Appl Intell (2013) 39:217–235
DOI 10.1007/s10489-012-0406-4

BA*: an online complete coverage algorithm for cleaning robots

Hoang Huu Viet · Viet-Hung Dang · Md Nasir Uddin Laskar · TaeChoong Chung

Published online: 19 December 2012

Abstract This paper presents a novel approach to solve the online complete coverage task of autonomous cleaning robots in unknown workspaces based on the boustrophedon motions and the A* search algorithm (BA*). In this approach, the robot performs a single boustrophedon motion to cover an unvisited region until it reaches a critical point. To continue covering the next unvisited region, the robot wisely detects backtracking points based on its accumulated knowledge, determines the best backtracking point as the starting point of the next boustrophedon motion, and applies an intelligent backtracking mechanism based on the proposed A* search with smoothed path on tiling so as to reach the starting point with the shortest collision-free path. The robot achieves complete coverage when no backtracking point is detected. Computer simulations and experiments in real workspaces prove that our proposed BA* is efficient for the complete coverage task of cleaning robots.

Keywords A* search algorithm · Boustrophedon motions · Cleaning robot · Complete coverage

H.H. Viet · M.N.U. Laskar · T.C. Chung (✉)
Department of Computer Engineering, Kyung Hee University, 1-Seocheon, Giheung, Yongin, Gyeonggi, 446-701, South Korea
e-mail: tcchung@khu.ac.kr

H.H. Viet
e-mail: viethh@khu.ac.kr

M.N.U. Laskar
e-mail: nasir@khu.ac.kr

V.-H. Dang
Research and Development Center, Duy Tan University, K7/25 Quang Trung, Da Nang City, VietNam
e-mail: dangviethungha@gmail.com

1 Introduction

In recent years, the complete coverage mission for service robots has attracted significant attention from the robotics research community due to the robots' crucial role in a wide range of applications, such as floor cleaning, lawn mowing, mine hunting, harvesting, and so forth [5, 19, 29]. In known workspaces, the complete coverage task is the determination of a path that allows a service robot to cover every part of a workspace while minimizing the path length, journey time, or energy consumption. The complete coverage task can be planed using genetic algorithms [8], neural networks [19, 29], cellular decomposition [1, 4, 6, 20], spanning trees [11, 12], spiral filling paths [13, 14], and the ant colony method [3, 16]. Although these approaches can successfully solve the complete coverage task, they are inadequate for dealing with unknown workspaces. These methods require a full map or a complete model of the workspaces before the coverage task is executed, and thus they are merely offline methods. In contrast, the complete map is not available to the robot in unknown workspaces and can only be discovered online by the robot's exploration such as executing actions and observing the action effects. At each instant in time during the cleaning process, the robot senses only the local contents of the workspace, including its current position, and the immediate neighboring positions. Moreover, the robot must make a decision about the "next action" as soon as it obtains the information by interacting with the environment and by sensing the interactions at that time. Obviously, an autonomous cleaning robot in unknown workspaces requires an online algorithm to estimate the coverage path instead of the mentioned offline algorithms, which are insufficient when a map is unavailable.

Although online algorithms are necessary for unknown environments, few research works on the online complete

coverage mission exist in literature. The main reason for this deficiency is that almost all cleaning robots, including commercial robots, are equipped with only a few simple touch and distance sensors, making it difficult to navigate and learn an environment. Therefore, these robots are limited to simple random path-planning algorithms to achieve complete coverage. Examples of these robots include the Roomba designed by iRobot, the RC3000 designed by Karcher, and the Trilobite designed by Electrolux. All of these robots use the random straight path-planning algorithm [24, 25], which drives them to go straight until collision and turn a random angle before going straight again. To improve the cleaning robot's performance, Oh et al. [23] present a complete coverage navigation method for cleaning robots using a triangular-cell-based map representation. While a rectangular-cell-based map has eight navigation directions, the triangular-cell-based map increases the number of navigation directions of the robot to twelve. This increase shortens the navigation path of the robot from the end point of the current covered region to the starting point of the next uncovered region. Unfortunately, the length of the navigation path depends strongly on the selection of the size of the triangular cells, and it is difficult, if not impossible, to determine the best size to attain an optimal navigation path. Furthermore, although the navigation path of the robot obtained from the triangular-cell-based map is shorter than that obtained from the rectangular-cell-based map, the path is still unnecessarily long because the limited number of navigation directions of the triangular-cell-based map representation usually results in a broken line path instead of a line-of-sight path. As a result, this method may be good for computer simulations with simple workspaces, but not for real scenarios, which are more complicated. Wong [28] introduces an approach for cleaning robots where the complete coverage task is decomposed into two phases. The first phase is the construction of a map of the workspace with the information gathered by the robot's sensors. The second phase is the use of the constructed model to plan the complete coverage path. To construct the workspace map, the slide decomposition method, which is similar to the Morse decomposition [1], is proposed to decompose the workspace into non-overlapping cells, where each cell can be covered by a zigzag pattern, and the boundaries of the cells are detected by landmarks such as walls and corridors. To plan the coverage path, a path-planning algorithm is developed to generate paths from the incomplete map. Although this approach fulfills the complete coverage task, it does not have an optimal backtracking mechanism when the robot finishes covering the current cell and moves to other uncovered cell. Therefore, this method is inefficient for the complete coverage mission of the robot.

For online methods, an autonomous cleaning robot should have the ability to reach and clean all accessible areas in its workspace. The robot is assumed to be capable of correctly detecting obstacles by using necessary sensors during the navigation time, regardless of the sizes, shapes, and positions of the obstacles. The workspace must also be closed and the number of obstacles in the workspace must be finite. Under these assumptions, our work attempts to deal with five main challenges [23]. First, the cleaning robot has no prior knowledge about its workspace and its initial position can be random. Second, the coverage path of the robot must ensure the coverage of all accessible areas. Third, the coverage path must be as short as possible in order to reduce the cleaning time. Fourth, the complete coverage navigation should be composed of sequential and continuous operations, and simple operation paths are preferred. Finally, the complete coverage algorithm is preferably systematic and structured, avoiding complex mixtures of cases and conditions. From this point of view, we propose a computationally low-cost but efficient approach for the online complete coverage task in unknown workspaces based on the boustrophedon motions and the A* search algorithm, titled BA*. The term boustrophedon literally means "*the way of the ox*." The boustrophedon motion mimics the simple back-and-forth motion of an ox when plowing a field [6] (Fig. 1). In the proposed BA*, the robot performs a single boustrophedon motion to cover an unvisited region until it arrives at a critical point. To reach the next starting point of an unvisited region via the shortest collision-free path, the robot wisely detects backtracking points based on its accumulated knowledge, determines the best backtracking point as the next starting point by using a greedy strategy, and applies an intelligent backtracking mechanism based on the proposed A* search with a smoothed path on tiling. The robot finishes its coverage mission if no backtracking point is detected. Computer simulations and experiments in real workspaces prove that our proposed BA* executed by a robot works properly in unknown workspaces with arbitrarily shaped obstacles. Furthermore, BA* is efficient for cleaning robots in terms of the robot design cost with few and simple sensors, the length of the coverage path, the total number of boustrophedon motions, and the simple coverage path with a small number of heading changes.

The rest of this paper is organized as follows: Section 2 gives the background of our approach. Section 3 presents our approach to the online complete coverage problem. Section 4 discusses the simulations and evaluations. Section 5 describes our experiments on *iRobot Create* in real workspaces. Finally, Sect. 6 concludes our work.

2 Background

2.1 The boustrophedon cellular decomposition

A boustrophedon motion is a sequence of simple back-and-forth motions as shown in Fig. 1(a). An application of

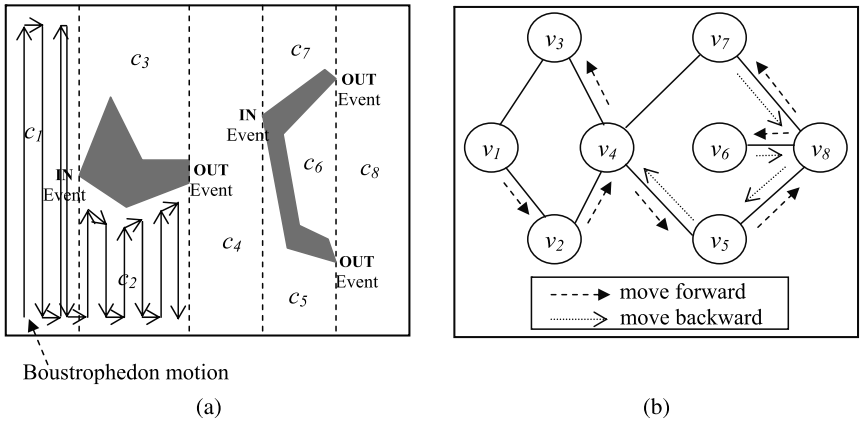


Fig. 1 (**a**) A boustrophedon motion and the BCD; (**b**) The adjacent graph describes the BCD and the walk on the graph starts at vertex v_1

the boustrophedon approach to the coverage path-planning problem for cleaning robots, called Boustrophedon Cellular Decomposition (BCD), is proposed by Choset and Pignon [4, 6]. In this method, the accessible area of the robot is decomposed into non-overlapping cells as illustrated in Fig. 1(a). The cells are formed by two types of events: IN and OUT. These events occur while a vertical line, called a *slice*, sweeps from left to right through a bounded workspace. An IN event occurs when the slice intersects a vertex of a polygonal obstacle so that the current cell is closed and two new cells are opened. Conversely, an OUT event occurs when the slice intersects a vertex of a polygonal obstacle so that the two current cells are closed and one new cell is opened. Through such cellular decomposition, the coverage in each cell can be achieved by a boustrophedon motion. If each cell is represented by a vertex and the adjacent cells are connected to form the edges of the graph, then the coverage problem is reduced to determine a walk through the graph in which each vertex is visited at least once, i.e., the traveling salesman problem. The BCD for the complete coverage problem is resolved by Algorithm 1.

An illustration of BCD is shown in Fig. 1. The workspace is decomposed into eight cells $c_1, c_2, \ldots, c_8$, and then the adjacent graph is constructed as in Fig. 1(b), where vertex v_i represents cell c_i $(i = 1, 2, \ldots, 8)$ of the decomposition. Starting from vertex v_1, the walk is $\mathcal{V} = \{v_1, v_2, v_4, v_5, v_8, v_7, v_8, v_6, v_8, v_5, v_4, v_3\}$, which is depicted by arrows as shown in Fig. 1(b). To perform the complete cleaning task, the robot covers an unvisited cell by a single boustrophedon motion. It visits cells in the order of c_1, c_2, c_4, c_5, c_8, and c_7, and then backtracks to cover cells c_6 and c_3.

Although BCD can achieve complete coverage of workspaces, it faces two major challenges when solving the complete coverage problem of cleaning robots: (i) BCD can generate too many cells, thereby lengthening the coverage path; and (ii) it is not suitable for the robots to solve the complete coverage problem in unknown workspaces because it requires the workspace map beforehand.

Algorithm 1 BCD algorithm

Step 1. Decompose the accessible area of the workspace into non-overlapping cells.

Step 2. Construct an adjacent graph where each vertex is a cell and each edge connects two vertices corresponding to two adjacent cells.

Step 3. Determine an exhaustive walk through the adjacent graph based on a depth-first-like graph search algorithm such that each vertex is visited at least once. Let $\mathcal{V}$ be the list that represents a consecutive sequence of vertices of a walk through the adjacent graph. $\mathcal{V}$ is determined using the depth-first-like graph search algorithm as follows:

Step 3.1. Start with any cell resulting from the decomposition. Add it into $\mathcal{V}$ and mark it as visited.

Step 3.2. Move to the first counterclockwise unvisited cell in the neighboring cells of the current cell. Add this cell into $\mathcal{V}$ and mark it as visited.

Step 3.3. Repeat *Step 3.2* until reaching a cell whose neighboring cells have all been visited.

Step 3.4. Backtrack and add each visited cell into $\mathcal{V}$ until a cell with an unvisited neighboring cell is reached. Go to *Step 3.2*.

Step 3.5. If no cell with an unvisited neighboring cell is found during the backtracking process, then the walk is the consecutive sequence of vertices in $\mathcal{V}$.

Step 4. As $\mathcal{V}$ is determined, drive the robot to start at the first cell corresponding to the first vertex of $\mathcal{V}$ and move it to the next cell based on $\mathcal{V}$. Perform the cleaning task by using a boustrophedon motion only when the robot enters an unvisited cell. Repeat moving and cleaning until the cell corresponding to the final vertex in $\mathcal{V}$ is reached.

2.2 A* search algorithm

A* search [15] is a widely-known form of the best-first search algorithms. It can be applied to solve the path-planning problem of mobile robots on grids [22, 30] or to find the shortest path from a given starting vertex to a given goal vertex in graphs. A* search defines a "*heuristic estimate*" f-value $f(s) = g(s) + h(s)$, which is an estimate of the length of the shortest path from the starting vertex via vertex s to the goal vertex, where the g-value $g(s)$ of vertex s is the length of the shortest path from the starting vertex to vertex s that has been found so far, and the h-value $h(s)$ of vertex s is an estimate of the distance from vertex s to the goal vertex. A* search is optimal if the h-value $h(s)$ of vertex s is an admissible heuristic, i.e., $h(s)$ never overestimates the path cost from vertex s to the goal [26].

Given a starting vertex s_s and a goal vertex s_g as inputs, A* search determines the shortest path consisting of a consecutive sequence of vertices from s_s to s_g as an output. To do this, it uses three lists of vertices: *closed*, *open*, and *parent*. The *closed* list is a set of vertices that have been expanded to ensure that every vertex of the graph is expanded at most once. The *open* list is a priority queue of vertices that have been generated but not yet expanded. The *parent* list is a set of vertices in which the index represents the child vertex's name and the value at this index represents the name of the vertex's parent. This list is ultimately used to extract a path from the starting vertex to the goal vertex as A* search ends. The A* search steps are described in Algorithm 2 [22].

When A* search is employed to deal with the robot path-planning problem, the common technique consists of two steps. First, the environment of the robot is represented as a graph $G = (S, E)$, where S is the set of possible locations of the robot and E is a set of edges that represent connections among these locations. Second, the path for the robot is achieved by applying A* search on this graph, in which the cost of each edge represents the cost of transition between the two locations.

Algorithm 2 A* algorithm

Inputs: The starting vertex s_s and the goal vertex s_g
Outputs: The solution path from s_s to s_g

Step 1. Assign the *open* and *closed* lists to the empty list.

Step 2. Assign the g-value of vertex s_s to zero, the parent of vertex s_s to vertex s_s, and the f-value of vertex s_s to the g-value of vertex s_s plus the h-value of vertex s_s.

Step 3. Add vertex s_s and its f-value to the *open* list.

Step 4. Check whether the *open* list is empty. If yes, the A* search stops with no solution.

Step 5. Find a vertex s with the minimum f-value in the *open* list. If this vertex is the goal vertex s_g, then A* returns the solution path, which is the path in reverse order of the path retrieved by following the *parent* list from the goal vertex to the starting vertex; otherwise, do the following steps:

Step 5.1. Remove vertex s from the *open* list and expand it by adding this vertex to the *closed* list.

Step 5.2. Check whether each neighboring vertex s' of vertex s is in the *closed* list. If yes, ignore it and return to Step 4; otherwise, assign the g-value of vertex s' to infinity and the parent of vertex s' to vertex s'.

Step 5.3. Check whether the g-value of vertex s plus the straight line distance from vertex s to vertex s' is smaller than the g-value of vertex s'. If yes,

(i) Assign the g-value of vertex s' to the g-value of vertex s plus the straight line distance from vertex s to vertex s', assign the f-value of vertex s' to the g-value of vertex s' plus the h-value of vertex s', and set the parent of vertex s' to vertex s.
(ii) Check whether vertex s' is in the *open* list. If yes, update the f-value of vertex s' in the *open* list; otherwise, add vertex s' and its f-value to the *open* list.

Step 6. Go to Step 4.

3 The proposed BA* method

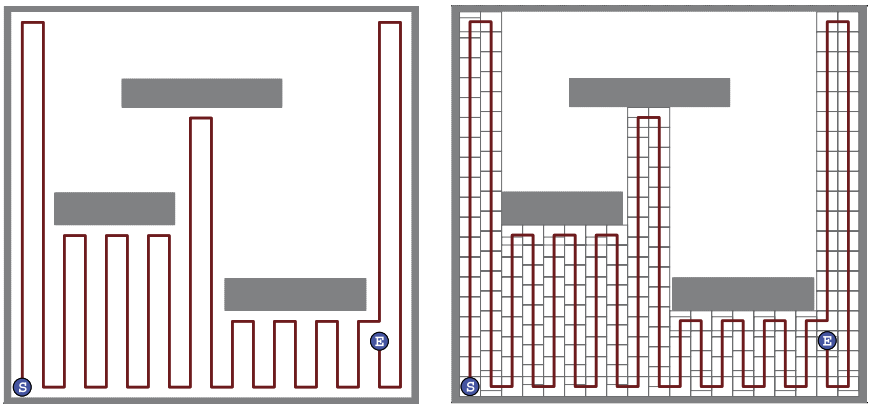
This section describes our approach to the online complete coverage task of an autonomous cleaning robot in unknown workspaces. At each time step, based on its knowledge so far and the states of neighboring positions, the robot has to perform a strategy to completely cover the accessible area with an as short as possible path.

3.1 Representing the robot's configuration

The representation of an autonomous mobile robot is usually the *point robot* in its workspace. Specifically, the robot can be represented as a point $(x, y) \in \mathcal{R}^2$ in the continuous Cartesian plane. The point (x, y) fully describes the coordinate position of the robot in the plane. Nevertheless, this representation is insufficient for a rigid robot that is capable of translating and rotating in the plane. In this work, the orientation of the robot at each position needs to be considered, and the *configuration* of the robot is defined as

$$q = [x, y, \theta]^T, \tag{1}$$

where (x, y) is the center position and θ is the heading angle of the robot in the fixed frame F_w of the workspace $\mathcal{W} = \mathcal{R}^2$ [18]. The robot is assumed to be modeled by a circle with



(a) (b)

Fig. 2 (**a**) A boustrophedon path is generated by the robot's moves, where positions S and E are the starting and ending positions of the robot, respectively; (**b**) A part of the tiling model is constructed incrementally as the robot moves

radius r. At the current configuration $q = [x, y, \theta]^T$, if the robot moves forward some distance d, the next configuration $q' = [x', y', \theta']^T$ is determined as

$$q' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} d\cos(\theta) \\ d\sin(\theta) \\ 0 \end{bmatrix}. \tag{2}$$

Meanwhile, if the robot turns at an angle α, the next configuration $q' = [x', y', \theta']^T$ is determined as

$$q' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \alpha \end{bmatrix}. \tag{3}$$

If $\alpha > 0$, the robot rotates in the counterclockwise direction, and vice versa.

When the robot moves in workspace $\mathcal{W}$, it cannot access all the regions of $\mathcal{W}$. The accessible regions are assumed to be connected and can be reached by the robot from any initial position. Since the configuration q of robot $\mathcal{A}$ is a specification of the physical state of $\mathcal{A}$ with respect to a fixed position in $\mathcal{W}$, a configuration q of robot $\mathcal{A}$ is valid if robot $\mathcal{A}$ can access the position specified by q in $\mathcal{W}$. The set of all valid configurations of robot A is defined as *free space*. The configuration q in $\mathcal{W}$ that robot $\mathcal{A}$ cannot access is either called invalid or corresponds to *obstacles*. In other words, the configuration space of robot $\mathcal{A}$ is the space $\mathcal{C}$ of all possible configurations of $\mathcal{A}$ in the workspace $\mathcal{W}$ [9]. Let O_i be an obstacle in $\mathcal{W}$, the union of all O_i becomes the *obstacle region* in $\mathcal{W}$. Since the obstacle region prohibits certain configurations of the robot, the *free space* of the robot is defined as

$$\mathcal{C}_{free} = \left\{ q \in \mathcal{C} \mid \mathcal{A}(q) \cap \left(\bigcup_i O_i \right) = \emptyset \right\}, \tag{4}$$

where $\mathcal{A}(q)$ is the portion of $\mathcal{W}$ occupied by robot $\mathcal{A}$ when the robot is in configuration q.

3.2 Constructing the boustrophedon motion

Similar to exact cellular decomposition methods [1, 6, 28] for the complete coverage task, our method has to break the free space $\mathcal{C}_{free}$ of the robot into non-intersecting regions. The difference is that our approach constructs the regions in an incremental manner via boustrophedon motions, so each region is called a *boustrophedon region*. This is one of the key ideas of this work, that is, to reduce the number of non-intersecting regions even when the workspace is unknown to the robot. The smallest boustrophedon region is defined to be a square tile that is the size of the robot's diameter, which means that if the robot covers all boustrophedon regions, then the free space will definitely be covered completely.

To construct a boustrophedon motion, the robot moves at each time step from the current position to one of its four adjacent positions based on the four directions: north, south, east, and west, provided that the directions do not lead the robot into a *blocked position*, which is either an obstacle or a covered tile, as illustrated in Fig. 2(a). As a result, a *tiling model* is constructed incrementally as the robot moves as shown in Fig. 2(b). Each element of the tiling model is a *tile* determined when the robot passes a distance of its diameter. Therefore, the tile has the size of the square bounding the robot. In addition, the tiles can be overlapped when the robot moves close to obstacles or to the boundaries of the workspace. By such a representation, the covered positions of the robot are identified as the positions of the tiles, while the tiles are undefined for uncovered positions. The key purpose of generating the model of the workspace is to build a map for a backtracking mechanism which will guide the robot to the next uncovered region.

The algorithm for the robot to perform a boustrophedon motion is shown in Algorithm 3, where the next robot's configuration is determined as in Eqs. (2) or (3). In this algorithm, the robot uses sensors on its front, left, and right sides to identify obstacles, and uses its memory to identify the covered tiles. In other words, the robot checks a blocked po-

Algorithm 3 Boustrophedon motion (BM) algorithm

Inputs: The robot's configuration and the model $\mathcal{M}$ of the workspace

Outputs: Updated version of the robot's configuration and the model $\mathcal{M}$ of the workspace

Step 1. Check to find the first available direction in the priority of north-south-east-west. If all directions are blocked, then the critical point has been reached. Break the loop.

Step 2. Move one step along this direction.

Step 3. Generate the tile $s = (x, y, 2r)$, i.e., the size of the robot's diameter at the robot's position.

Step 4. Add the tile s to the mode $\mathcal{M}$. Go to Step 1.

sition not only by using sensors, but also by using its accumulated knowledge. At each algorithm iteration, the robot moves only one step along the direction leading to the uncovered position, which means that if the robot moves north, it keeps checking north and moves forward via iterations. While moving south, the robot checks north (already covered) by its memory and then checks south and moves south, which means it will keep moving south if the south direction is uncovered. When the robot encounters a blocked position, the east (or west) direction will be checked and the robot moves east (or west) one step. Then, it will be in the new iteration and will again check the north-south-east-west direction in priority. Obviously, this priority mechanism allows the robot to construct the boustrophedon path until the *critical point*, a tile all of whose neighboring tiles are blocked, is reached. At each iteration, the robot moves one step of its diameter length to the next valid configuration $q' = [x', y', \theta']^T$. If the robot hits an obstacle before completing the given distance, it will stop and establish a new configuration based on the measured distance while moving. When the robot hits an obstacle, the tile will overlap with the previous tile. Let $s = (x, y, 2r)$ denote a square tile bounding the robot, where the center (x, y) of the tile is at the center of the robot and the size of the tile is $2r \times 2r$. The tiling model $\mathcal{M}$ of the workspace $\mathcal{W}$ is constructed gradually as the robot moves and each element of $\mathcal{M}$ is a square tile s:

$$\mathcal{M} = \{s | s \in \mathcal{C}_{free}\}. \tag{5}$$

3.3 Constructing the backtracking mechanism

The backtracking mechanism refers to the determination of the backtracking point as the next starting point of an uncovered region and the shortest collision-free path to guide the robot from the critical point of the current boustrophedon motion to the next starting point.

3.3.1 Determining the backtracking point

In most cases, the complete coverage cannot be completed with a single boustrophedon motion. Figure 3(a) shows an illustrative example, where four boustrophedon motions labeled 1 through 4 are performed to completely cover the free space $\mathcal{C}_{free}$. Each boustrophedon motion starts at a position S and ends at a position E. Figure 3(b) shows the complete tiling model constructed after the robot has covered the entire workspace. To construct non-intersecting boustrophedon regions in an incremental manner, whose union is the entire free space, the starting point for a new boustrophedon motion must be determined. The starting point of a new boustrophedon motion can be at any uncovered position in the workspace. However, randomly choosing the next starting point is not a wise decision because it fragments the remaining accessible area, thereby increasing the number of boustrophedon regions and the complete coverage path length. As a result, the mechanisms to determine

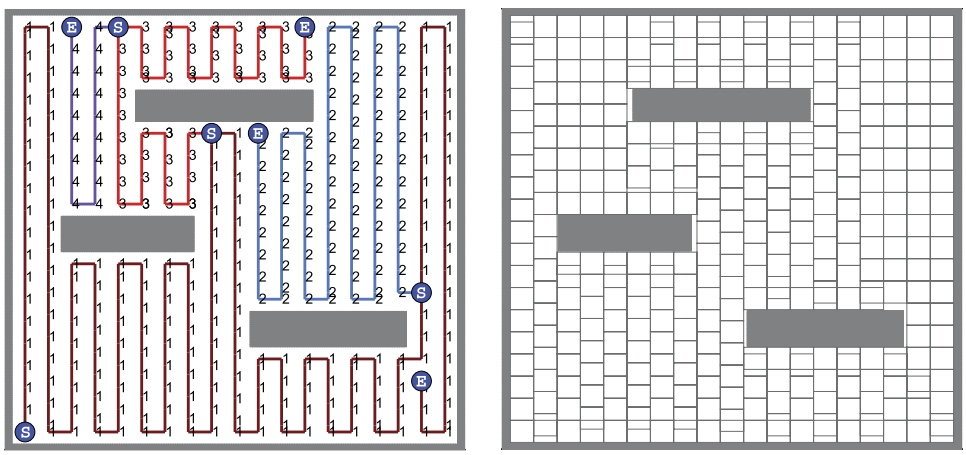


Fig. 3 (**a**) Four boustrophedon regions are generated by four boustrophedon motions labeled 1 through 4. Each boustrophedon motion starts at the position S and ends at the position E; (**b**) A complete tiling model is constructed after the robot has covered the entire free space

the "*best starting point*" for the next boustrophedon motion and to guide the robot from the current critical point to that point must be included. The starting point of the next boustrophedon motion must be detected from the information of the covered positions, i.e., from the tiling model $\mathcal{M}$ that has been constructed so far because the workspace is unknown in advance to the robot. In other words, a starting point is a *backtracking point* from which the robot can start a future boustrophedon motion.

To indicate the candidates for the next starting point, a set of backtracking points must be detected and stored in a *backtracking list* when the robot arrives at the critical point of the current boustrophedon motion. Basically, a backtracking point can be any tile where at least one cell with the size of the robot's diameter in its eight neighboring cells is uncovered. If so, numerous backtracking points can become the potential starting points and must be all maintained, leading to the inefficiency of deciding which point is the best for the next starting point. Many backtracking points on the side of the uncovered regions can also result in many boustrophedon motions, and thus lengthen the coverage path. To overcome these drawbacks, we propose a method to reduce the total number of backtracking points and the number of boustrophedon motions in which eight neighboring cells of tile s are used to determine the existence of the candidates of the starting point. Figure 4 shows all of the cases in which tile s is decided to be a backtracking point, where each black cell denotes a *blocked position* and the white cell denotes a *free position* uncovered by the robot. Meanwhile, each grey cell represents the "*don't care*" status, or the grey cell can be any blocked or uncovered cell. Specifically, let $\mathcal{N}(s) = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ be the set of eight neighboring cells of tile s in eight directions, east, north-east, north, north-west, west, south-west, south, and south-east, respectively. For any two cells $s_i \in \mathcal{N}(s)$ and $s_j \in \mathcal{N}(s)(i, j = 1, 2, \ldots, 8)$, we define the

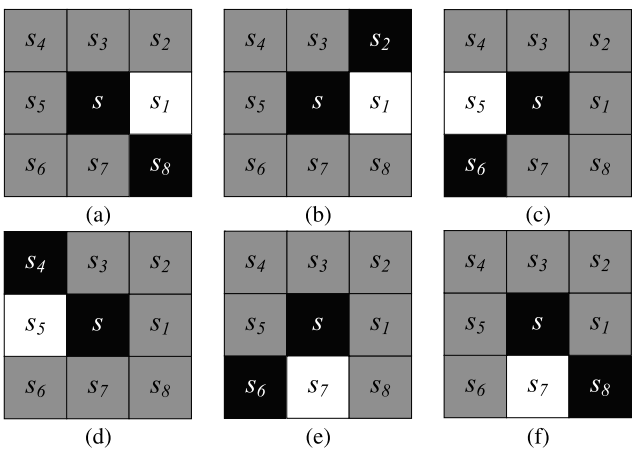


Fig. 4 Conditions of backtracking points

function

$$b(s_i, s_j) = \begin{cases} 1, & \text{if } (s_i \text{ is free) and } (s_j \text{ is blocked);} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

and the sum function

$$\mu(s) = b(s_1, s_8) + b(s_1, s_2) + b(s_5, s_6) + b(s_5, s_4) + b(s_7, s_6) + b(s_7, s_8), \tag{7}$$

where each component function $b(s_i, s_j)$ corresponds to each subfigure in Fig. 4. The tile s is defined to be a backtracking point if $\mu(s) \geq 1$. With this condition, the backtracking points are only the tiles at the corners of the boustrophedon regions. These conditions can reduce both number of backtracking points and number of boustrophedon regions because a starting point on the side of an uncovered polygon will create more fragments than a starting point at the corner of a polygon. Therefore, a fewer number of boustrophedon motions leads to a shorter path for complete coverage, which is one of the main advantages of the proposed approach. Let $\mathcal{L}$ be a list of the backtracking points and determined as

$$\mathcal{L} = \{s \mid s \in \mathcal{M} \text{ and } \mu(s) \geq 1\}. \tag{8}$$

Obviously, determining backtracking points while moving is difficult because the robot does not have enough information of all the corner neighbors (i.e., all s_2, s_4, s_6, and s_8) as shown in Fig. 4. Moreover, this is inefficient because some backtracking points added from the prior and current boustrophedon motion can be off the backtracking list $\mathcal{L}$ during the current boustrophedon motion. For this reason, in our method, the robot does not determine the backtracking points while constructing the boustrophedon motion until a critical point is detected. At the critical point, the robot "recalls" the accumulated knowledge and creates the backtracking list based on the proposed rules (i.e., Eqs. (6), (7), and (8)). Therefore, when examining whether a noncritical visited tile to be a backtracking point, the robot has enough information of all the neighbors around the tile.

Next, the robot must determine the best backtracking point as the starting point of the next boustrophedon motion based on a specific measurement. However, the robot has no information about the workspace in advance; only partial information accumulated from the prior covering process. Therefore, the starting point of the next boustrophedon motion cannot be determined so that the complete coverage path is the global optimization in terms of the length and the total number of boustrophedon regions. Instead, the starting point must be determined from the accumulated knowledge or, more specifically, from the backtracking list $\mathcal{L}$. The selection of the best backtracking point can be estimated based on different criteria, such as the nearest point to the current

critical point using the Euclidean distance, Manhattan distance, or even the shortest path using a distance propagation algorithm based only on the covered tiles. In general, the best backtracking point or the starting point s_{sp} of the next boustrophedon motion is proposed in the form of a greedy strategy:

$$s_{sp} = \arg\min_{s \in \mathcal{L}}(f(s, s_{cp})), \tag{9}$$

where $f(s, s_{cp})$ is a distance-based cost function that defines the distance cost between an element s in the backtracking list $\mathcal{L}$ and the critical point s_{cp}.

3.3.2 Planning the backtracking path

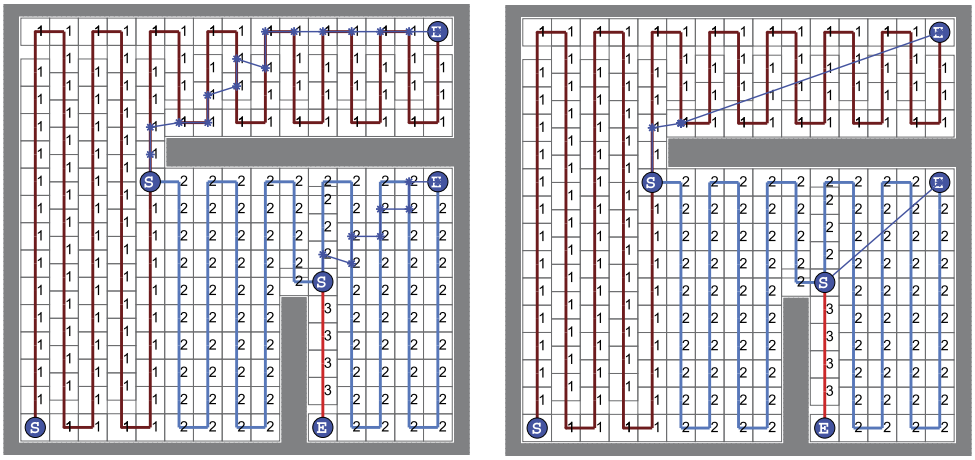
This section proposes a backtracking mechanism to guide the robot from the critical point to the next starting point while avoiding obstacles. The easiest way for the robot to achieve the collision-free movement is to backtrack along the old boustrophedon motion until it reaches the next starting point. However, the movement obtained by this way is unrealistic and lengthens the coverage path. With the model $\mathcal{M}$ found so far, the path-planning algorithms such as Dijkstra's algorithm [7], A* search [15], or even ED-FCM [21] can be used to plan a backtracking path for the robot.

The A* search is a popular solution for the path-planning problem [22, 30] because of its efficiency. Given a consistent heuristic $h(s)$, A* search expands only vertices whose total cost, $f(s) = g(s) + h(s)$, is less than c, the cost of an optimal solution [17, 26]. This means A* search does not explore the entire search space while guaranteeing an optimal solution. Hence, A* search is employed in this work to determine the backtracking path for the robot. At the critical point, the robot plans the shortest collision-free path to the next starting point using A* search based only on the regions it had already visited, i.e., the tiling model $\mathcal{M}$ that has been build so far, called *A* search on tiling*. Since the solution is constructed from tiles that have been accessed by the robot, A* search always gives a solution. Furthermore, we define the $h(s)$ of A* search to be the straight-line distance from tile s to the next starting point. The shortest path between any two points is a straight line. Therefore, the straight-line distance is admissible or $h(s)$ cannot be an overestimate. This means that the collision-free path found by A* search from the critical point to the next starting point is the shortest path based on the connections between adjacent tiles (see proof in [26]).

The model $\mathcal{M}$ of workspace $\mathcal{W}$ constructed in the BM algorithm is the tiling model. Therefore, the path obtained from the A* search on the tiling model $\mathcal{M}$ is constrained to the adjacent tiles, and the path is not equivalent to the true shortest path. Moreover, the robot usually performs many heading changes while moving along this path from the current critical point to the next starting point. One can improve the precision of the path by increasing the tiling resolution. However, in that case, the state space of the model will increase and the path-planning process of the robot will be inefficient. A*PS [2] and Theta* [22] are both variants of A* with smoothing process, which can be used to overcome this drawback. A* search gives the shortest path through adjacent tiles and its variants smoothen the path obtained by A* search. These smoothing A* variants are contemporarily the best methods for finding the shortest path between positions on grids. A*PS smoothens the resulting path obtained by the A* search in a post-processing step. Meanwhile, Theta* inserts the smoothing task into the iterations of the searching process with higher computation cost and more complexity to control. Therefore, we choose A*PS and add a small modification, called *A* search with smoothed path on tiling* (A*SPT). Our modification is that at the current considered point, we try to find and connect to the farthest line-of-sight point on the path while the other variants find the nearest non-line-of-sight point on the path and then connect to previous neighbors of this point. The modification not only reduces the computation cost, but also shortens the path length in most cases. The efficiency of this approach can be seen through a simple example: assume that the path found by A* is $\mathcal{P} = [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9]$ and lines-of-sight exist between s_2 and s_5, and s_2 and s_8. Other smoothing A* variants then give the smooth path $\hat{\mathcal{P}} = [s_1, s_2, s_5, s_6, s_7, s_8, s_9]$ while our approach gives $\hat{\mathcal{P}} = [s_1, s_2, s_8, s_9]$. Moreover, by sweeping back from the goal to the current point to find the farthest line-of-sight point, the robot does not have to check all of the points as the other A* variants do. In other words, A*SPT is a good choice for the shortest path to the backtracking path problem. A line-of-sight between two points is the straight line segment with the width of $2r$ (i.e., the width of the robot) and does not intersect any obstacles.

A*SPT is described in Algorithm 4. It is assumed that the path obtained by A* search on tiling is a consecutive sequence of square tiles or $\mathcal{P} = [s_1, s_2, \ldots, s_n]$, where s_1 is the critical point s_{cp} of the current boustrophedon motion and s_n is the starting point s_{sp} of the next boustrophedon motion. A*SPT returns the path connecting the tiles, i.e., $\hat{\mathcal{P}} = [\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_k]$ with $\hat{s}_1 = s_{cp}$ and $\hat{s}_k = s_{sp}$, which is guaranteed to be shorter and smoother than $\mathcal{P}$ obtained by A* search on tiling. Figure 5 illustrates an example where the boustrophedon motions labeled 1 through 3 are obtained by the BM algorithm, and the remaining paths are obtained by A* search on tiling (Fig. 5(a)) and by the A*SPT algorithm (Fig. 5(b)).

To follow the backtracking path $\hat{\mathcal{P}}$ from the current critical point $\hat{s}_1 = s_{cp}$ to the next starting point $\hat{s}_k = s_{sp}$, the robot has to determine the angle and the distance between two tiles $\hat{s}_i$ and $\hat{s}_{i+1} (i = 1, 2, \ldots, k-1)$. When the robot is at tile $\hat{s}_i = (x_i, y_i, 2r) = (q \cdot x, q \cdot y, 2r)$, the current direction of the robot is $q \cdot \theta$ and the direction of the robot



(a) (b)

Fig. 5 The paths connecting the boustrophedon motions are obtained from A* search on the tiling (**a**) and from A*SPT in a continuous two-dimensional plane (**b**)

Algorithm 4 A* search with smoothed path on tiling (A*SPT) algorithm

Inputs: The path $\mathcal{P} = [s_1, s_2, \ldots, s_n]$ found by A* search and the model $\mathcal{M}$ found by BM algorithm

Outputs: The smoothed path $\hat{\mathcal{P}} = [\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_k]$

Step 1. Initialize $k = 1$ and add tile s_k to $\hat{\mathcal{P}}$.

Step 2. Find the farthest tile s_i $(i = n, n-1, \ldots, k+1)$ with a line-of-sight from s_k to s_i based on the model $\mathcal{M}$.

Step 3. Add tile s_i to $\hat{\mathcal{P}}$.

Step 4. Increase k by 1.

Step 5. Check whether tile s_k is the critical point (*i.e.*, $s_k = s_n$). If yes, return to path $\hat{\mathcal{P}}$; otherwise, go to Step 2.

required to move to tile $\hat{s}_{i+1} = (x_{i+1}, y_{i+1}, 2r)$ is

$$\beta = \arctan \frac{y_{i+1} - q \cdot y}{x_{i+1} - q \cdot x}, \quad \text{where } \beta \in (-\pi, \pi]. \tag{10}$$

Therefore, to move to $\hat{s}_{i+1}$, the robot needs to turn angle α and then move distance d as

$$\alpha = \beta - q \cdot \theta, \quad \text{where } \alpha \in (-\pi, \pi], \text{ and}$$
$$d = \sqrt{(x_{i+1} - q \cdot x)^2 + (y_{i+1} - q \cdot y)^2}. \tag{11}$$

When the robot reaches the next starting point $\hat{s}_k = s_{sp}$, its heading angle, $q \cdot \theta$, is the direction of the vector $\overrightarrow{\hat{s}_{k-1}, \hat{s}_k}$. To begin a new boustrophedon, the robot needs to change its direction before moving forward to one of four neighboring positions in the priority directions of north, east, south, or west. In other words, the robot has to rotate an angle of

$$\alpha = \gamma - q \cdot \theta, \quad q \cdot \alpha \in (-\pi, \pi], \tag{12}$$

where γ is $\pi/2, 0, -\pi/2$, or π depending on whether the next position is in the direction of north, east, south, or west, respectively.

3.4 BA* algorithm

In this section, we systematically present a BA* algorithm for the online complete coverage task of autonomous cleaning robots in unknown workspaces based on our proposed techniques. The steps of the BA* algorithm executed by a robot are shown in Algorithm 5. Initially, the robot has no prior knowledge about its workspace, which means that the model $\mathcal{M}$ of the workspace in the robot's memory is empty. The robot performs the coverage mission from its initial position until no backtracking point is detected.

BA* is an online algorithm because the robot does not require a prior full map of the workspace. Instead, it has only the local information that it gathers from its sensors to fulfill the complete coverage mission. An online algorithm does not have information on the whole workspace and is forced to make decisions that may not be optimal. Despite this drawback, BA* can give the nearly-optimal solution in terms of the coverage path length because it constructs boustrophedon regions that have the following features: (i) they are not overlapped, (ii) the union of all of the regions is the free space $\mathcal{C}_{free}$, (iii) the next starting point is at a corner of the next boustrophedon region, and (iv) the next starting point is linked to the critical point of the current boustrophedon region through the shortest collision-free path found by A*SPT. With the proposed BA*, the robot covers the accessible area with sequential operations and simple motion paths.

Algorithm 5 BA* algorithm

Inputs: The robot's configuration consists of the initial position and the heading angle
Outputs: None

Step 1. Initialize $\mathcal{M} = \emptyset$.

Step 2. Cover the workspace based on the BM algorithm as described in Algorithm 3. As soon as the robot arrives at the critical point, it finishes the current boustrophedon motion.

Step 3. Detect the backtracking list based on the model $\mathcal{M}$ found so far [i.e., Eq. (8)].

Step 4. Check whether the backtracking list $\mathcal{L}$ is empty, i.e., no backtracking point is detected. If yes, the coverage task of the robot ends; otherwise, do the following steps:

Step 5. Determine the best backtracking point as the starting point s_{sp} of the next boustrophedon motion to the critical point s_{cp} based on the backtracking list $\mathcal{L}$ [i.e., Eq. (9)].

Step 6. Plan a collision-free path from the critical point s_{cp} to the next starting point s_{sp} using A* search on model $\mathcal{M}$.

Step 7. Shorten the path obtained from A* using the A*SPT algorithm as described in Algorithm 4.

Step 8. Follow the path obtained by A*SPT from the critical point s_{cp} to the next starting point s_{sp} [i.e., Eq. (11)].

Step 9. Adjust the heading angle at the next starting point s_{sp} [i.e., Eq. (12)]. Then, go to Step 2 to cover the next uncovered region.

4 Simulations

In this section, we conduct computer simulations and compare our proposed BA* with the BCD [4, 6] for efficiency evaluation.

4.1 BA* working demonstration

This section presents simulations implemented with the Matlab software on a Core i5-2430M CPU 2.4 GHz with 4 GB RAM computer. The simulations are designed to evaluate the performance of our BA* algorithm for a cleaning robot in unknown workspaces with arbitrarily shaped obstacles. The input workspaces are binary images of 200×200 pixels, in which each pixel is marked to indicate whether it belongs to an obstacle or not based on its value being one or zero, respectively. We also assume that the robot is modeled by a circle with a radius of $r = 5$ pixels.

The first two simulations are implemented to verify the successful coverage rate of a cleaning robot. Initially, the robot is placed at the bottom-left corner of the workspaces. The best backtracking point of each critical point is determined as the nearest backtracking point based on the Euclidean distance. Figures 6(a) and 6(b) show the resulting paths of these simulations. The points S and E represent the starting point and the ending point of the robot, respectively. In Fig. 6(a), five boustrophedon paths obtained from the BM function and the remaining paths obtained from the A*SPT function are connected to form a path covering the entire workspace. Similarly, Fig. 6(b) shows seven boustrophedon paths obtained from the BM function and the remaining paths obtained from the A*SPT function. The path lengths required to cover the workspaces of Figs. 6(a) and 6(b) are 344.83 robot diameters and 352.05 robot diameters, respectively. If the best backtracking point of each critical point is determined by the A*SPT-based cost function

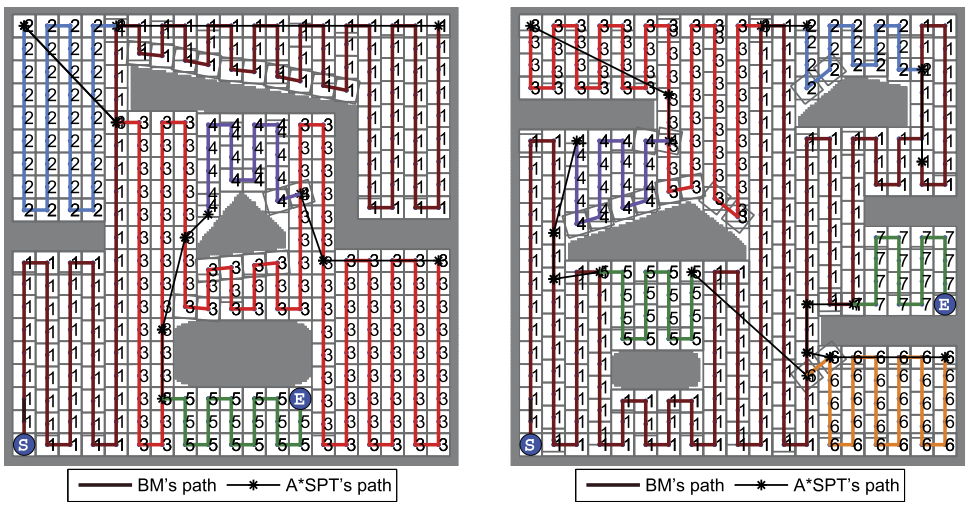


Fig. 6 The coverage paths are obtained from BA* with the Euclidean distance-based cost function. Boustrophedon paths are labeled 1 through 5 and 1 through 7 for cases (**a**) and (**b**), respectively. The remaining paths are obtained from A*SPT

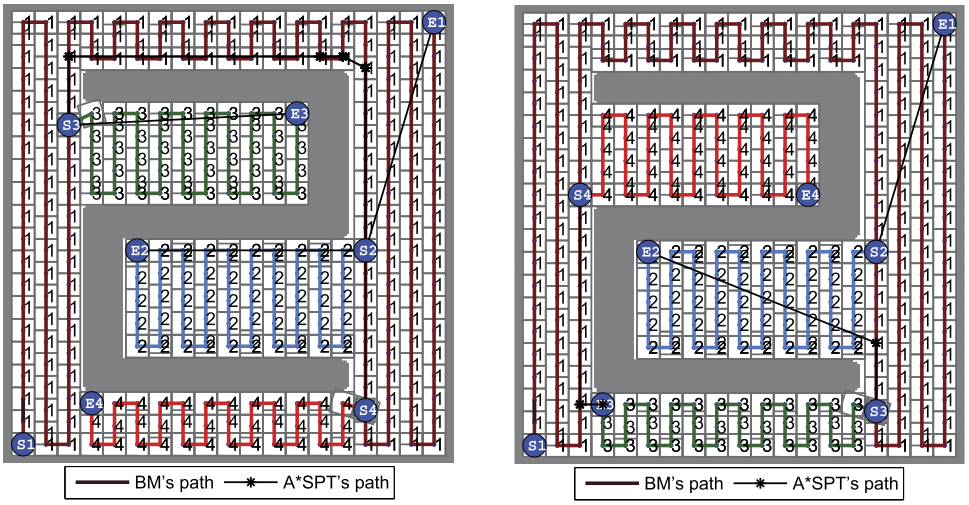


Fig. 7 The coverage paths are obtained from BA* with the Euclidean distance-based cost function (**a**), and the A*SPT-based cost function (**b**)

(i.e., the distance propagation of A*search with smoothed path on tiling), then the path lengths required to cover the workspaces of Figs. 6(a) and 6(b) are also 344.83 robot diameters and 352.05 robot diameters, respectively. In other words, under different distance cost functions (i.e., the Euclidean distance and A*SPT), the resulting path lengths are the same in this simulation set. These simulation results show that the BA* algorithm succeeds in controlling the robot to cover almost all of the free regions of the workspace, except for some small portions along the obstacle boundaries, which are insignificant compared with the whole workspace. If the successful coverage rate is defined as the ratio of the covered pixel number and the accessible pixel number, then the successful coverage rates of the robot in Figs. 6(a) and 6(b) are 96.92 % and 97.06 %, respectively.[1]

Although the simulations in Figs. 6(a) and 6(b) show that the coverage path for which the best backtracking points were determined by the Euclidean distance-based cost function is equal to that for which the best backtracking points were determined by the A*SPT-based cost function. This is not generally true for workspaces with arbitrarily shaped obstacles. If the workspace contains U-shaped obstacles, then the length of paths determined by using these two cost functions are unlikely to be equal. This is because the function based on the Euclidean distance measures the distance from the critical point to the best backtracking point does not consider whether the straight line between these two points goes through obstacles. To reach the best backtracking point from the critical point while avoiding obstacles, the robot has to follow the collision-free path found by A*SPT instead of the straight line to the best backtracking point. Thus, the backtracking determination based on the Euclidean distance will fail to achieve the shortest collision-free path. Meanwhile, A*SPT determines the best backtracking point based on the shortest collision-free path, and the robot follows this path to the determined point. Therefore, the coverage path with the A*SPT-based cost function is always shorter or equal to that of the Euclidean distance-based cost function. Furthermore, the number of boustrophedon regions with the A*SPT-based cost function is always smaller than or equal to that with the Euclidean distance-based cost function.

To verify this conclusion, Figs. 7(a) and 7(b) show two simulations where the starting points are determined by the Euclidean distance-based and the A*SPT-based cost functions, respectively. In these workspaces, positions Si and Ei $(i = 1, 2, 3, 4)$ represent the starting point and critical point of the ith boustrophedon motion, respectively. Figure 7(a) shows that when the robot finishes the second boustrophedon motion, it arrives at position $E2$. Based on the Euclidean distance, $S3$ is the backtracking point that is nearest to $E2$. Therefore, $S3$ becomes the next starting point. To reach this position, the robot has to follow the shortest collision-free path found by A*SPT, not the straight line from $E2$ to $S3$. Meanwhile, Fig. 7(b) shows that when the robot finishes the second boustrophedon motion, it also arrives at $E2$, which is the same position in Fig. 7(a). Based on the Euclidean distance, $S3$ is not nearest to $E2$, but the backtracking path from $E2$ to $S3$ is much shorter than the path from $E2$ to $S3$ in Fig. 7(a). The lengths of the coverage paths in Figs. 7(a) and 7(b) are 383.15 robot diameters and 331.75 robot diameters, respectively. This infers that the coverage path with the best backtracking points determined by the A*SPT-based cost function is shorter than that obtained with the best backtracking points determined by the

[1] If our method is implemented in a commercial cleaning robot such as the Samsung Robotic Vacuum Cleaner VC-RL87W (2011), then the robot may cover the entire workspace (100 %) because it is equipped with side brushes.

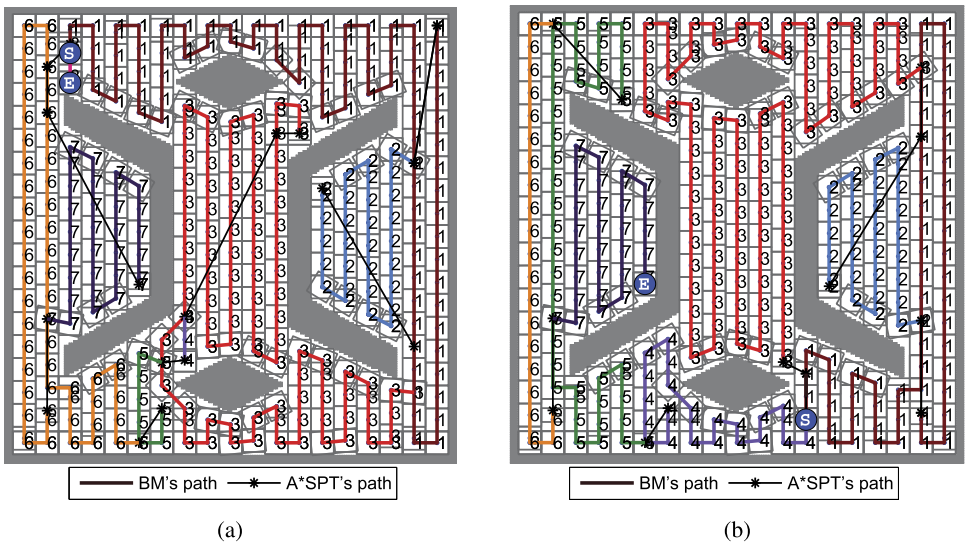


Fig. 8 The coverage paths obtained when the robot starts in the middle of the workspace

Euclidean distance-based cost function. In other words, the A*SPT-based cost function should be used to choose the best backtracking point to enhance the efficiency of BA* in terms of the coverage path length and the total number of boustrophedon regions.

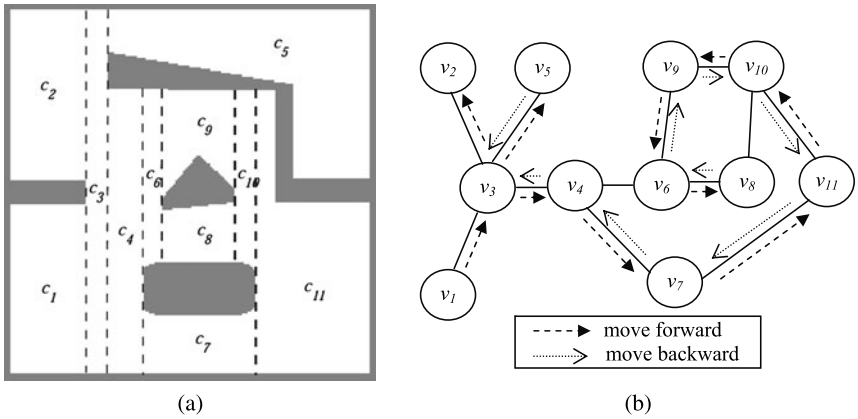
In the above simulations, the robot always starts at the bottom-left corner of the workspaces to perform the coverage mission. However, our BA* does not depend on the initial position of the robot, so the robot can start at any position in the workspace and still cover the entire workspace because BA* ends only when no backtracking point is detected. In other words, if an uncovered cell exists, the robot equipped with the intelligent backtracking mechanism moves to that cell to cover it. The next two experiments shown in Fig. 8 illustrate the situations in which the robot starts at random positions in the workspace. Specifically, points S and E represent the starting and ending points of the robot in each simulation, respectively. Figure 8(a) shows that the robot needs to perform eight boustrophedon motions with a total coverage path length of 346.84 robot diameters to cover the entire workspace. Meanwhile, Fig. 8(b) shows that the robot needs seven boustrophedon motions with a total coverage path length of 329.42 robot diameters to cover the entire workspace.

4.2 BCD and BA* comparison

In this section, simulations are conducted in order to compare the coverage path length, the number of boustrophedon regions, and the coverage rate of our BA* with those of the BCD approach [4, 6], which was reviewed in Sect. 2.1. Regardless of the fact that our BA* is an online method and BCD is an offline, BCD is chosen for comparison with our results because of three main reasons. First, BCD is an offline method, i.e., a map is available to determine a coverage path, which enables BCD to achieve the shortest coverage path. Second, both BCD and BA* use the boustrophedon motion technique to solve the complete coverage problem. Finally, BCD decomposes cells based on an exact cellular decomposition method, and thus it can yield a small number of decomposed cells.

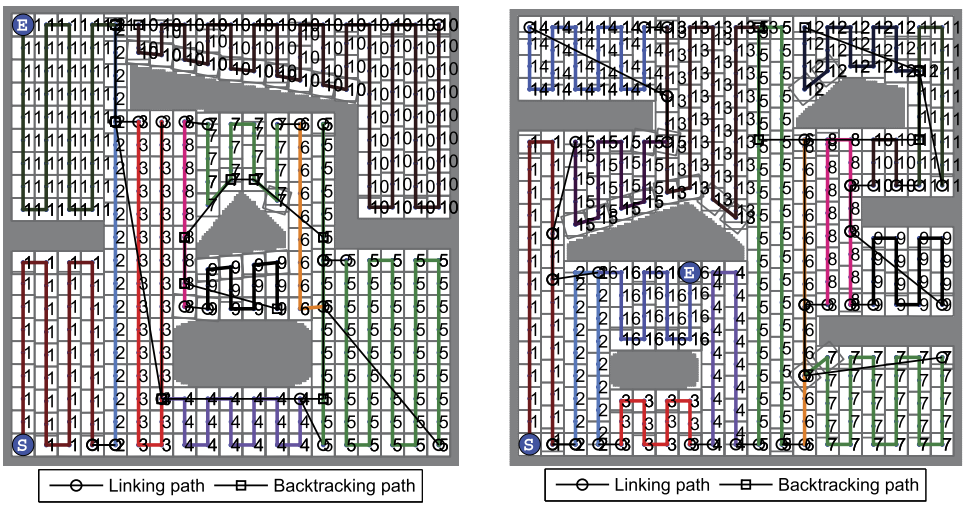
Fifteen workspaces are designed to implement the simulations as follows. The first four workspaces are the ones in Figs. 6(a), 6(b), 7, and 8, respectively. The next six workspaces and the last five workspaces are binary images of 300×300 and 400×400 pixels, respectively. Each workspace contains from four to six obstacles: L-shaped, U-shaped, triangular-shaped, and ellipse-shaped obstacles; and one or two bar-shaped obstacles for workspaces consisting of five or six obstacles. It should be remarked that except for the first four workspaces, others are generated by putting the obstacles randomly in terms of position and angle. The area of each obstacle is 4 % to 8 % of the workspace's area. The robot is modeled by a circle with a radius of $r = 5$ pixels.

The steps to implement BCD are shown as Algorithm 1 in Sect. 2.1. For instance, the workspace in Fig. 6(a) is decomposed into cells as in Fig. 9(a), and then the adjacent graph is constructed as in Fig. 9(b), where the vertex v_i represents the cell c_i $(i = 1, 2, \ldots, 11)$ of the decomposition. The walk is $\mathcal{V} = [v_1, v_3, v_4, v_7, v_{11}, v_{10}, v_9, v_6, v_8, v_6, v_9, v_{10}, v_{11}, v_7, v_4, v_3, v_5, v_3, v_2]$, which is illustrated in Fig. 9(b). The coverage path obtained from walk $\mathcal{V}$ is shown in Fig. 10(a). Specifically, the robot starts at position S. After covering cell c_1 (i.e., the boustrophedon motion labeled by 1), it continues to cover cell c_3 (i.e., the boustrophedon motion labeled by 2). At the end of cell c_3, the robot has to backtrack until it reaches the position where it can go to cell c_4 (i.e., the boustrophedon motion labeled by 3) and so forth. The robot stops at position E after covering the



(a) (b)

Fig. 9 (**a**) The BCD of the workspace in Fig. 6(a); (**b**) The adjacent graph describes the BCD and the walk on the graph starting at vertex v_1



(a) (b)

Fig. 10 The coverage paths are found by the BCD of the workspaces in Figs. 6(a) and 6(b)

entire workspace. The solid lines marked by circles depict the linking paths between two consecutive cells. The solid lines marked by squares depict the backtracking paths of the robot. The results of this simulation show that the coverage path length and the number of boustrophedon regions are 398.59 robot diameters and 11, respectively. Similarly, the coverage path obtained from the workspace in Fig. 6(b) is shown in Fig. 10(b). The length of this path and the number of boustrophedon regions are 381.12 robot diameters and 16, respectively.

To compare the coverage path length, the number of boustrophedon regions, and the coverage rate of our BA* with those of the BCD approach, we perform 15 pairs of simulations for BA* and BCD for each workspace, respectively. In each pair of simulation, the initial position of the robot is the same for both BA* and BCD and is placed randomly in order to generate diversity situations, which make the results more reliable under the statistical view in total of 450 simulations. Figure 11 demonstrates the comparison of the average length of the coverage paths achieved by 15 simulations with BA* and BCD for each workspace, respectively. The simulation results show that the average length of the coverage paths achieved by BA* is 8.06 % (for Workspace 3) to 14.43 % (for Workspace 1) shorter than that achieved by BCD. Meanwhile, Fig. 12 shows the number of decomposed cells in BCD and the average number of boustrophedon regions in BA*. The number of decomposed cells does not depend on the initial position of the robot in BCD but it does in BA*. Nevertheless, it can be seen in the figure that the maximum number of decomposed cells achieved by BA* is always smaller than that achieved by BCD. The average number of decomposed cells of 15 simulations for each workspace achieved by BA* is 18.89 % (for Workspace 3) to 55.83 % (for Workspace 2) smaller than that achieved by BCD. Under the statistical view, our BA* evidently dominates the BCD approach in terms of reducing the coverage path length and the number of decomposed cells.

From the simulation results of our BA* and the BCD approach, the five main observations can be summarized as follows:

(a) Our BA* is an online method for autonomous cleaning robots, whereas BCD is an offline method.

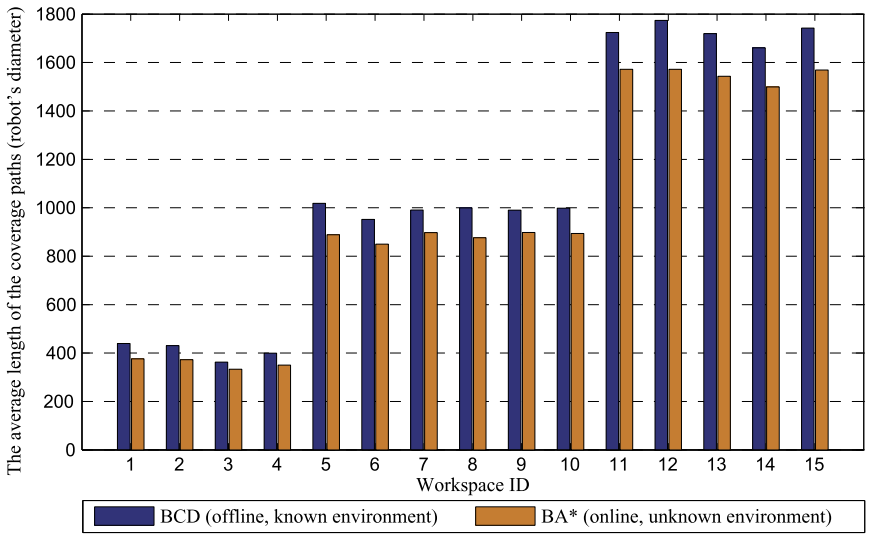


Fig. 11 The average length of the coverage paths of 15 simulations in the workspace

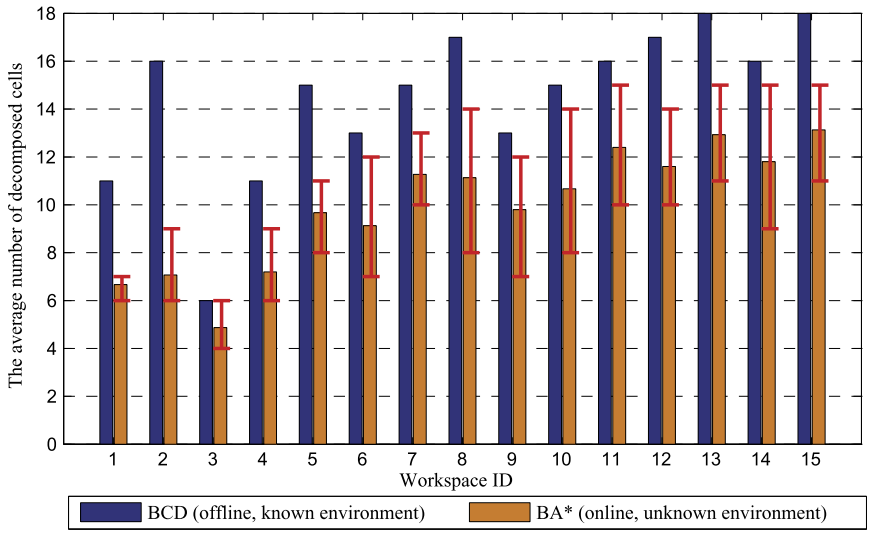


Fig. 12 The average number of decomposed cells of 15 simulations in the workspace

(b) BA* does not require a map of the workspace, whereas BCD needs to know the map before the covering process is performed.

(c) The coverage path achieved by our BA* is much shorter than that achieved by BCD because of three reasons:

(i) BA* guides the robot to the next uncovered region based on the shortest collision-free path found by A* search with smoothed path on tiling (i.e., A*SPT), whereas BCD controls the robot to the next uncovered region based on backtracking through the visited regions. The intelligent backtracking mechanism equipped for the robot is the main contribution to the significant reduction of the coverage path length.

(ii) In the BCD approach, the robot can repeat a lengthwise motion (i.e., a vertical line) to move to the first counterclockwise unvisited cell (e.g., the lengthwise motion at the end of the second boustrophedon motion or the third boustrophedon motion in Fig. 10(a), and the lengthwise motion at the end of the sixth in Fig. 10(b)). By contrast, the robot moves to the empty neighboring cell instead of repeating the lengthwise motion in the BA* approach. This advantage contributes to the decrease of the coverage path length.

(iii) BA* decomposes the accessible area of the robot into regions based on the boustrophedon motions, whereas BCD decomposes the accessible area of the robot into cells relying on the IN and OUT events determined by the changes in connectivity of the slice. Therefore, the region formed by a boustrophedon motion of BA* can contain more than one cell formed by BCD (e.g., the region formed by the first boustrophedon motion in Fig. 6(a) contains cells c_1, c_3, and c_5 in Fig. 9(a)). This means that the number of regions in BA* is

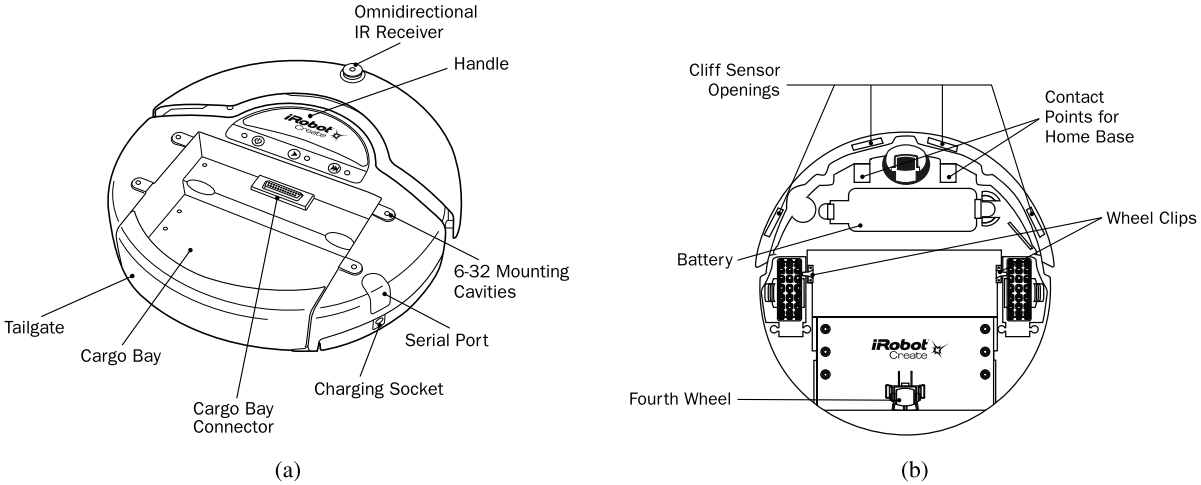


Fig. 13 (**a**) The top view of *iRobot Create*; (**b**) The bottom view of *iRobot Create* (Source: www.iRobot.com)

less than that in BCD, or in other words, the number of boustrophedon motions achieved by the proposed BA* is much smaller than that achieved by BCD. As a result, the length of the backtracking path shrinks significantly, which contributes to the shortening of the coverage path length.

(d) The coverage rate of the robot achieved by BA* and BCD is the same because both BA* and BCD use boustrophedon motions as the basic technique to cover workspaces. Furthermore, both approaches cannot cover the small portions along the obstacle boundaries because boustrophedon motions consider the heading angle of the robot in the directions of north, east, south, and west when the robot touches obstacles instead of directions that are perpendicular to the boundaries of the obstacles. However, the uncovered portions are small compared with the whole workspace.

(e) BA* does not depend on the initial position of the robot. In other words, the robot can start at any position in the workspace to cover it because BA* ends only when no backtracking point is detected.

Based on the simulations of the BA* working demonstration and the BCD comparison, the proposed BA* dominates BCD in terms of the working manner (online or offline), the necessity of prior knowledge about workspaces, the length of the coverage path, and the number of boustrophedon regions. Overall, the BA* algorithm effectively deals with the five challenges of the complete coverage problem of autonomous cleaning robots in unknown workspaces.

5 Experiments

This section presents two experiments in real environments to demonstrate and evaluate the proposed approach for autonomous cleaning robots. The implementation is applied using *iRobot Create* ♯4400 [27], which is a reprogrammable version of the Roomba robot vacuum cleaner for educational and research purposes in the robotics field. The anatomy, including the top and bottom views of *iRobot Create*, is shown in Fig. 13. Users can use their personal computers (PCs) to control the robot by reading its sensors and sending low-level numerical commands over serial ports or Bluetooth connections. In our experiments, the *iRobot Create* toolbox [10] is utilized to establish and control *iRobot Create* via a wireless connection. The *iRobot Create* toolbox replaces the native low-level numerical commands by using a set of Matlab functions, which are used to (i) create a link between a PC and *iRobot Create* using the PC's serial port or Bluetooth connections; (ii) read the bump and cliff sensors, then get the distance driven and the angle turned; and (iii) send commands to drive the robot. In our experiments, the main functions used to read the sensors and to control *iRobot Create* are listed as follows:

(a) *[serPort] = RoombaInit(portNum)* for initializing a serial port to connect the computer with the robot.

(b) *travelDist(serPort, speed, distance)* for controlling the robot movement with the specified distance and speed. If the given distance is a positive value, the robot moves forward; otherwise, the robot moves backward. The speed should be between 0.025 and 0.5 m/s.

(c) *[Distance] = DistanceSensorRoomba(serPort)* for determining the distance that the robot has traveled in meters since the function *travelDist* was last requested.

(d) *turnAngle(serPort,speed,angle)* for turning the robot some angle in degrees and some speed in rad/s.

(e) *[Angle] = AngleSensorRoomba(serPort)* for determining the angle that the robot has turned in radians since the function *turnAngle* was last requested.

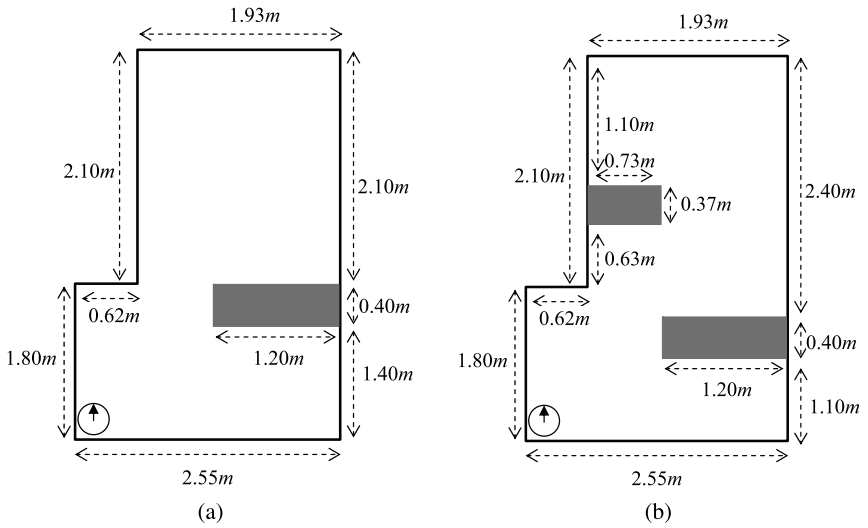


Fig. 14 The workspace and its parameters for the first (**a**) and the second experiment (**b**)

(f) *SetDriveWheelsCreate(serPort, rightWheel, leftWheel)* for moving the robot with the velocities of the right and left wheels. If $rightWheel = 0$ and $leftWheel = 0$, then the robot stops.

(g) *[BumpRight, BumpLeft, WheDropRight, WheDropLeft, WheDropCaster, BumpFront] = BumpsWheelDropsSensorsRoomba(serPort)* for determining the states of the bump and wheel drop sensors.

To implement BA* in *iRobot Create*, the robot is connected to a laptop via the Bluetooth connection, and BA* is developed on the Matlab platform to control it. All coded programs are implemented, stored, and executed on the laptop, and not on the *iRobot Create*. In other words, the laptop functions as the processing unit of *iRobot Create*. To avoid a long observation time and the difficulties of presentation when the robot performs too many boustrophedon motions, the two workspaces in these experiments are not as complicated as those described in the simulations. However, the real workspaces designed for these experiments still adequately highlight the characteristics of our BA*: the combination of boustrophedon motions and the A* search with smoothed path on tiling. The values of the basic parameters are set as follows: the robot velocity is 0.15 m/s and the distance of each robot movement is 0.3 m (i.e., equal to the diameter of *iRobot Create*). Each experiment has an L-shaped workspace, whose size details are shown in Figs. 14(a) and 14(b). The experiments are recorded by a digital camera at 24 frames/second and then saved to video files.[2] To obtain the trajectory of *iRobot Create* in the video file, we implement an algorithm by subtracting frames (the next frame is selected at an interval of 60 frames from the current frame) from the background frame to extract the robot's positions, and adding the extracted positions to the background.

The result of the first experiment is shown in Fig. 15, where the dashed arrows denote the paths obtained by the boustrophedon motions, the solid arrows describe the path obtained by A*SPT, and the circle and the bold solid arrow describe the current position and heading direction of *iRobot Create*, respectively. Initially, *iRobot Create* is placed at the South-West corner of the workspace as shown in Fig. 15(a). Then, *iRobot Create* performs the first boustrophedon motion to cover an accessible region of the workspace until a critical point is detected as in Fig. 15(b). Next, it changes direction and follows the path found by A*SPT as in Fig. 15(c), to the best backtracking point. Finally, the robot covers the remaining region by the second boustrophedon motion as shown in Fig. 15(d). The experimental time to cover the workspace is 4.32 min.

Figure 16 shows the result of the second experiment. In this experiment, *iRobot Create* performs three boustrophedon motions to cover the entire workspace. After finishing the first boustrophedon motion as in Fig. 16(b), *iRobot Create* changes direction and follows the path obtained by A*SPT to the best backtracking point as shown in Fig. 16(c). Then it performs the second boustrophedon motion as shown in Fig. 16(d). Similarly, after finishing the second boustrophedon motion, *iRobot Create* moves to the best backtracking point as shown in Fig. 16(e) and covers the final accessible region as shown in Fig. 16(f). The experimental time to cover the workspace is 4.35 min.

Although *iRobot Create* is equipped with only three bump sensors on the front, left, and right sides and without other sensors to determine its location in the workspace, the experiments show that the three touch sensors are enough for the robot to work properly. In BA*, a blocked cell means it can be either of an obstacle or visited. Therefore, the robot checks a blocked position not only by using the sensors,

[2]The experimental video files are available at the URL: http://163.180.116.135:8080/.

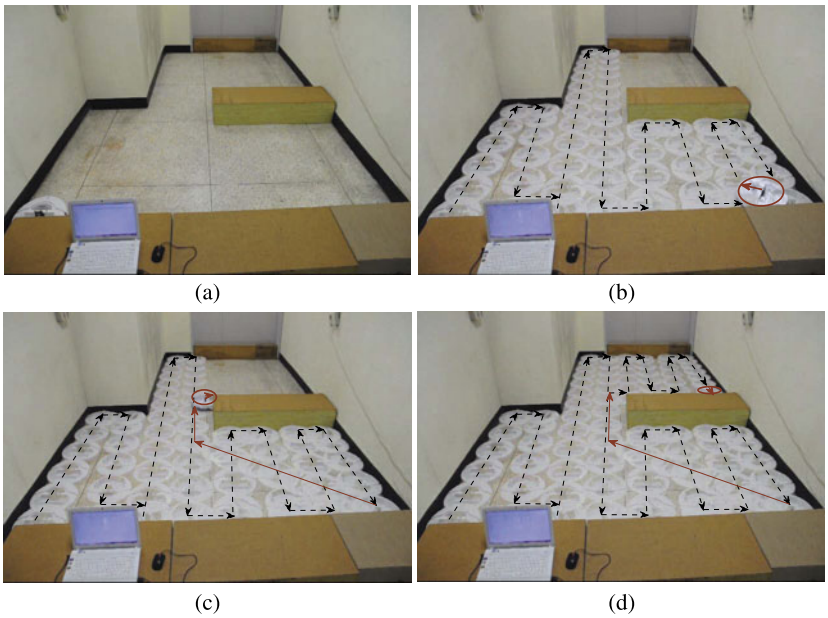


Fig. 15 The trajectory of *iRobot Create* in the first experiment

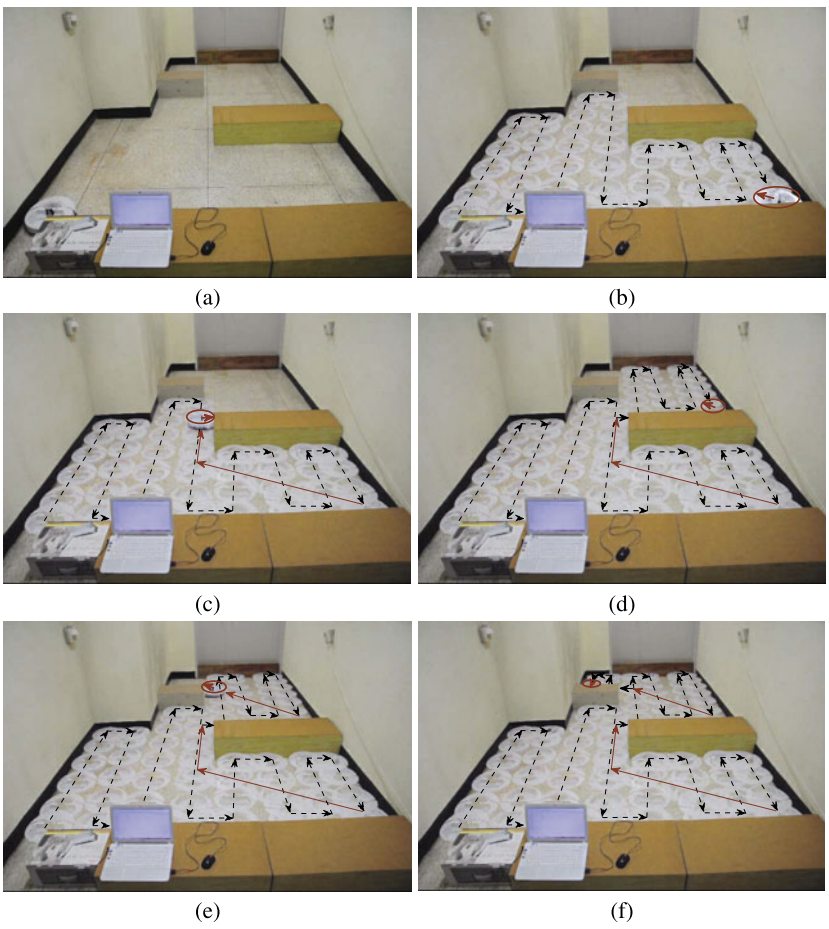


Fig. 16 The trajectory of *iRobot Create* in the second experiment

but also by the knowledge accumulated so far. When the robot enters a tile, the recently visited neighbor tile is certainly blocked. Based on its memory and touch sensors, the robot can determine the rest of the three neighbors of the main movement directions of north, east, south, and west. If these three neighbors are also blocked, then the current tile is detected as the critical point. As mentioned before, the robot uses the accumulated knowledge and creates the backtracking list at the critical point. Therefore, when examining whether a noncritical visited tile to be a backtracking point, it has enough information of all the neighbors around the tile. Although we already assume that the cleaning robots need to be equipped with necessary sensors to detect all obstacles around its position, the experiments show that *iRobot Create* can fulfill the complete coverage mission with only three touch sensors without hindering the performance of BA*.

6 Conclusions

In this paper, we present an online complete coverage algorithm for autonomous cleaning robots in unknown workspaces based on the boustrophedon motions and the A* search with smoothed path on tiling, called BA*. In this algorithm, the robot covers an unvisited region by using a single boustrophedon motion until it reaches a critical point. To continue covering the next unvisited region, the robot detects backtracking points based on its accumulated knowledge, determines the best backtracking point as the starting point of the next boustrophedon motion by applying a greedy strategy, and applies an intelligent backtracking mechanism based on the proposed A* search with smoothed path on tiling so as to reach the starting point with the shortest collision-free path. The robot achieves complete coverage when no backtracking point is detected. Computer simulations show that our proposed BA* works properly in unknown workspaces with arbitrarily shaped obstacles. It is superior to the BCD approach [4, 6], an exact cellular decomposition approach that can yield the shortest complete coverage path, in terms of the working manner (online/offline), the length of coverage path, and the number of boustrophedon motions. In addition, the experiments with *iRobot Create* in real environments prove that even when *iRobot Create* is equipped with only a few bump sensors and no sensors for the location determination, our BA* can enable the robot to store and reach the desired backtracking points correctly with exact navigation, as well as fulfill the complete coverage mission online. In summary, both analyses and practical experiments prove that our BA* algorithm is one of the best approaches for cleaning robots to deal with the coverage problem in unknown workspaces.

Acknowledgements The authors are grateful to the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0012609) for its tremendous support to this work's completion.

References

1. Acar EU, Choset H, Rizzi AA, Atkar PN, Hull D (2002) Morse decompositions for coverage tasks. Int J Robot Res 21(4):331–344
2. Botea A, Müller M, Schaeffer J (2004) Near optimal hierarchical path-finding. J Game Dev 1(1):7–28
3. Chibin Z, Xingsong W, Yong D (2008) Complete coverage path planning based on ant colony algorithm. In: Proceedings of the 15th international conference on mechatronics and machine vision in practice, Auckland, New-Zealand, pp 357–361
4. Choset H (2000) Coverage of known spaces: the boustrophedon cellular decomposition. Auton Robots 9(1):247–253
5. Choset H (2001) Coverage for robotics—a survey of recent results. Ann Math Artif Intell 31(1–4):113–126
6. Choset H, Pignon P (1997) Coverage path planning: the boustrophedon cellular decomposition. In: Proceedings of the international conference on field and service robotics, Canberra, Australia
7. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1(1):269–271
8. Dlouhy M, Brabec F, Svestka P (2000) A genetic approach to the cleaning path planning problem. In: Proceedings of the 16th European workshop on computational geometry, Eilat, Israel
9. Dudek G, Jenkin M (2010) Computational principles of mobile robotics, 2nd edn. Cambridge Univerity Press, Cambridge
10. Esposito JM, Barton O, Koehler J, Lim D (2011) Matlab toolbox for the create robot. www.usna.edu/Users/weapsys/esposito/roomba.matlab/
11. Gabriely Y, Rimon E (2001) Spanning-tree based coverage of continuous areas by a mobile robot. Ann Math Artif Intell 31(4):77–98
12. Gabriely Y, Rimon E (2002) Spiral-STC: an on-line coverage algorithm of grid environments by a mobile robot. In: Proceedings of the IEEE international conference on robotics and automation, Washington, DC, USA, pp 954–960
13. González E, Aristizábal PT, Alarcón MA (2002) Backtracking spiral algorithm: a mobile robot region filling strategy. In: Proceeding of the 2002 international symposium on robotics and automation, Toluca, Mexico, pp 261–266
14. González E, Álvarez O, Díaz Y, Parra C, Bustacara C (2005) BSA: a complete coverage algorithm. In: Proceedings of the IEEE international conference on robotics and automation, Barcelona, Spain, pp 2040–2044
15. Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107
16. Koenig S, Liu Y (2001) Terrain coverage with ant robots: a simulation study. In: Proceedings of the international conference on autonomous agents, Montreal, Quebec, Canada, pp 600–607
17. Korf RE, Reid M, Edelkamp S (2001) Time complexity of iterative-deepening-A*. Artif Intell 129(1–2):199–218
18. Latombe JC (1991) Robot motion planning. Kluwer Academic, Amsterdam
19. Luo C, Yang SX (2008) A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments. IEEE Trans Neural Netw 19(1):1279–1298

20. Mannadiar R, Rekleitis I (2010) Optimal coverage of a known arbitrary environment. In: Proceedings of the IEEE international conference on robotics and automation, Anchorage, Alaska, USA, pp 5525–5530
21. Mendonça M, de Arruda LVR, Jr FN (2012) Autonomous navigation system using event driven-fuzzy cognitive maps. Appl Intell 37(1):175–188
22. Nash A, Daniel K, Koenig S, Felner A (2007) Theta*: any-angle path planning on grids. In: Proceedings of the AAAI conference on artificial intelligence, Vancouver, Canada, pp 1177–1183
23. Oh JS, Choi YH, Park JB, Zheng YF (2004) Complete coverage navigation of cleaning robots using triangular-cell-based map. IEEE Trans Ind Electron 51(3):718–726
24. Palacín J, Palleja T, Valgañón I, Pernia R, Roca J (2005) Measuring coverage performances of a floor cleaning mobile robot using a vision system. In: Proceedings of the IEEE international conference on robotics and automation, Barcelona, Spain, pp 4236–4241
25. Palleja T, Tresanchez M, Teixido M, Palacin J (2010) Modeling floor-cleaning coverage performances of some domestic mobile robots in a reduced scenario. Robot Auton Syst 58(1):37–45
26. Russel SJ, Norvig P (2003) In: Artificial intelligence a modern approach. Pearson Education, Upper Saddle River
27. The iRobot Create Team: iRobot Create owner's guide (2006). www.irobot.com/hrd_right_rail/create_rr/create_fam/createFam_rr_manuals.html
28. Wong S (2006) Qualitative topological coverage of unknown environments by mobile robots. PhD dissertation, The University of Auckland, New Zealand
29. Yang SX, Luo C (2004) A neural network approach to complete coverage path planning. IEEE Trans Syst Man Cybern, Part B, Cybern 34(1):718–724
30. Yap P (2002) Grid-based path-finding. In: Lecture notes in artificial intelligence, vol 2338. Springer, Berlin, pp 44–55

Hoang Huu Viet received the B.S. degree in Mathematics from Vinh University, Nghean, Vietnam, in 1994, and the B.S. and M.S. degrees in Computer Science from Hanoi University of Technology, Hanoi, Vietnam, in 1998 and 2002, respectively. He is now working toward a Ph.D. degree at Artificial Intelligence Laboratory, Department of Computer Engineering, Kyung Hee University, Republic of Korea. His current research interests include Artificial Intelligence, Reinforcement Learning, and Robotics.

Viet-Hung Dang graduated as an electric-electronics engineer from HoChiMinh University of Technology in 2003, Vietnam. He got his M.S. from the same university in 2005. After 4 years of pursuing, he received his Ph.D. degree in Computer Science at Kyung Hee University, Korea in Feb 2012. He is now working for Research and Development Center in Duy Tan University, Vietnam. His research interests are Machine Learning, Artificial Intelligence, Distributed Computing and problems on Localization and Navigation.

Md Nasir Uddin Laskar received his B.S. degree in Computer Science and Engineering from the University of Dhaka, Bangladesh in 2008. He is a Faculty member in the dept. of Computer Science and Engineering, University of Information Technology and Sciences (UITS), Dhaka, Bangladesh. At present, he is pursuing his MS degree in Artificial Intelligence Lab, Dept. of Computer Engineering, Kyung Hee University, South Korea. His current research interests include Mobile Robotics with special focus on Robot motion planning and EKF-SLAM, human computer interaction, and machine learning.

TaeChoong Chung received the B.S. degree in Electronic Engineering from Seoul National University, Republic of Korea, in 1980, and the M.S. and Ph.D. degrees in Computer Science from KAIST, Republic of Korea, in 1982 and 1987, respectively. Since 1988, he has been with Department of Computer Engineering, Kyung Hee University, Republic of Korea, where he is now a Professor. His research interests include Machine Learning, Meta Search, and Robotics.