A Log-Approximation for Coverage Path Planning with the Energy Constraint

Minghan Wei, Volkan Isler
University of Minnesota
{weixx526, isler}@umn.edu

Abstract

We consider the problem of covering an environment with a robot when the robot has limited energy budget. The environment is represented as a polygon with a grid, whose resolution is proportional to the robot size, imposed on it. There is a single charging station in the environment. At each time step, the robot can move from one grid cell to an adjacent one. The energy consumption when moving in the environment is assumed to be uniform and proportional to the distance traveled. Our goal is to minimize both the total distance and the number of visits to the charging station. We present a coverage path planning algorithm which has $O(\ln D)$ approximation factor for both objectives, where D is the distance of the furthest cell in the environment measured on the grid.

Introduction

Many practical applications, such as autonomous sweeping, vacuum cleaning, and lawn mowing, require robots to fully cover an area. Coverage path planning is a well-studied problem in robotics. The goal is to plan paths for the robots so that the robot can visit every point in the area.

A common assumption when modeling the coverage problem is that the robots have an unlimited energy budget for moving arbitrary long distances. Therefore, a single path is planned to cover the given environment. This version of the problem is well studied in the literature (Galceran and Carreras 2013). Many algorithms have been proposed, where the robots know the boundaries (including obstacles) of the environment, such as the boustrophedon decomposition coverage with back-and-forth motion (Choset 2000) (Mannadiar and Rekleitis 2010), the spiral path coverage (Gonzalez et al. 2005), and the spanning-tree based coverage (Gabriely and Rimon 2001). Yoav and Elon restrict the robot to move rectilinearly and prove that their algorithm is optimal when there is no zero-thickness in the spanning tree (Gabriely and Rimon 2001). The boustrophedon coverage and spiral path can be adapted to perform online (Viet et al. 2015) (Choi et al. 2009). In the online setting, the robot does not know the environment at the beginning, but it can accumulate the knowledge of the environment.

In practice, robots operate with energy constraints. A battery-powered robot needs to go back to the charging station to get recharged before the battery runs out. The coverage path planning problem with energy constraints is a relatively new topic. With energy constraints, instead of a single path, we need to plan multiple paths for the robot since the robot cannot visit all the points in the environment after a full recharge. As an example, consider an environmental monitoring application with an aerial robot, as shown in Fig. 1. The battery can last 15 mins after a full recharge. We need to plan multiple paths to fully cover the environment.

Existing literature has integrated energy constraints by monitoring the energy level of the robot during the coverage. Strimel et al. use boustrophedon cellular decomposition to cover the environment. The robot returns to the charging station when its energy level is too low to continue (Strimel and Veloso 2014). Mishra et al. design a coverage system consisting of multiple robots (Mishra et al. 2016). To continuously cover the environment, the robots are divided into two groups, namely workers and helpers. When a worker needs to go back to recharge, an associated helper will continue the worker's coverage. These methods completely cover the environment, but they do not compare the results with the optimal solution to get a performance guarantee.

Shnaps and Rimon study this problem. They model the energy consumption by the length of the path (Shnaps and Rimon 2016). They present an approximation algorithm with a factor of $\frac{1}{1-\rho}$ when the environment is known, where ρ is the ratio between the furthest distance in the environment and half of the energy budget. This factor can be arbitrarily large when ρ approaches 1. They also propose an

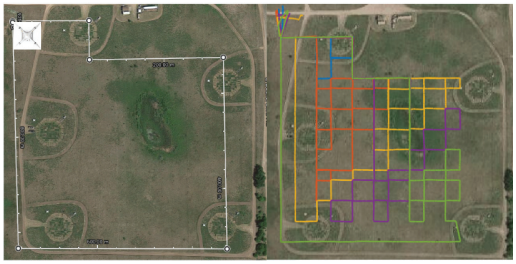


Figure 1: With energy constraints, the drone needs multiple paths to cover the environment. Each path is shown with a different color.

approximation algorithm and a lower bound for this problem in the online setting.

A related topic in the literature is the coverage of a graph. The goal is to design paths to visit every vertex of the given graph. Without energy constraints, it becomes the well-known Traveling Salesperson Problem (TSP). With energy constraints, this coverage problem is Vehicle Routing Problem (VRP) (Laporte 1992). One version of VRP is the Distance Vehicle Routing Problem (DVRP), which models the energy consumption proportional to the distance traveled. For DVPR on tree metrics, (Nagarajan and Ravi 2012) proposed a 2-approximation algorithm. Li et al. (Li, Simchi-Levi, and Desrochers 1992) uses a TSP-partition method and the algorithm has a similar approximation rate to the work in (Shnaps and Rimon 2016).

Coverage with multiple robots has also received a lot of attention. In some cases, the paths planned for a single robot under energy constraints can also be executed by multiple robots by assigning the paths to the robots. Using a single robot or multiple robots does not affect the total energy cost.

Our contribution: We revisit the setting in Shaps and Rimon's work (Shnaps and Rimon 2016). We present a $O(\ln D)$ approximation algorithm for the coverage path planning problem under the energy constraint when the robot is restricted to rectilinear motion. Compared with prior work, the deviation of the cost of our algorithm from the optimal cost remains bounded. We also demonstrate our algorithm with a simulation.

Problem Formulation

The environment P is represented as a unit-grid laid out on a polygon with a single charging station S. The robot is represented as a unit square that moves rectilinearly in P. The robot's energy consumption is assumed to be proportional to the distance traveled. We use path length and energy cost interchangeably. The robot is subject to the energy constraint and can only move at most B (the energy budget) units after a full charge. Then it has to go back to the charging station to get recharged. The goal is to find a set of paths, $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, for the robot such that $\bigcup_{i=1}^{n} \pi_i = P$ and the number of paths in Π is minimized. In addition, each path should start and end at S and the path length of each π_i should be within the energy budget.

Use $|\pi_i|$ to denote the length of π_i ($|\pi_i| \leq B$). A related goal is to minimize the total length of paths in Π: $\sum_{i=1}^{n} |\pi_i|$.

Preliminaries

In this section, we give the definition of *contour-connected* environments. We also discuss the robot's motion model. Both are critical to our analysis.

We use *approximate cellular decomposition* method to process the environment (Choset 2001). In other words, we decompose the environment into unit grids of the same size as the robot. We call these grids *cells*. Let S be the grid with the charging station. An *equi-distance contour* is a polyline where the cells on it have the same distance to S (as shown by the lines in Fig. 2). In the rest of the paper we use the term *contour* for short. The cells on a contour can be *ordered* from one side to the other. It is easy to see that we can also *order* the contours based on their distance to S in a strictly increasing fashion.

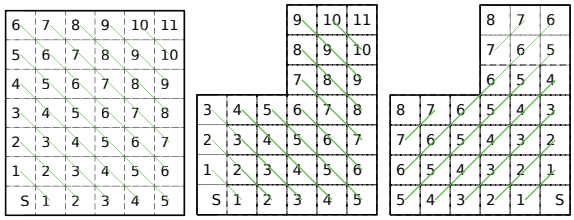


Figure 2: From left to right, (1)convex contour-connected. (2) non-convex contour-connected. (3) non-contour-connected. The value in each cell shows its $L1$ distance to the charing station S.

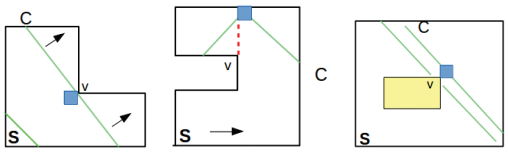


Figure 3: The two types of split cells (as shown by the square) $(a), (b)$: type-i. The inner rectangle (yellow) represents an obstacle. (c) type-ii.

We use $d(\cdot)$ to denote the distance to S. $d(c)$ is the distance from a cell c to S, and $d(C)$ is the distance from a contour C (i.e., the distance of any cell on C) to S. If $d(C_j) = d(C_i) + 1$, we say C_j is C_i's next contour. We refer to the contour C with $d(C) = 1$ as the first contour.

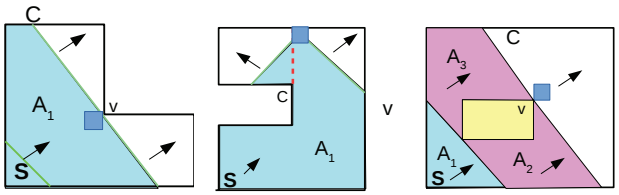
A *split cell* is a cell on a contour which is adjacent to the boundary and not on the endpoints of the contour. Let v be a split cell on a contour C. Based on C's next contour, there are two types of split cells. (i) From v the next contour splits into two segments, as shown in Fig. 3(a) and 3(b). (ii) Two contour segments merge into one at v, as shown in Fig. 3(c).

A *contour-connected* environment refers to an area without split cells. That is, if two cells have the same distance to S, they are on the same contour. Fig. 2 shows two contour-connected environments and one non-contour-connected environment. Comparing the second and third subfigure, we know that the contour-connected property also depends on the location of the charging station S.

In our model, the robot moves rectilinearly in the environment without rotations. In our analysis, we assume that the robot can cover a contour by moving along it though it is not rectilinear. Under this assumption, the robot needs two units of energy to visit a cell on the contour.

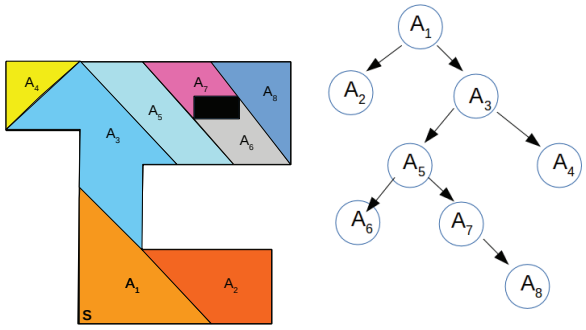
Environment Representation

In this section, we first introduce a sweeping algorithm to partition a general environment into contour-connected sub-



(a) (b) (c)

Figure 4: Two cases of meeting a split cell in the sweeping processes. The colored area is the contour-connected subareas.



(a) (b)

Figure 5: (a) The partition of an environment into the contour-connected subareas. The inner rectangle (black) represents an obstacle. (b) The corresponding tree structure.

areas. After the partition, we present an ordering of these areas represented as a tree. We also introduce two propositions which are necessary for the performance proof of our algorithm.

Recall that the first contour is the one which has unit distance to S. The sweeping algorithm starts from this contour, as shown in Fig. 4. The sweeping direction is along the distance-increasing direction of the contours. When the sweeping polyline meets a split cell v on the contour C, we stop the sweeping process and form a contour-connected subarea with the swept area. If there are multiple split cells, we process them from one side of the contour to the other side. There are two possible cases based on the types of the split cell. (i) v is of type-i. We start two new sweeping processes with the two segments of C split by v, as shown in Fig. 4(a) and 4(b). (ii) v is of type-ii. We start one new sweeping process with the merged contour, as shown in Fig. 4(c). When there are no more contours in a sweeping process, the area swept by this process forms a new subarea.

An example of partitioning an environment into contour-connected subareas is shown in Fig. 5(a). The common boundaries of two adjacent subareas correspond to the contours which have split cells on them.

Next, we impose an ordering of the partitioned subareas, which can be represented as a tree. Each node corresponds to a contour-connected subarea. At the beginning the tree has only one root node that represents the contour-connected subarea with S. We build the tree recursively. If a node N induces two subareas due to a split cell of type (i), we add the two nodes that represent the induced subareas as child nodes of N. If two nodes N_i and N_j induce a single subarea due to a split cell of type (ii), we add the node that represents the induced subarea and arbitrarily set N_i or N_j as its parent. We call this tree *partition tree* of the environment. The partition tree of the example is shown in Fig. 5(b).

Let N be a node and $\{N_1, N_2, ..., N_k\}$ be N's child nodes in the partition tree. Let C be the furthest (last) contour to S in N and C_i be the closest (first) contour to S in N_i, $i = 1, 2, ..., k$. It is easy to see that $d(C_i) = d(C) + 1$. In the geometric setting, we can *order* these child nodes from one side to the other based on the order of cells on C.

For simplicity, throughout the paper, when we refer to *covering* an environment (or a subarea), we mean that the robot goes to the closest cell which has not been visited, and then follows the contours in distance-increasing order. When all cells in the node are visited, the robot traverses the tree in pre-order from leftmost child nodes to the rightmost and continues to follow the contours in each node. Here the *leftmost* child node is defined to be the node closest to the robot when it finishes following the furthest (last) contour of the parent node.

To understand the coverage process without energy constraint, we present two properties next. The first property, Proposition 1, focuses on covering contour-connected environments. The second one is for general environments.

Proposition 1. *Without considering the energy constraint, the robot can fully cover a contour-connected environment and each cell is visited only once if we do not require the robot to return to S after fully covering the environment.*

Proof. In contour-connected environments, the contours can be ordered based on their distance to S. The robot can always enter the next contour from one side of the current contour. So by following the contours in distance-increasing order, the robot can fully cover the environment without revisiting any cell. □

Proposition 2. *Without considering the energy constraint, the robot can fully cover any environment in such a way that each cell is visited at most twice.*

Proof. Let N be any node of the partition tree. Let $\mathcal{N} = \{N_1, N_2, ..., N_t\}$ be the child nodes of N from left to right ($\mathcal{N}$ is empty if N is a leaf node). Any cell in N is visited the first time when following the contours in N. Then the robot needs to traverse the subtrees rooted at N's child nodes. When the robot transits from $subTree(N_i)$ to $subTree(N_{i+1})$ after $subtree(N_i)$ is fully covered, the cells on the furthest (last) contour in N which connect N_i and N_{i+1} are visited the second time (as shown by the colored cell in fig. 6). When the last child node $subTree(N_t)$ is fully covered, the robot leaves N. In this process, the robot visits one cell per contour in N and these cells are visited the second time. After $subTree(N)$ is fully covered, the robot no longer needs to come to this subtree. Thus no cells will be visited the third time in $subTree(N)$. Fig. 6 demonstrates a

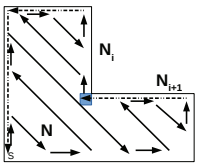


Figure 6: A coverage example without energy constraint. The colored (blue) cell connects the two child nodes of N.

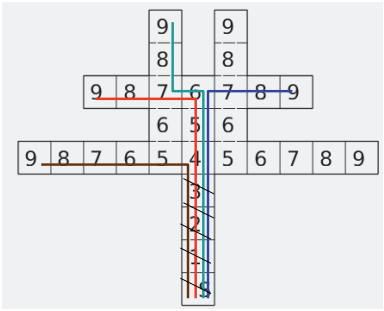


Figure 7: Many cells need to be revisited many times.

coverage example where the revisiting portion is shown by dashed lines. □

Shnaps and Rimon also use the method of following the contours in their work (Shnaps and Rimon 2016). They state that without 'narrow' corridors (an area of unit width) this method visits a cell only once without considering the energy constraint. Here our Proposition 2 adds that in the general case a cell can be visited at most twice.

With the energy constraint, the robot needs to go back and forth in the environment. As a result, the cells may be revisited many times. Consider the example shown in Fig. 7. Each path needs to go through the shadowed area when it goes to the further subareas. Thus the cells in the shadowed area are visited many times.

Grouping Nodes of the Partition Tree

In this section, we introduce our algorithm to group the nodes into subregions such that each subregion A_i, when A_i is empty, cannot be fully covered by a single path. Here *empty* means that no cells of A_i has been visited. *Fully cover* means that all the cells of A_i are visited. The purpose of forming the 'large' subregions is to allow each path to work in the individual subregion without going to the others. Later we show that if each path works in the 'large' subregions, the newly visited cells by this path is comparable to that by the optimal paths. We call the set of grouped subregions as the working zone.

We reuse the notation $d(\cdot)$ for areas. $d(A)$ denotes the distance of the closest uncovered cell in A to S.

The grouping algorithm is described in Algorithm 1. We start grouping the nodes of the partition tree from bottom upwards toward the top. Let $\{N_1, N_2, ..., N_p\}$ be the child nodes of N. Recall that we can order these child nodes by N's furthest (last) contour C. At each level of the tree, we group nodes in the direction of C in two steps. First (line 3 to 9), if the subtree rooted at N_i cannot be fully covered by a single path, we delete $subTree(N_i)$ from T and add it as a subregion in the working zone. In the second step (line 10 to 23), for the remaining nodes in this layer, we start from one side of nodes and group it with its siblings until the grouped forest cannot be fully covered by a single path (line 12 to 16). We add the grouped forest to the working zone. We do nothing if all the siblings together can be fully covered by a single path. After the two steps, the algorithm goes to the upper layer and repeats the process.

Algorithm 1 Group nodes.

Input: The tree structure of the environment T.
Output: working zone $\mathcal{A} = \{A_1, A_2, ..., A_m\}$. Each A_i cannot be covered by one path.

1: $\mathcal{A} = \emptyset$; D= depth of T.
2: **for** k=D:-1:1 **do**
3: $\{N_1, N_2, ..., N_p\}$ are the nodes on depth k from left to right;
4: **for** i=1:1:p **do**
5: **if** $subTree(N_i)$ cannot be fully covered by a single path **then**
6: Add $subTree(N_i)$ to $\mathcal{A}$;
7: Delete $subTree(N_i)$ from T;
8: **end if**
9: **end for**
10: $\{N'_1, N'_2, ..., N'_{p'}\}$ are the remaining nodes on depth k from one side to the other;
11: **while** $i \leq p'$ **do**
12: $q \leftarrow$the number of right siblings of N'_j;
13: $N = N_j$; $j = 1$;
14: **while** N can be fully covered by one path **do**
15: $N = N \bigcup N_{i+j}$; $j = j + 1$;
16: **end while**
17: **if** N cannot be fully covered by one path **then**
18: Add N to $\mathcal{A}$;
19: Delete N from T;
20: **end if**
21: i=i+j;
22: **end while**
23: **end for**
24: **if** T is not empty **then**
25: Add T to $\mathcal{A}$;
26: **end if**

Lemma 1. *Let $\mathcal{A} = \{A_1, A_2, ..., A_m\}$ be the working zone of any environment. There are three types of subregions in $\mathcal{A}$. (i) A single node, (ii) a subtree, and (iii) a forest. No subregions in $\mathcal{A}$ can be fully covered by a single path.*

The correctness of this lemma is easy to see based on how we form subregions in the working zone in Algorithm 1.

Coverage Strategy

The general idea of our approach is as follows. We use Algorithm 1 to group the nodes to form the working zone $\mathcal{A}$. Each path π_i chooses the A_i from $\mathcal{A}$ that has the closest uncovered

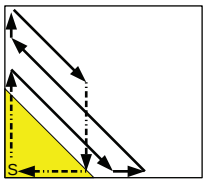


Figure 8: An example of the coverage phase of a path. The filled area is covered by previous paths. The coverage phase of this path is drawn with solid lines.

cell c to S and covers A_i from c. Let P_i be the remaining uncovered area after the first $(i-1)$ paths in our solution. Let a_i^* be the maximum number of cells visited by any path in the optimal solution. We show that in P_i, the number of visited cells by the ith path in our solution is at least a constant factor of a_i^* (Lemma 3). Then we prove that this greedy approach has an $O(\log(D))$ approximation factor compared to the optimal solution, where D is the distance of the furthest cell to S.

Our coverage algorithm is described in Algorithm 2. Each path π_k in the solution set Π_{sol1} chooses from the working zone the subregion A_i which has the closest uncovered cell c to S. Then it covers A_i from c until the energy is insufficient to continue. After π_k, we check if the remaining unvisited cells of A_i can be fully covered by a single path. If it can, we use an additional path to fully cover A_i and these additional paths are stored in a separate set Π_{sol2}. Note that when choosing c, if there are multiple cells with the same distance to S, we choose the one which is adjacent to the cell where a previous path retreats, or the one adjacent to the boundary.

Algorithm 2 Coverage Algorithm.

Input: The working zone $\mathcal{A} = \{A_1, A_2, ..., A_m\}$.
Output: Solution paths Π_{sol1}, Π_{sol2}.

```
1: Π1 = ∅; Π2 = ∅; i = 1; j = 1;
2: while P is not fully covered do
3:     Start recording path π_i;
4:     c_0 ∈ A_t ← closest uncovered cell in P; Move to c_0;
5:     cover(A_r);
6:     if remains(subTree(A_t)) can be fully covered by
       one path then
7:         Add π_j to cover remains(subTree(A_t));
8:         Add π_j to Π_sol2; j = j + 1;
9:     end if
10:    Add π_i to Π_sol1. j = j + 1;
11: end while
```

For the ith path π_i in Π_{sol1}, the environment is partially covered by the previous paths $\{\pi_1, \pi_2, ..., \pi_{i-1}\}$. We define the *coverage phase* of π_i to be the portion from the first uncovered cell on π_i to the cell where π_i starts to retreat to S. Fig. 8 illustrates a coverage phase example.

With the definition of the coverage phase, we have Lemma 2.

Lemma 2. *Each cell is visited at most twice by the coverage phase of the paths in* Π_{sol1}.

Proof. Let A_i be a subregion from the working zone and $\Pi_i = \{\pi'_1, \pi'_2, ..., \pi'_j\}$ be the paths whose coverage phase starts in A_i. The paths in Π_i are in the same order as they are in Π_{sol1}. According to Lemma 1, A_i can be a single node, a tree, or a forest. The lemma is proved by examining all the three cases.

(i) A_i is a single node. Let π'_j and π'_{j+1} be any two adjacent paths in Π_i. π'_{j+1} starts its coverage phase from where π'_j retreats to S. Both the coverage phases of π'_j and π'_{j+1} follow the contours in distance-increasing order. So if we put the coverage phase of the paths in Π_i together, they follow the contours in distance-increasing order. The energy for returning to S is not counted as coverage phase. By Proposition 1, each cell is visited at most once. An example of covering a single node (a contour-connected subarea) is shown in Fig. 9(a).

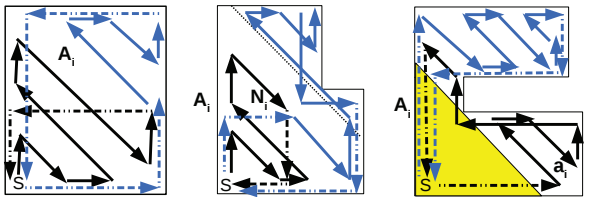
(ii) A_i is a tree. Let N_i be the root node of A_i and $(A_i - N_i)$ be the other nodes in A_i. The paths in A_i must start their coverage phase in N_i. Otherwise, let π_t be a path whose coverage phase starts in $(A_i - N_i)$, then the remaining uncovered area of A_i can be fully covered by π_t, since $(A_i - N_i)$ can be fully covered by a single path according to Algorithm 1. Then this path should not appear in Π_{sol1} by Algorithm 2. Now similar to case (i), the coverage phase of the paths in Π_i, when connected together, covers A_i by following the contours in distance-increasing order. By Proposition 2, each cell is visited at most twice. An example of covering a subtree is shown in Fig. 9(b).

(iii) A_i is a forest. Let a_i as the left most subtree of A_i and $(A_i - a_i)$ be the other nodes in A_i. By Algorithm 1, both $(A_i - a_i)$ and a_i can be fully covered by a single path. So actually A_i is fully covered by two paths while the second path is not included in Π_{sol1}. Each node in A_i is covered by the coverage phase of the path by following the contours in distance increasing-order. By Proposition 2, each cell in A_i is visited at most twice. We should also note that when the robot transits from one subtree of A_i to the next one, it will visit the A_i's parent node N_p. The robot needs to visit the furthest (last) contour in N_p. It is easy to see that the cells on the furthest (last) contour of N_p will only be visited once when the paths in N_p follow the contours. And these cells are visited the second time when the robot transits from one child node of N_p to the other child node. An example of covering a forest is shown in Fig. 9(c).

□

By Algorithm 2, each path in Π_{sol2} has a precedent path in Π_{sol1}. For analysis purposes, now we combine Π_{sol1} and Π_{sol2}. That is, let π_{i2} be a path from Π_{sol2} whose precedent path is π_{i1} from Π_{sol1}. We *attach* π_{i2} to π_{i1}, which means the area covered by π_{i2} is regarded as being covered by π_{i1}.

Lemma 2 states that a cell can be visited at most twice by the coverage phase of two paths. Note that this does not mean that a cell is visited at most twice by Π_{sol1} since the coverage phase starts from the first uncovered cell on the path.



(a) Coverage of a single node (b) Coverage of a subtree (c) Coverage of a forest. The colored (yellow) area is the root of this forest

Figure 9: The coverage examples of the subregions in the working space. The coverage phases are drawn by solid lines.

The next lemma counts the number of cells covered by the coverage phase of each path in Π_{sol1}. To prove it, we need to specify this: for two paths whose coverage phases visit the same cell, which path this cell should be allocated to as a *covered* cell. A naive way is that this cell is allocated to the path whose coverage phase visits this cell first. But for analysis purposes, we do it in the following way. Let C be the set of cells visited by the coverage phases of two paths π_i and π_j. By Lemma 2, C is not visited by any other paths' coverage phase. In our analysis, half of C is covered by π_i, and the other half is covered by π_j. Then we have Lemma 3.

Lemma 3. *Let $\Pi_{sol1} = \{\pi_1, \pi_2, ..., \pi_n\}$ be the first path set from Algorithm 2. Let P_i be the remaining uncovered area after the first $(i-1)$ paths in Π_{sol1}. Let s_i be the number of the covered cells (area) in P_i by π_i. Let s_i^* be the number of the covered cells in P_i by any path in Π^*. Then $s_i \geq \frac{1}{8}s_i^*$, for $i = 1, 2, ..., n$.*

Proof. Since a cell is of unit size as the robot, the number of visited cells is equal to the covered area measured in the unit of the robot's size. We use the number of cells and covered area interchangeably. When the robot follows the path π_i, it needs $d(c_i)$ energy to go the the first uncovered cell in P_i. $\frac{B}{2}$ energy is guaranteed to be enough for retreating to S. The robot needs two units of energy to visit a cell when following the contours. So π_i's coverage phase visits at least $(B - \frac{B}{2} - d(c_i))/2 = \frac{B}{4} - \frac{d(c_i)}{2}$ cells in P_i before retreating. According to Lemma 2, any cell on the coverage phase of π_i is visited at most twice by the coverage phases of the paths in Π_{sol1}. Let C_1, C_2, and C_3 be the cell sets on the coverage phase of π_i, where C_1 is the cell set visited only once by π_i, C_2 is the cell set visited twice by π_i itself, and C_3 is the cell set also visited by the coverage phases of the other paths in Π_{sol1}. Then π_i covers $s_i = |C_1| + \frac{|C_2|}{2} + \frac{|C_3|}{2} \geq ((\frac{B}{2} - d(c_i))/2)/2 = (\frac{B}{8} - \frac{d(c_i)}{4})$ cells in P_i.

For any path in Π^*, it needs at least $d(c_i)$ energy to go to the first uncovered cell in P_i and needs at least $d(c_i)$ energy to retreat to S. So any path in Π^* can cover at most $s_i^* = (B - 2d(c_i))$ cells in P_i. Thus $s_i \geq \frac{1}{8}s_i^*$. □

Theorem 1. *Let n^* be the number of paths in the optimal solution Π^*. Let n be the total number of paths from Algorithm 2. $n \leq 16n^* \ln |P|$, where $|P|$ denotes the total area of the environment P.*

Proof. Let P_i be the remaining uncovered area after the first $(i-1)$ paths in Π_{sol1}. P_i is fully covered by the n^* paths in Π^*. So on average each path in Π^* covers $\frac{|P_i|}{n^*}$ area of P_i. According to Lemma 3, the number of visited cells in P_i by the ith path π_i in Π_{sol1} is more than one eighth of that by any path from Π^*. So π_i covers at least $\frac{|P_i|}{8n^*}$ cells in P_i. Thus,

$$|P_{i+1}| \leq |P_i|(1 - \frac{1}{8n^*})$$

Let n_1 be the number of paths in Π_{sol1}. After n_1 paths, the remaining uncovered area should be less than 1. We get

$$|P_{n_1}| = |P|(1 - \frac{1}{8n_1^*})^{n_1} < 1 \quad (1)$$

$$n_1 <= 8n^*(\ln |P|) \quad (2)$$

Note that this analysis combines the paths from Π_{sol2} and Π_{sol1}. Each path in Π_{sol2} is attached to one path in Π_{sol1}. It is easy to see that no two paths in Π_{sol2} are attached to the same path in Π_{sol1}. So $n \leq 2n_1$. We get

$$n <= 16n^*(\ln |P|) \quad (3)$$

The theorem is proved. □

Recall that D is the distance of the furthest cell to S measured in the robot's size. It is easy to see that $|P| \leq 4 * \frac{D(D+1)}{2} = 2D(D+1)$. So by Theorem 1, the approximation factor is $O(\ln D)$.

Theorem 2. *Let l^* be the total length of the paths in the optimal solution, and l be the total length of the paths from Algorithm 2. Then, $l \leq 32(\ln |P|)l^*$.*

Proof. Let n_l be the number of paths of the optimal solution when minimizing the total length. Then $n_l \geq n^*$. No two paths in the optimal solution can have the sum of length less than B. Otherwise they can be combined to form a shorter path. So

$$l^* \geq \frac{B}{2}n^*$$

The total length of paths from Algorithm 2 is

$$l \leq 16n^*(\ln |P|)B$$

So

$$l \leq 32(\ln |P|)l^*$$

□

Similarly we know that the approximation factor is $O(\ln D)$.

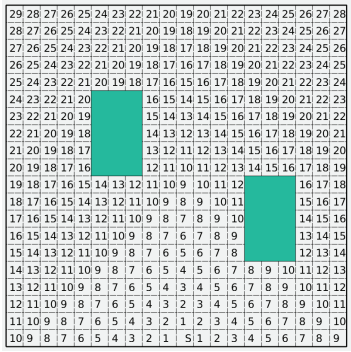
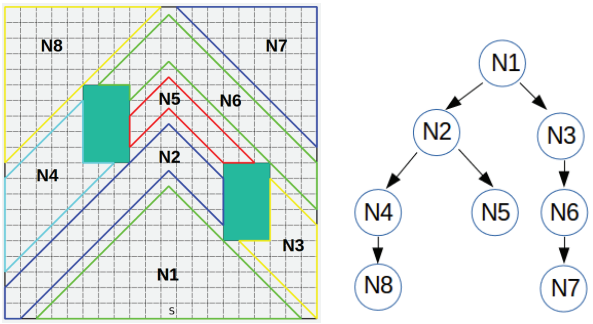


Figure 10: The simulated environment with two inner obstacles. The value in each cell is its distance to S.



(a) (b)

Figure 11: (a) Decomposing the environment into contour-connected subareas.(b) The corresponding partition tree of the environment.

An Implementation Example

In this section we demonstrate a simulated execution of our algorithm. The environment to cover is shown in Fig. 10.

In Fig. 11(a), the environment is partitioned into the contour-connected subareas. The cells on the boundaries of each subarea are included in that subarea. The corresponding partition tree is shown in Fig. 11(b).

In the simulation, we set the energy budget B to be 80 units. Note that B has to be greater or equal to $2D$ to make it possible to fully cover the environment. With B we can run the grouping algorithm. The nodes N_5, N_3 can be fully covered by a single path. The working zone of the environment is shown in Fig. 12.

Fig. 13 demonstrates the planned paths to cover the working zone. We plot the paths in each subregion together. Here the paths are rectilinear, which means we plot the robot's actual trajectory when it follows the contours.

To improve the quality of the resulting paths, we use a heuristic. As shown in Fig. 13(a), when the robot fully covers $N1$ and goes to $N3$, it takes the shortest path to $N3$ instead of following the last contour of $N1$ again. It is easy to see that taking the shortest path to $N3$ results in less revisiting. So our previous analysis still holds.

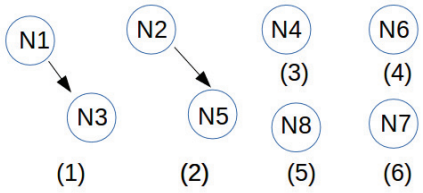


Figure 12: The working zone of the environment in Fig. 11.

Conclusion and Future Work

In this paper, we presented an algorithm for the energy-constrained coverage path planning problem and showed that the cost of its solution is always within a factor of $\log D$ of the optimal solution. In our model, we used a grid to represent the environment and restricted the robots motion to the four main directions. In the future, we would like to generalize this result to continuous robot motion.

Our problem formulation further assumes that the environment's boundary (including obstacles) is given. This knowledge may not be always available in which case online algorithms are needed. In (Shnaps and Rimon 2016) an $\Omega(\log D)$ lower bound is presented for the online version. We would like to extend our algorithm to the online case and match this lower bound.

Acknowledgment

This work is supported in part by NSF Award # 1525045, a MnDrive RSAM Industrial Partnership grant with The Toro Company, and a grant from Minnesota State LCCMR Program.

References

Choi, Y.-H.; Lee, T.-K.; Baek, S.-H.; and Oh, S.-Y. 2009. Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5788–5793. IEEE.

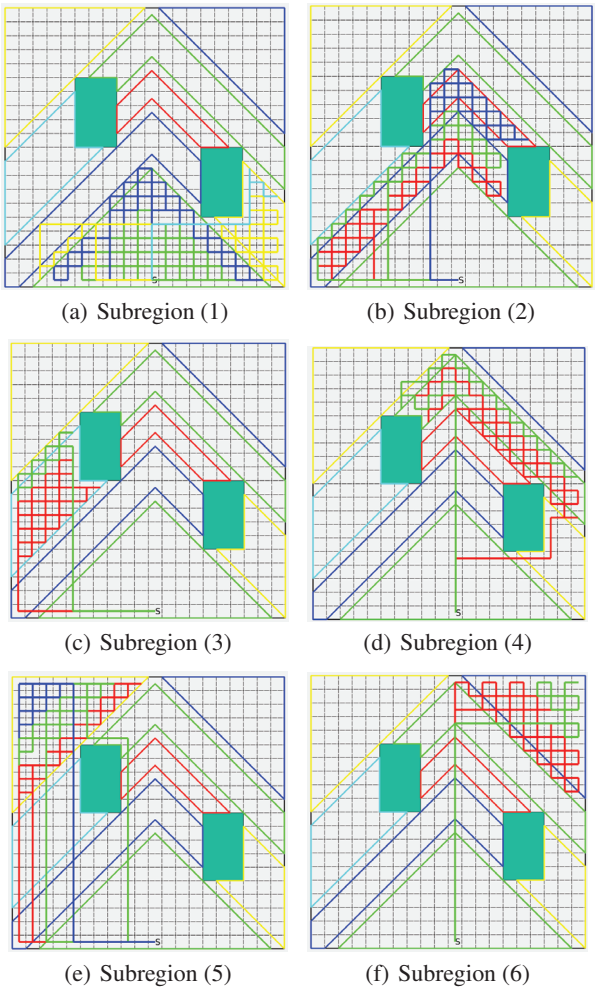
Choset, H. 2000. Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots* 9(3):247–253.

Choset, H. 2001. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence* 31(1):113–126.

Gabriely, Y., and Rimon, E. 2001. Spanning-tree based coverage of continuous areas by a mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, 1927–1933. IEEE.

Galceran, E., and Carreras, M. 2013. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems* 61(12):1258–1276.

Gonzalez, E.; Alvarez, O.; Diaz, Y.; Parra, C.; and Bustacara, C. 2005. Bsa: a complete coverage algorithm. In *IEEE International Conference on Robotics and Automation*, 2040–2044. IEEE.



(a) Subregion (1) (b) Subregion (2)

(c) Subregion (3) (d) Subregion (4)

(e) Subregion (5) (f) Subregion (6)

Figure 13: Coverage of each subregion of the working zone.

Laporte, G. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59(3):345–358.

Li, C.-L.; Simchi-Levi, D.; and Desrochers, M. 1992. On the distance constrained vehicle routing problem. *Operations research* 40(4):790–799.

Mannadiar, R., and Rekleitis, I. 2010. Optimal coverage of a known arbitrary environment. In *IEEE International Conference on Robotics and Automation (ICRA)*, 5525–5530. IEEE.

Mishra, S.; Rodriguez, S.; Morales, M.; and Amato, N. M. 2016. Battery-constrained coverage. In *IEEE International Conference on Automation Science and Engineering (CASE)*, 695–700. IEEE.

Nagarajan, V., and Ravi, R. 2012. Approximation algorithms for distance constrained vehicle routing problems. *Networks* 59(2):209–214.

Shnaps, I., and Rimon, E. 2016. Online coverage of planar environments by a battery powered autonomous mobile robot. *IEEE Transactions on Automation Science and Engineering* 13(2):425–436.

Strimel, G. P., and Veloso, M. M. 2014. Coverage planning with finite resources. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, 2950–2956. IEEE.

Viet, H. H.; Dang, V.-H.; Choi, S.; and Chung, T. C. 2015. Bob: an online coverage approach for multi-robot systems. *Applied Intelligence* 42(2):157–173.