ELSEVIER

Computational Geometry 24 (2003) 197–224

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Competitive on-line coverage of grid environments by a mobile robot

Yoav Gabriely, Elon Rimon *

Technion, Israel Institute of Technology, Department of Mechanical Engineering, Haifa 32000, Israel

Received 2 August 2001; received in revised form 20 May 2002; accepted 22 July 2002

Communicated by R. Klein

Abstract

We describe in this paper two on-line algorithms for covering planar areas by a square-shaped tool attached to a mobile robot. Let D be the tool size. The algorithms, called *Spanning Tree Covering* (STC) algorithms, incrementally subdivide the planar area into a grid of D-size cells, while following a spanning tree of a grid graph whose nodes are $2D$-size cells. The two STC algorithms cover general planar grids. The first, *Spiral-STC*, employs uniform weights on the grid-graph edges and generates spiral-like covering patterns. The second, *Scan-STC*, assigns lower weights to edges aligned with a particular direction and generates scan-like covering patterns along this direction. Both algorithms cover any planar grid using a path whose length is at most $(n+m)D$, where n is the total number of D-size cells and $m \leqslant n$ is the number of *boundary cells*, defined as cells that share at least one point with the grid boundary. We also demonstrate that any on-line coverage algorithm generates a covering path whose length is at least $(2-\varepsilon)l_{\text{opt}}$ in worst case, where l_{opt} is the length of the optimal off-line covering path. Since $(n+m)D \leqslant 2l_{\text{opt}}$, the bound is tight and the STC algorithms are worst-case optimal. Moreover, in practical environments $m \ll n$, and the STC algorithms generate close-to-optimal covering paths in such environments.

Keywords: Mobile robot covering; Competitive covering; Sensor based covering; On-line spanning tree construction

1. Introduction

In the *mobile robot covering problem*, a tool of a specific planar shape is attached to a mobile robot. Given a continuous planar work-area bounded by obstacles, the mobile robot has to move the tool along a path such that every point of the work-area is eventually covered by the tool. The definition of the problem

* Corresponding author.
E-mail addresses: meeryg@tx.technion.ac.il (Y. Gabriely), elon@robby.technion.ac.il (E. Rimon).

0925-7721/02/$ – see front matter
PII: S0925-7721(02)00110-4

allows for both closed and non-closed covering paths. In the on-line version of the problem the robot has no a priori information about the environment. Rather, the robot must collect information about obstacles in the environment during the coverage process. Note that on-line coverage is *distinct* from other on-line mobile robot tasks, such as map building [5,21] and vehicle routing [4]. The optimal off-line covering problem can be formulated as a generalization of the Traveling Salesperson Problem (TSP) to continuous domains, and thus is NP-hard [1]. Since the off-line covering problem is NP-hard, we seek competitive polynomial-time algorithms for the on-line version of the problem.[1] The mobile robot covering problem has several important applications, such as floor cleaning and coating [8], hazardous waste cleaning [11], and field demining [19].

Let us mention several relevant papers, focusing on aspects related to this paper. The mobile robot covering problem has been studied by Arkin et al. [1] and Ntafos [20]. Much like our approach, they impose a tool-based grid approximation over the continuous work-area. Ntafos considers only simple grids with no internal holes, and his algorithm covers the grid within 33% of its optimal covering path. Arkin et al. consider grids which do not possess any local cut nodes (i.e. nodes whose removal would locally disconnect the graph). Their algorithm covers such grids within 32.5% of their optimal covering path. The algorithms of Ntafos and Arkin run in $O(n)$ time, where n is the total number of grid cells. Two other algorithms that can cover general planar grids are by Grigni et al. [10] and Mitchell [18]. Both algorithms cover general planar grids within ε of their optimal covering path in $n^{O(1/\varepsilon)}$ time. A randomized algorithm by Arora [2] covers general planar grids within ε of their optimal covering path in $n(\log n)^{O(1/\varepsilon)}$ expected time. The non-random version of Arora's algorithm runs in $n^3(\log n)^{O(1/\varepsilon)}$ time. However, all of these algorithms are *strictly off-line*. While off-line coverage is suitable for applications such as pocket machining [1,12], mobile robot covering tasks usually take place in partially or completely unknown environments. We present on-line algorithms that can be implemented on a mobile robot with sensors.

The on-line covering problem has been first considered by Kalyanasundaram and Pruhs [17] in the context of on-line TSP tours on weighted planar graphs. Assuming that the traveling agent acquires local information about the graph while executing the tour, they have shown that the problem has a competitive ratio of at most sixteen. In the special case of graphs with uniform weights such as grid graphs, the trivial DFS algorithm covers the graph with a path whose length is at most twice the length of the optimal path. Our first algorithm, *Spiral-STC*, runs on uniform weight grid graphs and has the same competitive ratio as DFS. However, *Spiral-STC* generates a covering path whose length is bounded by $(n+m)D$ (n is the total number of grid cells, $m \leqslant n$ is the number of boundary cells defined below, and D is the tool size), while DFS generates a path whose length is always $2nD$. In practice $m \ll n$ and the *Spiral-STC* algorithm significantly improves on DFS. Our second algorithm, *Scan-STC*, generates fixed-scanning covering patterns and has the same $(n+m)D$ path-length bound. *Scan-STC* additionally strives to minimize the number of path-segments orthogonal to the desired scanning direction.

With the exception of a paper by Icking et al. discussed below, all competitive on-line covering algorithms for mobile robots use environmental markers such as pebbles [6,7] or pheromone-like traces [22] to aid in the robot's coverage process. In contrast, we rely solely on the robot's on-board recourses during the coverage process. A paper by Icking et al. proposes an on-line coverage strategy, called URC, that also relies on the robot's on-board recourses [14].[2] The authors conjecture that URC generates a

[1] An algorithm is *competitive* if its solution to every problem instance is a constant times the optimal solution to the problem with full information available [15].

[2] A preliminary version of the paper first appeared in [13].

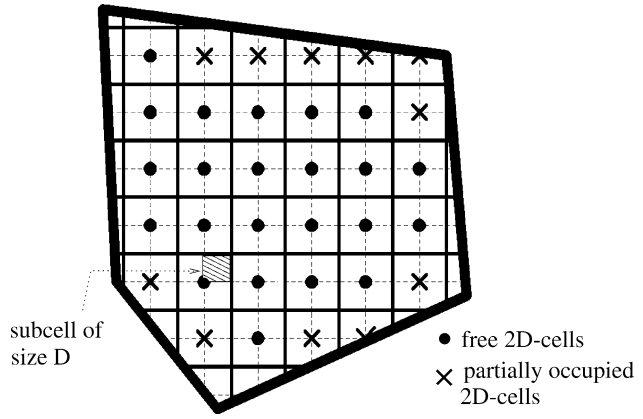


Fig. 1. Grid approximation of a given work-area.

covering path whose length in tool-size units is bounded by $n + \frac{1}{2}E + 3H - 2$, where n is the number of grid cells, E is the number of segments (i.e. edges of grid cells) on the grid boundary, and H is the number of holes in the grid. The parameter E roughly corresponds to our parameter m. Since H is usually much smaller than n and E, URC is likely to generate paths which are slightly longer than $n + \frac{1}{2}m$. Although the STC algorithms generate paths whose length is bounded by $n + m$, simulation studies indicate that on average they too generate paths slightly longer than $n + \frac{1}{2}m$. Moreover, URC generates paths that resemble the spiral paths generated by *Spiral-STC*. It is not clear whether URC can be generalized to generate scan patterns similar to the ones generated by *Scan-STC*.

In this paper we make the following assumptions. First, we assume that the tool is a square of size D. Second, the robot is allowed to move the tool only in directions orthogonal to the tool's four sides, without rotating the tool during this motion. Third, we focus on the path taken by the covering tool rather than the path taken by the robot. However, the covering tool is commonly mounted under the mobile robot's platform, and in this case the robot and covering tool follow the same path. Fourth, we approximate the continuous work-area by a discrete grid of D-size cells as shown in Fig. 1. Note that the Hamiltonian path problem on planar grids (i.e. the construction of a path that visits every node of the grid graph precisely once) is NP-complete [16]. The optimal covering of planar grids, being a more general problem, is NP-hard. Next, we assume that the environment is populated by stationary obstacles, so that the work-area grid is fixed. Finally, we assume that the robot has no a priori knowledge of the environment. Rather, the robot must use its sensors to detect obstacles while covering the work-area grid.

The paper is structured as follows. In Section 2 we present the *Spiral-STC* algorithm. This algorithm employs uniform weights on the grid-graph edges and generates spiral-like covering patterns. In Section 3 we present the *Scan-STC* algorithm. This algorithm assigns lower weights to edges aligned with a particular direction and generates scan-like covering patterns. The two STC algorithms incrementally follow a spanning tree whose nodes are $2D$-size cells. The algorithms can be interpreted as constructing on-line minimum spanning trees for coarse grid graphs whose nodes are $2D$-size cells. However, as noted in [17], the on-line generation of competitive covering paths requires more than minimum-spanning-tree considerations. In our STC algorithms, we carefully weave the traversal of the spanning-tree edges with the internal covering of individual $2D$-cells.

In Section 4 we analyze the two STC algorithms. The main result is that both algorithms cover any planar grid in $O(n)$ time using a path whose length is at most $(n + m)D$. In this bound n is the number

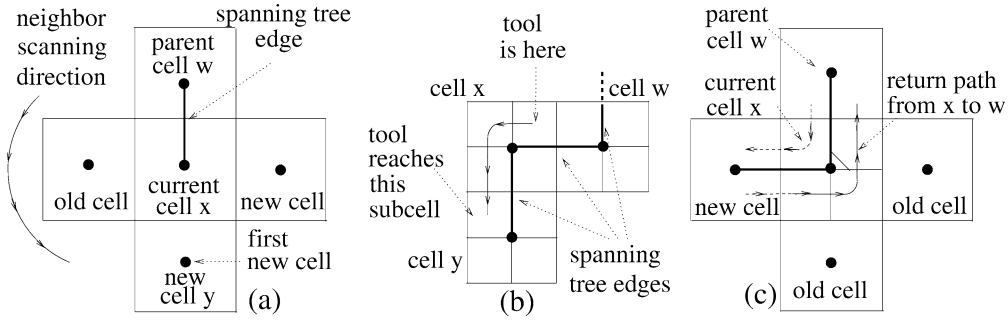


Fig. 2. (a) Counterclockwise scanning of four neighbors. (b) A move from x to a new cell y. (c) A return from x to a parent cell w.

of D-size cells and $m \leqslant n$ is the number of *boundary cells*, defined as cells that share at least one point with the grid boundary. Another result is a bound on the number of path-segments orthogonal to the scanning direction generated by *Scan-STC*. In Section 5 we demonstrate that any on-line coverage algorithm generates a covering path (either closed or non-closed) whose length is at least $(2-\varepsilon)l_{\text{opt}}$ in worst case, where l_{opt} is the length of the shortest covering path and ε is an arbitrarily small positive parameter. Interestingly, the paper by Icking et al. [14] contains a derivation of the same lower bound.[3] Since $l_{\text{opt}} \geqslant nD$, the path generated by the STC algorithms satisfies $(n+m)D \leqslant 2l_{\text{opt}}$. Consequently, the $(2-\varepsilon)l_{\text{opt}}$ bound is *tight*, and the STC algorithms are worst-case optimal. In Section 6 we run the STC algorithms on a simulated office-like environment. Finally, in the concluding section we mention several topics for further research in this area.

2. The Spiral-STC algorithm

We first describe a preliminary version of the algorithm called *2D-Spiral-STC*. The preliminary algorithm covers only $2D$-size cells which are completely free of obstacles. These cells are covered by an optimal path induced by a spanning tree whose nodes are the free $2D$-size cells. The full algorithm covers general grids by treating $2D$-size cells which are partially occupied by obstacles as special nodes of the spanning tree that incur repetitive coverage. Both versions of *Spiral-STC* use the following two sensors. The first is a *position-and-orientation sensor* that allows the robot to locally recognize the cells comprising the work-area. The second is a *range sensor* capable of identifying obstacles in the four $2D$-cells neighboring the robot's current $2D$-cell (Fig. 2(a)). For notational simplicity $2D$-size cells are simply called *cells*, while D-size cells are called *subcells*.

2.1. The preliminary 2D-Spiral-STC algorithm

The *2D-Spiral-STC* algorithm incrementally subdivides the work-area into cells of size $2D$, and discards cells which are partially occupied by obstacles. The free cells induce a graph structure whose nodes are the center points of the cells, and whose edges are the line segments connecting centers of adjacent cells (Fig. 1). The algorithm incrementally constructs a spanning tree for this graph, and uses

[3] Our derivation was obtained independently and in parallel to the one reported in [14].

the spanning tree to generate a coverage path as follows. During the spanning tree construction, the robot subdivides every cell it encounters into four identical subcells of size D, each being identical to the covering tool in size and shape. The covering tool follows a path of subcells that circumnavigates the incrementally constructed spanning tree, until the entire collection of free cells is covered. In the following description of the algorithm, a free cell is *new* when its four subcells have not yet been covered, otherwise the cell is *old*.

2D-Spiral-STC Algorithm:
Sensors: A position and orientation sensor. A 4-neighbors obstacle detection sensor.
Input: A starting cell S, but no a priori knowledge of the environment.
Recursive function: $STC1(w, x)$, where x is the current cell and w the parent cell in the spanning tree.
Initialization: Call $STC1(Null, S)$, where S is the starting cell.
$STC1(w, x)$:
1. Mark the current cell x as an *old* cell.
2. While x has a *new* obstacle-free neighboring cell:
   2.1 Scan for the first new neighbor of x in counterclockwise order, starting with the parent cell w. Call this neighbor y.
   2.2 Construct a spanning-tree edge from x to y.
   2.3 Move to a subcell of y by following the right-side of the spanning tree edges as described below.
   2.4 Execute $STC1(x, y)$.
End of while loop.
3. If $x \neq S$, move back from x to a subcell of w along the right-side of the spanning tree edges as described below.
4. Return. (End of $STC1(w, x)$.)

We now discuss several details of the algorithm. First, the robot runs a DFS algorithm during the incremental spanning tree construction. The counterclockwise scanning of neighbors specified in step 2.1 ensures that the covering tool circumnavigates the incrementally constructed spanning tree in counterclockwise order (Fig. 2(a)). The robot may equivalently choose to scan the neighboring cells in clockwise order, but then the covering tool would circumnavigate the spanning tree in clockwise order. Also note that the starting cell S has no parent cell, and in step 2.1 any neighbor of S can be designated as its parent. Second, in step 2.3 the covering tool is located in a subcell of x and has to move into a new cell y. By construction there is already a spanning-tree edge from x to y. The covering tool moves from its current subcell in x to a subcell of y by following the right-side of the spanning tree edges, measured with respect to the tool's direction of motion (Fig. 2(b)). It is shown in Lemma A.1 in Appendix A that the covering tool can always move from x to y through an empty subcell path that follows the right-side of the spanning tree. When the covering tool returns to a parent cell w in step 3, it again moves through subcells that lie on the right-side of the spanning-tree edge connecting x with w (Fig. 2(c)).

An execution example of the algorithm is illustrated in Fig. 3(a). It can be seen that the robot moves the covering tool through subcells that lie on the right-side of the spanning-tree edges, measured with respect to the tool's direction of motion. The circumnavigation of the spanning tree generates a simple closed path that brings the covering tool back to the starting cell. The figure possesses two unrealistic features, which were added for clarity. The tool size D is shown unrealistically large with respect to the work-area size, and the covering tool path is shown curved while it is rectilinear according to the algorithm.

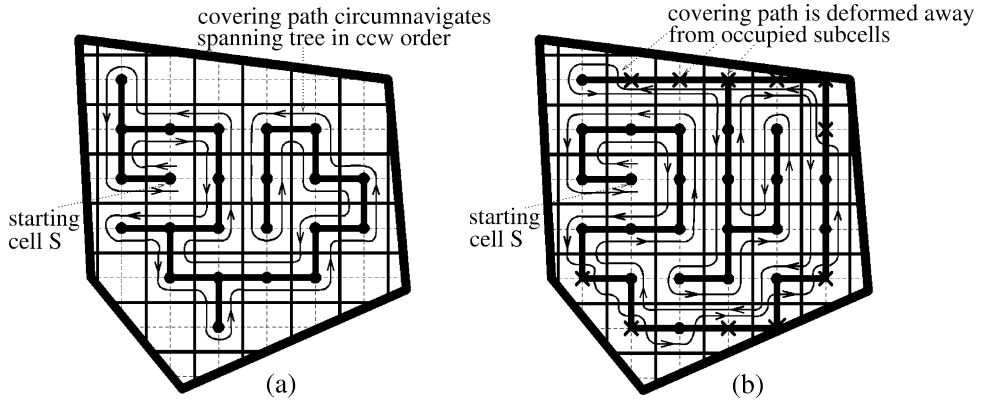


Fig. 3. An execution example of (a) *2D-Spiral-STC* and (b) the full *Spiral-STC*.

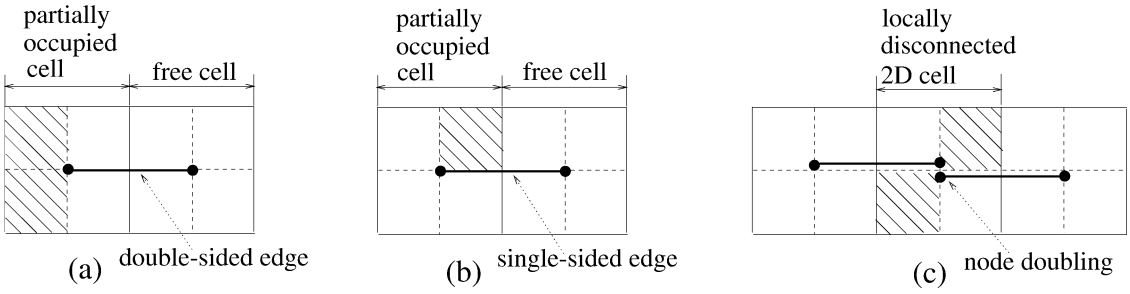


Fig. 4. (a) A double-sided and (b) a single-sided edge. (c) Node doubling at a disconnected cell.

2.2. The full Spiral-STC algorithm

The full *Spiral-STC* algorithm augments the preliminary algorithm with coverage of *partially occupied* 2*D-cells*, defined as cells that contain at least one obstacle-free D-size subcell. The full algorithm uses the following augmented graph structure. The nodes of the augmented graph are the center points of all free and partially occupied cells. The edges of this graph connect center points of adjacent cells that contain free subcells with a common boundary. This construction gives rise to two types of edges. The first type, called *double-sided edges*, possess only free subcells on both sides (Fig. 4(a)). The other type, called *single-sided edges*, possess at least one occupied subcell on either side (Fig. 4(b)). *Spiral-STC* incrementally constructs a spanning tree for the augmented grid graph, while guiding the covering tool along a subcell path that circumnavigates the spanning tree. However, the circumnavigation of a single-sided edge incurs repetitive coverage of certain subcells associated with this edge. Another new structure in the augmented grid graph involves *node doubling*. A partially occupied cell with diagonally opposite free subcells is locally disconnected. Since the covering tool can reach the free subcells from neighboring cells, we represent such a cell with two nodes, each having edges to adjacent cells from which a free subcell can be accessed (Fig. 4(c)).

The full *Spiral-STC* algorithm has the same structure as the preliminary algorithm. Hence we only describe the recursive function, called $STC2(w, x)$, which is modified in order to account for the two types of edges that occur in the augmented grid graph.

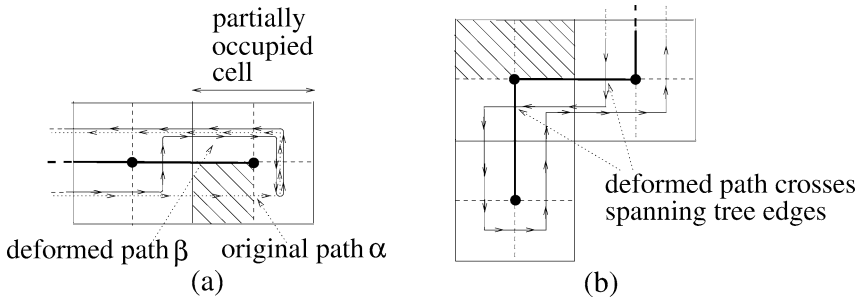


Fig. 5. (a) Path deformation along a single-sided edge. (b) Crossing of spanning-tree edges.

Full Spiral-STC Algorithm:
Initialization: Call *STC*2(*Null*, *S*), where *S* is the starting cell.
*STC*2(w, x):
1. Mark the current cell x as an *old* cell.
2. While x has a *new* free or partially occupied neighboring cell:
   2.1 Scan for the first new neighbor of x in counterclockwise order, starting with the parent cell w. Call this neighbor y.
   2.2 Construct a spanning-tree edge from x to y.
   2.3 Move to a subcell of y along the spanning tree edges, using a path determined by the type of edge from x to y as described below.
   2.4 Execute *STC*2(x, y).
End of while loop.
3. If $x \neq S$, move back from x to a subcell of w along a path determined by the type of edge from x to w as described below.
4. Return. (End of *STC*2(w, x).)

Much like the preliminary algorithm, here too the robot runs DFS during the incremental spanning tree construction. We now describe the subcell path taken by the covering tool along a single-sided edge in steps 2.3 and 3 of the algorithm. Consider for concreteness the partially occupied cell depicted in Fig. 5(a). This cell is a leaf node in the spanning tree, and the spanning-tree edge entering this cell is single-sided. Let α be the subcell path that would have been taken by the covering tool if the cell were completely free. Then the actual path taken by the covering tool, denoted β, is obtained by *deforming α away from the occupied subcell without changing the path's sense of direction, until every point of β lies at a distance of $D/2$ from the occupied subcell.* Fig. 5(a) shows the original path α as well as the deformed path β. Note that we use a maximum metric[4] in the path deformation, to ensure that β takes the covering tool only through obstacle-free subcells. Note, too, that the deformed path β crosses the spanning-tree edge. The deformed path may cross other edges emanating from a partially occupied cell, as shown in Fig. 5(b).

An execution example of the full algorithm on the environment used to execute the preliminary algorithm is illustrated in Fig. 3(b). Another execution example appears in Fig. 9. In the tight environment of Fig. 9 the central cell is covered each time the covering tool enters a new corridor. It is shown below

[4] The maximum metric is given by $d(p, q) = \max\{|p_x - q_x|, |p_y - q_y|\}$.

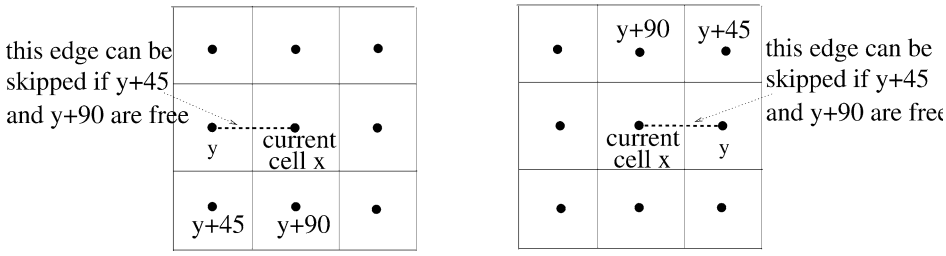


Fig. 6. The cells inspected by *2D-Scan-STC* when considering whether to skip the construction of a horizontal spanning-tree edge.

that the total number of such repetitive coverages is bounded by the number of boundary subcells in the work-area grid. Finally, simulations presented below reveal the spiral-like covering pattern typical to this algorithm.

3. The Scan-STC algorithm

Practical covering scenarios often require generation of fixed-scanning covering patterns. We now present a spanning-tree based algorithm, *Scan-STC*, that generates fixed-scanning covering patterns by considering larger neighborhoods of the grid. *Scan-STC* requires the same sensors as the previous algorithm. However, previously the range sensor was required to detect obstacles in the *four* $2D$-cells neighboring the current cell. Now it must detect obstacles in the ring of *eight* $2D$-cells surrounding the current cell. We first present a preliminary version of the algorithm called *2D-Scan-STC*, that covers only $2D$-size cells which are completely free of obstacles. Then we present the full algorithm that covers general grids of D-size cells.

3.1. The preliminary 2D-Scan-STC algorithm

The input to the algorithm is a desired scanning direction for the work-area, which we assume to be the *vertical direction*. Using the robot's orientation sensor, the algorithm aligns the incrementally constructed work-area grid with the vertical scanning direction. The algorithm skips the construction of horizontal spanning-tree edges by inspecting cells in the ring surrounding the current cell. These cells are denoted as follows. Given a neighbor cell y, the cells $y+45$ and $y+90$ are the cells that form 45° and 90° angles with the line from x to y in counterclockwise order (Fig. 6). Recall that a free $2D$-cell is *new* when its four subcells have not yet been covered, otherwise the cell is *old*.

2D-Scan-STC Algorithm:
Sensors: A position and orientation sensor. An 8-cells obstacle detection sensor.
Input: A starting cell S, but no apriori knowledge of the environment.
Recursive function: $STC3(w, x)$, where x is the current cell and w the parent cell in the spanning tree.
Initialization: Call $STC(Null, S)$, where S is the starting cell.
$STC3(w, x)$:
1. Mark the current cell x as an *old* cell.

2. While x has a *new* obstacle-free neighboring cell which has not been inspected:
   2.1 Scan in counterclockwise order for the first new neighbor of x which has not been inspected yet, starting with the parent cell w. Call this neighbor y.
   2.2 If y is a *horizontal* neighbor of x and the cells $y+45$ and $y+90$ are free: *Skip* the construction of a spanning-tree edge from x to y. Goto Step 2.
   2.3 Construct a spanning-tree edge from x to y.
   2.4 Move to a subcell of y by following the right-side of the spanning tree edges.
   2.5 Execute $STC3(x, y)$.

End of while loop.

3. If $x \neq S$, move back from x to a subcell of w along the right-side of the spanning tree edges.
4. Return. (End of $STC3(w, x)$.)

The *2D-Scan-STC* algorithm has the same general structure as the *2D-Spiral-STC* algorithm. In particular, *2D-Scan-STC* still expands new neighbors in counterclockwise order. Hence Lemma A.1 in Appendix A applies here too, and the covering tool can always move to the next new neighbor through an empty subcell path that follows the right-side of the spanning-tree edges. Consider now step 2.2, where the algorithm skips the construction of spanning-tree edges to horizontal neighbors. Let us verify that the covering tool still reaches all the free cells accessible from the starting cell. First, when the covering tool enters a new vertical column of cells, the entire column is covered since the algorithm never skips vertical grid-edges. Second, the covering tool skips the construction of a horizontal spanning-tree edge to a neighbor y *only when it knows that* y *has a neighboring free cell in its own column*, the cell $y+45$, that can be accessed from the current column at a later stage of the covering process, from the cell $y+90$. Otherwise the algorithm constructs a horizontal spanning-tree edge that takes the covering tool to a neighboring column. It follows that the spanning tree reaches all the free cells accessible from the starting cell, and the covering tool covers these cells during the spanning tree circumnavigation.

An execution example of the algorithm is illustrated in Fig. 7(a). The figure shows the spanning tree constructed on-line by the algorithm. It can be seen that the spanning-tree edges are vertically aligned, except for a single horizontal edge between every pair of adjacent columns. The algorithm in general strives to minimize the number of horizontal edges, and the number of horizontal edges generated by the algorithm is quantified below. The figure also shows the subcell path taken by the covering tool. Note

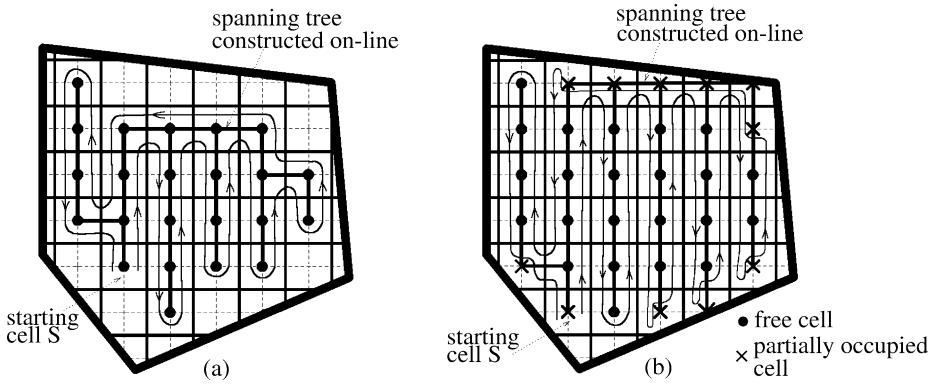


Fig. 7. An execution example of (a) *2D-Scan-STC* and (b) the full *Scan-STC*.

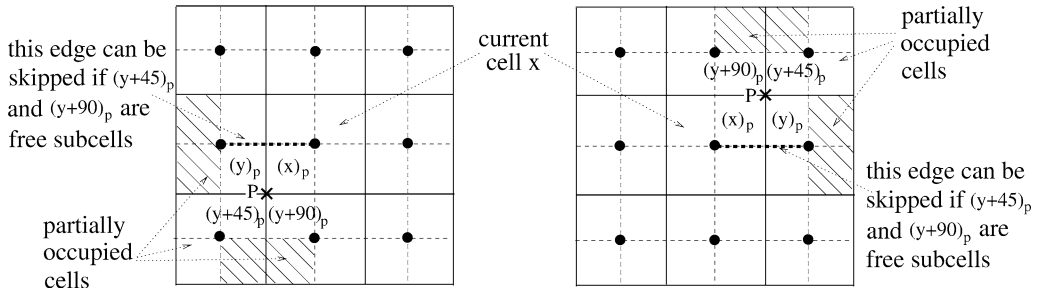


Fig. 8. The subcells inspected by the full *Scan-STC* when considering whether to skip the construction of a horizontal spanning-tree edge.

that the covering tool follows an optimal covering path. This optimality is not maintained by the full algorithm presented in the next section.

3.2. The full Scan-STC algorithm

The full *Scan-STC* algorithm employs the same augmented grid graph used by the full *Spiral-STC* algorithm. We briefly repeat here the structure of this graph. Recall that a *partially occupied cell* is a $2D$-cell that contains at least one obstacle-free D-size subcell. In the augmented grid graph, adjacent free and partially occupied cells are connected by an edge if they contain free subcells with a common boundary. This construction gives rise to double-sided edges (Fig. 4(a)) and single-sided edges (Fig. 4(b)). Last, a partially occupied cell with diagonally opposite free subcells is represented by two nodes, each having edges to adjacent cells from which a free subcell can be accessed (Fig. 4(c)).

The full *Scan-STC* algorithm incrementally constructs a spanning tree for the free and partially occupied cells. However, for practical reasons we wish to ensure that the algorithm inspects only subcells which are directly visible from the covering-tool's current location. Hence the full algorithm inspects only the following subcells. Let p be the point where the cells x, y, $y+45$ and $y+90$ meet. Then the full algorithm inspects the four subcells surrounding p. These subcells are denoted x_p, y_p, $(y+45)_p$, $(y+90)_p$ and are depicted in Fig. 8. The full algorithm has the same structure as the preliminary algorithm. Hence we only describe the recursive function which is now called $STC4(w,x)$.

Full Scan-STC Algorithm:
Initialization: Call $STC4(Null, S)$, where S is the starting cell.
$STC4(w,x)$:
1. Mark the current cell x as an *old* cell.
2. While x has a *new* free or partially occupied neighboring cell which has not been inspected:
   - 2.1 Scan in counterclockwise order for the first new neighbor of x which has not been inspected yet, starting with the parent cell w. Call this neighbor y.
   - 2.2 If y is a *horizontal* neighbor of x and the four subcells x_p, y_p, $(y+45)_p$ and $(y+90)_p$ are obstacle-free: *Skip* the construction of the spanning-tree edge from x to y. Goto Step 2.
   - 2.3 Construct a spanning-tree edge from x to y.
   - 2.4 Move to a subcell of y along the spanning tree edges, using a path determined by the type of edge from x to y as described below.

2.4 Execute $STC4(x, y)$.
End of while loop.
3. If $x \neq S$, move back from x to a subcell of w along a path determined by the type of edge from x to w as described below.
4. Return. (End of $STC4(w, x)$.)

Let us discuss several details of the full algorithm. The subcell path taken by the covering tool in steps 2.4 and 3 is the same deformed path taken by the tool in the full *Spiral-STC* algorithm. As depicted in Fig. 5, the deformed path is obtained by moving the original unobstructed path a distance of $D/2$ away from the occupied subcells, without changing the path's sense of direction. The resulting path incurs repetitive coverage which is quantified below. Next consider step 2.2, where the algorithm skips the construction of a horizontal spanning-tree edge. The rule specified in this step is the subcell-version of the corresponding rule in the preliminary algorithm. Given a horizontal neighbor y, the cell $y + 45$ is a vertical neighbor of y. The algorithm skips the construction of a spanning-tree edge from x to y when it knows that y can be accessed from the subcell $(y + 45)_p$ at a later stage of the covering process from the subcell $(y + 90)_p$. The four subcells involved in this test are visible from any subcell of the current cell x, for the following reason. If the subcell x_p is occupied by an obstacle, the test fails and there is no need to inspect the other three subcells. Otherwise x_p is free and the other three subcells are directly visible from any subcell of x. Finally, the full algorithm need only inspect the ring of subcells surrounding the current cell, rather than the ring of $2D$-cells required by the preliminary algorithm. However, the ring-of-subcells neighborhood is still larger than the neighborhood inspected by the full *Spiral-STC* algorithm.

An execution example of the full algorithm on the environment used to execute the preliminary algorithm is illustrated in Fig. 7(b). As discussed below, here too the total number of repetitive coverages is bounded by the number of boundary subcells in the work-area grid. Simulations of the full *Scan-STC* algorithm are presented in Section 5.

4. Analysis of the STC algorithms

In this section we first consider several properties of the STC algorithms, then analyze the amount of repetitive coverage generated by the algorithms, and finally characterize the scanning pattern generated by *Scan-STC*.

Lemma 4.1 (completeness). *The* Spiral-STC *and* Scan-STC *algorithms cover every free subcell accessible from the starting cell S*.

Proof. Both algorithms construct a spanning tree that reaches every free and partially occupied $2D$-size cell in the accessible work-area grid. These cells are partitioned into subcells of size D. By construction, every obstacle-free subcell touches the spanning tree either at a point (a leaf node), or along a segment (part of an edge emanating from a node). Consider now the subcell path, denoted α, generated by circumnavigating the spanning tree. (Note: α passes through empty as well as occupied subcells.) Since every free subcell touches the spanning tree, α passes through every free subcell accessible from S. Both algorithms deform the path α away from the occupied subcells, so that the deformed path passes only through free subcells. The deformed path is a closed path that passes through every free subcell which lies along the original path α. Hence every accessible free subcell is covered. □

The next lemma gives the run-time and memory requirement of the STC algorithms.

Lemma 4.2. *Let n be the total number of free subcells accessible from the starting cell S. Then* Spiral-STC *and* Scan-STC *cover these subcells in* $O(n)$ *time using* $O(n)$ *memory.*

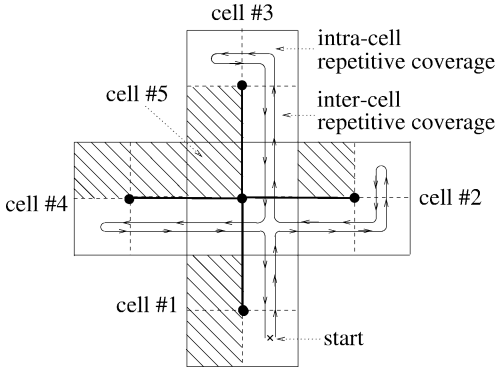
Let us sketch the derivation of these bounds. In each step the covering tool enters either a new subcell or a previously visited subcell. According to Theorem 1 below, the total number of repetitive visits is bounded by m, where $m \leqslant n$ is the number of boundary subcells. Hence there are $O(n)$ covering steps. The $O(n)$ memory requirement allows the algorithm to identify which cells of the grid have already been visited by the covering tool. This memory requirement imposes a strict limitation on the size of the work-areas that can be covered by a bounded-memory robot. In [9] we describe an STC algorithm that uses cell markers to identify previously visited subcells. This algorithm requires only $O(1)$ memory.

The following theorem establishes a bound on the length of the covering path generated by the STC algorithms. Recall that *boundary subcells* are obstacle-free subcells that share either a point or a segment with the boundary of the work-area grid.

Theorem 1. *Let n be the total number of free subcells in the accessible work-area grid. Let $m \leqslant n$ be the total number of boundary subcells. Then* Spiral-STC *and* Scan-STC *cover the work-area grid using a path of total length $l \leqslant (n+m)D$, where D is the tool size.*

Proof. We use the following terminology. An *entry edge* of a cell is a spanning-tree edge along which the covering tool enters a cell for the first time. All other spanning-tree edges that emanate from a cell are *exit edges*. Note that every exit edge is an entry edge for the other cell joined by this edge. Next, we define *repetitive coverage* of a point p as the situation where p is being covered, then exposed, and later covered again. We distinguish between two types of repetitive coverages. An *inter-cell repetitive coverage* is a repetitive coverage that occurs when the covering tool moves between subcells of different cells. An *intra-cell repetitive coverage* is a repetitive coverage that occurs when the covering tool moves between subcells of the same cell (Fig. 9).

We use the following convention for counting repetitive coverages. A single-sided edge always generates an inter-cell repetitive coverage in the cell from which it exits, but not in the cell into which



cell	inter-cell repetitive coverage	intra-cell repetitive coverage	total repetitive coverage	number of boundary subcells
1	0	1	1	2
2	1	2	3	3
3	1	2	3	3
4	1	1	2	2
5	1	2	3	3
total			12	13

Fig. 9. An execution example of the full *Spiral-STC* with a listing of the repetitive coverages in each cell. (The full *Scan-STC* generates the same path in this environment.)

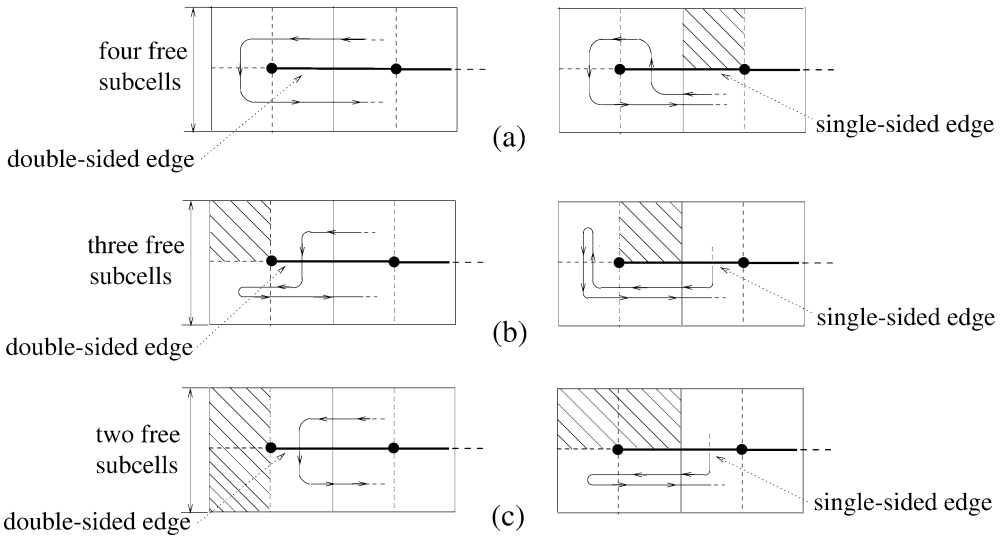


Fig. 10. A cell with (a) four free subcells, (b) three free subcells and (c) two free subcells.

it enters. However, we add this inter-cell repetitive coverage to the count at the cell for which this edge is an entry edge. The total count of repetitive coverages is not affected by this convention, since every edge is an exit edge for one cell and an entry edge for some other cell. Intra-cell repetitive coverages are counted by the cell itself. The counting convention is illustrated in Fig. 9, which includes a table that lists the number of repetitive coverages in each cell.

We now show that the total number of repetitive coverages is bounded by the number of boundary subcells. It suffices to show that the number of repetitive coverages in each cell is bounded by the number of boundary subcells in this cell. We consider four cases, classified by the number of free subcells in the cell. Each of the four cases is further classified into two sub-cases, depending whether the entry edge into the cell is single-sided or double-sided. We do not need to consider the exit edges from a cell, since a double-sided exit edge incurs no repetitive coverage, while the repetitive coverage associated with a single-sided exit edge is counted by the cell into which the edge enters.

First consider a cell with four free subcells (Fig. 10(a)).[5] A double-sided edge entering the cell incurs no repetitive coverage. If the edge entering the cell is single-sided, there is one inter-cell repetitive coverage and one intra-cell repetitive coverage, or a total of *two* repetitive coverages. The presence of a single-sided entry edge implies that the adjacent cell has an occupied subcell. Since this subcell borders our cell, our cell contains *two* boundary subcells—one that shares a boundary segment and one that shares a point with the occupied subcell.

Next consider a cell with three free subcells (Fig. 10(b)). Here, too, a double-sided edge entering the cell incurs no inter-cell repetitive coverage. However, the presence of an occupied subcell in our cell generates one intra-cell repetitive coverage. If the edge entering the cell is single-sided, there is one inter-cell repetitive coverage and at most two intra-cell repetitive coverages, or a total of *three* repetitive

[5] Note that Fig. 10 depicts only specific examples of the eight possible cases.

coverages. The number of boundary subcells in such a cell is *three*, since all three subcells touch the occupied subcell.

Now consider a cell with two free subcells (Fig. 10(c)). If the edge entering the cell is double-sided, this edge incurs no repetitive coverage. If the edge entering the cell is single-sided, there is one inter-cell repetitive coverage and at most one intra-cell repetitive coverage, or a total of *two* repetitive coverages. The number of boundary subcells is also *two*, since the two free subcells touch the occupied subcells. Last consider a cell with a single free subcell. The entry edge into this cell is necessarily single-sided. Hence there is one inter-cell repetitive coverage. Since this type of cell incurs no intra-cell repetitive coverage, there is *one* repetitive coverage, and this is also the number of boundary subcells.

To summarize, the total number of repetitive coverages is at most m, and the length of the covering path consequently satisfies $l \leqslant (n+m)D$. □

The $m+n$ bound does not reflect an important property of the STC algorithms, that their repetitive coverage is localized around partially occupied cells. The following corollary provides a tighter bound that emphasizes this property. Consider the collection of occupied subcells contained in the partially occupied cells. Let k be the total number of free subcells that share either a point or a segment with these occupied subcells. Note that k is a property of the grid imposed by the robot, and is independent of the STC covering strategy. By definition $k \leqslant m$, and the length of the covering path is bounded in terms of k as follows.

Corollary 4.3. *Let n be the total number of free subcells in the accessible work-area grid. Let $k \leqslant m$ be the total number of free subcells that touch occupied subcells contained in partially occupied cells. Then* Spiral-STC *and* Scan-STC *cover the work-area grid using a path of total length $l \leqslant (n+k)D$, where D is the tool size.*

Proof. In the proof of Theorem 1, the number of repetitive coverages is counted per cell. According to the counting convention used in the proof, a free cell incurs one repetitive coverage when its entry edge is single-sided. In this case the neighbor cell has an occupied subcell touching two subcells of the free cell. All other cases considered in the proof of the theorem count repetitive coverages in partially occupied cells. In all of these cases the number of repetitive coverages never exceeds the number of free subcells in the cell. The total number of free subcells contained in partially occupied cells, together with free subcells in free cells that touch occupied subcells contained in partially occupied cells, is bounded by the number k appearing in the corollary. □

We conclude with two comments regarding the STC algorithms. First consider the relation of the algorithms to DFS. The path length of the STC algorithms satisfies $l \leqslant (n+m)D \leqslant 2nD$, since $m \leqslant n$. But DFS can also be executed on-line on the work-area grid, and the length of its covering path would be $2nD$. However, *DFS always generates a covering path of length $2nD$, no matter what is the particular geometry of the work-area grid.* In contrast, the STC algorithms can be interpreted as running DFS on a coarser grid of 2D-cells, allowing them to adapt the internal coverage of cells according to the cells' particular geometry. In practical environments $m \ll n$, and in such environments the STC algorithms generate paths whose length is close to nD. Next consider the competitive ratio of the STC algorithms, defined as l/l_{opt}, where l_{opt} is the length of the optimal coverage path. The total area of the work-area grid is $A = nD^2$, and the area occupied by the boundary subcells is $\partial A = mD^2$. The optimal

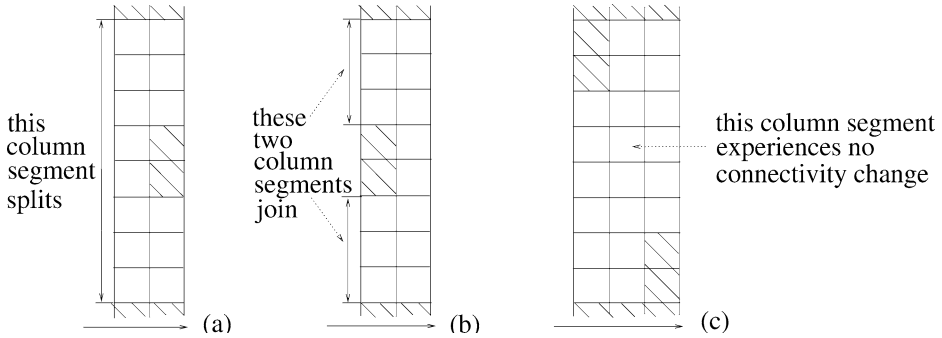


Fig. 11. (a)–(b) Split and join connectivity changes in a column-segment. (c) A column-segment can deform without experiencing any connectivity change.

coverage path satisfies $l_{\text{opt}} \geqslant A/D$, and the length of the coverage path generated by the STC algorithms satisfies $l \leqslant (n+m)D = A/D + \partial A/D$. It follows that the competitive ratio satisfies the inequality $l/l_{\text{opt}} \leqslant 1 + \partial A/A$. In practice $\partial A \ll A$, and the STC algorithms achieve a close-to-optimal coverage of practical work-area grids.

The last result of this section concerns the covering pattern generated by *Scan-STC*. The objective of *Scan-STC* is to cover the work-area grid with a vertical-scanning pattern that minimizes the number of horizontal path segments. We present an upper bound on the number of horizontal path segments using the following terminology. A *column segment* is a contiguous sequence of free cells arranged vertically and bounded by the boundaries of the work-area grid. Consider now the possible connectivity changes that can occur in a column-segment as we move horizontally, say from left to right. A column-segment can split into two disjoint column-segments (Fig. 11(a)) or two column-segments can join into a single connected column-segment (Fig. 11(b)). In both cases we say that the column-segment experiences a *connectivity change*. For clarity, Fig. 11(c) shows a case where a column-segment experiences no connectivity change. The following proposition provides an upper bound on the number of horizontal path segments in terms of the number of horizontal spanning-tree edges constructed by the algorithm. The bound is stated for the simpler *2D-Scan-STC* algorithm.

Proposition 4.4. *The number of horizontal spanning-tree edges constructed by* 2D-Scan-STC*, denoted* h*, is bounded by*

$$h \leqslant h^* + p + 1, \tag{1}$$

where h^* *is the off-line optimal number of horizontal spanning-tree edges,*[6] *and* p *is the number of connectivity changes in the column-segments of the grid.*

Let us discuss some aspects of the proposition, leaving its proof to Appendix A. First, the total length of the horizontal segments along the covering path is $2Dh$, since the covering tool moves on both sides of each spanning-tree edge. Second, the unity term in (1) corresponds to a wasteful horizontal spanning-tree edge that can occur in the column-segment containing the starting cell (Fig. 16(e)). This term can be eliminated if the covering tool always starts on the grid boundary. Last, it seems that any on-line covering

[6] h^* is equal to the number of column-segments in the grid minus one.

algorithm must "pay" for the connectivity changes. For instance, let a column-segment C experience a join connectivity change. Then C can be accessed from two column-segments on its left side. Since only local sensory information is available to the robot, it may lead the covering tool into C from both routes. Not knowing that it has arrived to the same column-segment, the robot may independently cover several portions of C, generating a wasteful covering pattern.

5. A universal on-line coverage bound

In this section we derive a lower bound on the path-length of any on-line covering algorithm for planar environments. First let us demonstrate the bound in a simple corridor whose width is identical to the tool size D. Suppose the robot executing the algorithm has a range sensor with a detection range of $1-\delta$, where δ is a small positive parameter. We initially place the robot in an infinite corridor and wait until it covers a length-L portion of the corridor, where $L \gg 1$. At that instant we truncate the corridor at a unit distance from both sides of the length-L portion, as shown in Fig. 12. Since the robot has no way of knowing about this change in the environment, its on-line behavior would be identical if it starts at the same point in the truncated corridor. The length of the on-line covering path is $l \geqslant 2L+3$, since the covering tool must visit both ends of the corridor before coverage is complete. The shortest off-line covering path, achieved by sweeping the corridor from end to end, has length $l_{\text{opt}} = L+2$. Thus $l/l_{\text{opt}} \geqslant 2-\varepsilon$, and this is the lower bound we wish to establish. However, *the corridor environment does not support the bound for covering tours where the robot must return to its starting point.* Another caveat with the corridor is that the shortest off-line covering path does not start at the same point as the on-line algorithm.

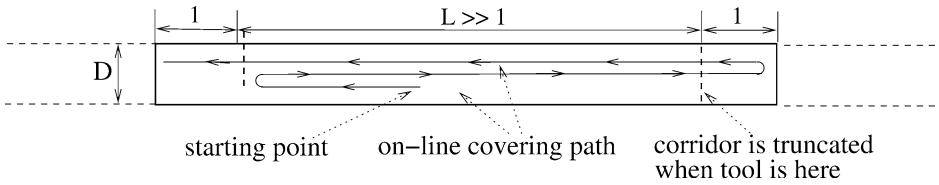


Fig. 12. A corridor environment with an on-line covering path depicted.

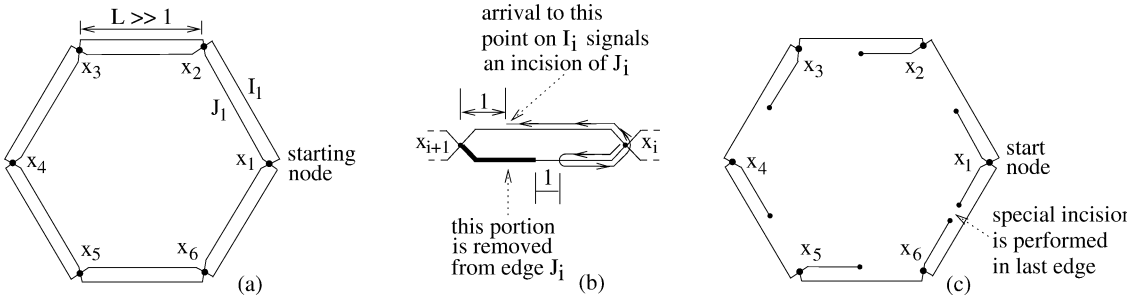


Fig. 13. (a) A double-ring environment. (b) The local incision performed on the edge J_i. (c) The environment resulting from k incisions.

The more sophisticated environment presented in Fig. 13 is circular, and the lower bound is established in this environment both for covering paths and tours. Moreover, the circular environment gives no advantage to any particular initial point. Hence the off-line covering path might as well start at the same point as the on-line algorithm.

Theorem 2. *Every on-line covering algorithm of a bounded planar environment generates in worst case a covering path or tour whose length is at least* $(2-\varepsilon)l_{\text{opt}}$*, where* l_{opt} *is the length of the shortest off-line covering path, and* ε *is an arbitrarily small positive parameter.*

Moreover, in planar grid environments the bound is ε*-tight, since there exist on-line algorithms* (*such as the STC algorithms and DFS*) *that generate covering paths whose length is at most* $2l_{\text{opt}}$.

The on-line coverage bound is valid under the following general conditions. The planar environment may be continuous or a discretized grid. The covering tool may have any convex shape with non-empty interior, and it may move along arbitrary paths. Finally, the robot's range sensor may have an arbitrary finite detection range.

Proof. In the proof we identify the robot with the covering tool. Furthermore, we consider for simplicity a point robot covering a graph. We loose no generality by this simplification, as a thickening of the graph to rectangular corridors of width D and lengths that are integer multiples of D would imply the same bound in grid as well as continuous planar environments. The robot is able to detect nodes of the graph by "looking" along edges of the graph, and it has a detection range of $1-\delta$ where δ is a small positive parameter. Given an on-line coverage algorithm, we execute the algorithm in the *double-ring environment* depicted in Fig. 13(a). The ring consists of $2k$ edges arranged in parallel pairs, where $k \gg 1$. Each edge has length L such that $L \gg 1$. Adjacent edges are connected by an $\times$-shaped node that allows the robot to freely move between the four edges meeting at the node. During the execution of the algorithm, we perform k *local incisions* that remove portions of the edges. However, the removed portions always lie beyond the robot's detection range.

The local incision procedure is as follows. Let $x_1, \ldots, x_k$ be the nodes of the ring, arranged in counterclockwise order. Let I_i and J_i be the parallel edges connecting x_i and x_{i+1}, where indexing is modulo k. Let the starting point be at the node x_1. The robot initially covers portions of the four edges incident to x_1. The first incision occurs at the instant when the robot reaches within a distance of *one unit* from one of the nodes adjacent to x_1. Suppose this event occurs first on the edge I_1 which leads to x_2. We remove from the parallel edge, J_1, the portion that lies at a distance of *one unit* beyond the covered portion of J_1 (Fig. 13(b)). There is no problem with this incision, since $L \gg 1$ and the covered portion of J_1 is more than one unit away from x_2. Note that the portion removed from J_1 has not been detected by the robot. The robot next continues with its covering path, and at a certain instant it would reach a new node. We may assume that the robot visits the nodes in counterclockwise order, until it reaches the last node x_k. (The robot may visit the nodes in any other order, but the covering path would only be longer.) The same local incision is repeated in every edge J_i for $i = 1, \ldots, k-1$, while the edges I_i are left intact. The last edges I_k and J_k have been already partially covered by the robot, since these edges are incident to the starting node x_1. In these edges only, we wait until the robot approaches within *one unit* distance from the covered portion of either I_k or J_k. Suppose this event occurs first in I_k. Then we remove the middle portion of J_k which lies *one unit* away from the covered portions of J_k. The environment resulting from the k incisions is depicted in Fig. 13(c).

The modified environment consists of a loop of k edges $I_1, \ldots, I_k$ each having a length L. Every node x_i also has a truncated edge emanating from it (x_1 has two truncated edges). For purpose of path-length computation, let H_i be the length of the portion of J_i that has been covered up to the instant of incision in J_i. (The piece of each edge J_i has a total length of $H_i + 1$.) Also, let

$$H = \sum_{i=1}^{k} H_i.$$

A key property of the modified environment is that *the on-line algorithm has covered a length-H_i portion of the piece of J_i emanating from x_i when it reaches within one-unit distance from x_{i+1} along the edge I_i*. In the best scenario the robot first completes the coverage of the edges $I_1, \ldots, I_k$, then performs a second loop while pausing at each node to cover the truncated edge (two truncated edges in the case of x_1) emanating from it. The total length of the motion along the edges I_i is $2kL$. During the first circumnavigation the robot covers a length-H_i portion of the piece of J_i emanating from x_i, then retreats through x_i into the parallel edge I_i. Hence the total length of the motion along the pieces of the edges J_i is $2H$ during the first round, and an additional $2(H + k)$ during the second round. The total length of the on-line coverage path is therefore $l \geqslant 2kL + 4H + 2k$. Note that the assumption $k \gg 1$ allows us to ignore the small difference in the total length of a covering path and a covering tour for this particular environment.

Let us now compute the off-line optimal covering path. Starting from x_1, the optimal covering path circumnavigates the modified environment once, stopping at each node to cover the edge-piece (two edge-pieces in the case of x_1) emanating from it. The total length of the optimal covering path is $l_{\text{opt}} = kL + 2(H + k)$, since the robot must enter and retreat from every piece of the edges J_i. Comparing l with l_{opt}, we obtain that $l \geqslant (2 - \varepsilon)l_{\text{opt}}$, where $\varepsilon = 2k/(kL + 2(H + k)) \ll 1$ since $L \gg 1$. □

6. Simulation results

We now present simulations of the four algorithms *2D-Spiral-STC*, full *Spiral-STC*, *2D-Scan-STC* and full *Scan-STC*. The algorithms are run on an environment that resembles several rooms populated by pieces of furniture. The first example shows several execution stages of the *2D-Spiral-STC* algorithm (Fig. 14). In Fig. 14(a) the covering tool follows the right-side of the spanning-tree edges along a subcell path that spirals outward from the starting cell S. The spiral shape is a result of the algorithm's selection of new neighbors in counterclockwise order, starting with the parent cell. This selection rule forces the covering tool to turn right whenever possible, and in empty regions this policy yields spiral paths. When the covering tool reaches the outer boundary, it follows this boundary into the lower-left room. In Fig. 14(b) the covering tool spirals inward to the center of the lower-left room. The cell at the center of this room is a leaf node of the spanning tree, and the covering tool backtracks outward along the complementary spiral. In Fig. 14(c) the covering tool proceeds to the lower-right room, and in Fig. 14(d) the room is covered with the same double-spiral pattern. Finally, the covering tool spirals inward to the starting cell S, resulting in a complete coverage of the grid. The spanning tree constructed by *2D-Spiral-STC* appears in Fig. 14(e). Note that the covering path is optimal, in the sense that there is no repetitive coverage of any point in the area underlying the grid of free $2D$-cells.

The next example shows several execution stages of the full *Spiral-STC* algorithm (Fig. 15). In contrast with the previous grid of $2D$-cells, the discretization of the environment into D-size subcells yields

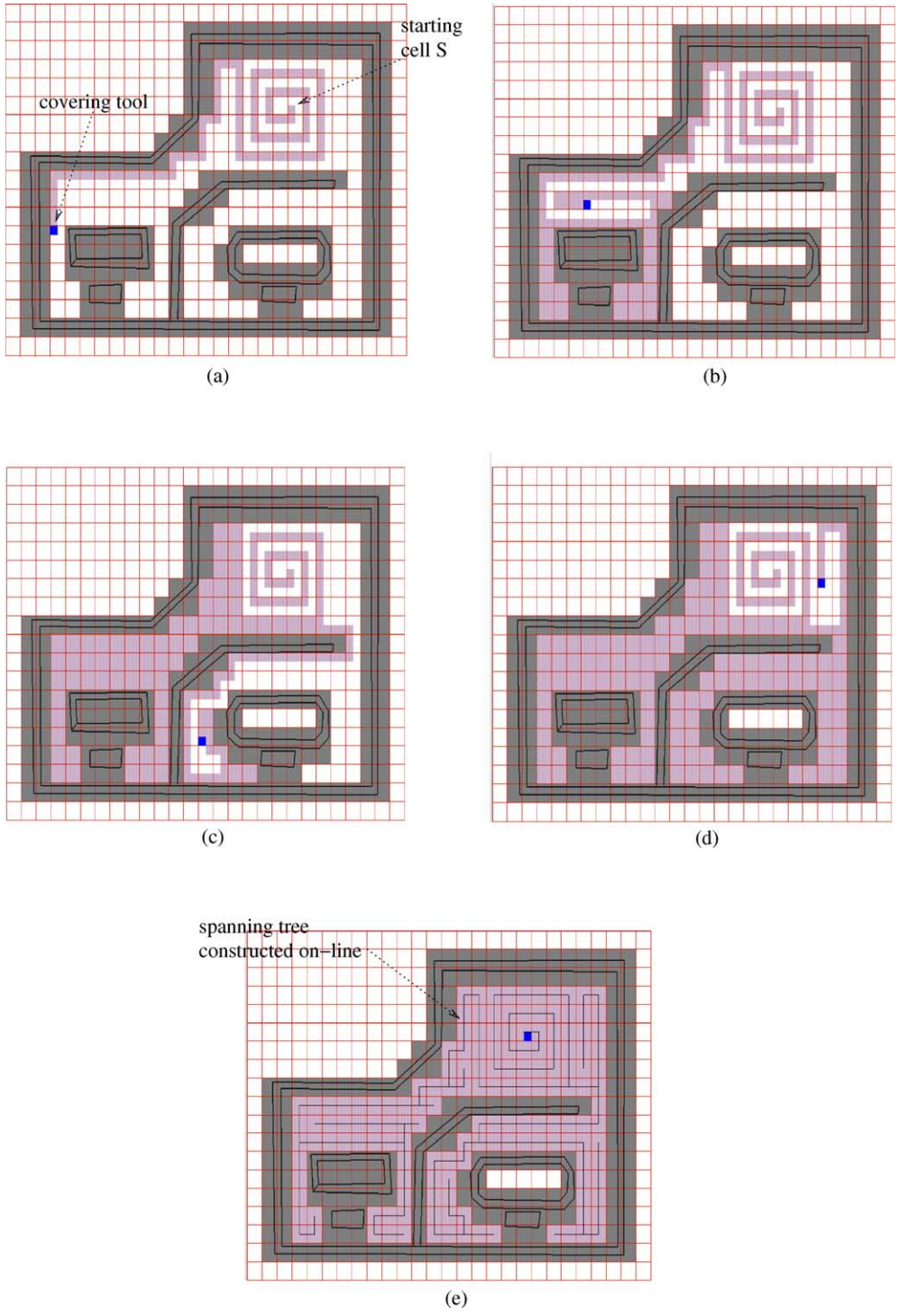


Fig. 14. Five covering stages of the *2D-Spiral-STC* algorithm.

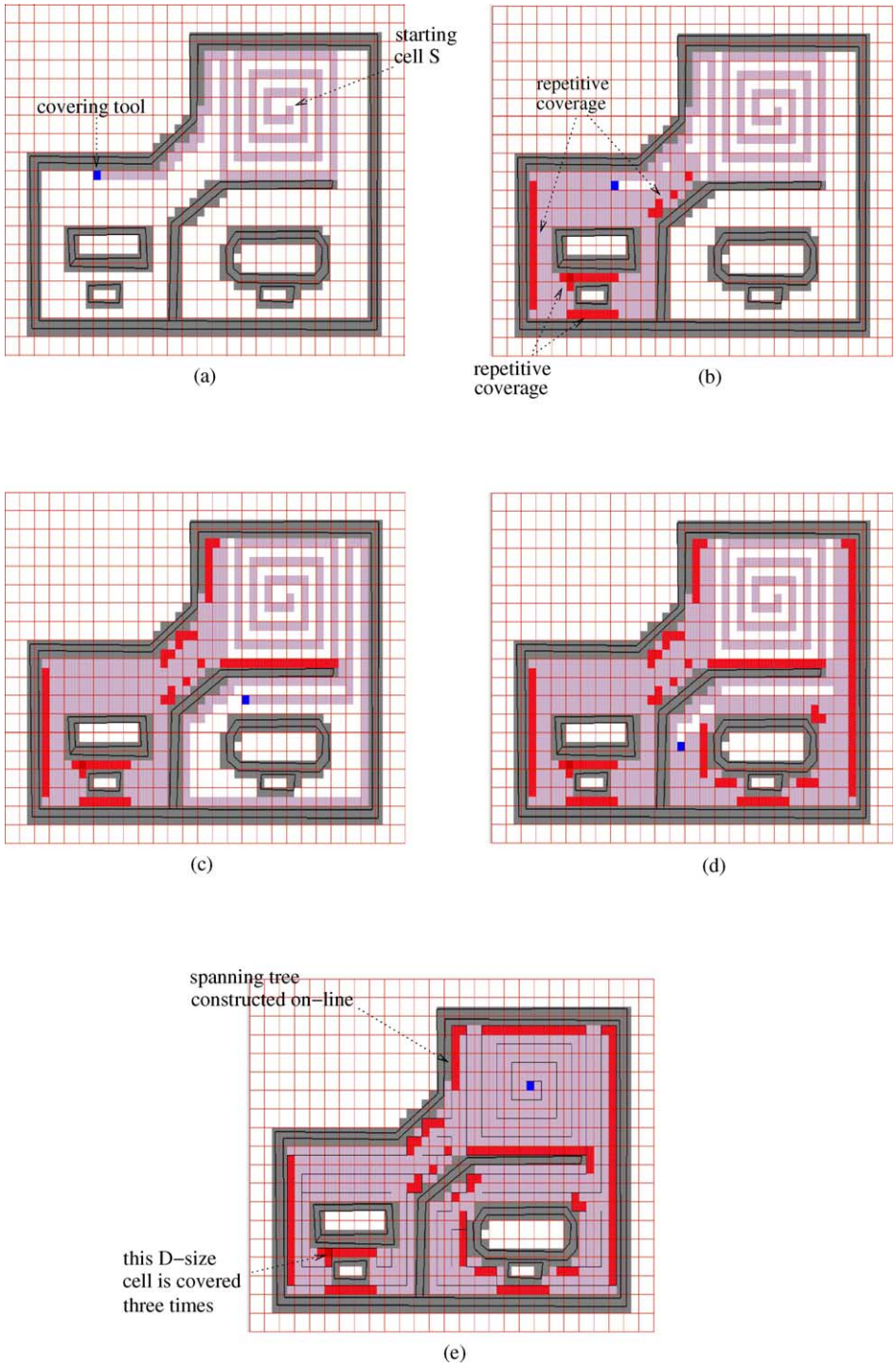


Fig. 15. Five covering stages of the full *Spiral-STC* algorithm, with repetitive coverage depicted in dark gray.

a completely general grid. Fig. 15(a) shows that here, too, the covering tool initially spirals outward. However, now the covering tool crosses the spanning-tree edges at places where subcells on the right-side of the spanning-tree edges are occupied by obstacles. In Fig. 15(b) the covering tool follows the outer boundary into the lower-left room. The room is covered with a double-spiral pattern whose inner part lies between the internal obstacles (see Fig. 15(e) for the corresponding spanning tree). The covering tool next proceeds to cover the lower-right room as shown in Fig. 15(c)–(d). The narrow gaps between the internal obstacles in both rooms incur repetitive coverage of subcells in these gaps. This repetitive coverage is fundamental—any on-line coverage algorithm must in worst-case repetitively cover all subcells that locally disconnect the work-area grid. Fig. 15(e) shows the completion of coverage, where the covering tool spirals back to the starting cell. The figure shows in dark gray the subcells that were covered twice (one subcell was covered three times in this example). In total 18% of the subcells in this environment were repetitively covered. While we do not know the precise length of the globally optimal covering path in this example, *Spiral-STC* generated a covering path whose length is at most 18% longer than the globally optimal path.

The third example shows the *2D-Scan-STC* algorithm (Fig. 16). In Fig. 16(a) the covering tool follows a vertical-scanning pattern that extends leftward from the starting cell. The spanning tree constructed during this stage is overlayed, and its distinctive comb-like shape should be noted. The comb's "spine" lies a distance D away from the lower wall, leaving an empty subcell-path that would allow the covering tool to retreat into the right-side of the environment. Fig. 16(b) shows the stage where the covering tool retreats from the lower-left room along the wall. It returns to the column of the starting cell and completes the coverage of the initial room with a vertical-scanning pattern that extends rightward (Fig. 16(c)). Intuitively speaking, *2D-Scan-STC* exhibits a desired behavior, whereby it covers entire rooms except for an escape route which is left along the room's walls. This feature is not shared by the previous algorithms, and its practical significance is discussed in the concluding section. Next, Fig. 16(d) shows the covering of the lower-right room, and Fig. 16(e) shows the completion of coverage. Note that the spanning tree constructed by the algorithm almost minimizes the number of horizontal spanning-tree edges. It contains a single wasteful horizontal edge at the top of the column containing the starting cell. The corresponding covering path is not only optimal in its length, but is also almost optimal in terms of the number of horizontal path-segments.

The last example shows the full *Scan-STC* algorithm (Fig. 17). Here, too, the discretization of the environment into a grid of D-size subcells incurs repetitive coverage. In this example 17% of the subcells were repetitively covered, making the covering path at most 17% longer than the globally optimal path. The average amount of repetitive coverage generated by the full *Scan-STC* and *Spiral-STC* algorithms can be characterized as follows. Recall that repetitive coverage occurs at places where the subcells on the right-side of the spanning-tree edges are occupied by obstacles. In particular, all the repetitive coverages occur in subcells that share a point or an edge with the grid boundaries. On average, half of these boundary subcells lie on the left-side of the spanning-tree edges, and the covering tool must pass through these subcells twice during the spanning tree circumnavigation. On average slightly more than 50% of the boundary subcells are repetitively covered, due to additional repetitive coverage incurred each time the covering tool crosses the spanning-tree edges. We have not attempted to rigorously measure the average amount of repetitive coverage in simulations. In the examples described here, *Spiral-STC* repetitively covered 56% of the boundary subcells (Fig. 15(e)), while *Scan-STC* repetitively covered 52% of the boundary subcells (Fig. 17(e)).

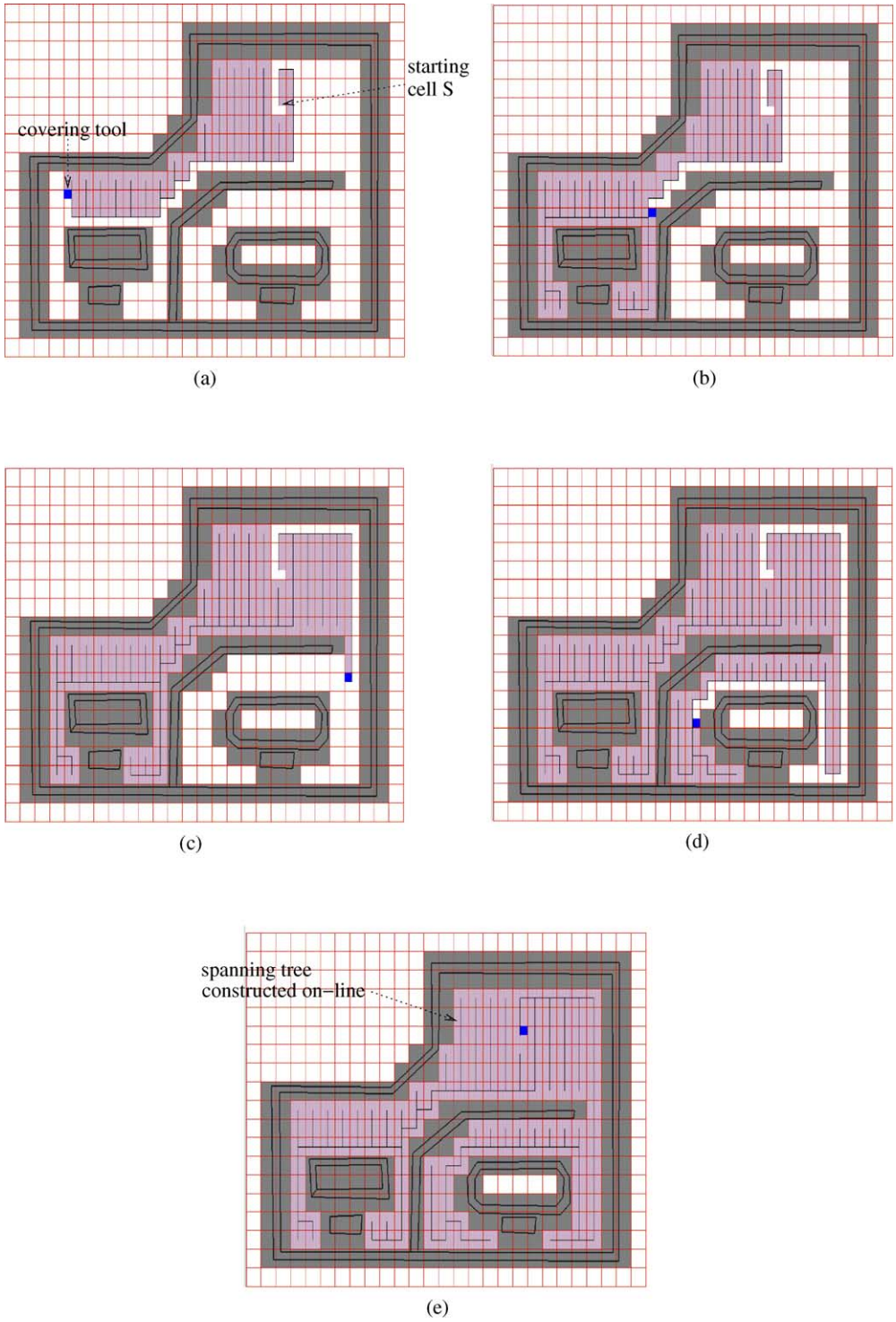


Fig. 16. Five covering stages of the *2D-Scan-STC* algorithm, with the spanning tree overlayed in each stage.

Fig. 17. Five covering stages of the full *Scan-STC* algorithm, with repetitive coverage depicted in dark gray.

7. Conclusion

We presented and analyzed two on-line mobile robot covering algorithms. Both algorithms incrementally discretize the work-area into a grid of D-size cells, then follow a path that circumnavigates a spanning tree constructed for a coarser grid of $2D$-size cells. The two algorithms differ in the incremental rule they employ to construct the spanning tree. *Spiral-STC* treats the grid edges as having equal weights and generates spiral-like covering patterns. *Scan-STC* assigns lower weights to grid edges aligned with a desired scanning direction and generates scan-like covering patterns. Both algorithms treat partially occupied $2D$-cells as special nodes of the spanning tree that incur repetitive coverage. The algorithms cover any planar grid of D-size cells in $O(n)$ time using a covering path whose length is at most $(n+m)D$, where n is the number of D-size cells and $m \leqslant n$ is the number of D-size cells along the grid boundary. The algorithms satisfy an even tighter path-length bound of $(n+k)D$, where $k \leqslant m$ is the number of D-size cells that lie along the grid boundary in the vicinity of partially occupied $2D$-cells. We also quantified the quality of the scanning pattern generated by *2D-Scan-STC*. The number of path-segments orthogonal to the desired scanning direction is bounded by $h^* + p + 1$, where h^* is the off-line minimum number of path-segments orthogonal to the scanning direction, and p is the number of connectivity changes in the columns of the grid aligned with the scanning direction.

Both algorithms require $O(n)$ memory, which imposes a limit on the size of the work-areas that can be covered by a limited-memory robot. The STC algorithms can be adapted for the use of cell markers, and in that case they would require only $O(1)$ memory [9]. Finally, we showed that any on-line coverage algorithm generates a covering path whose length is at least $(2-\varepsilon)l_{\text{opt}}$ in worst case. Since $(n+m)D \leqslant 2l_{\text{opt}}$, the bound is tight and the STC algorithms are worst-case optimal. The trivial DFS is also worst-case optimal. However, DFS always generates a path whose length is $2nD$. In contrast, the length of the path generated by the STC algorithms is at most $(n+m)D$, where $m \ll n$ in practical environments.

Let us mention several open issues concerning on-line mobile robot coverage. The first issue is coverage within guaranteed return-time bounds. Given a cell in the interior of the grid, it is desirable to complete the coverage of the cell as well as its surrounding cells within a guaranteed time bound. The practical motivation for this requirement is the accumulative error incurred by the robot's position sensors. If the robot does not complete the coverage of any particular area in a reasonably short time, it will have to compensate for its position uncertainty by employing wasteful overlap in its covering pattern. *Spiral-STC* is problematic in this respect, since it leaves in each room a spiral pattern that has to be covered at a later stage. In contrast, *Scan-STC* covers contiguous areas leaving only cells along the grid boundary. Unfortunately *Scan-STC* does not guarantee any return-time bound, and to our knowledge no other on-line covering algorithm guarantees such a bound. A second issue is cooperative coverage by several mobile robots. Existing results on cooperative coverage employ cell markers to communicate the state-of-coverage between the group members [22]. It is more desirable to achieve cooperative coverage using local communication between the robots. A fundamental question in this context is to identify the minimal amount of communication that would still allow a reasonable speedup of the cooperative coverage. A third issue is on-line coverage of curved terrains by a mobile robot. In this case a representation of the terrain's surface by a uniform grid may not be well justified, since the surface's area-form varies along the surface while the covering-tool's shape is constant. This issue becomes even more acute when a robot arm has to cover a three-dimensional structure [3].

Appendix A. Details concerning the STC algorithms

The following lemma asserts that the STC algorithms cover the subcells of every cell in counterclockwise order.

Lemma A.1. *The* 2D-Spiral-STC *and* 2D-Scan-STC *algorithms cover the subcells of each cell in counterclockwise* (*but not necessarily contiguous*) *order. Hence there is always an empty subcell path leading from the current subcell to the next new neighbor in counterclockwise order.*

Proof. Consider the *2D-Spiral-STC* algorithm. If a cell x has no new neighbors, x is a leaf-node of the spanning tree and the covering tool circumnavigates this node as depicted in Fig. 18(a). If x has new neighbors, the algorithm selects the first neighbor in counterclockwise order, starting with the parent cell w. Let y denote this neighbor. Since the covering tool follows the right-side of the spanning-tree edges, when it exits into y through a particular subcell of x, it must return from y into the next subcell of x in counterclockwise order. As Fig. 18(b)–(d) depicts, for each combination of new neighbors the subcells of x are consequently covered in counterclockwise order. □

Next we establish an upper bound on the number of horizontal spanning-tree edges constructed by *2D-Scan-STC*. Proposition 4.4 is repeated here with a simplifying assumption that the starting cell lies on the grid boundary.

Proposition 4.4. *Let the starting cell lie on the grid boundary. Then the number of horizontal spanning-tree edges constructed by* 2D-Scan-STC, h, *is bounded by*

$$h \leqslant h^* + p,$$

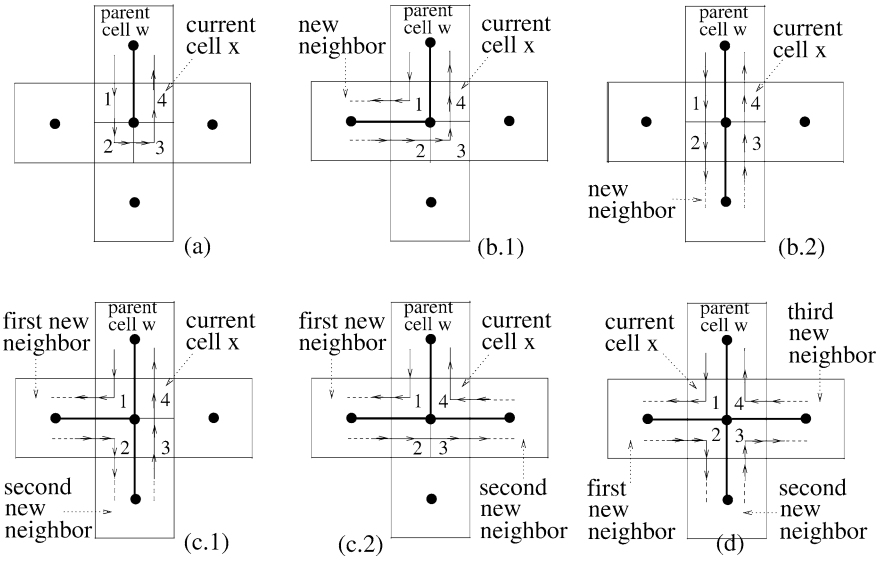


Fig. 18. The subcell paths associated with (a) a cell with no new neighbors, (b)–(c) a cell with one or two new neighbors, (d) a cell with three new neighbors.

where h^* is the off-line optimal number of horizontal spanning-tree edges, and p is the number of connectivity changes in the column-segments of the grid.

The proof of the proposition relies on two lemmas. The first lemma characterizes the amount of excessive horizontal spanning-tree edges, $h - h^*$.

Lemma A.2. *The number of excessive horizontal spanning-tree edges constructed by* 2D-Scan-STC *is equal to the number of vertical grid-edges not included in the spanning tree.*

Proof. In general, a spanning tree for a grid with N cells must have $N - 1$ edges. The spanning tree with the minimum number of horizontal edges necessarily contains all the vertical edges in the grid. Since the total number of spanning-tree edges is fixed, every excessive horizontal spanning-tree edge corresponds to some vertical grid-edge which is not included in the spanning tree. □

The second lemma characterizes the location of the horizontal spanning-tree edges.

Lemma A.3. *Each horizontal spanning-tree edge constructed by* 2D-Scan-STC *is incident to a cell that lies along the grid boundary.*

Proof. The algorithm constructs a horizontal spanning-tree edge from the current cell x to a horizontal neighbor y when one of the cells $y + 45$ and $y + 90$ is occupied. Thus either x or y has a neighbor which is occupied by an obstacle. □

We are now ready to prove the proposition.

Proof of Proposition 4.4. The proof is structured as follows. First we consider a column-segment that has no connectivity change. We show that all the vertical grid-edges of such a column-segment are included in the spanning tree constructed by the algorithm. Next we consider a column-segment that experiences a single connectivity change. We show that in this case at most one vertical grid-edge of the column-segment is not included in the spanning tree. Last, we generalize the result to column-segments that experience several simultaneous connectivity changes.

Let C_0 be a column-segment that experiences no connectivity change. Let C_1 and C_2 denote its left and right neighboring column-segments. (There is at most one column-segment on each side of C_0.) Lemma A.3 implies that the algorithm can possibly construct two horizontal spanning-tree edges between C_0 and C_1, and two horizontal edges between C_0 and C_2. These edges are indicated with bold lines in Fig. 19(a). The construction rule specified in the algorithm further implies that the covering tool can enter C_0 only through the top cell of C_1, or through the bottom cell of C_2. Similarly, the covering tool can exit C_0 only from the bottom cell of C_0, or from the cell adjacent to the top cell of C_2. Suppose the covering tool enters C_0 through the top cell of C_1. Then the right-turn policy of the algorithm would take the covering tool downward in C_0 as depicted in Fig. 19(a). It can be shown by induction that *the covering tool must have visited all the cells of C_1 that border C_0 before entering C_0*. Hence the covering tool cannot exit from C_0 back into C_1 at the bottom cell of C_0. Rather, it backtracks upward in C_0 until it exits into C_2. At that instant all the cells of C_0 that border C_2 have been visited by the covering tool. Hence the covering tool must return to C_0 along the same horizontal spanning-tree edge that led it into C_2. The

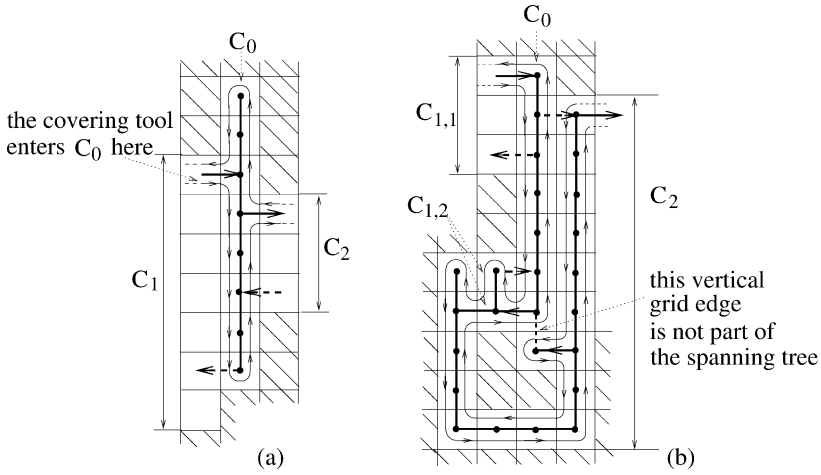


Fig. 19. (a) The covering pattern of a column-segment with no connectivity change. (b) The covering pattern of a column-segment with one connectivity change.

covering tool next visits the cells that remain in the top portion of C_0, then returns to C_1. During this excursion all the vertical grid-edges in C_0 are included in the spanning tree.

Next consider the case where C_0 experiences a single connectivity change, as depicted in Fig. 19(b). In this case one of the neighboring column-segments, say C_1, consists of two disjoint components. Let $C_{1,1}$ and $C_{1,2}$ denote these two components. Lemma A.3 implies that at most two horizontal spanning-tree edges can exist between C_0 and each of the column-segments $C_{1,1}$, $C_{1,2}$ and C_2. Moreover, for each pair of horizontal spanning-tree edges associated with a neighboring column-segment, the covering tool can enter C_0 along one of these edges and exit C_0 along the other edge. Suppose the covering tool enters C_0 through the top cell of $C_{1,1}$. Here again, it can be shown by induction that the covering tool must have visited all the cells of $C_{1,1}$ before entering C_0. The covering tool therefore proceeds downward in C_0, until it exits into $C_{1,2}$ as indicated in Fig. 19(b). There are now two sub-cases to consider. Let $\mathcal{C}$ denote the cell of column-segments to the left of $C_{1,2}$. In the first sub-case $\mathcal{C}$ can be accessed only through $C_{1,2}$. In that case the covering tool completes the coverage of $\mathcal{C}$ then returns to C_0 along the same horizontal spanning-tree edge that led it into $C_{1,2}$. The remaining lower portion of C_0 is next covered, resulting in a spanning tree that includes all the vertical grid-edges of C_0. In the second sub-case $\mathcal{C}$ can be also accessed from its left side (Fig. 19(b)). In this case the covering tool can move through a sequence of cells that loops around an obstacle and brings it back into an uncovered portion of C_0. Since C_0 experiences a single connectivity change, the covering tool must return to C_0 through a cell of C_2. Moreover, this event can occur at most once, since there exits only one feasible entry edge from C_2 into C_0. Upon entering an uncovered portion of C_0, the covering tool covers this portion until it encounters the covered portion of C_0. The gap between the two covered portions of C_0 consists of a single vertical grid-edge, and the spanning tree is therefore missing one vertical grid-edge of C_0.

Finally consider the case where C_0 experiences several connectivity changes. (The probability of such simultaneous events approaches zero as the tool size D approaches zero.) Every connectivity change adds one more component to the neighboring column-segments. Two horizontal spanning-tree edges can possibly exist between C_0 and each neighboring component. Moreover, the covering tool can enter C_0 along one of these edges and exit C_0 along the other edge. Once the covering tool enters C_0,

each subsequent exit from C_0 can lead to a path that takes the covering tool around an obstacle into some uncovered portion of C_0. The number of such exit-and-entry events is bounded by the number of connectivity changes in C_0. Moreover, by construction the covering tool must cover each portion of C_0 until it encounters a covered portion of C_0. Hence the different covered portions of C_0 are separated from each other by single vertical grid-edges. The number of vertical grid-edges in C_0 which are missing from the spanning tree is therefore bounded by the number of connectivity changes in C_0. □

References

[1] E.M. Arkin, S.P. Fekete, I.S. Mitchell, Approximation algorithms for lawn mowing and milling, Computational Geometry 17 (2000) 25–50.
[2] S. Arora, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, J. ACM 45 (5) (1998) 753–782.
[3] P.N. Atkar, H. Choset, A.A. Rizzi, E.U. Acar, Exact cellular decomposition of closed orientable surfaces embedded in $\mathbb{R}^3$, in: IEEE Int. Conf. on Robotics and Automation, 2001, pp. 699–704.
[4] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, M. Talamo, Competitive algorithms for the on-line traveling salesman, in: Proc. of 4th WADS, in: Lecture Notes in Computer Science, Vol. 955, Springer-Verlag, Berlin, 1995, pp. 206–217.
[5] X. Deng, T. Kameda, C. Papadimitriou, How to learn an unknown environment, in: 31th IEEE Symp. on Foundations of Computer Science, 1991, pp. 298–303.
[6] X. Deng, A. Mirzaian, Competitive robot mapping with homogeneous markers, IEEE Trans. on Robotics and Automation 12 (4) (1996) 532–542.
[7] G. Dudek, M. Jenkin, E. Milios, D. Wilkes, Robotic exploration as graph construction, IEEE Trans. on Robotics and Automation 7 (6) (1991) 859–865.
[8] H. Enders, W. Feiten, G. Lawitzky, Field test of a navigation system: Autonomous cleaning in supermarkets, in: IEEE Int. Conf. on Robotics and Automation, 1998, pp. 1779–1781.
[9] Y. Gabriely, E. Rimon, Spanning-tree based coverage by a mobile robot, Technical Report, Dept. of ME, Technion, http://www.technion.ac.il/~robots, 1999.
[10] M. Grigni, E. Koutsoupias, C. Papadimitriou, An approximation scheme for planar graph TSP, in: 36th IEEE Symp. on Foundations of Computer Science, 1995, pp. 640–645.
[11] S. Hedberg, Robots cleaning up hazardous waste, AI Expert (May 1995) 20–24.
[12] M. Held, On the Computational Geometry of Pocket Machining, Springer-Verlag, New York, 1987.
[13] C. Icking, T. Kamphans, R. Klein, E. Langetepe, On the competitive complexity of navigation tasks, in: 16th European Workshop on Computational Geometry, Eilat, 2000, pp. 140–143.
[14] C. Icking, T. Kamphans, R. Klein, E. Langetepe, On the competitive complexity of navigation tasks, in: Sensor Based Intelligent Robots, in: Lecture Notes in Computer Science, Vol. 2238, Springer-Verlag, Berlin, 2002, pp. 245–258.
[15] C. Icking, R. Klein, Competitive strategies for autonomous systems, in: H. Bunke, T. Kanade, H. Noltemeier (Eds.), Modelling and Planning for Sensor Based Intelligent Robot Systems, World Scientific, Singapore, 1995, pp. 23–40.
[16] A. Itai, C. Papadimitrio, J. Szwarcfiter, Hamiltonian paths in grid graphs, SIAM J. Comput. 11 (1982) 676–686.
[17] B. Kalyanasundaram, K.R. Pruhs, Constructing competitive tours from local information, in: Lecture Notes in Computer Science, Vol. 700, Springer-Verlag, 1993, pp. 102–113.
[18] J.S.B. Mitchell, Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems, SIAM J. Comput. 28 (1999) 1298–1309.
[19] J.D. Nicoud, M.K. Habib, The PEMEX autonomous demining robot: Perception and navigation strategies, in: IEEE/RSJ Int. Conf. on Intelligent Robot Systems, 1995, pp. 419–424.
[20] S. Ntafos, Watchman routes under limited visibility, Computational Geometry 1 (1992) 149–170.
[21] N.S.V. Rao, S.S. Iyengar, Autonomous robot navigation in unknown terrains: visibility graph based methods, IEEE Trans. Systems Man Cybernet. 20 (6) (1990) 1443–1449.
[22] I.A. Wagner, M. Lindenbaum, A.M. Bruckstein, Distributed covering by ant-robots using evaporating traces, IEEE Trans. on Robotics and Automation 15 (5) (1999) 918–933.