
Annals of Mathematics and Artificial Intelligence **31:** 77–98, 2001.

Spanning-tree based coverage of continuous areas by a mobile robot

Yoav Gabriely and Elon Rimon *

Department of Mechanical Engineering, Technion, Israel Institute of Technology, Israel
E-mail: {yoav@robots;elon@robby}technion.ac.il

This paper considers the problem of covering a continuous planar area by a square-shaped tool attached to a mobile robot. Using a tool-based approximation of the work-area, we present an algorithm that covers every point of the approximate area for tasks such as floor cleaning, lawn mowing, and field demining. The algorithm, called *Spanning Tree Covering* (STC), subdivides the work-area into disjoint cells corresponding to the square-shaped tool, then follows a spanning tree of the graph induced by the cells, while covering every point precisely once. We present and analyze three versions of the STC algorithm. The first version is off-line, where the robot has perfect apriori knowledge of its environment. The off-line STC algorithm computes an optimal covering path in linear time $O(N)$, where N is the number of cells comprising the approximate area. The second version of STC is on-line, where the robot uses its sensors to detect obstacles and construct a spanning tree of the environment while covering the work-area. The on-line STC algorithm completes an optimal covering path in time $O(N)$, but requires $O(N)$ memory for its implementation. The third version of STC is "ant"-like. In this version, too, the robot has no apriori knowledge of the environment, but it may leave pheromone-like markers during the coverage process. The ant-like STC algorithm runs in time $O(N)$, and requires only $O(1)$ memory. Finally we present simulation results of the three STC algorithms, demonstrating their effectiveness in cases where the tool size is significantly smaller than the work-area characteristic dimension.

1. Introduction

The *mobile robot covering problem* can be formulated as follows. Let a tool of a specific planar shape be attached to a mobile robot, and let $\mathcal{A}$ be a continuous planar work-area bounded by obstacles. Then the mobile robot has to move the tool along a path such that every point of $\mathcal{A}$ is covered by the tool along the path. The covering problem is currently receiving considerable attention for several reasons. First, sensor-based coverage by mobile robots seems amenable to the geometric planning techniques developed for the sensor-based robot navigation problem, a problem that recently received considerable attention [1,7,25]. A second reason is current interest in competitive algorithms[1] for autonomous systems that operate with incomplete information [5,13,18]. The optimal covering problem can be formulated as a generalization of the Traveling Sales-

* This research is supported by Friendly Machines.

[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.

person Problem (TSP) for a continuous domain, and thus is NP-hard [2,4]. It is therefore natural to seek competitive algorithms for the covering problem. Finally, recent work on ant-like coverage [30,31] has spurred interest in the possibility of achieving complex coverage behavior by bounded-resource agents who leave pheromone-like markers in the environment.

Besides its theoretical interest, the mobile robot covering problem has several important applications. One such application area is automatic floor cleaning and coating in facilities such as supermarkets [14] and train stations [32]. Other application areas are lawn mowing [22], hazardous waste cleaning [15], and field demining [20]. Furthermore, although we focus on mobile robot coverage, the same methods apply to other robotic coverage tasks, such as part machining [16], car painting, and wall finishing during house construction.

Previous efforts on the covering problem have focused mainly on the *discrete* version of the problem, where an agent has to visit all nodes of an unknown graph by traversing its edges. Examples are the discrete covering algorithms of Dudek et al. [11] and Deng and Mirzaian [10], who achieve competitive graph coverage by using pebble markers that can be placed and later picked up by the searching agent. More recently, Wagner et al. [30,31] have proposed discrete coverage algorithms that use pheromone-like markers to guide the coverage process and to coordinate cooperation between several agents. The *continuous* covering problem, where a robot must sweep a tool over a continuous area, has been studied by Ntafos [21]. Much like in our approach, Ntafos imposes a tool-based grid approximation over the work-area. Then he subdivides the grid into rectangular sub-grids that can be covered optimally, and computes a TSP path that visits all sub-grids. Ntafos considers only simple grids with no internal holes, and his algorithm covers the grid within 33% of its optimal covering path. The continuous covering problem has also been studied by Arkin et al. [3]. They propose an algorithm that covers simple grids within 20% of the optimal covering path. They also extend their algorithm to non-simple grids which possess no local cut nodes[2], and their algorithm covers such grids within 32.5% of the optimal covering path. In contrast, we consider general grids, we make no attempt at subdividing the grid into sub-grids, and our algorithm produces an optimal covering path. Furthermore, the algorithms of Ntafos and Arkin are strictly off-line while ours has an on-line version. The continuous coverage problem has also been studied by Butler et al. [6], Choset and Pignon [8], and Pirzadeh and Snyder [23]. All of these works emphasize the importance of achieving on-line coverage based on the robot's sensors.

Finally, we note that map building or area exploration is distinct from the covering problem considered here. In the map building problem an autonomous agent has to completely scan an unknown environment using its sensors [9,12,17,21,24,29]. Map building can thus be interpreted as achieving sensory coverage of an unknown environment. In contrast, area coverage requires physical sweeping of a tool over every point of a given work-area.

[2] A *local cut node* is a node whose removal locally disconnects the graph induced by the grid.

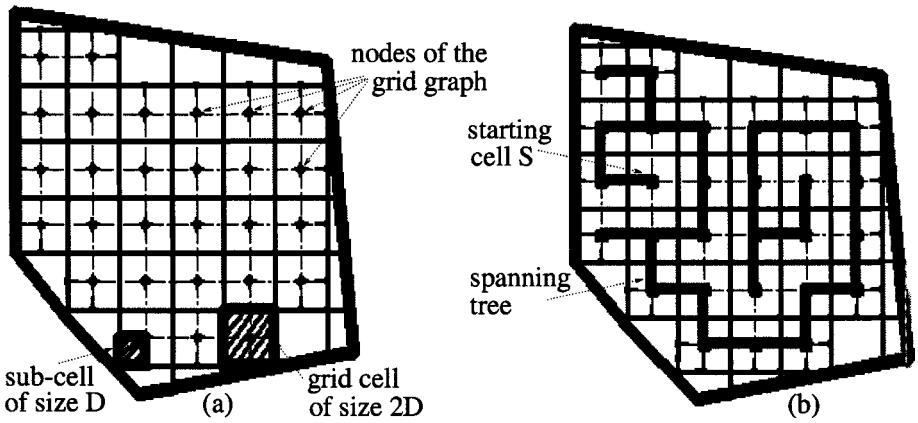

Figure 1. (a) Grid approximation of a given work-area. (b) A spanning tree for the grid.

In this paper we make the following simplifying assumptions on the mobile robot covering problem. 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 subdivide the work-area into square cells of size $2D$, and discard cells which are partially covered by obstacles (figure 1). As later discussed with the analysis and simulations, the quality of the approximation depends on the tool size D being significantly smaller than the work-area characteristic dimension. The algorithm, called *Spanning Tree Covering* (STC), follows a spanning tree of the graph induced by the cells, while covering every point precisely once. The STC algorithm can be interpreted in two ways. First, from a graph-theoretic viewpoint, the algorithm strives to embed a Hamiltonian cycle[3] having the largest number of cells in the given work-area. The resulting Hamiltonian cycle is automatically an optimal covering path in terms of path length, since the tool covers every cell (and hence every point in the underlying continuous area) precisely once. It should be noted that the problem of constructing a Hamiltonian cycle in a general grid-like graph is NP-complete [19]. Intuitively speaking, we show that for covering purposes it is advantageous to first construct a finer grid of D-size cells in $O(N)$ steps, then construct a Hamiltonian cycle for this grid in an additional $O(N)$ steps, where N is the total number of grid cells. Second, from a robot motion-planning viewpoint, we decompose the environment into simple cells and use a spanning tree as an adjacency graph for these cells. However, unlike previous covering paradigms, we carefully weave the traversal of the adjacency graph with the covering of individual cells. This integration of inter-cell traversal with intra-cell coverage is a novel approach that has not appeared in the literature before.

In the next two sections we present and analyze three versions of the STC algorithm. The first version is off-line, where the robot has perfect apriori knowledge of its environment. The off-line STC algorithm computes an optimal covering path for the approximate work-area in linear time $O(N)$. The second version of STC is on-line, where

[3] A *Hamiltonian cycle* is a closed path in a graph which visits every node of the graph precisely once.

the robot uses its on-board sensors to detect obstacles and construct a spanning tree of the environment while covering the work-area. The on-line STC algorithm completes an optimal covering path in time $O(N)$. However, it requires $O(N)$ memory, and this requirement limits the size of work-areas that can be covered by the on-line algorithm. The third version of STC is "ant"-like. In this version, too, the robot has no apriori knowledge of the environment, but it may leave pheromone-like markers during the coverage process. The ant-like STC algorithm completes an optimal covering path in time $O(N)$, and requires only $O(1)$ memory. The analysis section also contains a fourth version of the STC algorithm, which is an adaptation of the on-line STC to an algorithm that achieves *exact* coverage of the work-area. (This adaptation includes a pass along the obstacles' boundaries which requires re-orientation of the covering tool.) The exact STC algorithm is amenable to competitive analysis, and we show that it achieves complete coverage with a competitive ratio of $1+3(\Pi D/A)$, where D is the tool size, Π is a quantity related to the obstacles' total perimeter, and A the total area. In practice $\Pi D \ll A$, and STC typically achieves an almost optimal coverage of the work-area. Finally we present simulation results of the three STC algorithms, demonstrating their effectiveness in cases where the tool size is significantly smaller than the work-area characteristic dimension. The concluding section describes work in progress, where we augment the work-area grid with partially occupied cells to achieve a tighter approximation of the given work-area.

2. The STC algorithm

We describe three versions of the *Spanning Tree Covering* (STC) algorithm, starting with its off-line version.

2.1. The off-line STC algorithm

The input to the off-line STC algorithm is a geometrical description of a bounded planar environment populated by piecewise-smooth and stationary obstacles. The algorithm first subdivides the work-area into cells of size $2D$, and discards cells which are partially covered by obstacles. We define a graph structure, $G(V, E)$, by defining as nodes V the center points of each cell, and as edges E the line segments connecting centers of adjacent cells (figure 1(a)). The algorithm next constructs a spanning tree for G (figure 1(b)), and uses this tree to generate a covering path as follows.

Off-line STC algorithm
Input. A geometrical description of the environment, converted to a $2D$-size grid with a graph structure G as described above. A starting cell S.
Pre-processing. Starting from S, construct a *spanning tree* for G using any spanning-tree construction algorithm. Subdivide every $2D$-size cell into four identical *sub-cells* of size D.
Covering action. Starting at a sub-cell of S, move between neighboring sub-cells (each

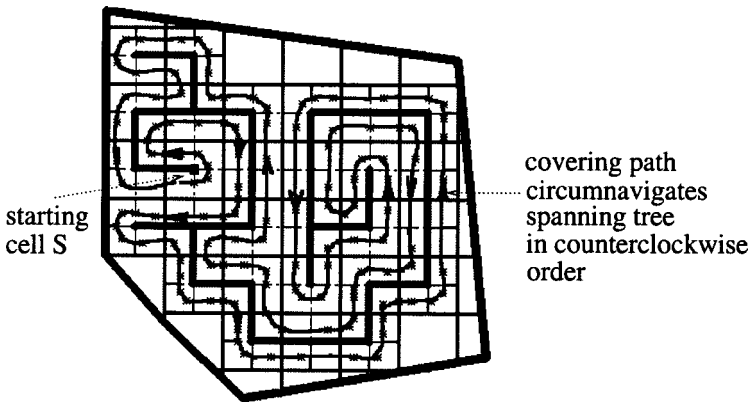

Figure 2. An execution example of the off-line STC algorithm.

being identical to the tool's shape) along a path which *circumnavigates the spanning-tree along a counterclockwise direction*. Halt when the starting sub-cell is encountered again.

An execution example is illustrated in figure 2. It can be seen that the robot moves the covering tool through sub-cells that lie on the right-side of the spanning-tree edges, measured with respect to the tool's direction of motion. Since a tree is contractible to a point, the circumnavigation of the spanning tree generates a simple closed path that brings the covering tool back to the starting sub-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. The off-line STC algorithm can compute a spanning tree for G using DFS. However, it can also assign edge weights and compute a minimal spanning tree using any minimum-spanning-tree algorithm such as Kruskal's or Prim's algorithms [28]. We can use this feature to affect the shape of the spanning tree and hence the covering-path geometry. For example, suitable edge weighting can generate covering paths that tend to scan the work-area along a particular coordinate direction, or covering paths that strive to follow obstacle perimeters. These possibilities are illustrated below.

2.2. *The on-line STC algorithm*

In this version of STC the robot has no prior knowledge about the environment, except that obstacles are stationary. Rather, the robot must use its on-board sensors to detect obstacles and plan its covering path accordingly. We assume that the robot has position and orientation sensors, allowing it to locally recognize the $2D$-size cells comprising the work-area. We also assume a range sensor, capable of identifying obstacles in the four cells neighboring the robot's current cell (figure 3(a)). The practically important issues of sensor selection, sensor measurement errors, and sensor fusion are not

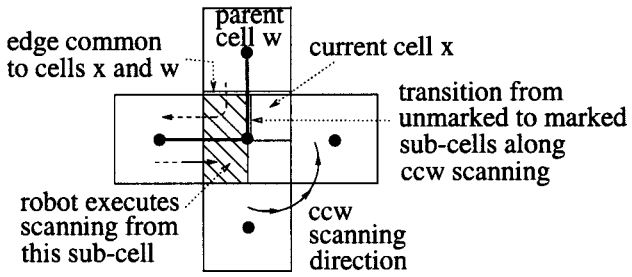

Figure 3. (a) Counterclockwise scanning of four neighbors. (b) A move from x to a new cell y. (c) A return from x to a parent cell w.

considered here. Rather, we assume ideal sensors that provide perfect readings. In the on-line STC algorithm, the robot incrementally constructs a spanning tree for the grid representing the work-area. During the spanning tree construction, the robot subdivides every cell it encounters into four identical sub-cells of size D, each being identical to the tool size. The covering tool follows a sub-cell path which circumnavigates the incrementally constructed spanning tree, until the entire work-area grid is covered. A description of the algorithm follows.

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

We now discuss several details of the algorithm. First note that the robot runs a DFS algorithm during the spanning tree construction. The counterclockwise scanning of neighbors specified in step 2 is crucial for ensuring that the covering tool circumnavigates the incrementally constructed spanning tree in counterclockwise order (figure 3(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. Second, in step 2.2 the covering tool is located in a sub-cell 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 sub-cell in x to a sub-cell of y by following the right-side of the spanning tree edges, measured with respect to the tool's direction of motion (figure 3(b)). It is shown below that the robot covers the sub-cells of x in counterclockwise order. Since the robot selects the first new neighbor y in counterclockwise order, the covering tool always has a sub-cell path from x to y which follows the right-side of the spanning tree edges. When the covering tool returns to a parent cell w, it again moves through sub-cells that lie on the right-side of the spanning-tree edge connecting x with w, as shown in figure 3(c). Finally, unlike the off-line STC, now the spanning-tree construction is part of the covering process. Since the spanning-tree construction must occur along a continuous path, weighted spanning-tree algorithms are not useful here, as these algorithms require discontinuous jumps between nodes. As a result, the on-line STC algorithm lacks a means to affect the shape of the spanning-tree being constructed.

2.3. The ant-like STC algorithm

The ant-like version of STC is also an on-line algorithm, and it uses the same sensors described above. However, now the robot has the ability to leave markers in the sub-cells it covers, using color, odor, heat, or pebble markers [26,27]. The particular nature of the markers has no importance, as long as they allow the robot to identify sub-cells that have been already covered. We additionally assume the availability of a detection device, capable of inspecting the sub-cell markers in the current cell and its four immediate neighbors. Using this detection device, the robot identifies a neighboring cell as *new* when its four sub-cells are all unmarked.

Much like the on-line STC algorithm, here too the robot uses DFS to incrementally construct a spanning tree for the work-area grid. However, rather than store the spanning tree in its memory, the robot uses the sub-cell markers to identify parent cells along the spanning tree as follows. By construction, the robot covers and marks the sub-cells of a given cell in counterclockwise order. Hence as long as the four sub-cells of the current cell are not all marked, the robot need only scan the sub-cells in counterclockwise order and identify the transition between unmarked and marked sub-cells. (The transition from unmarked to marked sub-cells along a counterclockwise direction is unique.) The two sub-cells bordering this transition have an exterior edge in common. As depicted in figure 4, this edge is necessarily the boundary edge between the current cell and its parent cell in the spanning tree. Moreover, the robot has no need to identify a parent cell from a cell whose four sub-cells are already marked, since once the robot covers the four sub-cells of a given cell it never returns to this cell again. A description of the algorithm follows.

Ant-like STC algorithm
Marking device. A device that marks the sub-cell currently being covered.
Sensors. A position and orientation sensor. An obstacle detection sensor. A marker detector.

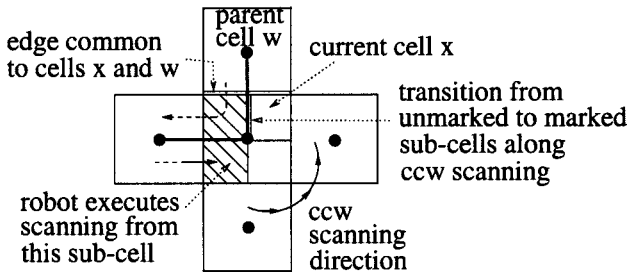

Figure 4. Parent detection using sub-cell markers.

Input. A starting cell S, but no apriori knowledge of the environment.
Find parent function. A function *Parent*(x) which returns the parent of the current cell x in the spanning tree, using the technique described above.
Find new neighbor function. A function *Neighbor*(w, x) which accepts as inputs the current cell x and its parent cell w, and returns the first *new* obstacle-free neighbor of x in counterclockwise order, starting with w. Returns *Null* if x has no new neighbor.
Initialization. Set $x = S$.
While S has unmarked sub-cells:

1. Determine the parent cell of x, $w = Parent(x)$.
2. Determine a *new* neighbor of x, $y = Neighbor(w, x)$.
3. If y is not *Null*:
   3.1. Move to a sub-cell of y, while marking the sub-cells of x being covered.
   3.2. Set $x = y$.
4. Else (y is *Null*):
   4.1. Return from x to a sub-cell of w, while marking the sub-cells of x being covered.
   4.2. Set $x = w$.

End of while loop.

We now discuss several details of the algorithm. In the function *Parent*(x), the parent of the starting cell S is not well defined. In this case *Parent* (x) identifies the transition between unmarked and marked sub-cells of S in counterclockwise order, and returns the neighbor cell adjacent to this transition. In steps 3.1 and 4.1, the move between cells is executed by following the right-side of the spanning tree edges, as depicted in figure 3. However, in the ant-like STC the robot does not explicitly construct the spanning tree. Rather, it computes the two spanning-tree edges required for the local motion of the covering tool. The edge from the current cell x to its parent cell is computed with *Parent*(x). The edge from x to a new neighbor (step 3.1), or from a neighbor back to x (step 4.1), is computed from knowledge of the neighbor. Finally, consider the sub-cell marking specified in the algorithm. As discussed above, *Parent*(x) can identify the parent cell of x only when not all four sub-cells of x are marked. Hence *the robot marks a sub-cell only when the covering tool leaves this sub-cell.* In particular, when the covering tool returns to a parent cell w in step 4.1, the robot marks the sub-cells of x being covered during the

return motion, but not the sub-cell of w into which the covering tool has entered. This feature allows the robot to identify the parent cell of w in the next step of the algorithm.

3. Algorithm analysis

In this section we first discuss several properties of the three STC algorithms, establish their correctness and derive performance bounds. Then we present an adaptation of the on-line STC algorithm which achieves exact coverage and is amenable to competitive analysis. Additional results concerning the quality of the grid approximation appear in the appendix.

3.1. Properties of the STC algorithm

For purposes of analysis, we define a *repetitive coverage* of a point p as the situation where p is being covered by the robot's tool, then exposed, and later covered again. We first establish that the three STC algorithms do not generate any repetitive coverage.

Lemma 3.1 (Non-repetitive coverage). The three STC algorithms: off-line STC, on-line STC, and ant-like STC, do not repetitively cover any work-area point.

Proof. In the three algorithms, the robot guides the tool from one sub-cell to an adjacent sub-cell using translational motion in the direction perpendicular to the edge common to both sub-cells. Since each sub-cell is identical to the tool in shape and size, the motion between adjacent sub-cells involves no repetitive coverage of either sub-cell. To complete the proof, we have to show that no sub-cell is visited more than once by the covering tool. In the three algorithms, the robot moves the tool through a sequence of sub-cells that circumnavigates the spanning tree (figure 2). If we consider the spanning-tree edges as having some non-zero thickness, the curve which circumnavigates the spanning tree is a simple closed curve. By construction, any two spanning-tree edges which do not emanate from the same node lie at least two sub-cells apart (figure 1). Hence, the curve circumnavigating the spanning-tree cannot cross the same sub-cell twice. Consequently, the tool which traces this curve cannot visit any sub-cell twice. This, together with the non-repetitive covering of adjacent sub-cells, implies the result. □

The following proposition considers the covering pattern of individual cells.

Proposition 3.2. The three STC algorithms cover the sub-cells of any particular cell in a counterclockwise (but not necessarily contiguous) order.

Proof. In all three algorithms, the robot moves the covering tool along a sub-cell path which locally follows the right-hand side of the spanning-tree, measured with respect to the tool's direction of motion. If a cell x has no new neighbors, the center of x is a leaf-node of the spanning tree, and the tool circumnavigates this node in counterclockwise

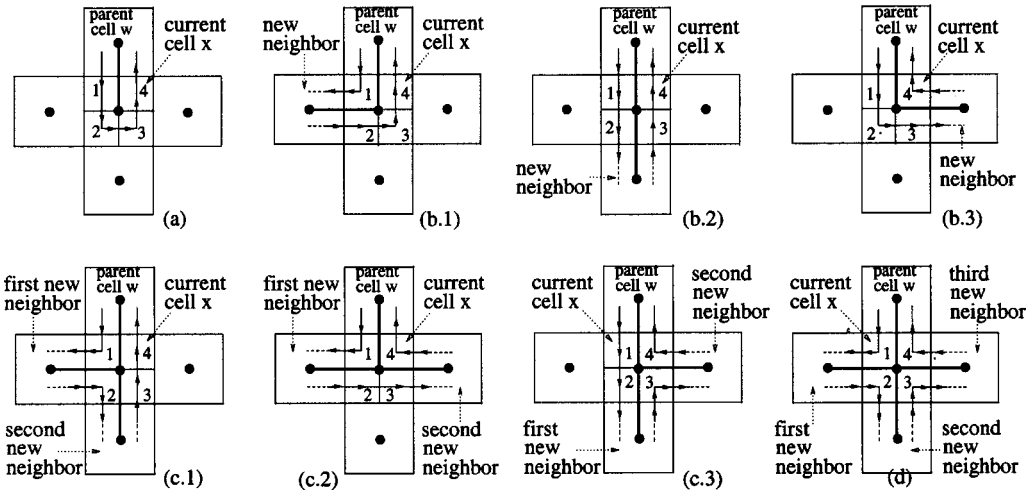

Figure 5. The sub-cell paths associated with (a) a cell with no new neighbors, (b)–(c) a cell with one or two new neighbors in three possible locations, (d) a cell with three new neighbors.

order as depicted in figure 5(a). If x has new neighbors, the robot selects the *first* neighbor in counterclockwise order, starting with the parent cell of x. Let y denote the first neighbor and let w denote the parent cell. The covering tool next follows the right-side of the spanning tree to a sub-cell of x which allows it to exit into y. (This sub-cell is determined by the spanning-tree edge from x to y.) As depicted in figures 5(b)–(d), motion along the right-side of the spanning tree implies that the tool moves through sub-cells of x in counterclockwise order.

When the covering tool returns from y to x, it follows the same spanning-tree edge that led it from x to y. Hence the tool re-enters x through a sub-cell which is adjacent to the sub-cell from which it left x. Let x' and x'' denote the sub-cells of x associated with exiting from x into y and returning from y to x. We now show that x'' is adjacent to x' in counterclockwise (ccw) order. There are two cases to consider. In the first case x' is the first sub-cell of x into which the tool entered from w. In this case x'' cannot lie before x' in ccw order, since the sub-cell before x' borders the parent cell w, and this would imply that $y = w$. In the second case x' is not the first sub-cell of x. In this case the sub-cell before x' in ccw order is already covered. It follows that x'' cannot lie before x' in ccw order, since according to lemma 3.1 no sub-cell is covered twice. When the covering tool returns from a neighbor y to a sub-cell x'' of x, it resumes its counterclockwise motion through sub-cells of x. The last sub-cell of x lies on the right-side of the spanning-tree edge that leads from x to w, and the covering tool returns from this sub-cell to w. □

The proposition has two important implications. First, since the robot enters new neighbors of the current cell in counterclockwise order, the proposition ensures that the covering tool can always access a new neighbor of the current cell x through empty sub-cells of x which follow the right-hand side of the spanning tree. Figure 5 illustrates these sub-cell paths for the four possible locations of a target neighbor. Second, the

proposition asserts that all cells are covered by the *same* counterclockwise pattern, which always starts with a sub-cell adjacent to the parent cell. This feature is crucial for the ant-like STC, which relies on this fixed coverage pattern in order to uniquely identify the parent cell using sub-cell markers. The next lemma considers the completeness of the three STC algorithms.

Lemma 3.3 (Complete coverage). The three STC algorithms: off-line STC, on-line STC, and ant-like STC, cover every cell which is accessible from the starting cell S.

Proof. Every obstacle-free cell defines a node in the work-area graph, and every pair of adjacent cells shares an edge in this graph. Hence the component of the work-area grid accessible from S is a connected graph. The three STC algorithms construct a spanning tree that reaches *every* cell accessible from S. These cells are partitioned into sub-cells. By construction, every sub-cell touches the spanning tree either at a point (a leaf node), or along a segment (an edge-segment emanating from a node). Since every sub-cell touches the spanning tree, the sub-cell path generated by circumnavigating the spanning tree passes through *every* sub-cell of the cells accessible from S. Since each cell is completely covered once its four sub-cells are visited by the covering tool, all cells accessible from S are covered. □

The following theorem summarizes the coverage properties of the three STC algorithms.

Theorem 1. The three STC algorithms: off-line STC, on-line STC, and ant-like STC, completely cover the continuous area underlying the accessible work-area grid. Moreover, the covering is optimal in the sense that there is no repetitive coverage of any point in this area.

The theorem follows from lemmas 3.1 and 3.3. The next theorem lists several performance bounds for the three STC algorithm. For simplicity, we assume that the off-line STC runs DFS rather than a weighted spanning-tree construction algorithm.

Theorem 2. Let N be the number of cells in the grid representing the accessible work area. Then the three STC algorithms cover this area in $O(N)$ steps, using a covering path of total length $L = 4ND$ where D is the tool size. Furthermore, the on-line STC requires $O(N)$ memory, while the ant-like STC requires $O(1)$ memory.

Let us sketch the derivation of these bounds. First, the bound $O(N)$ on the number of covering steps reflects the way all three STC algorithms operate. In each step a new sub-cell is being covered. Since there are $4N$ sub-cells in an N-cell grid, all three algorithms cover the grid in $4N$ steps. Next consider the covering path total length. The covering path is a rectilinear curve, in each of whose segments the tool traverses a distance D between adjacent sub-cells. Since there are $4N$ sub-cells, the total length of the

covering path is $4ND$. The O(N) memory requirement of the on-line STC algorithm allows it 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. However, this limitation can be alleviated by allowing the robot to operate in environments whose total area is estimated in advance. By installing memory according to the expected work-areas size, the on-line STC algorithm can cover these work-areas irrespective of their particular geometry. In contrast, the ant-like STC algorithm uses sub-cell markers to locally identify the spanning tree. The ant-like STC stores only the following two items, which require O(1) memory. The first is the coverage state of the starting cell S, which allows the algorithm to determine completion of coverage. (This item can be eliminated in a more sophisticated formulation of the algorithm.) The second is the identity of the spanning-tree edge along which the covering tool entered the current cell. This knowledge allows the robot to uniquely identify the four sub-cells of the current cell, which in turn allows the robot to properly scan the neighboring cells.

3.2. Competitive analysis

We now adapt the on-line STC algorithm to an algorithm called *exact STC*, which achieves exact coverage. Then we subject the exact STC algorithm to competitive analysis, that compares the length of the covering path generated by the algorithm to the optimal covering path with full information available. First we make several simplifying assumptions. We assume that the tool size D is sufficiently small so that the connectivity of the work-area is preserved by the grid approximation. According to lemma A.1 in the appendix, this assumption is automatically satisfied when the minimal gap between obstacles in the environment is at least $2R$, where $R = 2\sqrt{2}D$. The ensuing analysis holds true for environments with narrower gaps between obstacles, but we wish to avoid the overhead incurred by discussing such environments.

In order to achieve exact work-area coverage, we use lemma A.2 in the appendix. According to this lemma, any point of the work-area not included in the grid approximation lies at a distance of at most R from one of the obstacle boundaries. The robot thus augments the grid coverage with a sweep of a thickness-R neighborhood about the obstacle boundaries. Our third assumption is that the covering tool can freely rotate while following the contour of an obstacle boundary. Since the tool size is D and $R = 2\sqrt{2}D$, the freely rotating tool can sweep a thickness-R neighborhood about an obstacle using $2\sqrt{2}$ passes.

The exact STC algorithm simply runs the on-line STC algorithm with the following modification. Each time the current cell x has a neighbor cell occupied by an obstacle, the robot checks if this is a new obstacle. If it is a new obstacle, the robot halts the grid coverage and sweeps a thickness-R neighborhood about the obstacle. During this sweep, the robot marks the obstacle boundary as being already covered. The robot completes the sweep at the cell x, and from this cell it resumes the grid coverage according to the

on-line STC. Note that the exact STC algorithm requires an additional $O(N)$ memory in order to recognize which boundary cells have been already traced by the covering tool.

The covering path generated by the exact STC algorithm consists of segments of two types. Segments which correspond to the grid coverage, and segments associated with the sweep around obstacle boundaries. The total length of the segments of the first type is $4ND$ according to theorem 2, where N is the number of grid cells. The total length of the segments of the second type is as follows. The coverage of a thickness-R neighborhood about the obstacles requires *three* passes at distances of $D/2$, $3D/2$, and $5D/2$ from the obstacles' boundaries. It can be verified that under fairly general conditions[4] the length of the curve at a distance $3D/2$ is the average of the length of the other two curves. Hence the total length of the segments of the second type is 3Π, where Π is the total length of the curves at a distance $3D/2$ from the obstacles. Our next task it to derive a lower bound on the length of the optimal covering path. In order to simplify the derivation, we assume that the covering tool moves along a piecewise linear path, maintaining a fixed orientation along each linear segment.

Lemma 3.4 (Optimality bound). Let a continuous and connected work-area have a total area A. Then the length L of any piecewise-linear covering path of the work-area satisfies $L \geqslant A/D_{\max}$, where $D_{\max}$ is the tool's maximal sweeping width.

Proof. Let L_i be the length of the ith segment in the piecewise-linear covering path, and let D_i be the tool's covering width during this segment. Then for a complete work-area coverage $A \leqslant \sum_i L_i D_i \leqslant D_{\max} \sum_i L_i$. The covering path total length is given by $L = \sum_i L_i$. Hence any piecewise-linear covering path satisfies $L \geqslant A/D_{\max}$. □

In our case we assume that the covering tool always uses a sweeping width of D. The total length of the covering path generated by the exact STC algorithm is $4ND + 3\Pi \leqslant A/D + 3\Pi$, since $4ND^2 \leqslant A$. It follows that the *competitive ratio* of the exact STC algorithm is:

$$\frac{A/D + 3\Pi}{A/D} = 1 + 3\frac{\Pi D}{A}.$$

In practice $\Pi D \ll A$, and the exact STC algorithm typically achieves an almost optimal coverage of the work-area. However, the exact STC algorithm may perform poorly in certain environments. Figure 6 depicts a very long corridor of length H, whose width is twice the tool size D. Both ends of the corridor enter small rooms whose size is negligible with respect to the corridor's size. The work-area grid is aligned with the corridor and enters both rooms. The exact STC algorithm covers the grid using an optimal path of length $2H$, where we have neglected the covering of the two small rooms. The globally optimal covering path also has a length of $2H$. However, the exact STC algorithm additionally sweeps a thickness-R neighborhood about the outer walls. Since

[4] A sufficient condition is that the boundary of every obstacle is surrounded by a free neighborhood of thickness $3D$.

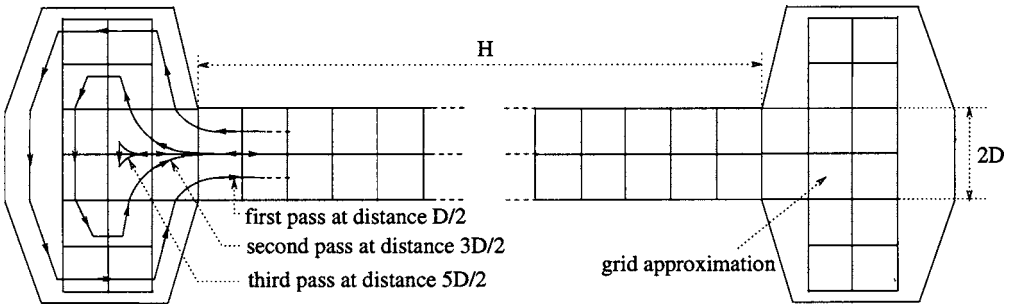

Figure 6. An adversarial environment for the exact STC algorithm.

the algorithm requires a complete sweep around the outer walls, the covering tool must pass through the corridor three times in each direction, or a total of six times, in order to complete three loops around the outer walls. Neglecting again the two rooms, the total length of the six passes through the corridor is $6H$. The total length of the covering path generated by the exact STC is $8H$, and the competitive ratio for this environment is $8H/2H = 4$. The poor performance of the exact STC algorithm in this environment is due to its inefficient sweep of a thickness-R neighborhood about the obstacles. It is reasonable to expect that more sophisticated boundary following methods would yield a tighter competitive ratio in such environments.

4. Simulation results

In this section we present simulation results of the three STC algorithms. We begin with an example of the off-line STC algorithm which demonstrates the possibility of using edge weighting to influence the covering pattern. Then we consider an example of the on-line and ant-like STC algorithms. Finally we consider an office environment in which the tool size D is an order-of-magnitude smaller than a door's width. In this case the grid closely approximates the work-area, and on average the STC algorithms achieve a 90% coverage of the work-area.

Our first example executes the off-line STC algorithm on the environment shown in figure 7. The off-line STC algorithm can use Prim's algorithm rather than DFS to pre-compute a spanning tree for the work-area. Prim's algorithm admits arbitrary edge weighting, and here we assign edge weights that give preference to vertical edges over horizontal edges. The resulting spanning-tree edges tend to be vertically aligned, and the covering tool consequently scans the work-area along vertical strips. (For comparison, the spanning tree generated for the same environment by DFS appears in figure 8(f).) Figure 7 shows four stages of the covering process. Figure 7(a) shows the spanning tree and the initial motion of the covering tool along the right-side of the spanning tree edges. Figure 7(b) shows the covering tool approaching the left edge of the work-area, using vertical covering sweeps. Figure 7(c) shows the covering tool entering the lower-right room. Finally, figure 7(d) shows the completion of coverage, where the covering tool has completed the circumnavigation of the spanning tree and returns to the starting cell.

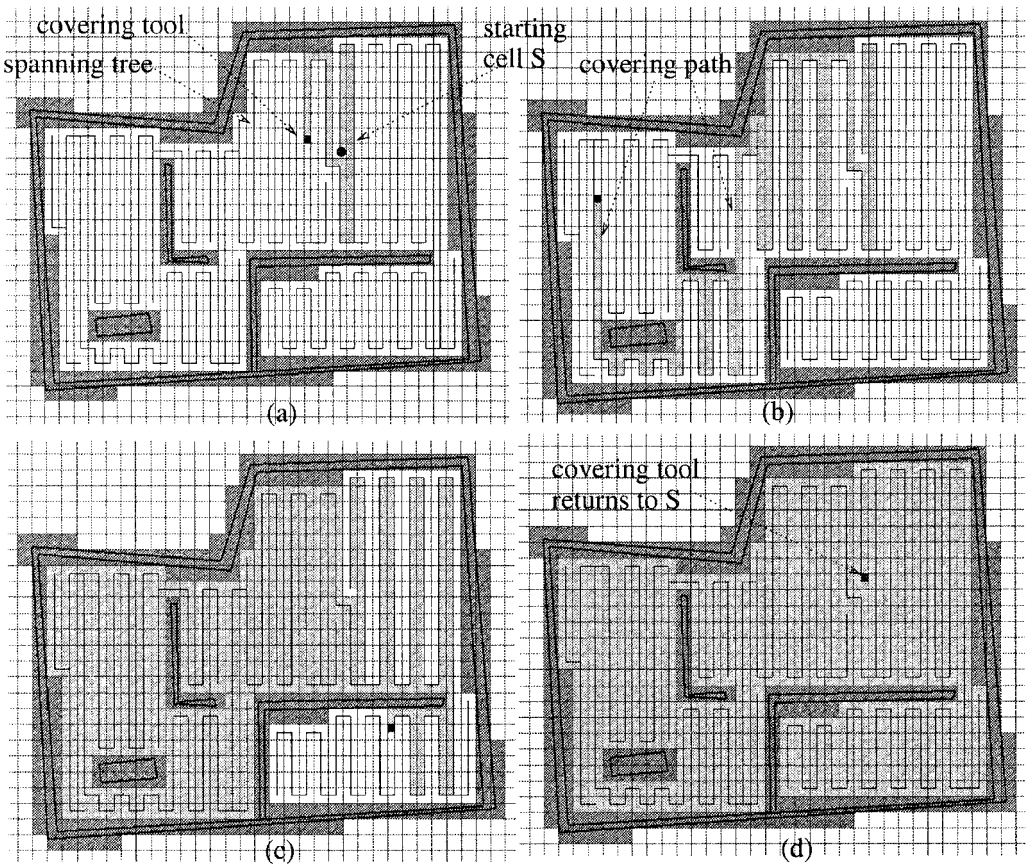

Figure 7. Four covering stages of the off-line STC algorithm using edge weighting.

The next example, shown in figure 8, runs the on-line STC and ant-like STC algorithms on the same environment. The two algorithms generate identical covering paths, and the covering stages shown in figure 8 represent the execution of both algorithms. However, the two algorithms operate internally in completely different ways. The on-line STC maintains a data structure of the incrementally constructed spanning tree, while the ant-like STC uses sub-cell markers to locally identify the spanning tree. In figure 8(a) the covering tool follows the right-side of the spanning tree edges along a sub-cell path that spirals outward from the starting cell S. In figure 8(b) the covering tool has reached the outer boundary, and it follows this boundary until it reaches the entrance to the lower-right room. In figure 8(c) the covering tool spirals inward to the center of the lower-right room, and in figure 8(d) it spirals outward along the complementary spiral, until it reaches the room's entrance. At this stage the lower-right room is completely covered. The covering tool next proceeds to the left portion of the environment. This portion is populated by two internal obstacles which divide the area into three cavities. Each of these cavities is covered by the same double-spiral pattern, and the resulting

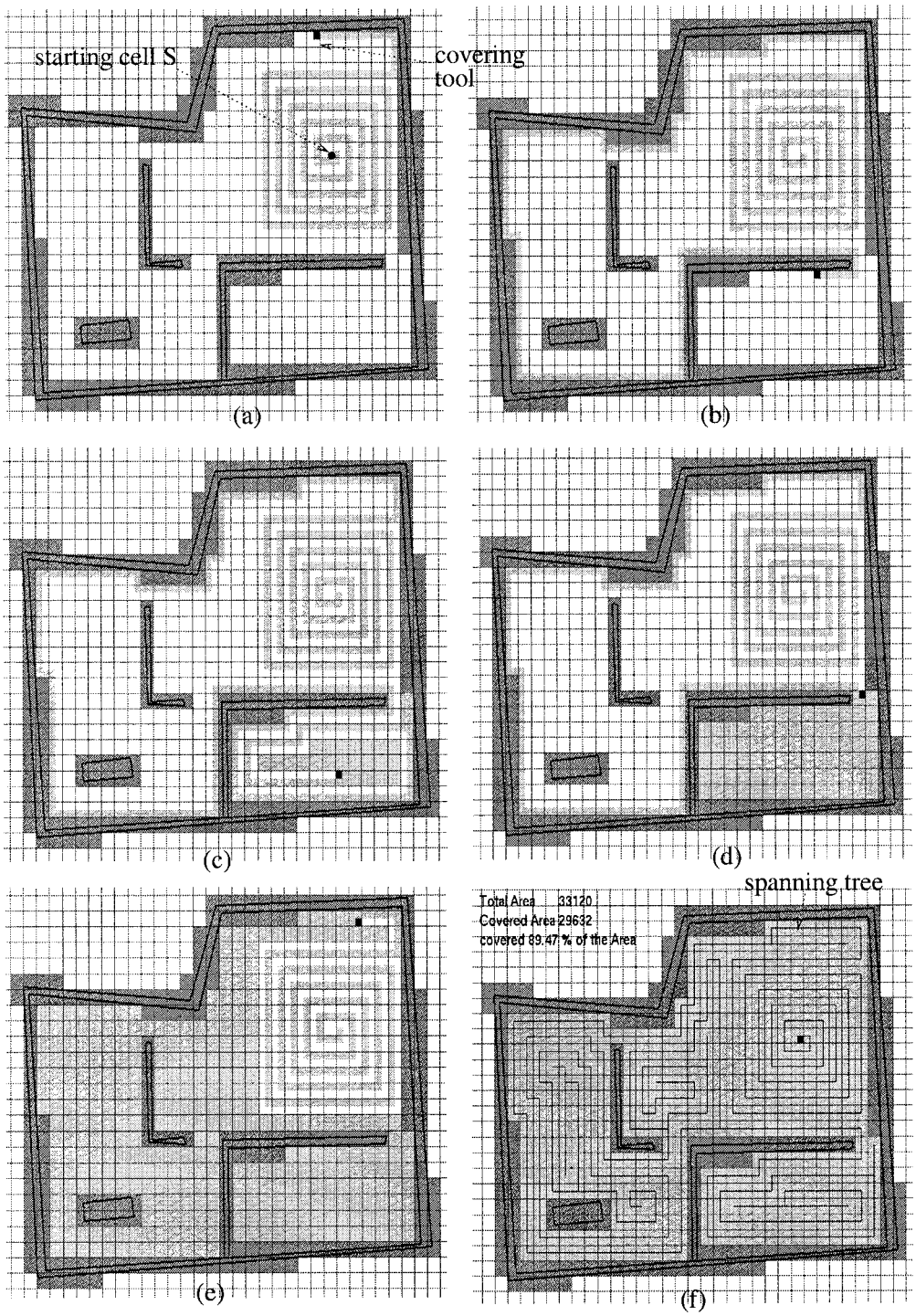

Figure 8. Six covering stages of the on-line and ant-like STC algorithms.

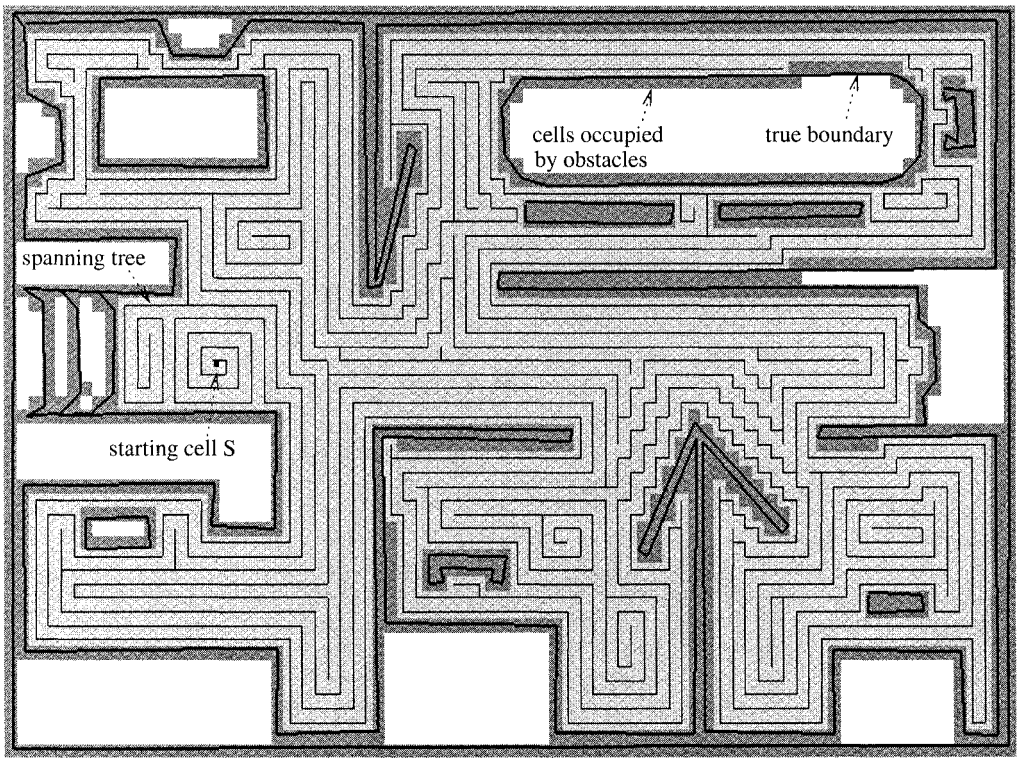

Figure 9. Coverage of an office environment using a tool size an order-of-magnitude smaller than a door's width.

coverage is shown in figure 8(e). Finally, the covering tool spirals inward to the starting cell S, resulting in a complete coverage of the work-area grid. The spanning tree constructed incrementally by the on-line STC algorithm appears in figure 8(f). The same spanning tree is locally traced by the ant-like STC algorithm using sub-cell markers. It should be noted that the covering path is optimal, in the sense that there is no repetitive coverage of any point in the area underlying the grid.

The last example, shown in figure 9, attempts to provide a realistic office environment of a startup company making robotic vacuum-cleaners for office and home environments. This example consists of several rooms and offices which are populated by pieces of furniture and other items. The tool size D is selected to be an order-of-magnitude smaller than a door's width. The execution of the on-line STC and ant-like STC algorithms on 100 instances of this environment yields a 85–95% coverage of the total area. In the particular office environment shown in figure 9, the covered area is 92% of the total area. Note that the portions of the work-area not covered by the STC algorithms are concentrated around the obstacle perimeters. Hence, it is possible to augment the grid coverage with a sweep around the obstacles' boundaries as described above, in order to achieve exact coverage of the entire work-area.

5. Conclusion

We presented and analyzed three spanning-tree based covering algorithms for continuous work-areas. The algorithms impose a grid approximation on the given work-area, then subdivide each cell into four identical sub-cells of size D, where D is the tool size. All three algorithms use the same procedure of constructing a spanning tree for the grid, then following a sub-cell path that circumnavigates the spanning-tree edges. Since each sub-cell is identical to the covering tool and the spanning tree visits every cell, the resulting sub-cell path completely covers the work-area grid in optimal time $O(N)$, where N is the number of grid cells. The quality of coverage depends on the tool size being significantly smaller than the gaps between obstacles in the environment. Our simulations show that in a typical office environment a tool size which is an order-of-magnitude smaller than a door's width yields an average cover of 90% of the work-area.

Each of the three STC algorithms has its own merits and caveats. The off-line STC algorithm requires apriori information about the environment. However, this algorithm is able to pre-compute spanning trees that guide the covering tool along desired patterns, such as parallel-line scanning. The on-line STC algorithm covers an environment whose geometry is not apriori known to the robot. This algorithm is therefore suitable for office and home settings, where the location of furniture and other items may vary between successive covering sessions. However, the on-line STC algorithm requires $O(N)$ memory, which imposes a limit on the size of the work-areas that can be covered by a limited-memory robot. The third STC algorithm uses ant-inspired markers which are left at each sub-cell being covered. Using these sub-cell markers, the ant-like STC algorithm is able to achieve complete coverage with only $O(1)$ memory. However, sub-cell marking mechanisms are still in an experimental development stage and are not readily available. Finally, we also described a fourth STC algorithm, which augments the coverage of the work-area grid with a sweep around the obstacles' boundaries. (The boundary sweep involves re-orientation of the covering tool.) The latter algorithm achieves an exact coverage with a competitive ratio of $1 + 3(\Pi D/A)$, where Π is related to the obstacles' perimeter and A is the total area. In typical office-like environments $\Pi D \ll A$, and in these environments the augmented STC algorithm achieves an almost optimal coverage of the entire work-area.

Following are several issues currently under investigation. First, we seek ways to control the geometry of the spanning tree constructed incrementally by the on-line STC algorithm. Current algorithms that compute spanning trees for weighted graphs are not suitable for on-line coverage, since these algorithms require discontinuous jumps between nodes. Second, we are investigating the use of several cooperative agents under the ant-like STC and on-line STC algorithms. In the ant-like STC algorithm, the agents can simply share the sub-cell markers during the coverage process. However, it has been shown that adding covering agents does not necessarily improve coverage time [10, 31]. The challenge here is to characterize this gain under the ant-like STC algorithm. In the on-line STC algorithm, it is currently not clear how to effectively coordinate a cooperative covering. Finally, we wish to develop covering algorithms that do not restrict

the covering tool to a rectilinear path. We are currently investigating ways to embed *curved* covering paths in a given work-area, such that the path's geometry is directly dictated by the work-area geometry.

We conclude with a brief description of ongoing work, where we extend the work-area grid to include partially occupied cells. First we impose a $2D$-size grid on the given work-area, then divide each $2D$-size cell into four D-size sub-cells. We define a node at every cell which contains at least one unoccupied sub-cell, and an edge between any two cells that share adjacent unoccupied sub-cells. The basic algorithm remains the same: construct a spanning-tree for the grid graph, then circumnavigate the spanning tree to achieve complete coverage of all unoccupied sub-cells. However, the circumnavigation through partially occupied cells incurs repetitive coverage of certain sub-cells. Our analysis indicates that a covering path can be computed in linear time with the following bound on its length. Let n be the total number of unoccupied sub-cells. Let $m \leqslant n$ be the number of *boundary sub-cells*, defined as sub-cells that share at least one point with the boundary of the work-area grid. Then the covering path length is bounded by $(n+m)D$. This bound is tight for certain non-Hamiltonian environments, and it provides a new type of approximation for the TSP problem on general grid graphs.

Acknowledgements

We wish to thank Israel Wagner, for his patient and wise advise during numerous discussions.

Appendix A. Properties of the work-area grid

The STC algorithm subdivides the work-area into $2D$-size cells and discards cells which are occupied by obstacles. In this appendix we characterize the quality of this approximation in terms of the work-area parameters. Given a starting cell S, the *accessible work-area* is the connected component of the work-area which contains the cell S and is accessible by the mobile robot. The following lemma characterizes the connectivity of the grid approximation.

Lemma A.1. The grid approximation of the accessible work-area contains all cells which can be accessed from the starting cell S by a path whose minimal distance from the obstacles is at least $R = 2\sqrt{2}D$, where D is the tool size.

The bound is obtained by considering two cells arranged diagonally with a vertex in common. If these two cells are occupied by obstacles, they may disconnect the work-area grid. We can in principal compute the minimum passage between obstacles in the work-area, then use the lemma to determine a tool size D which guarantees a grid approximation that preserves the connectivity of the accessible work-area. The next lemma characterizes the thickness of the area which is left out by the grid approximation.

Lemma A.2. Any point of the accessible work-area not included in the grid approximation lies at a distance of at most R from one of the obstacle boundaries, where $R = 2\sqrt{2}D$.

The lemma follows from the fact that the diameter of a $2D$-size cell is $2R$. Hence all points which lie in a cell occupied by obstacles are at most R away from one of the obstacles. The lemma provides a justification for the fourth version of the STC algorithm, which achieves exact coverage by augmenting the grid coverage with a sweep of an area of thickness R about the obstacle boundaries. The last lemma gives a formula for the total area not included in the work-area grid. First we introduce some notation. The total area of the given work-area is denoted A, while the total area of the grid approximation is denoted $\widehat{A}$. In the case of a polygonal environment, convex and concave obstacle vertices are defined as follows. At a convex vertex an obstacle protrudes into the free space, while at a concave vertex the free space protrudes into an obstacle. In the case of an environment bounded by smooth obstacles, we parametrize the obstacle boundaries by arc-length, using the curve $\alpha(s)$.

Lemma A.3. If the work-area is bounded by polygonal obstacles, the total area not included in the grid approximation is bounded by

$$A - \widehat{A} \leqslant \Delta R - \frac{1}{2}R^2\left(\sum_i (\pi - \alpha_i) - \sum_j \cot\left(\frac{\beta_j}{2}\right)\right),$$

where $R = 2\sqrt{2}D$, Δ is the obstacles' total perimeter, α_i are the angles of the concave obstacle vertices, and β_j are the angles of the convex obstacle vertices.

If the work-area is bounded by smooth obstacles, the total area not included in the grid approximation is bounded by

$$A - \widehat{A} \leqslant R\int_0^{\Delta} \left\|\alpha'(s)\right\| \mathrm{d}s + \frac{1}{2}R^2 \int_0^{\Delta} \frac{\kappa(s)}{\|\alpha'(s)\|^2}\,\mathrm{d}s,$$

where $\alpha(s)$ parametrizes the obstacles' boundaries, $\alpha'(s)$ is the unit tangent at $\alpha(s)$, and $\kappa(s)$ is the curvature of the obstacles' boundaries at $\alpha(s)$, for $0 \leqslant s \leqslant \Delta$.

A proof of the lemma is omitted for the sake of brevity. The bounds for $A - \widehat{A}$ are quadratic in D, and in principal can be used to determine a tool size which would guarantee coverage within a desired area tolerance.

References

[1] D. Angluin, J. Westbrook and W. Zhu, Robot navigation with range queries, in: *Proc. of STOC* (1996) pp. 469–478.

[2] E.M. Arkin, S.P. Fekete and J.S. Mitchell, The lawnmower problem, in: *Proc. of 5th Canadian Conf. on Computational Geometry*, Waterloo, Canada (1993) pp. 461–466.

[3] E.M. Arkin, S.P. Fekete and J.S. Mitchell, Approximation algorithms for lawn mowing and milling, ftp.zpr.uni-koeln.de/pub/paper/zpr97-255.ps.gz 97.255, University of Koein (1997).
[4] E.M. Arkin and R. Hassin, Approximation algorithms for the geometric covering salesman problem, Discrete Appl. Math. 55 (1994) 197–218.
[5] A. Blum, P. Raghavan and B. Schieber, Navigating in unfamiliar terrain, in: *Proc. of STOC* (1991) pp. 494–504.
[6] Z.J. Butler, A.A. Rizzi and R.L. Hollis, Complete distributed coverage of rectlinear environments, in: *Proc. of Fourth Workshop on Algorithmic Foundations of Robotics*, Hanover, New Hampshire, 2000.
[7] H. Choset and J.W. Burdick, Sensor based planning, Part II: Incremental construction of the generalized Voronoi graph, in: *IEEE Int. Conference on Robotics and Automation* (1995) pp. 1643–1649.
[8] H. Choset and P. Pignon, Coverage path planning: The boustrophedon decomposition, in: *Proc. of Int. Conf. on Field and Service Robotics,* Canberra, Australia, December 1997.
[9] X. Deng, E. Milios and A. Mirzaian, Robot map verification of a graph world, in: Lecture Notes in Computer Science, Vol. 1663 (Springer, Berlin, 1999) pp. 86–97.
[10] X. Deng and A. Mirzaian, Competitive robot mapping with homogeneous markers, IEEE Trans. Robotics Autom. 12(4) (1996) 532–542.
[11] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, Robotic exploration as graph construction, IEEE Trans. Robotics Autom. 7(6) (1991) 859–865.
[12] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, Map validation and robot self-location in a graph-like world, Robotics Autonom. Syst. 22 (1997) 159–178.
[13] E. Bar Eli, P. Berman, A. Fiat and P. Yan, Online navigation in a room, J. Algorithms 17(3) (1996) 319–341.
[14] H. Enders, W. Feiten and G. Lawitzky, Field test of navigation system: Autonomous cleaning in supermarkets, in: *Proc. of IEEE Int. Conf. on Robotics and Automation* (1998) pp. 1779–1781.
[15] S. Hedberg, Robots cleaning up hazardous waste, AI Expert (May 1995) pp. 20–24.
[16] M. Held, *On the Computational Geometry of Pocket Machining* (Springer, New York, 1967).
[17] S. Hert, S. Tiwari and V.J. Lumelsky, A terrain covering algorithm for an AUV, Autonom. Robots 3 (1996) 91–119.
[18] C. Icking and R. Klein, Competitive strategies for autonomous systems, Technical Report CS:175, University of Hagen, Germany (1995).
[19] A. Itai, C. Papadimitrio and J. Szwarcfiter, Hamiltonian paths in grid graphs, SIAM J. Comput. 11 (1982) 676–686.
[20] J.D. Nicoud and M.K. Habib, The PEMEX autonomous demining robot: Perception and navigation strategies, in: *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robot Systems* (1995) pp. 1:419–424.
[21] S. Ntafos, Watchman routes under limited visibility, Comput. Geom. Theory Appl. 1 (1992) 149–170.
[22] U. Peles and S. Abramson, *RoboMow by Friendly Machines*, www.friendlymachines.com, Israel, 1998.
[23] A. Pirzadeh and W. Snyder, A unified solution to coverage and search in explored and unexplored terrains using indirect control, in: *Proc. of IEEE Int. Conf. on Robotics and Automation* (1990) pp. 2113–2119.
[24] N.S.V. Rao and S.S. Iyengar, Autonomous robot navigation in unknown terrains: visibility graph based methods, IEEE Trans. Systems Man Cybern. 20(6) (1990) 1443–1449.
[25] E. Rimon, Construction of c-space roadmaps from local sensory data: What should the sensors look for? Algorithmica 17(4) (1997) 357–379.
[26] R.A. Russell, Laying and sensing odor markings as a strategy for assisting mobile robot navigation tasks, IEEE Robotics Autom. Magazine 2 (1995) 3–9.
[27] R.A. Russell, Heat trails as short-lived navigational markers for mobile robots, in: *Proc. of IEEE Int. Conf. on Robotics and Automation* (1997) pp. 3534–3539.
[28] R.E. Tarjan, *Data Structures and Network Algorithms* (SIAM, Philadelphia, PA, 1983).
[29] C.J. Tayior and D.J. Kriegman, Vision based motion planning and exploration algorithms for mobile robots, in: *Proc. of Workshop on Algorithmic Foundations of Robotics* (A.K. Peters, 1995) pp. 69–83.

[30] I.A. Wagner, M. Lindenbaum and A.M. Bruckstein, Distributed covering by ant-robots using evaporating traces, IEEE Trans. Robotics Autom. 15(5) (1999) 918–933.
[31] I.A. Wagner, M. Lindenbaum and A.M. Bruckstein, Efficiently searching a graph by a smell-oriented vertex process, Ann. Math. Artif. Intell. 24 (1998) 211–223.
[32] H. Yaguchi, Robot introduction to cleaning work in the East Japan Railway Co, Adv. Robotics 10(4) (1996) 403–414.