

Path planning on grid graphs still remains inefficient, as the map size grows and there is a possibility of improving path quality where a graph is constructed with angle-restricted edges. To realize efficient path planning on grid graphs, the SPS

* Corresponding author.
E-mail addresses: jrjrhan@gmail.com (J. Han), skoenig@usc.edu (S. Koenig).

https://doi.org/10.1016/j.ins.2021.10.024
0020-0255/

(Surrounding Point Set) method, which avoids unimportant cells, was implemented in previous studies [13,14]. In this method, the search space for path planning is reduced by determining a subset of unblocked cells. To this end, a set of unblocked grid cells surrounding obstacles that collide with the straight line between the start and goal cells is identified; these unblocked grid cells are defined as CO (Critical Obstacles). Then, solely the cells around the obstacles and on the line are used for planning a path. This approach considerably reduced the number of cells and enhanced the efficiency of path planning.

The previous SPS method exhibits a limitation with respect to the construction of a graph based on the subset of unblocked cells before conducting a path planning search. Because the SPS method uses only a subset of grid cells, the typical four- or eight-neighbor grid graphs cannot help obtain a good solution for the path length. Furthermore, even with a small number of cells, constructing a full graph can consume a considerable amount of computational time because of the frequent line-of-sight checks required. Hence, connecting edges between cells within a density bound is used for graph construction. This density-bound based graph could provide more connectivity; however, the line-of-sight checks to verify whether edges are feasible should be implemented. Owing to these checks, the runtime for the density-bound based graph is higher than that for the simple eight-neighbor grid graph. Other limitations include the need to make decisions regarding the appropriate value for the bound and the impossibility to obtain distant connectivity beyond the bound value.

To avoid bound-based connectivity and use the grid graph as it is, we focus on a well-known any-angle path planning algorithm, Theta* [15–17]. The traditional shortest path planning algorithm, such as A* or Dijkstra, wherein a grid graph typically has connectivity to its eight neighbors, generally generates a longer path than the one in a continuous space and contains unnecessary heading changes in free space [7,18]. By contrast, any-angle path planning algorithms resolve the limitation of the shortest grid path planning algorithms by interleaving the path smoothing with search. In any-angle path planning, several algorithms, such as Theta*, Lazy-Theta*, and any-angle sub-goal graph algorithm, have been developed [15,16,19] and applied in various domains [20–22]. Recently, any-angle path planning algorithms have been extended to real-time environments with re-planning mechanism and to those considering path constraints for practical application [23,24]. These methods determine shorter paths than A* on the grid by propagating information across all cells of the grid, but not restricting the resulting paths to lie on the grid [25]. Unfortunately, Theta* needs to perform several line-of-sight checks to verify whether the edge is unblocked, which is time-consuming [26,27].

The present paper proposes a method called multi-SPS with the Theta* algorithm to efficiently plan a path on eight-neighbor grid graphs. This method determines a subset of cells that lie on certain lines (i.e., the line connecting the start and goal cells) and surround the clusters of blocked cells that intersect these lines. This subset of cells is then used by Theta* during the search rather than the entire set of cells. This approach resolves the limitations of the previous SPS method as the grid graph is used as it is and decisions regarding a bound value are no longer required. The approach avoids a feasibility test on each edge generated within the bound and achieves distant connections beyond the bound value by the parent-update mechanism in Theta*. From the perspective of Theta*, the advantage of the proposed method is that it resolves the issue of an expensive runtime at the expense of a slight increase in the length of the resulting path, since only a subset of cells is used by Theta* during the search, which reduces both its search space and its number of line-of-sight checks.

The proposed method can be applied to environments that require high resolution, such as game maps, as it can handle the complexity by reducing the search space [16,28]. Consequently, once geometric concepts in the proposed method are properly expanded to 3D space, it can be use of for a 3D environment because the size of set of surrounding cells would increase less sharply than the overall search space does from 2D to 3D. Moreover, it can be effectively applied to urban maps where environments are unstructured [12,29], since obstacles do not need to be defined as polygons in advance. These characteristics would ultimately contribute to the efficient adaptive path planning for dynamic environments once a certain mechanism such as modification on the set of surrounding cells is further included in the method. This is because that the only small part of search space would still be handled with respect to the environment changes, and the shape of obstacles could be easily adapted as the proposed method gets to identify it over the process of finding the set of surrounding cells.

The main contributions of this paper can be summarized as follows:

i) Resolving the issues related to bound-based graph construction in the previous SPS method by using a simple eight-neighbor grid graph with Theta*.
ii) Improving the previous CO and SPS concepts by developing a multiple operation of SPS in a practical manner.
iii) Simultaneously improving both SPS and Theta* in terms of solution quality and efficiency based on the complement incorporation.
iv) A significant decrease in the runtime of Theta* algorithm by considering only a subset of unblocked cells and reducing the number of line-of-sight checks at the expense of a slight increase in the path length.

The remainder of this paper is organized as follows. Theta* is briefly reviewed in Section 2. The definition and assumptions of path planning on grid graphs are provided in Section 3. An overview of the proposed method, along with a brief summary of the previous work, is presented in Section 4. Numerical experiments and simulation results are discussed in Section 5, and conclusions are drawn in Section 6.

2. Brief overview of Theta*

Theta* is a well-known any-angle path planning algorithm that generates a path with a connectivity between distant grid cells, rather than a limited-connectivity with a certain number of neighbors at fixed angles. It mainly consists of expansion (propagation) and a parent-update process based on line-of-sight checks. The expansion process has the same concept as the existing shortest path planning algorithms such as A*, in which each cell expands the cost to its neighbors and records itself as a parent of those expanded neighbors. This expansion is repeated until the goal cell is reached. When the goal cell is expanded with the smallest cost from the start cell, the path is obtained by tracking back from the goal to the start cell using the parent information. Theta*, by contrast, updates the parent information with the parent of the parent during expansion if the newly considered edge does not collide with blocked cells. For example, in Fig. 1, when Theta* generates a cell x while expanding a cell y, as depicted by the dotted line (a), it checks whether the straight line segment from parent z of the expanded cell to x has line-of-sight (b). If so, then it considers the short-cut that moves directly from z to x (c).

Typically, the path length can be shortened more by the parent-update process than the shortest path planning algorithms. Given the start and goal cell corners as S and G, Fig. 2 compares the paths obtained from traditional shortest path planning (left) and Theta* (right). In the left figure, each path segment is made only between eight neighbors in a fixed number of angles, demonstrating the possibility of further reduction in path length from an unnecessary heading change in free space. Simultaneously, Theta* resolves this issue by obtaining a longer and unblocked segment through the parent update, which results in shortening the overall path length as compared to the shortest grid path length.

3. Proposed approach

An efficient path planning method in a 2D grid graph is proposed based on the multiple surrounding point set approach and Theta* algorithm. The problem's definition and assumptions are presented in Section 3.1. Previous work on CO and SPS and the motivation for the proposed approach are described in Section 3.2, and the proposed method is developed in Section 3.3.

3.1. Problem definition and assumptions

This study aims to develop a method to efficiently find a path from a given start to a goal cell in a 2D grid graph by determining a subset of unblocked cells surrounding clusters of importantly considered blocked cells and using those cells for Theta*. The map is represented by a set of 2D cells, N, which is categorized into a set of unblocked or blocked cells, denoted as N_f and N_o, respectively. Start and goal cells, denoted as S and G, respectively, are given as unblocked cells. The environment is assumed to be static and known; hence, no blocked cells would move, and full information about the map is provided in advance. The edges are connected between the center coordinates of each cell. Each cell, c_i, has edges with eight neighbors. The path is represented by unblocked line segments from S to G and evaluated by its length.

Notations for the proposed approach are summarized in Table 1.

3.2. Previous work on the CO and SPS

The proposed method, multi-SPS with Theta* algorithm, is an advanced work based on the previously proposed method named CO and SPS [13,14,30]. This concept started from the observation that, in free space, not many cells remain uninvolved in the planning of the shortest path. Therefore, the method focused on identifying which cells should be considered more important than others to enhance the efficiency of path planning. A straight line between the start and goal cells was selected and used to determine which obstacles (clusters of blocked cells) are critical, denoted as CO. Once an obstacle intersects with the straight line, it is considered to be CO. Consequently, a subset of unblocked cells, denoted as SPS, was determined to completely surround the critical obstacle. Fig. 3 illustrates CO and SPS given the straight line segment between S and G. In (a), two obstacles in dark grey and unblocked cells in blue circles are identified by the straight line segment. Then, all surrounding cells for each obstacle in CO are determined (b), and finally, those surrounding cells with cells on the seg-

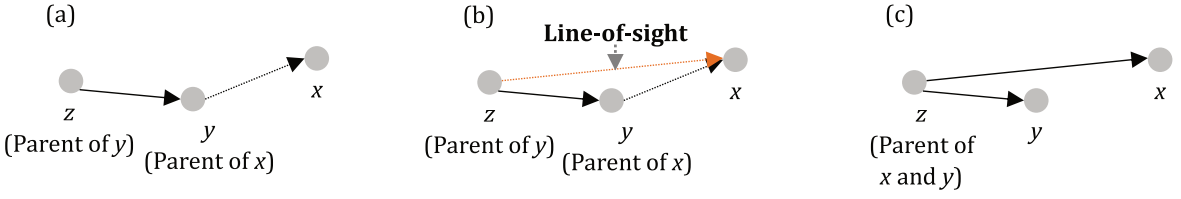

Fig. 1. Parent-update process in Theta*

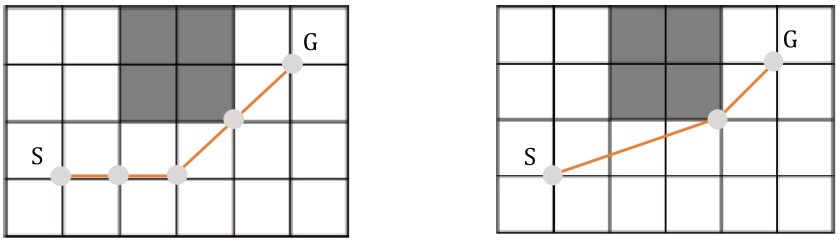

Fig. 2. Comparison between a shortest grid path planning and Theta*

Table 1
Notations.

Notation	Description
N	Set of grid cells
N_f	Set of unblocked grid cells,$N_f \subset N$
N_o	Set of blocked grid cells,$N_o \subset N$
c_i	i^{th} grid cell,$c_i \in N$
S	Start cell
G	Goal cell

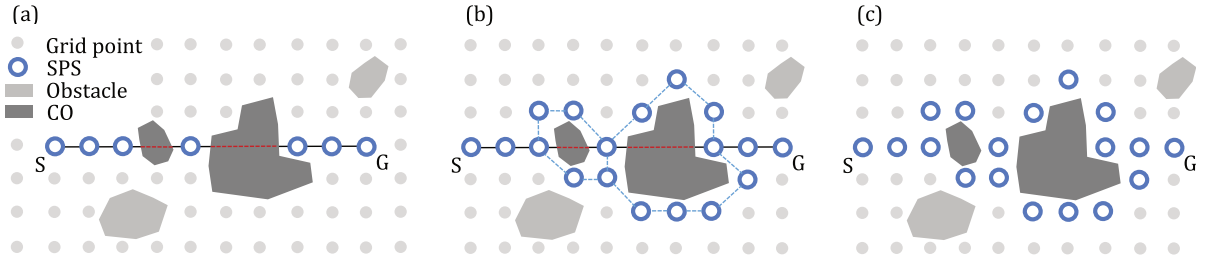

Fig. 3. Example of CO and SPS.

ment are defined as SPS (c). After SPS was identified, a graph was constructed based on the cells in SPS, and the path was found using a shortest path planning algorithm such as A* or Dijkstra algorithm.

The motivation of this study lies in the limitations of the SPS method with respect to graph construction. Given a subset of unblocked cells (SPS), a graph needs to be generated by considering the connectivity between cells such that a graph search based method can be applied for path planning. As shown in Fig. 4, the most general way to do this in grid graphs is to simply connect edges from a cell to its eight neighbors in the left, right, up, down, and diagonal directions. However, this might not provide sufficient connectivity in the SPS method, since several cells that were not chosen as SPS were excluded from the graph. Meanwhile, a full graph can be also considered; however, it has to go through the expensive computational time, since all possible edges need to be tested in terms of line-of-sight as well as the exponential increase in search space. Accordingly, the previous work adopted a density-bound based graph that connects edges from a cell to its adjacent cells within a bound. For example, the center plot in Fig. 4 shows a graph in which the bound value is set to 3. This approach was beneficial for obtaining a good solution in an efficient manner, as it lies between two extreme approaches. However, it would still take a

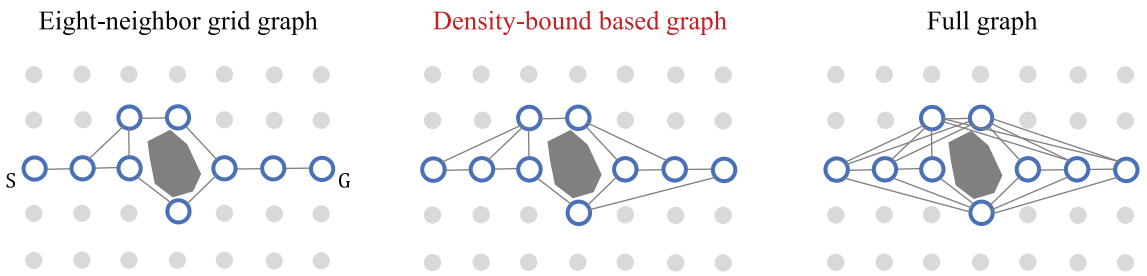

Fig. 4. Comparison of the graph constructions based on cells in SPS.

longer time than a simple eight-neighbor grid graph, as line-of-sight checks for graph construction would have to be implemented. There would be a longer path length than a full graph because its connectivity cannot exceed the bound value. In addition, we need to decide which bound should be used for a map, which has a critical impact on solution quality and time.

For simplicity, given the blocked cells that intersect with the straight line as an input parameter, we defined the SPS process as a function *FindSPS*() that returns the set of unblocked cells that surrounds the obstacles. Since we assume that an obstacle is not provided as a polygon in advance (rather, its cluster is identified by the SPS process), it also returns the cluster information of CO.

3.3. Multi-SPS with Theta* algorithm on grid graphs

The proposed approach is based on previous research on the SPS method that considers a subset of unblocked cells on a straight line between S and G and then uses them for planning a path. However, when applying Theta* to the subset of cells (the result of SPS), the edges made from Theta* would not be able to provide proper connectivity, since not all eight neighbor cells are considered for expansion. For example, in Fig. 5, which shows each edge after parent update in grey, when c_i expands to its neighbor c_j, the edge from parent of c_i, which is S, to c_j collide with blocked cells (see the red dashed line), making c_i the parent of c_j. As a result, the subsequent cells are connected with c_i, rather than making direct short-cuts to S (see the figure on the right). Based on this result of SPS with Theta*, multi-SPS, which implements the SPS process multiple times by considering additional lines, the straight line between S and G was developed.

The multi-SPS method can provide more connectivity in the searching phase of Theta* because it considers more critical obstacles and their surrounding cells based on multiple lines and consequently improves the results of Theta*. The unblocked cells resulting from multi-SPS would have fewer neighbors, since all cells that are not on the lines or surrounding the critical obstacles would not be considered. This could result in generating a long path; yet, connecting the cells to further parents in Theta* algorithm could shorten the path.

As multi-SPS is based on identifying the existence of critical obstacles in the way, it defines each way as a line segment. Note that a line segment l from $s_l = (s_{l,x}, s_{l,y})$ to $g_l = (g_{l,x}, g_{l,y})$ is converted to a series of cells, denoted as N_l, *LineInGridCells*() that is based on Bresenham's line-drawing algorithm with a the modification entailed adding missing cells [26,31]. Each line segment l is placed in the *line* list as a queue and popped off one at a time at each round as a referent line denoted as l^r. If there are blocked cells colliding with the reference line l^r, which are identified by the intersection of N_{l^r} (a series of cells for l^r) and N_o (a set of all blocked cells) (hence, $N_{l^r} \cap N_o \neq \varnothing$), then a set of surrounding unblocked cells and those covered obstacles, denoted as SPS_{l^r} and, CO_{l^r} respectively, are obtained from *FindSPS*(). Consequently, lines are added in the *line* list in which the line will pop-off the list as a reference line at each round of the SPS process. However, if no obstacles are colliding with l^r, indicating $N_{l^r} \cap N_o = \varnothing$, then no SPS process regarding l^r is required, and no new lines are added in the *line* list.

With respect to the line addition, the lines between certain cells in SPS_{l^r} and two cells, s_{l^r} and g_{l^r}, which define the current reference line, l^r are added in the *line* list. In the initial round, $s_{l^r} = (s_{l^r,x}, s_{l^r,y})$ and $g_{l^r} = (g_{l^r,x}, g_{l^r,y})$ are set to be S and G. To decide which cells in SPS_{l^r} should be used for the line addition, we focused on the most distant cells from the line because these cells would work as barriers and might have to be considered in generating a path to G. Therefore, given the general line representation for l^r to $Ax + By + C = 0$, where $A = g_{l^r,y} - s_{l^r,y}$, $B = s_{l^r,x} - g_{l^r,x}$, $C = s_{l^r,y} \cdot g_{l^r,x} - s_{l^r,x} \cdot g_{l^r,y}$, we divided the cells in SPS_{l^r} into the upper and lower cells from l^r, denoted as $SPS_{l^r,U}$ and $SPS_{l^r,L}$, respectively. For the upper side, the most distant cell c_U can be obtained by calculating the distance between each cell $c_i = (c_{i,x}, c_{i,y})$ in $SPS_{l^r,U}$ and l^r, and returning one that has the maximum distance (Eq. (1)). The most distant cell c_L on the lower side can also be obtained in a similar manner (Lines 27–43 in Algorithm 1). Then, in the initial round, four lines, $\{\overline{s_{l^r}c_U}, \overline{c_U g_{l^r}}, \overline{s_{l^r}c_L}, \overline{c_L g_{l^r}}\}$, as depicted by the dashed red lines in Fig. 6 (left figure), are placed in the *line* list (Line 17).

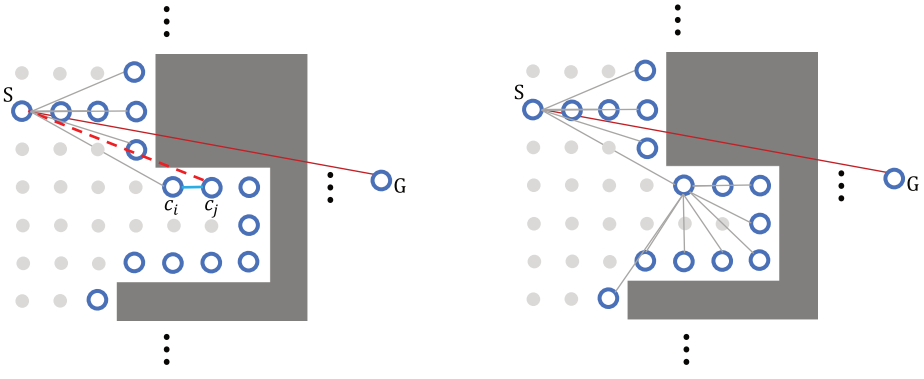

Fig. 5. Example of the parent-update process of Theta* based on SPS.

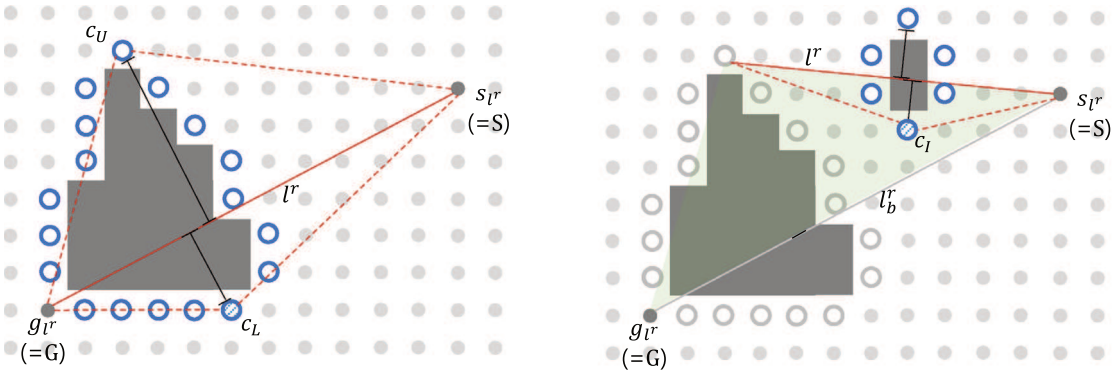

Fig. 6. Line addition of multi-SPS.

$$c_U := \underset{c_i}{\operatorname{argmax}} \frac{|Ac_{i,x} + Bc_{i,y} + C|}{\sqrt{A^2 + B^2}}, \ \forall c_i \in SPS_{l^r,U} \tag{1}$$

Following the initial round, a cell c_I is selected between c_U and c_L in the inner side between the current line l^r and its prior line. This method could restrict the number of SPS processes and prevent distant lines from being considered, which might not be effective, as prior lines generally play an important role in constructing the shortest path because the path length would have to be increased over rounds. To identify this inner cell, which line each line is generated from should be recorded as a back-line, denoted as l_b^r. When new lines are added to the *line* list, the current line l^r is set as the back-line l_b^r for those new lines and placed in the *backline* list as the number of new lines. Then, we construct a triangle polygon (green region in Fig. 6 (right figure)) based on l^r and l_b^r, and we select a cell c_I in the triangle using a function *intriangle*() where it returns which cell from a given set of cells (c_U and c_L) is in a given triangle (Line 20). Consequently, two lines $\{\overline{s_{l^r}c_I}, \overline{c_I g_{l^r}}\}$ between this inner cell and s_{l^r} and g_{l^r} are placed in the *line* list (Line 21).

At each round, the covered obstacles of the reference line, CO_{l^r}, would be removed from the set of remaining uncovered blocked cells to see if there exist any new obstacles that get in the way. Hence, $N_{o'}$ is defined to represent the set of uncovered blocked cells, which is set to be N_o in the beginning. It is updated by removing CO_{l^r} at each round as shown in Eq. (2).

$$N_{o'} := N_{o'} \{CO_{l^r} \tag{2}$$

Theorem 3.3.1. *The multi-SPS algorithm terminates after a finite number of iterations.*

Proof. Initially, assuming obstacles are present on the straight line between S and G, four lines are placed in the *line* list, generating two triangles: $\Delta(c_U, \mathrm{S}, \mathrm{G})$ and $\Delta(c_L, \mathrm{S}, \mathrm{G})$. For generality, we focus on the upper triangle. The multi-SPS algorithm runs with respect to i) the existence of blocked cells and ii) each line in the *line* list. For i), let us assume that there are n clusters of blocked cells in a triangle $\Delta(c_U, \mathrm{S}, \mathrm{G})$. These are removed through the intersection with each line, indicating that the number of clusters of blocked cells decreases over rounds. For ii), we only need to prove that lines generated over rounds lie inside the triangle $\Delta(c_U, \mathrm{S}, \mathrm{G})$. Note that the lines are the segments of a triangle. A triangle is constructed by one segment of its preceding triangle and c_I, which is an inner cell of the preceding triangle by definition, resulting in the succeeding triangle fully belonging to its preceding triangle. Then, all lines generated over the rounds lie inside the initial triangle $\Delta(c_U, \mathrm{S}, \mathrm{G})$. Therefore, the algorithm runs at most n times from the perspective of a triangle $\Delta(c_U, \mathrm{S}, \mathrm{G})$.

We denote the subset of cells as N_{sub}, which is set to be an empty set in the beginning. It is merged with the chosen cells with respect to each reference line l^r. If l^r is a feasible line (no intersection between N_{l^r} and $N_{o'}$), then N_{l^r} is merged to N_{sub}, as shown in Eq. (3) (Lines 9–10). However, for the opposite case, SPS_{l^r} and the unblocked cells obtained by excluding CO_{l^r} from N_{l^r} are merged to N_{sub} (Lines 11–13).

$$N_{sub} := \begin{cases} N_{sub} \cup N_{l^r} & \text{if} N_{l^r} \cap N_{o'} = \varnothing \\ N_{sub} \cup \{(N_{l^r} \{CO_{l^r}) \cup SPS_{l^r} & \text{otherwise} \end{cases} \tag{3}$$

Algorithm 1 Multi-SPS

$N_{sub} := \varnothing$;
$N_{o'} := N_o$;
$line := \overline{SG}$;
$backline := \overline{SG}$;
while $line \neq \varnothing$
    $l^r := line.Pop()$;
    $l_b^r := backline.Pop()$;
    $N_{l^r} := LineInGridCells(l^r)$;
    if $(N_{l^r} \cap N_{o'}) = \varnothing$
        $N_{sub} := N_{sub} \cup N_{l^r}$;
    else
        $[SPS_{l^r}, CO_{l^r}] := FindSPS(N_{l^r} \cap N_{o'})$;
        $N_{sub} := N_{sub} \cup \{(N_{l^r}\{CO_{l^r}) \cup SPS_{l^r}\}$;
        $N_{o'} := N_{o'}\{CO_{l^r}$;
        $[c_U, c_L] :=$ GetMaxCells(l^r, SPS_{l^r});
        if $l^r = \overline{SG}$
            $line := line \cup (\overline{s_{l^r} c_U}, \overline{c_U g_{l^r}}, \overline{s_{l^r} c_L}, \overline{c_L g_{l^r}})$;
            $backline := backline \cup \{arraycopy(l^r, 4)\}$;
        else
            $c_I := intriangle([c_U, c_L], [l^r, l_b^r])$;
            $line := line \cup (\overline{s_{l^r} c_I}, \overline{c_I g_{l^r}})$;
            $backline := backline \cup \{arraycopy(l^r, 2)\}$;
        end if
    end if
end while
return N_{sub}
function GetMaxCells(l^r, SPS_{l^r})
$c_U := c_L := Null$;
$\max dist_U := \max dist_L := 0$;
for all $c_i \in SPS_{l^r}$
    if $sign(Ac_{i,x} + Bc_{i,y} + C) > 0$
        if $\frac{|Ac_{i,x}+Bc_{i,y}+C|}{\sqrt{A^2+B^2}} > \max dist_U$
            $c_U := c_i$;
            $\max dist_U := \frac{|Ac_{i,x}+Bc_{i,y}+C|}{\sqrt{A^2+B^2}}$;
        end if
    else
        if $\frac{|Ac_{i,x}+Bc_{i,y}+C|}{\sqrt{A^2+B^2}} > \max dist_L$
            $c_L := c_i$;
            $\max dist_L := \frac{|Ac_{i,x}+Bc_{i,y}+C|}{\sqrt{A^2+B^2}}$;
        end if
    end if
end for
return c_U, c_L

In Theta*, a modification is made to obtain the neighbors of a popped off cell s from the *open* list, denoted as $nghbr_{vis}(s)$. It would expand all adjacent neighbors; however, any adjacent neighbors that are not in N_{sub} would be excluded for defining neighbors, and consequently, for expansion (see Eq. (4)). Fig. 7 shows an example of N_{sub} from multi-SPS where each considered line is depicted as a red dashed line, and the resulting path is shown as a blue solid line obtained by Theta* that is only used N_{sub} as an input set of cells. Each expansion is depicted as a grey edge, while generating a short-cut between distant

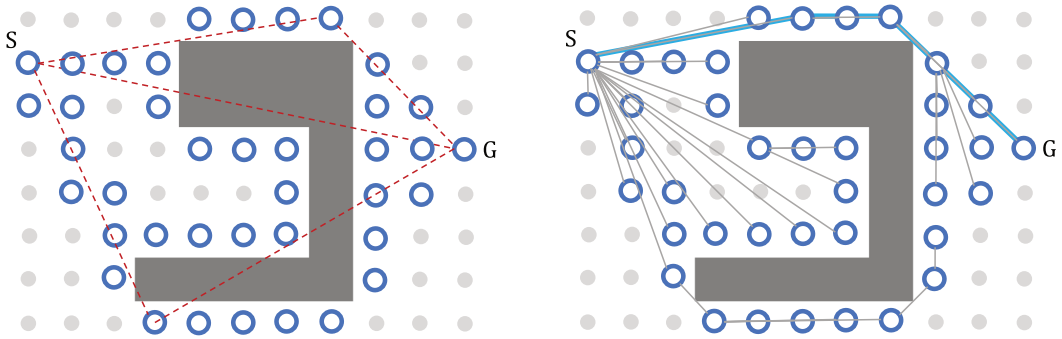

Fig. 7. Example of N_{sub} from multi-SPS (left) and a resulted path by Theta* (right).

cells through the parent update process. Because Theta* preserves parent information for each expansion, the shortest path can be simply obtained by tracking back from G to S.

$$nghbr_{vis}(s) := nghbr_{vis}(s) \cap N_{sub} \tag{4}$$

Theorem 3.3.2. *The multi-SPS with Theta* algorithm finds a feasible path if one exists.*

Proof. *Theta* is a complete algorithm, as it is basically identical to A* algorithm but includes short-cuts only if they succeed line-of-sight checks. Therefore, we only need to prove that the output of the multi-SPS, N_{sub}, guarantees feasible connectivity from S to G in an eight-neighbor grid graph. With respect to feasibility of N_{sub}, for a line l, a set of cells N_l is entirely merged with N_{sub} if l is feasible ($N_l \cap N_o = \varnothing$); further, this set is partially merged with N_{sub} as it excludes the clusters of blocked cells CO_l if l intersects with any blocked cells. SPS_l is also merged with N_{sub}, which is a set of unblocked cells surrounding obstacles by definition. Therefore, the cells in N_{sub} are unblocked. With respect to connectivity between cells in N_{sub}, note that each cell in N_{sub} comes from either a line (N_l) or SPS process (SPS_l). As the SPS process is conducted from an intersection with a line, at least one incoming and outgoing cell exists (in the direction of S to G), wherein these cells are both defined as N_l and SPS_l. Hence, each cell in N_{sub} other than S and G has at least two neighbors in an eight-neighbor grid graph. From this, we obtain feasible connectivity from S to G.*

4. Experiments

In this section, experiments are conducted to test the proposed approach regarding its effectiveness in reducing the search space and comparing the results to the existing methods in terms of the features of maps, solution quality, and efficiency. We also conduct performance tests to determine how well the existing and proposed methods perform with an increase in the map size.

4.1. Experimental design

We used 2D grid maps from Nathan Sturtevant's repository (http://movingai.com/benchmarks/). The following three types of maps were considered: randomly blocked maps of varying blockage rates from 10 to 40%, real-world street maps (Berlin, Boston, New York, and Paris), and room maps of varying room sizes from 8 × 8 to 64 × 64. All maps were 512 × 512 in grid size, but we also considered the map size from 256 × 256 to 1024 × 1024 for further test. The start and goal cells were selected to run the scenarios of a sufficiently long path. The experiments were run on an Intel i5-6300U (2.40 GHz) CPU and 4 GB of RAM, and the algorithms were written in MATLAB.

We simulated the proposed method in different maps to see how it performs and discuss the results. With respect to the previous work named COSPS [14] which conducts finding the set of surrounding cells only once based on a line segment between start and goal cells, we compared the results in terms of differences in number of lines, cells, as well as efficiency and solution quality. In particular, for COSPS, we constructed a graph by a density bound value as the previous work did where the density value was set to 0.1% of the number of unblocked grid cells (N_f). For comparative experiments, we first compared the multi-SPS + Theta* with A*, which is the most basic search method, and then with Theta* and variants of Theta*, such as Lazy Theta* and Theta* with re-expansion (denoted as Theta* R). We allowed these algorithms to consider all grid cells, so that we could discuss how much CPU runtime decreases and path length increases by reducing the number of searched cells. With respect to two variants of Theta* above, Lazy Theta* is a well-known any-angle path planning algorithm that delays the line-of-sight checks in Theta* to increase the computational efficiency [27], further, Theta* R allows a

vertex to be re-expanded and re-inserted in the open list even after it is placed on closed list by not maintaining a closed list to find a shorter path at an increase in runtime [15]. In addition, we combined the multi-SPS with those variants of Theta* and additionally with Theta* with key vertices (denoted as Theta* KV) to see how well the variants of Theta* would work with the multi-SPS. Theta* KV uses a key vertex as the parent of an expanding vertex for a short-cut and increases the number of visible neighbors of that key vertex, so that it can update the vertices later from expansion [15]. For the performance test, we considered the small and large maps that are available on the street map, where the number of grid cells would be 65,536, 262,144, and 1,048,576 for 256 × 256, 512 × 512, and 1024 × 1024 size, respectively.

4.2. Experimental results

The simulation results of the multi-SPS with Theta* and the previous work COSPS are summarized in Fig. 8 and Table 2. All blocked cells, including green cells, represent obstacles. The unblocked cells surrounding CO and on the lines are marked with blue dots.

As shown in Fig. 8 and Table 2, the number of generated lines tended to increase, as the map included more blocked cells, which consequently resulted in an increase in the number of CO, and the ratio of N_{sub} to N_f. For example, as the blockage rate increased from 10% to 40% for the random map, or the room size decreased from 64 × 64 to 8 × 8 for the room map, we observed that the ratio of N_{sub} to N_f grew based on the number of lines and CO. With regard to the different features between maps, the street map showed a relatively small number of lines, CO, and the ratio of N_{sub} to N_f, which were on average 13, 13.25, and 2.92, as compared to other maps (50.5, 427.25, and 3.11 for random maps and 26, 97.5, and 5.45 for the room map). This was because the clusters of obstacles were rather simple shapes and large, requiring less frequent iterations of SPS, while blocked cells were disjoint and scattered over the space in random or room maps, which resulted in a more frequent implementation of SPS.

Compared to the COSPS, the multi-SPS with Theta* generated the number of CO and N_{sub} almost 3 times larger than COSPS which is obvious since COSPS includes only one line segment between start and goal cells. On the other hand, the runtimes of the multi-SPS with Theta* and COSPS were 3.61 and 60.39 (sec) on average respectively. This shows the advantage of using the eight-neighbor grid graphs by implying that a density-bound graph construction had a significant impact on runtime since each cell had to go through the feasibility tests with adjacent cells within the bound to define edges that do not collide with any blocked cells. In addition, we observed that runtime tends to increase over an increase of N_{sub} and N_f, which is because the feasibility tests should be frequently conducted in those cases. With respect to path length, COSPS obtained 4.9% longer paths than the proposed method given the density bound. Knowing that the previous work used relatively small sized maps where the map size was 2000 for each axis and the width size was set to 100–200 while the map size in this paper are 512 for each axis and the width size is set to 1, the comparative results indicate that COSPS would work less effective in large maps resulted from a graph construction. Consequently, the multi-SPS with Theta* can resolve the limitation of the previous work by using an eight-neighbor grid graph as it is and adopting Theta* to obtain distant connectivity.

The results of comparative experiments in Fig. 9 and Table 3 show that when the number of unblocked cells was on average 203,510 across all maps, A*, Theta*, Lazy Theta*, and Theta* R searched 41%, 34%, 36%, and 35% of cells, respectively. However, the proposed method searched for only 3% of cells with Theta*, Lazy Theta*, Theta* R, and 6% with Theta* KV to generate the path. Note that Theta* KV is known for a significant increase in runtime due to the overhead of keeping the key vertices and large number of neighbors; it increased the number of searched cells twice as many as Theta*, Lazy Theta*, and Theta* R with the proposed method. The results of the number of expanded cells were related to the CPU runtime result, as overall runtime tended to increase with search space. However, we observed that line-of-sight checking worked expensively, since Theta* was approximately 5 times slower than A* despite a lower number of searched cells in Theta* than in A* (69,447 and 84,410 on average, respectively).

The longest and shortest runtimes were obtained by Theta* R and multi-SPS + Lazy Theta* (61.71 and 3.46 sec, respectively), where A*, Theta* (with its variants), and multi-SPS + Theta* (with its variants) showed 6.69, 36.19, and 4.70 sec on average. These results suggest that using multi-SPS with Theta* made it possible to effectively reduce the runtime from using all grids. With respect to differences in Theta* algorithms, allowing re-expansion such as Theta* R and Theta* KV turned out to be critical in runtime, as it increased almost twice as much as Theta* in both cases of all grids and multi-SPS. Using Theta* instead of Lazy Theta* slightly increased the runtime (by approximately 0.16 sec) when multi-SPS was applied, while a difference of 18.06 sec was present between them when all grid cells were considered. Meanwhile, the multi-SPS + Theta* KV showed a similar runtime with A* as 6.73 and 6.69, respectively, while other variants were 1.3–1.9 times faster than A*. With respect to the maps, the proposed method was particularly fast for street maps or maps with a low occupancy rate. The case where the proposed method provided a longer time, such as the random map with 40% rate, involved small and disjoint obstacle cells scattered over the space, and a map with a high occupancy rate of obstacles. When obstacles were disjoint and frequently scattered, the multi-SPS procedure had to be implemented more frequently, as many disjoint obstacles were encountered in the lines. By contrast, compared with other maps, street maps with obstacles as large polygons showed a higher efficiency for the proposed method. In addition, the proposed method yielded a higher efficiency in sparse maps. For example, when the occupancy rate decreased from 40 to 10% in the random map, or from 8 × 8 to 64 × 64 size in the room map, implying that the maps became sparse, the proposed method (unlike A* or Theta*) showed a trend of a decrease in runtime.

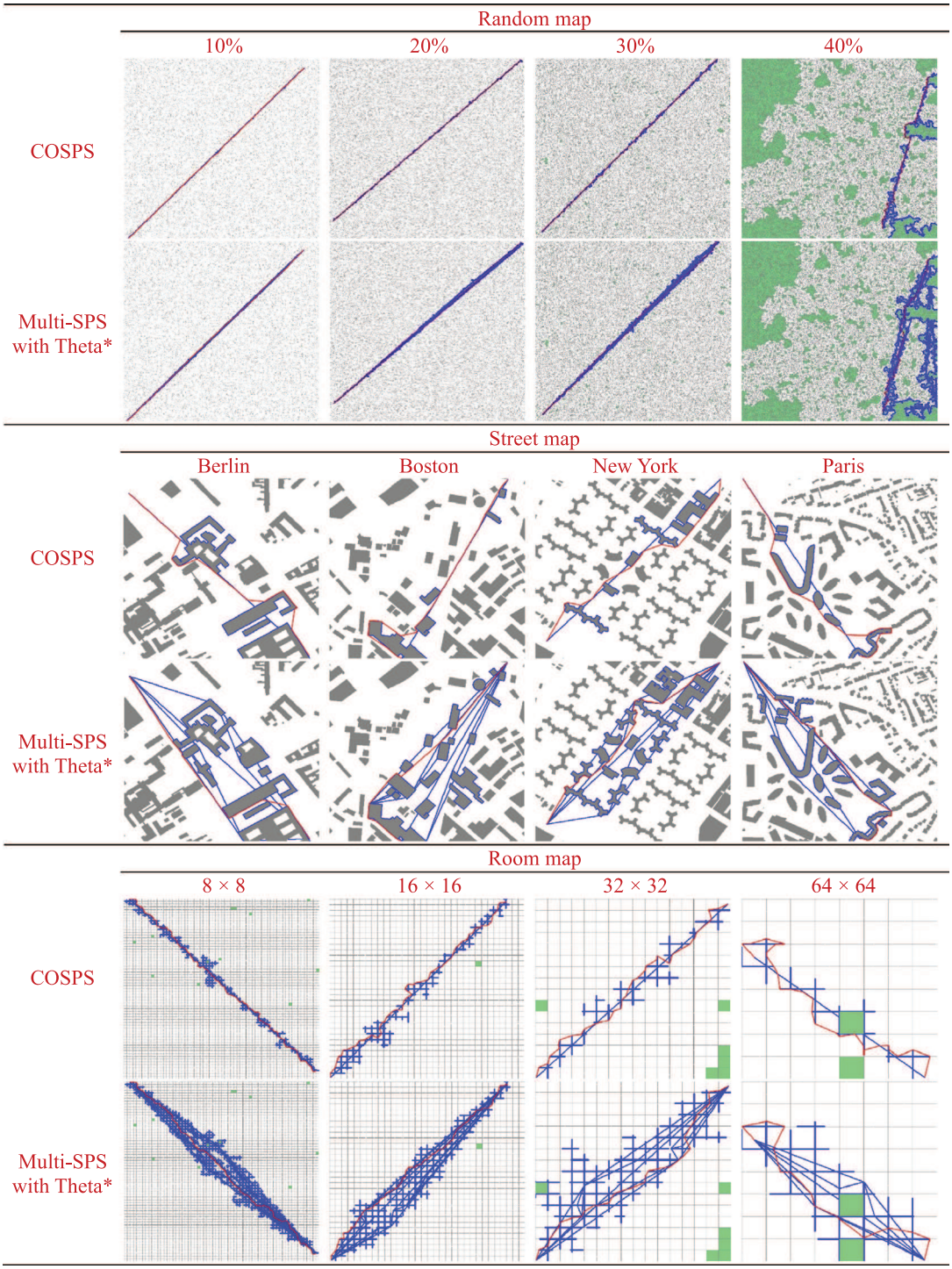

Fig. 8. Result plots of the proposed method compared to COSPS.

Table 2
Result summary of the proposed method compared to COSPS.

		No. of lines		No. of CO		N_{sub}		N_{sub} to N_f ratio (%)		Runtime (sec)		Path length	
		COSPS	Multi-SPS with Theta*	COSPS	Multi-SPS with Theta*	COSPS	Multi-SPS with Theta*	COSPS	Multi-SPS with Theta*	COSPS	Multi-SPS with Theta*	COSPS	Multi-SPS with Theta*
Random map	10%	1	26	81	203	1046	2549	0.44	1.08	29.33	1.78	671.57	671.44
	20%	1	40	148	482	1252	3631	0.60	1.73	31.61	3.54	685.96	683.66
	30%	1	58	172	565	1491	4201	0.83	2.33	35.07	4.76	714.50	699.25
	40%	1	78	84	459	2653	7678	2.53	7.31	52.22	7.54	570.37	536.68
Street map	Berlin	1	10	4	7	2342	5136	1.19	2.61	57.89	1.56	820.95	764.04
	Boston	1	16	5	17	1403	5827	0.71	2.96	36.38	2.41	796.36	717.59
	New York	1	12	5	14	2106	6312	1.07	3.21	51.95	2.32	713.17	759.00
	Paris	1	14	6	15	1950	5762	0.99	2.92	47.91	2.36	822.13	735.30
Room map	8×8	1	42	61	235	4323	14,351	2.09	6.94	95.13	5.62	822.47	791.53
	16×16	1	30	34	99	4644	13,436	2.00	5.79	107.84	4.56	818.69	770.62
	32×32	1	18	15	44	4082	13,434	1.70	5.58	99.07	4.12	826.88	808.75
	64×64	1	14	5	12	3113	8640	1.26	3.50	80.28	2.84	959.63	853.50

Regarding the path length, Theta* R and multi-SPS + Lazy Theta* generated the shortest and longest paths (704.30 and 735.03, respectively) across all maps, where A*, Theta* (with its variants), and multi-SPS + Theta* (with its variants) generated 734.24, 705.68, and 729.07 in path length, respectively. These results suggest that the proposed method has a 2.3% solution gap from using all grid cells. With respect to differences in Theta* algorithms, Lazy Theta* generated slightly longer paths than Theta* or Theta* R, as it delayed the line-of-sight checks for efficiency; further, Theta* and Theta* R obtained similar path lengths (with minor improvement). These results remained consistent after combining multi-SPS, as the multi-SPS + Theta*, Lazy Theta*, and Theta* R generated 732.61, 735.03, and 729.90 in path length, respectively. In particular, the multi-SPS generated the shortest path with Theta* KV, not with other Theta* algorithms. This implies that increasing the number of neighbors for key vertices in the case of short-cuts can play an important role under the limited set of grid cells. Compared to A*, the proposed method combined with Theta* algorithms allowed us to obtain similar path lengths and even find shorter ones with Theta*, Theta*R, or Theta* KV. Regarding the shape of paths, A* showed frequent heading changes in free space, as it restricts the connectivity to eight neighbors at fixed angles; however, Theta*, Lazy Theta*, Theta* R, and Theta* KV with or without the multi-SPS contained relatively longer line segments by any-angle connectivity.

Fig. 10 shows the average performance results as the map size increased from 256×256 to 1024×1024 for the street maps. As the map became larger (i.e., the number of cells increased from 65,536 to 1,048,576), the number of expanded cells significantly increased for A* (from 21,185 to 392,864) and Theta*, Lazy Theta*, and Theta* R (from 16,748 to 304,044 on average) compared with those with multi-SPS (from 2,266 to 12,979). More importantly, A* and Theta* algorithms showed relatively consistent ratios of searched cells as A* searched 44%, 53%, and 49% of grid cells, and Theta* algorithms searched 34%, 44%, and 38% of grid cells in 256×256, 512×512, and 1024×1024 maps, respectively. In contrast, the multi-SPS + Theta*, Lazy Theta*, Theta* R, or Theta* KV showed a decreasing pattern of 4.7%, 2.8%, and 1.6%, respectively. This result indicates that the proposed method exhibits superior performance for an increased map size and consequently explains the difference in the slope of runtime in the center plot. At runtime, the multi-SPS approach showed an insignificant increase from 1.35 to 7.56 sec, whereas A* and Theta* algorithms had steeper slopes from 1.57 to 40.46 sec and from 8.43 to 271.77 sec, respectively. Regarding the path length, differences were relatively insignificant compared with the differences in efficiency. We also noticed that the multi-SPS + Theta* KV appeared to be similar to Theta* algorithms in terms of path length. This result indicates that the proposed method, compared to the existing methods, tends to be more effective as the map size grows with respect to the increasing trend in path length and runtime.

In summary, the proposed method successfully reduced the set of grid cells and generated feasible paths from the given start to the goal. It obtained a minor increase in path length and a significant reduction in runtime compared with Theta*, Lazy Theta*, and Theta* R, which considers all existing grid cells. In particular, Theta* KV turned out to be beneficial in terms of computational time and solution quality when it was merged with the proposed method. Even with A*, the results showed a shorter runtime required to generate similar or even shorter path lengths. The efficiency of the proposed method increased when a map got sparse rather than had a high occupancy rate of obstacles, and where obstacles were positioned in relatively large shapes of polygons rather than being separated and scattered. This result indicates that the proposed method can be more efficient in real-world maps, such as street maps where regions for unblocked cells are relatively spacious. The results of the performance test demonstrated that the proposed method can be more advantageous for large maps, as it does not proportionally increase the computational time along with an increase in the map size. In addition, even though the proposed method removed several cells in search to enhance the efficiency of path planning, it did not necessarily lower the solution quality, showing a path length similar to that afforded by other methods such as A* and Theta* algorithms.

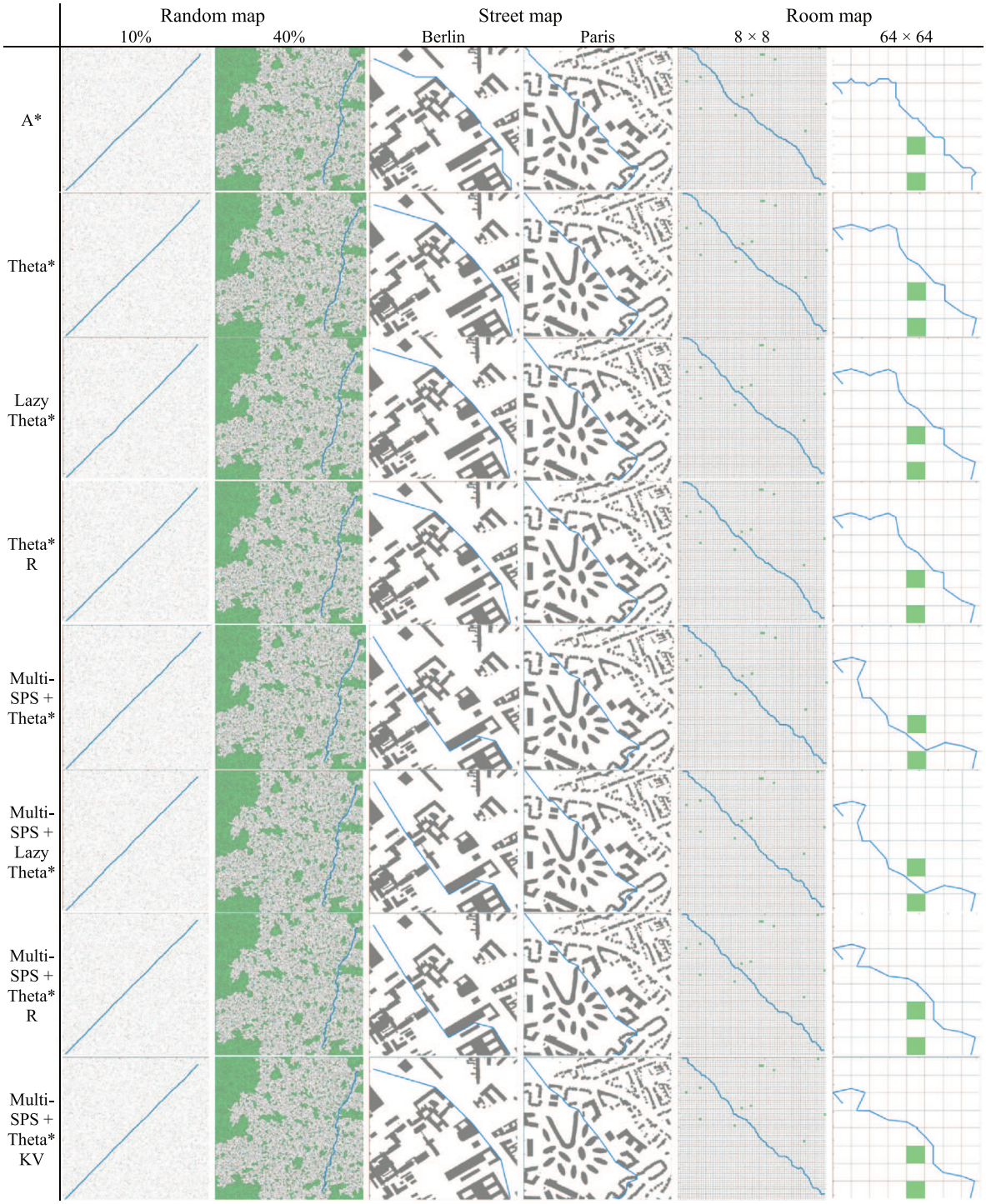

Fig. 9. Path results on 512 × 512 size maps.

Table 3
Comparison between algorithms on 512 × 512 size maps.

		Random map				Street map				Room map			
		10%	20%	30%	40%	Berlin	Boston	New York	Paris	8 × 8	16 × 16	32 × 32	64 × 64
Number of expanded cells	N_f	235,900	209,281	180,136	104,950	196,667	196,678	196,500	196,672	206,642	231,854	240,671	246,178
	A*	23,613	32,034	48,278	22,302	54,660	136,871	94,629	132,416	101,095	88,669	117,274	161,079
	Theta*	10,108	20,584	35,303	19,365	29,579	129,595	62,928	124,104	86,968	68,037	92,315	154,483
	Lazy Theta*	22,360	33,229	46,756	19,569	29,550	130,009	62,612	124,118	95,507	71,743	93,229	154,552
	Theta* R	20,679	22,951	38,781	20,071	29,901	133,583	57,600	126,186	89,431	69,062	92,673	154,804
	Multi-SPS + Theta*	2,450	3,588	3,997	4,324	2,101	5,748	5,249	5,588	14,144	12,972	11,710	8,339
	Multi-SPS + Lazy Theta*	2,531	3,608	4,058	4,324	2,101	5,748	5,249	5,580	14,158	13,010	11,724	8,339
	Multi-SPS + Theta* R	3,796	4,467	4,376	4,437	2,459	6,072	5,399	6,025	16,088	15,902	16,610	9,685
	Multi-SPS + Theta* KV	6,447	7,053	6,644	6,778	3,466	10,893	7,708	10,477	26,958	26,224	36,827	18,515
Runtime (sec)	A*	2.02	2.44	3.58	1.52	4.50	10.19	7.55	10.29	7.82	6.95	9.88	13.50
	Theta*	4.47	6.42	9.42	4.61	18.32	81.78	35.36	68.25	26.20	23.51	37.60	73.61
	Lazy Theta*	4.02	5.63	7.50	3.80	6.89	32.39	15.14	26.21	14.99	11.92	15.85	28.48
	Theta* R	13.35	13.46	17.83	10.28	33.99	168.41	61.17	133.25	45.99	43.54	65.39	133.87
	Multi-SPS + Theta*	1.78	3.54	4.76	7.54	1.56	2.41	2.32	2.36	5.62	4.56	4.12	2.84
	Multi-SPS + Lazy Theta*	1.51	3.19	5.63	7.67	1.55	2.26	2.30	2.08	5.25	3.83	3.68	2.63
	Multi-SPS + Theta* R	2.87	4.51	5.80	8.19	2.16	3.61	3.20	3.62	7.84	7.01	6.97	4.44
	Multi-SPS + Theta* KV	3.65	5.35	6.26	8.63	2.22	5.61	4.09	5.33	10.57	9.72	12.30	7.04
Path length	A*	679.46	694.49	708.89	518.07	745.2	728.68	725.79	752.27	799.44	771.51	836.64	850.4
	Theta*	670.52	681.39	690.95	501.36	709.94	698.36	688.79	716.76	767.36	737.58	789.12	805.07
	Lazy Theta*	673.78	688.40	700.35	503.04	709.95	698.93	688.64	717.03	777.57	742.18	790.61	805.48
	Theta* R	670.33	680.54	690.05	500.87	709.94	698.34	686.58	716.36	767.27	737.49	788.80	805.08
	Multi-SPS + Theta*	671.44	683.66	699.25	536.68	764.04	717.59	759.00	735.30	791.53	770.62	808.75	853.50
	Multi-SPS + Lazy Theta*	676.70	690.15	706.79	538.39	764.04	717.59	759.02	735.00	796.11	773.43	809.46	853.68
	Multi-SPS + Theta* R	671.26	682.37	698.32	536.34	763.94	717.59	755.14	734.89	787.74	767.90	803.00	840.33
	Multi-SPS + Theta* KV	671.26	682.35	698.13	535.25	709.93	713.75	690.46	727.51	787.74	768.18	801.14	839.25

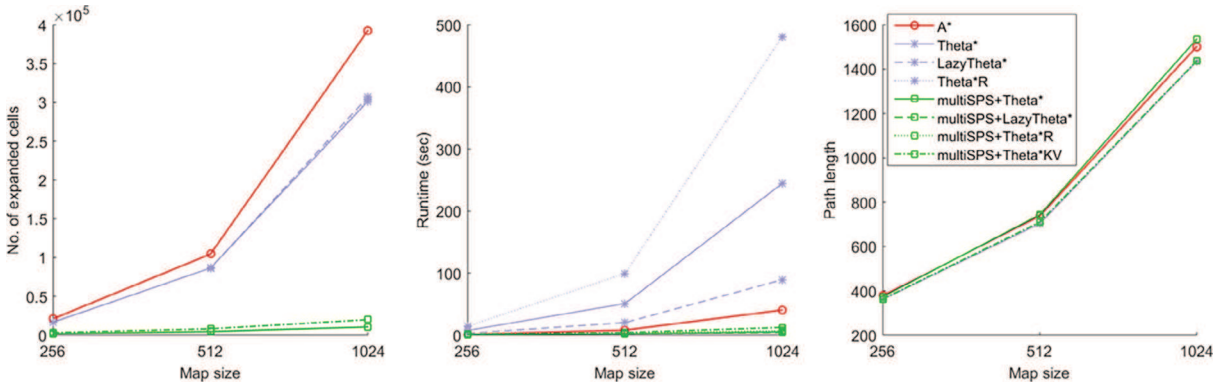

Fig. 10. Performance test by an increase of map size.

5. Conclusions

The algorithm proposed in the present paper improved the efficiency of path planning on 2D grid maps by considering unblocked cells around certain obstacles. Furthermore, the limitation of a long path length owing to the consideration of fewer cells rather than all existing grid cells was resolved by the parent updating mechanism in Theta* algorithm. The results of numerical experiments demonstrated that, compared with existing well-known path planning methods such as A* and several variants of Theta*, the multi-SPS with Theta* algorithm could significantly reduce the runtime without noticeable degradation in path lengths. In particular, Theta* that additionally considered vertices for expansion worked well with the multi-SPS, as its combination could obtain a path length similar to that of considering all grid cells.

The proposed method can be effectively applied in cases where a map is sparse with a low occupancy rate, and where obstacles are relatively large polygons, rather than being small and frequently scattered over the space, as it implements the procedure based on the existence of disjoint obstacles on the lines. More importantly, because the proposed method does not need to have the entire polygon information for obstacles in advance, it can be useful for urban maps. Furthermore, it can be useful for large or high-resolution maps such as game maps, as the computational time does not significantly increase over the map size, unlike the existing traditional path planning methods. Finally, the multi-SPS can be combined with other types of any-angle path planning algorithms as well as Theta* algorithm because it is basically an approach that determines a subset of cells on grid maps.

In future research, the proposed method can be implemented in 3D environments, wherein the line for dividing SPS into two sides and the triangle for identifying the farthest cell should be a plane and polyhedron, respectively. The approach can also be enhanced by developing additional post-smoothing methods based on the result of the straight line between start and goal. Dynamic modification and re-planning mechanism in partially known environments would be another research topic for real-time application. In addition, kinematic constraints such as angle smoothness or safe distance from obstacles can be considered for practical use of the proposed method in robotic environments. These restrictions might involve a multi-objective function in Theta* search or introduce safeness-considered proximity in determining a set of surrounding cells.

CRediT authorship contribution statement

Jihee Han: Conceptualization, Methodology, Software, Validation, Visualization, Writing – original draft. **Sven Koenig:** Conceptualization, Methodology, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. NRF-2019R1C1C1002798). The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, 1817189, 1837779, and 1935712. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

[1] S.M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, 2006.
[2] T. Lozano-Pérez, M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, Commun. ACM 22 (10) (1979) 560–570.
[3] J. Lee, D.-W. Kim, An effective initialization method for genetic algorithm-based robot path planning using a directed acyclic graph, Inf. Sci. 332 (2016) 1–18.
[4] Y. Björnsson, et al., Comparison of different grid abstractions for pathfinding on maps, in: IJCAI. 2003.
[5] B.C. Shah, S.K. Gupta, Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles, Twenty-Sixth International Conference on Automated Planning and Scheduling, 2016.
[6] P. Raja, S. Pugazhenthi, Optimal path planning of mobile robots: a review, Int. J. Phys. Sci. 7 (9) (2012) 1314–1320.
[7] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[8] A. Stentz, The focussed d^* algorithm for real-time replanning, in: IJCAI, 1995.
[9] S. Koenig, M. Likhachev, D. Furcy, Lifelong planning A*, Artif. Intell. 155 (1-2) (2004) 93–146.
[10] R. Song, Y. Liu, R. Bucknall, Smoothed A* algorithm for practical unmanned surface vehicle path planning, Appl. Ocean Res. 83 (2019) 9–20.
[11] B. Fu, L. Chen, Y. Zhou, D. Zheng, Z. Wei, J. Dai, H. Pan, An improved A* algorithm for the industrial robot path planning with high success rate and short length, Rob. Auton. Syst. 106 (2018) 26–37.
[12] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel, Path planning for autonomous vehicles in unknown semi-structured environments, Int. J. Robot. Res. 29 (5) (2010) 485–501.
[13] J. Han, Y. Seo, Mobile robot path planning with surrounding point set and path improvement, Appl. Soft Comput. 57 (2017) 35–47.
[14] J. Han, An efficient approach to 3D path planning, Inf. Sci. 478 (2019) 318–330.
[15] K. Daniel, A. Nash, S. Koenig, A. Felner, Theta*: any-angle path planning on grids, J. Artif. Intell. Res. 39 (2010) 533–579.
[16] T. Uras, S. Koenig, C. Hernández, Subgoal graphs for optimal pathfinding in eight-neighbor grids, Twenty-Third International Conference on Automated Planning and Scheduling, 2013.
[17] T. Uras, S. Koenig, An empirical comparison of any-angle path-planning algorithms, Eighth Annual Symposium on Combinatorial Search, 2015.
[18] R.L. Rardin, Optimization in Operations Research, Prentice Hall Upper Saddle River, NJ, 1998.
[19] T. Uras, S. Koenig, Speeding-up any-angle path-planning on grids, Twenty-Fifth International Conference on Automated Planning and Scheduling, 2015.
[20] M. Faria, I. Maza, A. Viguria, Applying frontier cells based exploration and Lazy Theta* path planning over single grid-based world representation for autonomous inspection of large 3D structures with an UAS, J. Intell. Rob. Syst. 93 (1-2) (2019) 113–133.
[21] L. De Filippis, G. Guglieri, F. Quagliotti, Path planning strategies for UAVS in 3D environments, J. Intell. Rob. Syst. 65 (1-4) (2012) 247–264.
[22] H. Kim, D. Kim, J.-U. Shin, H. Kim, H. Myung, Angular rate-constrained path planning algorithm for unmanned surface vehicles, Ocean Eng. 84 (2014) 37–44.
[23] H. Kim, D. Kim, H. Kim, J.-U. Shin, H. Myung, An extended any-angle path planning algorithm for maintaining formation of multi-agent jellyfish elimination robot system, Int. J. Control Autom. Syst. 14 (2) (2016) 598–607.
[24] K. Yakovlev, A. Andreychuk, Any-angle pathfinding for multiple agents based on SIPP algorithm, Twenty-Seventh International Conference on Automated Planning and Scheduling, 2017.
[25] A. Nash, S. Koenig, Any-angle path planning, AI Mag. 34 (4) (2013) 85–107.
[26] S. Choi, J. Lee, W. Yu, Fast any-angle path planning on grid maps with non-collision pruning, IEEE International Conference on Robotics and Biomimetics, IEEE, 2010.
[27] A. Nash, S. Koenig, C. Tovey, Lazy Theta*: Any-angle path planning and path length analysis in 3D, Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.
[28] R. Geraerts, M.H. Overmars, The corridor map method: a general framework for real-time high-quality path planning, Comput. Anim. Virtual Worlds 18 (2) (2007) 107–119.
[29] S.S. Mehta et al, Human-assisted RRT for path planning in urban environments, 2015 IEEE International Conference on Systems, Man, and Cybernetics, IEEE, 2015.
[30] J. Han, Y. Seo, Path regeneration decisions in a dynamic environment, Inf. Sci. 450 (2018) 39–52.
[31] J.E. Bresenham, Algorithm for computer control of a digital plotter, IBM Syst. J. 4 (1) (1965) 25–30.