Proceedings of the 2002 IEEE
International Conference on Robotics & Automation
Washington, DC • May 2002

Spiral-STC: An On-Line Coverage Algorithm of Grid Environments by a Mobile Robot

Yoav Gabriely and Elon Rimon
Technion, Israel Institute of Technology
meeryg@tx.technion.ac.il, elon@robby.technion.ac.il

Abstract *We describe an on-line sensor based algorithm for covering planar areas by a square-shaped tool attached to a mobile robot. Let D be the tool size. The algorithm, called* Spiral-STC, *incrementally subdivides the planar work-area into disjoint D-size cells, while following a spanning tree of the resulting grid. The algorithm covers general grid environments using a path whose length is at most $(n+m)D$, where n is the number of D-size cells and $m \leq n$ is the number of* boundary cells, *defined as cells that share at least one point with the grid boundary. We also report that any on-line coverage algorithm generates a covering path whose length is at least $(2-\epsilon)l_{opt}$ in worst case, where l_{opt} is the length of the optimal covering path. Since $(n+m)D \leq 2l_{opt}$,* Spiral-STC *is worst-case optimal. Moreover, $m << n$ in practical environments, and the algorithm generates close-to-optimal covering paths in such environments. Simulation results demonstrate the spiral-like covering patterns typical to the algorithm.*

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 accessible point of the work-area is eventually covered by the tool. In the on-line version of the problem the robot collects information about the obstacles during the coverage process. Note that on-line coverage is *distinct* from on-line mapping or area exploration by a mobile robot. In the latter problem an autonomous robot has to completely scan an unknown environment with its sensors [12, 20]. In contrast, area coverage requires physical sweep of a tool over every point of a given work-area. The mobile robot covering problem has several important applications such as floor cleaning and coating [7], hazardous waste cleaning [10], field demining [17], and lawn mowing [18]. Moreover, while we focus on mobile robot coverage, similar methods apply to other robotic coverage tasks such as part machining [11] and car painting [2].

An optimal solution of the off-line covering problem generalizes the Traveling Salesperson Problem (TSP) to continuous domains [1]. Hence it is natural to seek competitive algorithms for the on-line covering problem. (An on-line 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].) The on-line covering problem has been first considered by Kalyanasundaram and Pruhs [16], in the context of on-line TSP tours on weighted planar graphs. They have shown that the problem has a competitive ratio of sixteen. In the special case of graphs representing grid environments the edges have uniform weights. The trivial DFS algorithm covers such graphs with a path whose length is at most twice the length of the optimal path. Our *Spiral-STC* algorithm runs on general grid environments and has the same competitive ratio as DFS. However, *Spiral-STC* generates a covering path whose length is bounded by $(n+m)D$ (n being the total number of grid cells, $m \leq n$ the number of boundary cells defined below, and D the tool size), while DFS generates a path whose length is always $2nD$. In practice $m << n$, and *Spiral-STC* significantly improves on DFS.

In the robotics literature, all the competitive on-line covering algorithms proposed so far use environmental markers such as pebbles [5, 6] or pheromone-like traces [21] to aid the robot's coverage process. We solely rely on the robot's on-board resources during the coverage process. The mobile robot covering problem is also studied by Butler et al. [3], Choset and Pignon [4], Pirzadeh and Snyder [19], and Huang [13]. All of these works emphasize the importance of achieving on-line sensor based coverage.

We make the following assumptions. First, we assume that the covering 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 approximate the continuous work-area by a discrete grid of D-size cells as shown in Figure 1. A preliminary version of the algorithm was presented at Icra 2001 [9]. However, the preliminary algorithm covers

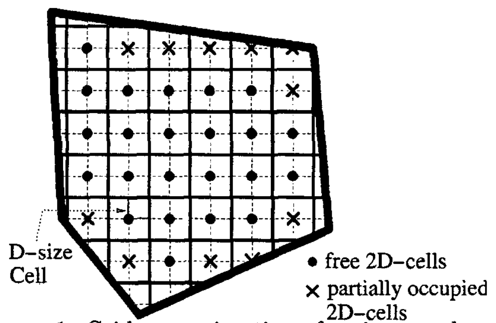

Figure 1: Grid approximation of a given work-area.

only special grids of $2D{\times}2D$ cells, while the full algorithm covers completely general grids of $D \times D$ cells. Next, we assume that the environment is populated by stationary obstacles. Finally, we assume that the robot has no apriori knowledge of the environment. Rather, the robot must use its sensors to detect obstacles while covering the work-area grid.

The paper makes the following two contributions. The first is description and analysis of the sensor-based covering algorithm *Spiral-STC.* The algorithm is presented in Section 2 and analyzed in Section 3. The main result is that *Spiral-STC* covers any planar grid environment in $O(n)$ time using a path whose length is at most $(n{+}m)D$. In this bound n is the number of D-size cells and $m \leq n$ is the number of *boundary cells,* defined as cells that share at least one point with the grid boundary. The second contribution reports a universal lower bound on the competitiveness of on-line coverage algorithms. It is demonstrated in Section 4 that any on-line coverage algorithm generates a covering path whose length is at least $(2{-}\epsilon)l_{opt}$ in worst case, where l_{opt} is the length of the shortest covering path and ϵ is an arbitrarily small positive parameter. In the case of *Spiral-STC,* $(n{+}m)D \leq 2l_{opt}$ (since $l_{opt} \geq nD$), hence *Spiral-STC* is worst-case optimal. Finally, simulation results of the algorithm demonstrate its effectiveness in office-like environments.

2 The Spiral-STC Algorithm

We first review 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. For notational simplicity $2D$-size cells are simply called *cells,* while D-size cells are called *subcells.*

2.1 A 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 (Figure 1). The algorithm incrementally constructs a spanning tree for this graph and uses 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 four-neighbors obstacle detection sensor.
Input: A starting cell $S$, but no apriori 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 counter-clockwise 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.
4. Return. (End of $STC1(w,x)$.)
```

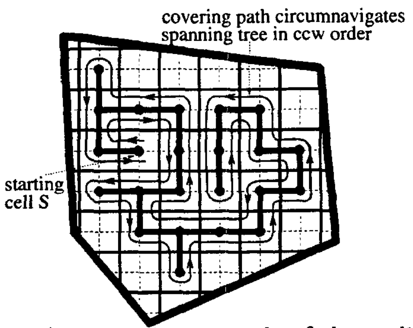

Figure 2: An execution example of the preliminary *2D-Spiral-STC* algorithm.

We now discuss several details of the preliminary algorithm. First, the counterclockwise scanning of neighbors specified in step 2.1 ensures that the covering tool circumnavigates the incrementally constructed spanning tree in counterclockwise order (Figure 2). 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. 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.

An execution example of the preliminary algorithm is illustrated in Figure 2. It can be seen that the circumnavigation of the spanning tree generates a 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.

2.2 The Full *Spiral-STC* Algorithm

The full algorithm augments the preliminary algorithm with coverage of *partially occupied* $2D$*-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

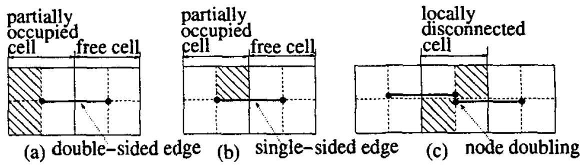

Figure 3: (a) A single-sided, and (b) double-sided edge. (c) Node doubling at a disconnected cell.

double-sided edges, possess only free subcells on both sides (Figure 3(a)). The other type, called *single-sided edges*, possess at least one occupied subcell on either side (Figure 3(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 may be able to 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 (Figure 3(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.

Full Spiral-STC Algorithm:

```
Initialization: Call $STC2(Null, S)$, where $S$ is the starting cell.
$STC2(w,x)$:
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 $STC2(x,y)$.
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 $w$.
4. Return. (End of $STC2(w,x)$.)
```

Let us describe the subcell path taken by the cover-

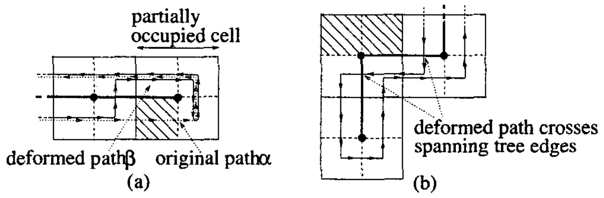

Figure 4: (a) Path deformation along a single-sided edge. (b) Crossing of spanning-tree edges.

ing tool along a single-sided edge in steps 2.3 and 3. Consider for concreteness the partially occupied cell depicted in Figure 4(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.* Figure 4(a) shows the original path α as well as the deformed path β. Note that we use a maximum metric[1] 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 spanning-tree edges emanating from a partially occupied cell, as shown in Figure 4(b).

An execution example of the full algorithm on the environment used to execute the preliminary algorithm is illustrated in Figure 5. Note that repetitive coverage occurs at places where the subcells on the right-side of the spanning-tree edges are occupied by obstacles. It is shown below 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 patterns typical to this algorithm.

3 Analysis of Spiral-STC

In this section we first consider several properties of *Spiral-STC*, then analyze the amount of repetitive coverage generated by the algorithm.

Lemma 3.1 (completeness). *The* Spiral-STC *algorithm covers every free subcell accessible from the starting cell S.*

Proof: The algorithm constructs a spanning tree that reaches every free and partially occupied cell in the accessible work-area grid. These cells are partitioned into subcells. By construction, every obstacle-free subcell touches the spanning tree either at a point (a leaf node), or along a segment (part of a spanning-tree edge). 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. The algorithm deforms the path α away from the occupied subcells, such that the deformed path passes only through free subcells. The deformed path passes through every free subcell which lies along the original path α. Hence every free subcell accessible from S is covered. □

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

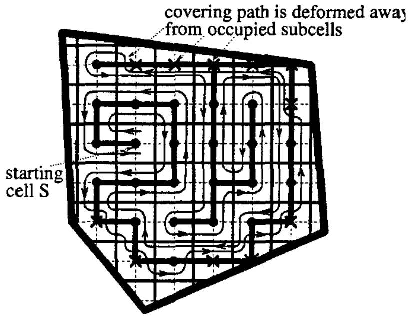

Figure 5: An execution example of the full *Spiral-STC* algorithm.

The next lemma gives the run-time and memory requirement of the *Spiral-STC* algorithm.

Lemma 3.2. *Let n be the number of free subcells accessible from S. Then* Spiral-STC *covers 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 $m \leq n$ 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 Ref. [9] we describe an STC algorithm that uses cell markers to identify previously visited subcells. This algorithm requires only $O(1)$ memory.

The following key theorem provides a bound on the length of the covering path generated by *Spiral-STC*. Recall that *boundary subcells* are obstacle-free subcells that share either a point or a segment with the

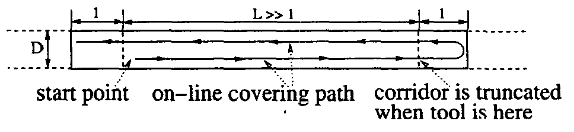

Figure 6: A corridor environment with the best on-line covering path depicted.

boundary of the work-area grid.

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

A proof of the theorem appears in Ref. [8]. The $m+n$ bound does not reflect an important property of the algorithm, that its repetitive coverage is localized around partially occupied cells (Figure 8). 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. Then $k \leq m$ and the length of the covering path is bounded in terms of k as follows.

Corollary 3.3. *Let n be the number of free subcells in the accessible work-area grid. Let $k \leq m$ be the number of free subcells that touch occupied subcells contained in partially occupied cells. Then* Spiral-STC *covers the work-area grid using a path of total length $l \leq (n+k)D$, where D is the tool size.*

Finally consider the competitive ratio of *Spiral-STC*, defined as l/l_{opt} where l_{opt} is the length of the optimal coverage path. The total area of the grid is $A = nD^2$, and the area occupied by the boundary subcells is $\partial A = mD^2$. The optimal coverage path satisfies $l_{opt} \geq A/D$, and the length of the coverage path generated by *Spiral-STC* satisfies $l \leq (n+m)D = A/D + \partial A/D$. It follows that the competitive ratio of *Spiral-STC* satisfies $l/l_{opt} \leq 1 + \partial A/A$. In practice $\partial A << A$, and *Spiral-STC* achieves a close-to-optimal coverage of practical work-area grids.

4 A Universal On-Line Coverage Bound

This section reports a lower bound on the path-length of any on-line covering algorithm for planar environments[2]. First let us demonstrate the bound in a simple corridor whose width is identical to the tool size D. Suppose the robot 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>>1. At that instant we truncate the corridor at a unit distance from both sides of the length-L portion, as shown in Figure 6. Since the robot has no way of knowing about this truncation, 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 $l \geq 2L+3$, since the covering tool must visit both ends of the corridor before coverage is complete. The optimal off-line covering path, achieved by sweeping the corridor from end to end, has length $l_{opt} = L+2$. Thus $l/l_{opt} \geq 2-\epsilon$.

[2]The same bound was independently obtained by Icking et al. in a paper currently in press [14].

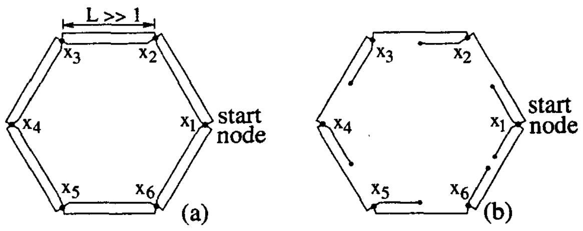

Figure 7: (a) A circular double-ring environment. (b) The modified environment resulting from k incisions.

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. The more sophisticated environment depicted in Figure 7 is circular, and the lower bound is established in this environment both for covering paths and covering 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 $(2-\epsilon)l_{opt}$, where l_{opt} is the length of the shortest off-line covering path, and $\epsilon > 0$ is an arbitrarily small parameter. Moreover, the bound is ϵ-tight, since there exist on-line algorithms (such as Spiral-STC and DFS) that generate covering paths whose length is at most $2l_{opt}$.*

The circular environment which is used to establish the bound consists of $2k$ edges arranged in parallel pairs (Figure 7(a)). Each edge has a length $L >> 1$, where the robot's detection range is $1-\delta$. During the execution of the algorithm, we perform k local incisions that remove portions of the edges whenever the robot reaches within one-unit distance of a new

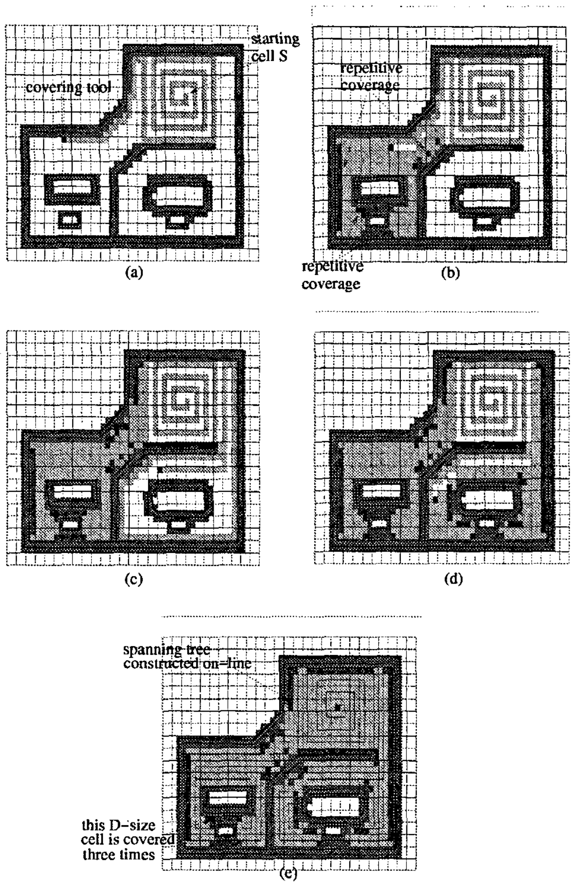

Figure 8: Five covering stages of *Spiral-STC*, with repetitive coverage depicted in dark gray.

node. Much like the corridor environment, here too the removed portions lie beyond the robot's detection range. Hence the robot generates the same on-line covering path in the modified environment (Figure 7(b)). The technical details of the proof appear in Ref. [8]. We note that the on-line coverage bound is valid for continuous as well as discretized planar environments. Moreover, the covering tool may move along arbitrary paths, and the robot's range sensor may have an arbitrary finite detection range.

5 Simulation Results

We now present simulations of *Spiral-STC* in an environment that resembles several rooms populated by pieces of furniture (Figure 8). In Figure 8(a) the covering tool follows the right-side of the spanning-tree edges along a subcell path that initially 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 Figure 8(b) the covering tool spirals inward to the area between the internal obstacles at the center of the lower-left room (the corresponding spanning tree is shown in Figure 8(e)). 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. Note that 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 Figure 8(c) the covering tool proceeds to the lower-right room, and in Figure 8(d) this room is covered with the same double-spiral pattern. 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. Figure 8(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.

6 Conclusion

We presented and analyzed an on-line sensor based mobile robot covering algorithm. The algorithm discretizes the given work-area into a grid of D-size cells, then follows a path that circumnavigates a spanning-tree constructed for a coarser grid of $2D$-size cells. The algorithm treats partially occupied $2D$-cells as special nodes of the spanning tree that incur repetitive coverage. The algorithm covers the 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 $m \leq n$ is the number of D-size cells along the grid boundary. The 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. Finally, we reported that any on-line coverage algorithm generates a covering path whose length is at least $(2-\epsilon)l_{opt}$ in worst case. Since $(n+m)D \leq 2l_{opt}$, *Spiral-STC* is worst-case optimal. The trivial DFS is also worst-case optimal. However, DFS always generates a path whose length is $2nD$. In contrast, *Spiral-STC* generates a path whose length is at most $(n+m)D$, where $m << n$ in practical environments.

Let us conclude with a mention of an open on-line coverage problem. 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 an uncovered spiral pattern that has to be covered at a later stage. To our knowledge, none of the existing on-line covering algorithms guarantees a return-time bound.

References

[1] E. M. Arkin, S. P. Fekete, and J. S. Mitchell. Approximation algorithms for lawn mowing and milling. Tech. report 97.255, Mathematics Inst., University of Koeln, 1997.

[2] P.N. Atkar, H. Choset, A.A. Rizzi, and E.U. Acar. Exact cellular decomposition of closed orientable surfaces embedded in $\mathbb{R}^3$. *Icra*, pages 699–704, 2001.

[3] Z. J. Butler, A. A. Rizzi, and R. L. Hollis. Complete distributed coverage of rectlinear environments. In *Fourth Int. Workshop on Algorithmic Foundations of Robotics*, Hanover, NH, 2000.

[4] H. Choset and P. Pignon. Coverage path planning: The boustrophedon decomposition. In *Int. Conf. on Field and Service Robotics*, Canberra, Australia, 1997.

[5] X. Deng and A. Mirzaian. Competitive robot mapping with homogeneous markers. *IEEE Trans. on Robotics and Automation*, 12(4):532–542, 1996.

[6] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Trans. on Robotics and Automation*, 7(6):859–865, 1991.

[7] H. Enders, W. Feiten, and G. Lawitzky. Field test of navigation system: Autonomous cleaning in supermarkets. *Icra*, pages 1779–1781, 1998.

[8] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. Tech. report, Dept. of ME, Technion, www.technion.ac.il/~robots, July 2001.

[9] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Icra*, pages 1927–1933, 2001.

[10] S. Hedberg. Robots cleaning up hazardous waste. *AI Expert*, pages 20–24, May 1995.

[11] M. Held. *On the Computational Geometry of Pocket Machining.* Springer-Verlag, 1987.

[12] S. Hert, S. Tiwari, and V. J. Lumelsky. A terrain covering algorithm for an AUV. *Autonomous Robots*, 3:91–119, 1996.

[13] W. H. Huang. Optimal line-sweep based decomposition for coverage tasks. *Icra*, pp 27–32, 2001.

[14] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. *To appear in Lecture Notes in Computer Science.*

[15] C. Icking and R. Klein. Competitive strategies for autonomous systems. In H. Bunke, T. Kanade, and H. Noltemeier, *Modelling and Planning for Sensor Based Intelligent Robot Systems*, 23–40. World Scientific, Singapore, 1995.

[16] B. Kalyanasundaram and K. R. Pruhs. Constructing competitive tours from local information. In *Lecture Notes in Computer Science*, volume 700, pages 102–113. Springer Verlag, 1993.

[17] J. D. Nicoud and M. K. Habib. The PeMex autonomous demining robot: Perception and navigation strategies. In *IEEE/RSJ Int. Conf. on Intelligent Robot Systems*, pages 1:419–424, 1995.

[18] U. Peles and S. Abramson. RoboMow by *Friendly Machines.* www.friendlymachines.com, Israel, 1998.

[19] A. Pirzadeh and W. Snyder. A unified solution to coverage and search in unexplored terrains using indirect control. *Icra*, pages 2113–2119, 1990.

[20] N. S. V. Rao and S. S. Iyengar. Autonomous robot navigation in unknown terrains: visibility graph based methods. *IEEE Trans. on Systems, Man and Cybernetics*, 20(6):1443–1449, 1990.

[21] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Trans. on Robotics and Automation*, 15(5):918–933, 1999.