Optimal Online Coverage Path Planning with Energy Constraints

Gokarna Sharma
Kent State University
Kent, Ohio
sharma@cs.kent.edu

Ayan Dutta
University of North Florida
Jacksonville, Florida
a.dutta@unf.edu

Jong-Hoon Kim
Kent State University
Kent, Ohio
jkim@cs.kent.edu

ABSTRACT

We consider the problem of covering an unknown polygonal environment possibly containing obstacles using a robot of square size $L \times L$. The environment is structured as a grid with resolution proportional to the robot size $L \times L$, imposed on it. The robot has a limited energy budget – it has to visit a charging station before it runs out of its energy; there is a single charging station in the environment. In a single time step, the robot can move from one grid cell to one of its four adjacent cells. The energy budget B allows the robot to travel at most B distance, i.e., B grid cells. The objective of the robot is to minimize both total distance traveled to cover the environment (visit each cell of the environment not occupied by obstacles) and the number of visits to the charging station. In this paper, we present the *first* online coverage path planning algorithm that achieves $O(\log(B/L))$-approximation for both objectives. Our bound is optimal since there exists a lower bound of $\Omega(\log(B/L))$ for this problem for both objectives. Simulation results show the efficiency of our approach.

KEYWORDS

Robotics; Coverage Path Planning; Energy Constraint; Unknown Environment; Approximation

ACM Reference Format:
Gokarna Sharma, Ayan Dutta, and Jong-Hoon Kim. 2019. Optimal Online Coverage Path Planning with Energy Constraints. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Coverage path planning which requires a robot (or a team of robots) to fully cover a given area (or a planar environment) possibly containing obstacles is a well-studied problem in robotics with many practical applications such as autonomous sweeping, vacuum cleaning, and lawn mowing. The goal is to plan path(s) so that the robot(s) can visit every point in the area not occupied by the obstacles.

For many years, this problem has been studied assuming that the robot(s) have an unlimited energy budget. Therefore, given a robot, a single path can be planned to cover the given environment (possibly containing obstacles) since it can be assumed that the robot can move arbitrarily long distances. The *offline* version of the problem where robot(s) have knowledge of the environment including obstacles has been well studied, e.g., see [7]. Many algorithms have been proposed such as the boustrophedon decomposition based coverage [4, 11], the spiral path coverage [8], and the spanning-tree based coverage [6]. The boustrophedon coverage and spiral path techniques can be adapted to solve the *online* version of the problem [3, 18] where the robot does not know the details of the environment, e.g., shapes and locations of the obstacles in the beginning, but it can accumulate the knowledge of the environment over time.

In practice, robots have energy constraints. This is due to the limited budget of the battery that they operate on, which will run out after moving certain finite distances. A battery-powered robot needs to go back to the charging station to get recharged before the battery runs out. This presents a new topic of coverage path planning under the limits on the distances a robot can move after a full change of battery. This may require planning multiple paths for the robot (instead of a single path in the unlimited energy case) since the robot may not be able to visit all the points in the environment after only a full charge.

Recently, there has been a growing interest in solving coverage path planning problem with energy constraints. Strimel and Veloso [16] used boustrophedon decomposition to cover the environment. The robot returns to the charging station when its energy level is too low to continue the coverage. Mishra *et al.* [12] designed a coverage planning algorithm consisting of multiple robots. To continuously cover the environment, the robots are divided into two groups, workers and helpers. When a worker needs to go back to recharge, an associated helper will continue the worker's coverage. These studies do not formally analyze their algorithms for performance guarantees, except proving that their methods correctly cover every point in the environment.

Shnaps and Rimon [15] modeled the energy consumption of the robot by the length of the path that robot traverses and studied both offline and online versions of the coverage path planning problem (denoted as OfflineCPP and OnlineCPP, respectively, in this paper). For OfflineCPP, they proposed an $1/(1-\rho)$-approximation algorithm, where ρ is the ratio between the furthest distance in the environment and half of the energy budget [15]. The approximation factor $1/(1-\rho)$ can be arbitrarily large when ρ approaches 1. For OnlineCPP, they proposed an $O(B/L)$-approximation algorithm, where B is the energy budget and L is the size of the robot (assuming a $L \times L$ square robot). For a unit square size robot (i.e., $L = 1$), the approximation factor becomes $O(B)$. They also established a lower bound of $\Omega(\log(B/4L))$-approximation for OnlineCPP (for $L = 1$, this lower bound becomes $\Omega(\log B)$). Recently, Wei and Isler first presented an $O(\log B/L)$-approximation algorithm for OfflineCPP in [18] and improved it to a constant-factor approximation in [17]. Our goal in this paper is to provide a better approximation algorithm for OnlineCPP. Particularly, our goal is to close the gap of

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

$B/\log B$ factor between the lower and upper bounds of Shnaps and Rimon [15] for OnlineCPP.

Objectives and Contributions. We consider the online coverage path planning problem OnlineCPP with a single robot that has a limited energy budget B. The planar polygonal environment P to cover by the robot is not known to it a priori; the number of obstacles including their shapes and locations are unknown. The robot is equipped with an obstacle detection sensor (e.g., laser rangefinder) as well as position sensor (e.g., GPS) to build the map of the environment. The robot is represented as a square $L \times L$ that moves rectilinearly in P. Initially, the robot is at the only charging station S that is inside P. The environment P is also represented as a grid ($L \times L$ size cells) laid out on the polygonal area. The energy consumption of the robot is assumed to be proportional to the distance traveled, i.e., the energy budget of B allows the robot to move B units distance (that is, $\lfloor B/L \rfloor$ cells)[1]. The robot has to go back to the charging station S to get its battery recharged before it runs out of it. It is assumed that the size of P is such that the robot cannot fully cover P in a single path; i.e., the number of cells in P is at least $\lfloor B/L \rfloor + 1$. The goal of OnlineCPP is to find a set of paths $\Pi = \{\pi_1, \ldots, \pi_n\}$ for the robot so that

- **Condition (a):** Each path π_i starts and ends at S,
- **Condition (b):** Each path π_i has length $|\pi_i| \leq B$, and
- **Condition (c):** The paths in Π collectively cover the environment P, i.e., $\cup_{i=1}^{n} \pi_i = P$.

We will show that any algorithm satisfying simultaneously conditions (a)–(c) correctly solves OnlineCPP. However, we are interested in finding a set of paths Π by an algorithm such that it optimizes the following two performance metrics:

- **Performance metric 1:** The *number of paths* (or number of visits to S) in Π, denoted as $|\Pi|$, is minimized, and
- **Performance metric 2:** The *total lengths of the paths* in Π, denoted as $\sum_{i=1}^{n} |\pi_i|$, is minimized.

In this paper, we establish the following main theorem for OnlineCPP for correctness and for both performance metrics.

Theorem 1.1 (**Main Result**). *Given an unknown polygonal environment P possibly containing obstacles and a robot r of size $L \times L$ consisting of position and obstacle detection sensors initially situated at a charging station S inside P with an energy budget of B, there is an online algorithm that correctly solves* OnlineCPP *and guarantees*

- *$O(\log(B/L))$-approximation to the number of paths, $|\Pi|$, compared to that for an optimal algorithm that knows everything about P a priori.*
- *$O(\log(B/L))$-approximation to the total lengths of the paths traversed by the robot, $\sum_{i=1}^{n} |\pi_i|$, compared to that for an optimal algorithm that knows everything about P a priori.*

The approximations achieved for both performance metrics in Theorem 1.1 are asymptotically optimal since there is a lower bound of $\Omega(\log(B/L))$ for both metrics for OnlineCPP due to Shnaps and Rimon [15]. The best previously known algorithm due to Wei and Isler [17] achieves constant-approximations for both metrics, but only for OfflineCPP, and their techniques cannot be extended to achieve the bound we obtain in this paper. Furthermore, our algorithm is a significant improvement to that of Shnaps and Rimon [15] which achieves $O(B/L)$ approximation for OnlineCPP.

Technique. We employ a new technique of traversing the unknown planar environment P using a *Depth First Search* (DFS) traversal used in distributed robotics in different contexts (e.g., see [2, 5]). The DFS traversal asks the robot r (which is at S in the beginning) to traverse the cells of P in the increasing order of depth through forward and backtrack phases. The map of the environment P is constructed incrementally while r is running the DFS traversal such that all the new frontiers (unvisited cells not occupied by obstacles) are eventually visited. To control the path costs, since the robot has a limited energy budget, it is asked to run the DFS traversal level by level. That means, the robot first runs its DFS traversal until some depth from the charging station S. The depth is computed based on its energy budget B (which is known to robot a priori). After all the cells within that certain depth are covered, then the robot is asked to increase the depth appropriately to cover some new cells of P. The DFS traversal running level by level is exploited to show that for every level, the cost of the technique that we use will ask the robot to follow a number of paths that is only a constant factor more than the number of paths that would have been taken by the robot in the optimal, offline scenario. Therefore, the approximation of our algorithm becomes proportional to the number of levels (which is $O(\log(B/L))$) compared to the optimal cost. Simulation results show the efficiency of our approach.

Roadmap. We discuss related work in Section 2 and model and some preliminaries in Section 3. We discuss some techniques to handle the unknown environment, including how to build a map of the environment on-the-fly and divide the tree map into levels, in Section 4. We then present and analyze the OnlineCPP algorithm in Sections 5 and 6, respectively. We then demonstrate our algorithm through simulation in Section 7 and conclude in Section 8.

2 RELATED WORK

The most closely related work to ours is due to Wei and Isler [17, 18] and Shnaps and Rimon [15] (as discussed in Section 1). The other related work 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) [1]. With energy constraints, this coverage problem becomes the *Vehicle Routing Problem* (VRP) [9]. 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 [13] proposed a 2-approximation algorithm. Li *et al.* [10] used a TSP-partition method and their algorithm has a similar approximation to the work in [15]. Most of these work studied the offline version where the environment P (including obstacles) is known to the robot a priori so that pre-processing on the environment can be done prior to exploration obtaining better approximation. This is also the case in the algorithm of Wei and Isler [17, 18] for OfflineCPP.

Coverage with multiple robots has also received a lot of attention (e.g., see [2, 5]). In some cases, the paths planned for a single robot under energy constraints can also be executed by multiple robots

[1] We do not consider the charging time of the battery of the robot r while at the charging station S since this delay does not impact the cost related to these metrics.

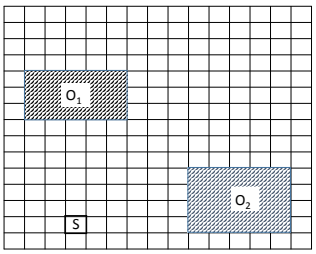

Figure 1: An example environment P with two obstacles O_1 and O_2 and a charging station S inside P. The perimeter of P is considered as a boundary of P. P is shown decomposed as cells of size $L \times L$ same as the size of the robot.

by assigning the planned paths to the robots; using a single robot or multiple robots does not affect the total energy cost.

3 MODEL AND PRELIMINARIES

Environment. The environment P is a planar polygon containing a single charging station S inside it. P may possibly contain polygonal, static obstacles. P has a boundary that is known to the robot. See Fig. 1 for an illustration of P with two obstacles.

Robot. We consider the robot r initially positioned at the charging station S. The robot r moves rectilinearly in P, i.e., it may move to any of the four neighbor cells (if the cell is not occupied by an obstacle) of the cell that it is currently positioned. We also assume that r has the knowledge of the global coordinate system, that means it knows left (West) and right (East) and up (North) and down (South) cells consistently from its current position cell.

We have the following observation on the size of **P**.

Observation 1. *If a cell of* **P** *is located at distance D from the charging station S and $D > B/2$, then robot r cannot fully cover P.*

Since the total energy budget is B, for any cell at distance D away from S, it needs D amount of energy to reach there from S and the same D amount of energy to return to S. So if a cell is such that $D > B/2$, r will not have enough energy left to either reach the cell or return to S after the cell is visited. Therefore, what Observation 1 intuitively means is that no cell of P can be farther than $B/2$ distance (measured in Manhattan distance) from the charging station S.

A path π_i is a list of neighboring cells that r visits starting and ending with S. Furthermore, a cell c in P is called *accessible* if there is (at least) a path π_i formed by consecutive cells of P from S leading to c such that each cell in π is not occupied by any obstacle.

Observation 2. *If there is a cell c in P such that there is no path connecting S to c, the robot r cannot cover c (and hence cannot fully cover P), even with an unlimited energy budget.*

Notice that this happens only when obstacles within P are located in such a way that they divide P into two obstacle-free sub-polygons P_1 and P_2 with P_1 and P_2 share no common boundary. Therefore, we assume that there is no such cell c in P, i.e., all cells not occupied by obstacles (i.e., free cells) are accessible by r.

Observation 3. *If there is at least a path connecting S to any cell c, then c has at least a neighboring cell (out of at most 4 neighbor cells) that is not occupied by any obstacle.*

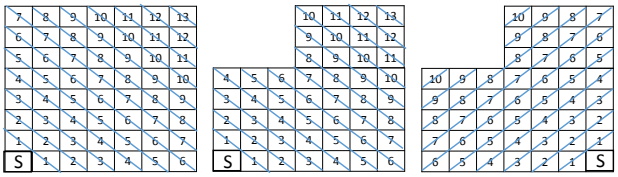

Figure 2: An illustration of contours in the environment P with the value in each cell the L_1 distance (or Manhattan distance) to it from the charging station S.

The Online Coverage Path Planning Problem OnlineCPP**.** The OnlineCPP problem can be formally defined as follows.

Definition 3.1. Given an unknown planar polygonal environment P possibly containing obstacles with a robot r having battery budget of B initially positioned at a charging station S inside P, OnlineCPP is for r to visit all the cells of P not occupied by obstacles and accessible from S through a set of paths Π so that

- Conditions (a)–(c) are satisfied, and
- Performance metrics (1) and (2) are minimized.

Approximation. OnlineCPP is a NP-hard problem [14, 15, 17, 18]. Therefore, our goal is to obtain a polynomial time algorithm solving OnlineCPP optimizing the performance metrics. We measure the efficiency of any algorithm for OnlineCPP in terms of *approximation*, which is defined as the worst-case ratio of the cost of the online algorithm for some environment P over the cost of the optimal, offline algorithm for the same environment P. It is assumed that the optimal algorithm has complete knowledge of P including obstacles a priori. We denote the cost of the optimal offline algorithm by OPT.

Definition 3.2. An algorithm solving OnlineCPP provides k-approximation if the cost of solving any instance p of the problem does not exceed k times the cost (OPT) of solving p using an optimal, offline algorithm.

4 HANDLING UNKNOWN ENVIRONMENT

4.1 Decomposition of the Environment P

Following Wei and Isler [17, 18] and Shnaps and Rimon [14, 15], we decompose the environment P into grids of same size as the robot r (i.e., robot r is assumed to be of size $L \times L$) using *approximate cellular decomposition* method of Choset [4]. We call these grids *cells*. Moreover, following [14, 15, 17, 18], we assume that the obstacles are such that they do not partially occupy any cell in P, i.e., for a cell, an obstacle either occupies it fully or does not occupy it at all (see Fig. 1 for an illustration where obstacles O_1 and O_2 do not partially occupy any cell of P). However, the obstacles may not necessarily be of rectangular shapes.

We assume that the charging station S is in a cell in P (Fig. 1). An *equi-distance contour* is a poly-line where the cells on it has the same distance to/from the charging station S (Fig. 2). The cells on a contour can be ordered from one side to the other (robot r is assumed to have the knowledge of the global coordinate system). It is easy to see that we can also order the contours based on their distance to S in a strictly increasing fashion.

Let c be a cell and C be a contour. Let $d(c)$ denote the distance to S from c and let $d(C)$ denote the distance to S from C. If $d(C_j) =$

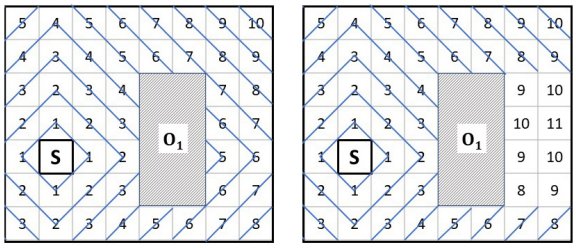

Figure 3: An illustration of changes on contour numbers for the cells of P in certain situations

$d(C_i) + 1$, we say that contour C_j is contour C_i's next contour. The contour C with $d(C) = 1$ is called the first contour; this contour has the at most 4 cells that are neighbors of S.

Notice that since P is unknown, the information about contours is not provided a priori and the robot r has to determine these contours on-the-fly while under coverage. The robot uses its position sensor to determine for the cell (it is currently positioned) in which contour the cell belongs to and also the at most 4 neighboring cells (that are not occupied by obstacles). The robot r can do this as follows. Since it has a position sensor with respect to S (which works as an origin of the coordinate system). It will measure its Manhattan distance. Consider the cell 12 on the top row in the middle of Fig. 2. When the robot is at cell 12, it considers a horizontal line passing through S and the vertical line passing through cell 12. This gives the information for r that the current position is 12.

This method works if the horizontal and vertical lines passing through S do not intersect any obstacle present in P. If intersection happens, then r uses the approach as described in Fig. 3 to appropriately modify in which contour they belong. This is needed since otherwise the algorithm we design may not visit those cells and the full coverage of P is not achieved. In the left of Fig. 3, since the horizontal line passing through S intersects obstacle O_1, the number of the cells based on pure distances from S is problematic for the cells on the right of O_1 (the cells numbered 6, 7 to 7, 8 bottom to top). This is because the DSF never visits those cells. The robot r detects the situation based on the obstacle sensor and corrects the numbering on-the-fly handing the issue (the right of Fig. 3).

4.2 Constructing a Tree Map of P On-the-fly

We denote the tree map by T_P which is constructed incrementally while the robot r is under coverage. Initially, the robot r is at the charging station S. In this case, the tree T_P has only one node S, which we call *the root* of T_P. If there is no obstacle present in P, each cell (except the cells in the boundary) in P has exactly four neighbor cells. See the left of Fig. 4 where each cell (except boundary cells) has 4 neighbor cell as there is no obstacle. Since r has the global coordinate system, each of the (at most) four neighbor cells of any cell c can be consistently labeled West, North, East, and South in the clockwise order starting from the cell in its left (i.e., West).

The robot r picks the first cell c_1 that is not occupied by any obstacle and includes it in T_P as a *child* of S. If the cell labeled West is not occupied by any obstacle, then r picks that cell. Otherwise, it goes in order of North, East, and South until it finds the first cell that is not occupied by any obstacle and includes it as a child of S. Recall

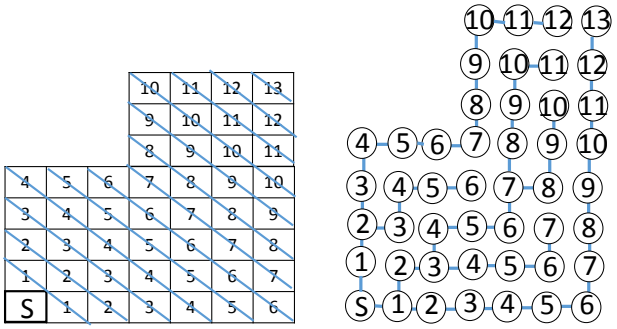

Figure 4: An example tree map T_P on the right constructed for the environment P shown in the left. It is guaranteed in T_P that each cell with distance d from S is included in T_P at exactly at depth d from the root S. This means that all the cells at any contour $C(d)$ are at depth d in T_P.

that there is at least one cell that is not occupied by any obstacle (Observation 3). We now have two nodes in $T_P = \{S, c_1\}$ with c_1 as a child node of S. Furthermore c_1 is a cell in the first contour C_1 (i.e., $d(C_1) = 1$). The right of Fig. 4 provides an illustration of the tree map T_P developed for the environment P given in the left.

Notice also that, since r is exploring P while building T_P, it will move to c_1 after it is included as a child in T_P. Robot r then again repeats the process of building T_P from its current position c_1. While at c_1, r is only allowed to make one of the neighbor cells of c_1 the child of c_1 in T_p and this child node will be in the second contour C_2, i.e., $d(C_2) = 2$. Furthermore, if some cell is already a part of T_P, then this cell will not be included in T_P again. This process then continues. Using this approach, all the cells in the first contour C_1 will be children of S (the root of T_P), all the cells in the second contour C_2 will be children of the nodes of T_P that are cells in the first contour C_1, and so on. In Fig. 4, each cell of contour C_i on the environment P on the left are at depth i in T_P shown on the right. Additionally, if one or more neighbor cells of the robot's current cell are occupied by obstacles, they will be detected by the position sensor (laser rangefinder) and will not be added to T_p.

Let $T_{P,final}$ be the final tree map of the environment P after all the cells of P (that are not occupied by the obstacles in P and accessible from S) are included in T_P. Let $Depth(T_{P,final})$ denote the depth of the tree $T_{P,final}$; the root S has depth 0, the child node of S has depth 1, and so on. Let N_x be the set of nodes in T_P such that the distance from S to each node in N_x is x.

Lemma 4.1. *In the tree T_P constructed above, each cell $c \in C$ such that $d(C) = x$ is positioned exactly at depth x in T_P.*

Proof. Follows easily from construction of T_P since the cells at contour C such that $d(C) = x$ become the children of the cells of contour C_p such that $d(C_P) = x - 1$, for $1 \leq x \leq B/2$. □

Lemma 4.2. *For any unknown polygonal environment P possibly containing unknown number of obstacles, $Depth(T_{P,final}) \leq B/2$.*

Proof. This lemma follows combining the results of Observations 1 and 2, and Lemma 4.1; otherwise covering all free and accessible cells of P is not possible by the robot r of energy B. □

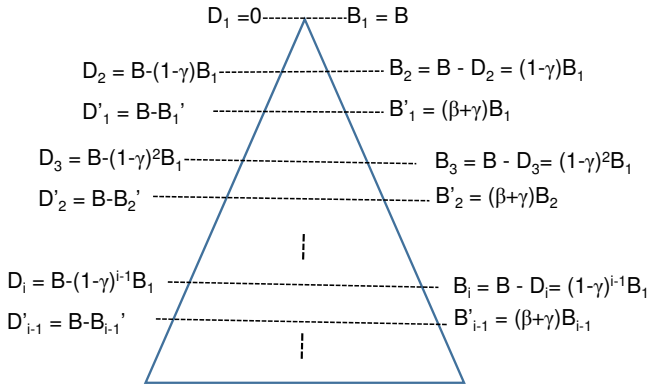

Figure 5: An illustration of the partitioning of the tree map T_P into levels. The levels are denoted as D_i (in the increasing order from the charging station S which is the root of T_P). The charging station S is at level $D_1 = 0$ and also the nodes up to depth D_2. The nodes of T_P starting from depth D_2 up to D_3 are at level 2, and so on. B_i denotes the energy budget for robot for level D_i (while at depth D_i); at level D_1, the energy budget $B_1 = B$. We also define D'_i, such that $D_{i+1} \leq D'_i \leq D_{i+2}$; for $i = 1$, $D_2 < D'_1 < D_3$. How the values for each D_i, B_i, D'_i, and B'_i are computed is also shown. These values depend on two parameters γ, β such that $0 < \gamma, \beta < 1$.

4.3 Partitioning the Tree Map into Levels

We denote a *level* of the tree T_P by i. To be able to partition T_P into levels, it is not required that T_P is known a priori. This helps in partitioning T_P into levels incrementally on-the-fly while under coverage. We guarantee that this does not have effect on how partitioning is obtained. This guarantee makes easy to run the algorithm for any unknown polygonal P containing obstacles. Our goal is to partition T_P into levels proportional to B/L (for simplicity, we assume $L = 1$ in the description below which works even if $L > 1$).

Figure 5 provides an illustration on how T_P is partitioned into levels, where $D_1, D_2, \ldots, D_i, \ldots$ denote the level boundaries and all the nodes of T_P from D_i to D_{i+1} are considered the nodes of level-i. Each level i of T_P is a set of nodes which are located at certain depth in T_P. We can also say that a level i is defined starting from a contour (say $C_{start,i}$) and ending at the counter (say $C_{end,i}$) such that $d(C_{start,i}) \leq d(C_{end,i})$. All the nodes of T_P starting from $C_{start,i}$ and ending at $C_{end,i}$ are called the *level-i nodes* of T_P. All the nodes of T_P at contour $C_{start,i}$ are called the *level-i roots* of T_P. Level-i roots are denoted by a set $N_i = \{v_{1i}, v_{2i}, \ldots\}$.

According to this definition, it is easy to see that the first level (level-1) consists of the root of T_P, which is the charging station S, and the nodes of T_P up to depth D_2 (excluding the nodes of T_P at depth D_2). Therefore, the starting contour $C_{start,1}$ for level 1 is S. We say $C_{start,1}$ is D_1 which is 0. At the charging station S, the energy budget is B. We say that the starting contour $C_{start,2}$ for level 2 is at depth $\lceil B - (1-\gamma)B \rceil$ from S, where $0 < \gamma < 1$; we will set the value of γ later. We denote $C_{start,2}$ by D_2. All the nodes of T_P from contour D_1 to D_2 are called the *level-1 cells*. Furthermore, all the cells at contour D_2 are called *level-2 roots*.

We say that the starting contour $C_{start,3}$ for level 2 is at depth $\lceil B - (1-\gamma)^2 B_1 \rceil$ from S. We denote $C_{start,3}$ by D_3. All the nodes of T_P from D_2 to contour D_3 are called the *level-2 cells*. Furthermore, all the cells at contour D_3 are called *level-3 roots*.

This definition then extends to any level i, $i > 2$. For any level-i, the starting and the end contours are $C_{start,i} = D_i$ and $C_{end,i} = D_{i+1}$, respectively, and all the cells from contour D_i to contour D_{i+1} are called the *level-i cells* and the cells in contour D_{i+1} are called *level-(i+1) roots*.

The energy budget for the robot at any level-i root is $B_i = (B - D_i)$, where D_i is the depth of the root level of the level-i. This can also be directly computed using the following formula $B_i = (1-\gamma)^{i-1}B$; we provide the proof of this below which will help in proving a bound on the number of levels in Lemma 4.4.

Lemma 4.3. *For any $i \geq 0$, $B_{i+1} = B - (1-\gamma)^i B$.*

Proof. We prove this lemma by induction. We use two variables D_i and B_i, where D_i denotes the depth of level-i from the root of T_P and B_i denotes the energy budget for level-i for the root of that level. We have that for level-1, $D_1 = 0$ and $B_1 = B$. For any level-i, $i > 1$, $D_i = D_{i-1} + \gamma B_{i-1}$, where $B_{i-1} = B - D_{i-1}$.

We argue through induction that, for any $i \geq 1$, $D_i = B - (1-\gamma)^{i-1}B$. For $i = 1$, $D_1 = B - (1-\gamma)^0 B = 0$, which is trivially true and serves as the base case. Assuming that $D_i = B - (1-\gamma)^{i-1}B$ holds for some $i > 1$, we now show that $D_{i+1} = B - (1-\gamma)^i B$, which will complete the proof. Particularly, we have that

$$\begin{aligned} D_{i+1} &= D_i + \gamma B_i \\ &= D_i + \gamma(B - D_i) \\ &= (1-\gamma)D_i + \gamma B \\ &= (1-\gamma)(B - (1-\gamma)^{i-1}B) + \gamma B \\ &= B - (1-\gamma)^{i-1}B - \gamma B + \gamma(1-\gamma)^{i-1}B + \gamma B \\ &= B - (1-\gamma)^{i-1}B + \gamma(1-\gamma)^{i-1}B \\ &= B - (1-\gamma)^{i-1}B(1-\gamma) \\ &= B - (1-\gamma)^i B. \end{aligned}$$

□

We are now ready to prove the following lemma regarding the number of levels for the partitioning approach discussed above.

Lemma 4.4. *When $L = 1$, the number of levels in the tree map T_P according to the partitioning algorithm discussed above is $\log_{\frac{1}{1-\gamma}}\left(\frac{B}{2}\right)$.*

Proof. Observe that the depth of the tree map T_P cannot be more than $\frac{B}{2}$ (Lemma 4.2). Therefore, the partitioning of T_P into levels stops as soon as $(1-\gamma)^i \left(\frac{B}{2}\right) \leq 1$, i.e., there is at most one contour of nodes of T_P in a level (notice that these nodes are the leaves in T_P). Therefore, setting $i = \log_{\frac{1}{1-\gamma}}\left(\frac{B}{2}\right)$, we have that $(1-\gamma)^i \frac{B}{2} \leq 1$ and hence we have $i = \log_{\frac{1}{1-\gamma}}\left(\frac{B}{2}\right)$ levels in T_P. □

When $L > 1$, a simple adaptation gives $\log_{\frac{1}{1-\gamma}}\left(\frac{B}{2L}\right)$ levels.

5 ALGORITHM SOLVING ONLINECPP

We first discuss the high level ideas of the algorithm and then the details. In the description of the algorithm and analysis for both correctness and performance, we assume that $L = 1$. A simple

adaptation of the algorithm and analysis will work when $L > 1$. The pseudocode is given in Algorithms 1 and 2.

Overview of the Algorithm. The main idea behind our algorithm is to incrementally explore the environment P by the robot r to fully cover P, while at the same time r constructing a tree map T_P of P to keep track of the new frontiers that need to be visited by it to solve OnlineCPP. We ask r to proceed covering of P level by level as defined in Section 4.3. In order to do this, r explores P and builds T_P up until a certain depth.

Specifically, while at level-1, r covers all the cells that are in level-1 and some cells that are in level-2. This additional extension of covering a few nodes at level-2 while covering the nodes at level-1 provides us extra information about exploring level-2 cells and makes the algorithm analysis easier for both lower and upper bounds on the performance metrics. In fact, while at level-1, the robot r covers the cells from depth D_1 to depth D'_1 as shown in Fig. 5, where $D_2 < D'_1 \leq D_3$.

After all the cells in level-1 (along with some cells in level-2) are visited, robot r then starts covering the cells in level-2 (along with some cells in level-3). After all the cells in level-2 (and some cells of level-3) are visited, r then starts covering cells in level-3 (and some cells of level-4) and this process continues until all the cells of P are covered. Notice from the description above that some cells of each level $i > 1$ are visited two times by r as they will be visited first time while r is covering the cells in level-$(i-1)$ and second time while r is covering the cells in level-i. We will argue that this has no asymptotic impact on the bounds of the algorithm.

Within a level, r uses the *Depth First Search* (DFS) traversal to visit all the cells in that level. In the beginning, r starts its DFS traversal from S and visits the leftmost cells of each contour with increasing depth until it reaches contour at depth D'_1 from S. After depth D'_1 is reached while exploring level-1, r backtracks to visit the remaining cells within depth D'_1 from S. After all the free cells until depth D'_1 are visited, then r starts visiting the cells of P in level-2. Robot r again performs the DFS traversal up to depth D'_2 (from S). After all the cells until D'_2 are visited using the DFS approach, r starts visiting level-3 cells and this process continues until all the cells of P in all the levels are visited.

Within any level-i, while covering the cells using DFS traversal, robot r keeps track of its current distance D_{cur} from S. D_{cur} is the shortest L_1 (or Manhattan) distance from S to the current cell of r. In fact, the path from S to the current cell of r is of length D_{cur} since T_P is a tree with root S. Robot r also keeps track of B_{cur} - the remaining energy budget. After fully charged at S, $B_{cur} = B$. As soon as it transitions to adjacent cell of S, $B_{cur} = B_{cur} - 1$. Therefore, as soon as $B_{remain} = D_{cur}$, r follows the shortest path leading to S and returns to S. This way r is positioned at S when $B_{cur} = 0$ and will be recharged to have energy budget B before starting next path to visit the remaining nodes of T_P.

To facilitate the exploration of the cells in any level-$(i+1)$, $i \geq 2$, r selects the nodes of level-i at depth $C(D_{i-1})$ as the root nodes N_i. That is, for level-2, the nodes in the end contour $C(D_1)$ act as the root nodes and stored in N_2. Generalizing this for any level-i, the nodes in the end contour $C(D_{i-1})$ act as the root nodes of N_i. And in level-i, the robot first moves to a node in N_i from S and from there it moves to the first node of that level that was not already visited in the previous round(s) of exploration. After visiting some new nodes at that level, r returns to S to be recharged again. If the path of r does not lead it to visit some new nodes (that were not already visited), then r does not go to that path.

Particularly, the new frontier of the covered part of T_P at level-i defines level-$(i+1)$. The algorithm is then called at each node $v \in N_i$ to cover the cells in level-$(i+1)$. The robot r then first moves from S to v to explore the new frontier of T_P using DFS traversal. When it is left with energy budget $B_{cur} = D_{cur}$, it heads to S following the shortest path and in the next path, it starts covering the cells of level-$(i+1)$ where it left off in the previous path.

Detailed Description of the Algorithm. We call our algorithm OnlineCPPAlg. The environment P is unknown to the robot r except that S is in P and P has a boundary of radius at most $B/2$ centered at S (in L_1 distance). Initially, the robot r is at S with the energy budget B. This is a special situation of OnlineCPPAlg, where $v = S$ and $B_1 = B$. In this case of $i = 1$, we ask r to explore P using a *Depth First Search* (DFS) traversal up to depth $D'_1 = \lfloor(\beta +$

Algorithm 1: OnlineCPPAlg

Input: The charging station S and the available energy budget B for r that is initially at S; the environment P is not known to r except that S is inside P and P has a boundary of radius at most $B/2$ centered at S;
$N \leftarrow \{S\}$;
for $i = 1, 2, \ldots, \lceil \log_{\frac{1}{1-\gamma}}(\frac{B}{2}) - 1 \rceil$ **do**
  $D_i \leftarrow \lfloor B - (1-\gamma)^{i-1}B \rfloor$;
  $D_{i+1} \leftarrow \lfloor B - (1-\gamma)^{i}B \rfloor$;
  $B_i = B - D_i$;
  $B'_i \leftarrow \lceil (\beta+\gamma)B_i \rceil$;
  $D'_i \leftarrow \lfloor B - B'_i \rfloor$;
  while there is at least a node of T_P within depth D'_i (from S) that is yet to be visited **do**
    Cover(S, i, D_i, D'_i, D_{i+1}, N);
  $N \leftarrow N'$;

Algorithm 2: Cover($S, i, D_i, D'_i, D_{i+1}, N$)

if *there is at least one unvisited node in tree T_P between depth D_i and D'_i (D'_i inclusive)* **then**
  $v \leftarrow$ the leftmost unexplored node on T_P that is closest from S (the root of T_P);
  move to a node $v_i \in N$ that is closest to v using a path in T_P;
  move to v from v_i through a shortest path in T_P;
  $D_v \leftarrow$ the distance from S to v;
  $B_{remain} = B - D_v$;
  while $B_{remain} \geq D_{v'}$ *for any node v' between depth D_i and D'_i (inclusive)* **do**
    explore the unvisited nodes between depth D and D' (inclusive) using a DFS traversal;
    insert each new node visited in tree T_P making a child appropriately;
    decrease B_{remain} by 1 for each new node the traversal visits;
    $N' \leftarrow$ a set of nodes of T_P that are at depth D_{i+1} (note that $D_i \leq D_{i+1} \leq D'_i$);
  return to S following the tree T_P;

$\gamma)B_1\rfloor$; $B_1 = B$. We set later the value of β such that $D_1 \leq D'_1 \leq D_2$, meaning that while at level-1 some cells (nodes) of level-2 are also visited. The value D_i is calculated such that $D_i = \lfloor B - (1-\gamma)^{i-1}B \rfloor$.

Recall also that the DFS traversal is a renowned approach which is defined as a walk followed by a robot starting at node v building T_P based on the DFS approach. As soon as robot r realizes that it has the remaining energy that is just enough to reach back to S, it goes back to S visiting the parent nodes of the tree T_P that is built.

For level-2, we ask r to explore P again using the DFS traversal up to depth $D'_2 = \lfloor(\beta+\gamma)\cdot B_2\rfloor$, $B_2 = B - D_2$, i.e., all the cells of P up to contour $C(D'_2)$. Note that $D_2 \leq D'_2 \leq D_3$.

Therefore, for any level-i, $i > 2$, we ask r to explore P using a *Depth First Search* (DFS) traversal up to depth $D'_i = \lfloor(\beta+\gamma)\cdot B_i\rfloor$, $B_i = B - D_{i-1}$, i.e., all the cells of P up to contour $C(D'_i)$. Note that $D_i \leq D'_i \leq D_{i+1}$. The scenario in the left of Fig. 3 is also handled on-the-fly running the dynamic contour modification method discussed in Section 4.1 so that the DFS can visit the cells affected by the change as shown in the right of Fig. 3.

6 ANALYSIS OF THE ALGORITHM

In this section, we provide a theoretical analysis of OnlineCPPAlg. We first prove correctness of OnlineCPP and then provide the analysis of the costs for the performance metrics (1) and (2).

6.1 Correctness Proof of the Algorithm

We prove the following theorem for the correctness of OnlineCPPAlg. We show that the robot r covers all the grid cells of the environment P that are free and accessible using OnlineCPPAlg.

Lemma 6.1. *Setting $\beta = 3/4$ and $\gamma = 1/10$ correctly runs* OnlineCPPAlg.

Proof. The root nodes in N_2 (level-2) has depth $D_2 = B - (1-\gamma)B_1 = 9/10B$; $B_1 = B$. Therefore, when r reaches a node $v \in N_2$, it has $B_2 = 9B/10$. While at level-1, the nodes up to depth $D'_1 = B - \lfloor(\beta+\gamma)b_1\rfloor = \lfloor(3/4 + 1/10)B_1 = B - 17B/20 = 3B/20$ are explored. This satisfies that $D_2 \leq D'_1 \leq D_3$.

The root nodes in N_3 has depth $D_3 = B-(1-1/10)^2B = 19B/100$; $B_3 = (1-1/10)^2B = 81B/100$. Therefore, when r reaches a root node $v \in N_3$, it has $B_3 = 81B/100$. While at level-2, the nodes up to depth $D'_2 = \lfloor(\beta+\gamma)B_2\rfloor = \lfloor(3/4+1/10)B_2 = 17/20 * 9B/10 = 765B/1000$ are explored. Note also that $D_3 \leq D'_2 \leq D_4$. Therefore, repeating this process and arguing through induction, for any level-i, we have that $D_{i+1} \leq D'_i \leq D_{i+2}$. □

Theorem 6.2 (Correctness). OnlineCPPAlg *correctly covers the environment P solving* OnlineCPP.

Proof. We will show that using OnlineCPPAlg, robot r correctly explores all the nodes of T_P that are at some level-i, $i \geq 1$. To explore the nodes of T_P at level-i, the robot has to move first to a root node of the level-i from S (the node in the set N_i). Let $v_{ni} \in N_i$ be a node on the root of level-i. After reaching v_{ni}, r is left with the energy budget B_i, where $B_i = B - D_i$. And, r is left with at least $B' = B - D'_i$ amount of energy budget while reaching the first unexplored node v in level i (the is because v may be at level D'_i), i.e., the nodes of level-i up to depth D'_i are explored while r was at level-$(i-1)$. The robot needs D'_i energy budget to return to S from v. Therefore, r is left with $B - 2D'_i$ energy budget to perform exploration at level-i. We have that $D'_i < \frac{B}{2}$ for any level i. Therefore, r can visit at least $B - 2D'_i \geq 1$ nodes of level i that were not visited before. Finally, after covering of level-i is finished, the algorithm goes to level-$(i+1)$ as T_P has the knowledge of at least one unexplored cell of P until all the free and accessible cells of P are covered by r. Since $\gamma > 0$, the algorithm continues until $\lceil\gamma B\rceil \geq 1$. The theorem follows. □

6.2 Analysis of Approximation Ratio

For the analysis purpose, we denote by T_v^δ the subtree of T_P rooted at some node v truncated to depth δ from v. We denote by $|T_P|$ the number of edges in T_P. Denote by N the total number of free and accessible cells in P. We prove the following lemma for OnlineCPPAlg to cover all the cells of P that are in level-1. We show that the number of paths taken by r while covering all the cells in level-1 are within a constant factor of the number of paths taken by r in an optimal offline algorithm *OPT*.

Lemma 6.3. *Using* OnlineCPPAlg, *the robot r initially at S covers all the cells in level-1 in X_{ALG} paths, where $X_{ALG} \leq c \cdot X_{OPT}$, where c is a positive constant and X_{OPT} are the number of paths generated by an optimal offline algorithm OPT.*

Proof. Let the sub-tree of T_P for level-1 be T_S^δ, where $v = S$ and $\delta = D'_1$. Initially, T_S^δ is completely unexplored. Since r uses the DFS traversal, the length of the DFS traversal is $2 \cdot |T_S^\delta|$.

Robot r uses $x = B - D'_1 = B - B + B'_1 = B'_1 = (\beta+\gamma)B = 17/20B$ amount of energy to traverse a part of this DFS traversal, for $\beta = 3/4$ and $\gamma = 1/10$. Therefore, we have that

$$x \cdot X_{ALG} \leq 2 \cdot |T_S^\delta|.$$

We also know that

$$2 \cdot |T_S^\delta| \leq 2 \cdot |T_P| \leq 2 \cdot X_{OPT} \cdot B.$$

Therefore,

$$X_{ALG} \leq \frac{2 \cdot X_{OPT} \cdot B}{17B/20} \leq c \cdot X_{OPT}.$$

□

Note that in above lemma $c = 40/17 = 2.35$. We now extend Lemma 6.3 for any level-i, $i > 1$.

Lemma 6.4. *Using* OnlineCPPAlg, *the robot r initially at S covers all the cells in any level-i, $i > 1$, in $X_{ALG,i}$ paths, where $X_{ALG,i} \leq c_i \cdot X_{OPT,i}$, where c_i is a positive constant and $X_{OPT,i}$ are the number of paths by an optimal offline algorithm OPT for level-i.*

Proof. For simplicity in the proof, we discard the costs of OnlineCPPAlg and *OPT* to reach to S from a node in level-i and come back from S to the node in level-i. This is because these costs for our algorithm and the optimal offline algorithm *OPT* are the same.

After r reaching level-i at some node v, it uses the remaining energy $B'_i = B - 2D'_i$ to explore the edges of T_v^δ of that level. Therefore, we have the following relation.

$$B'_i \cdot X_{ALG,i} \leq 2 \cdot |T_v^\delta|.$$

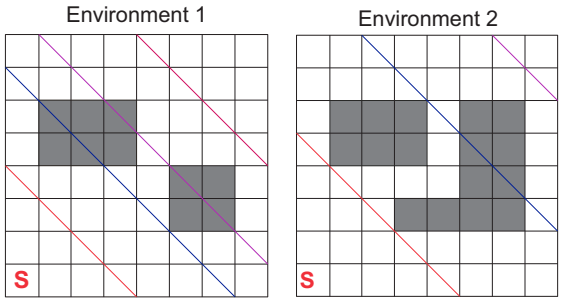

Figure 6: Two 8×8 test environments are shown. S denotes the charging station and the color-coded polylines denote the levels that the robot has covered: a) Environment 1 ($B = 32$); b) Environment 2 ($B = 48$). Due to the higher available budget, less number of levels are present in Environment 2.

This provides the following:

$$X_{ALG,i} \leq \frac{2}{B'_i} \cdot |T_v^{\delta}| \leq \frac{2}{B'_i} |T_P \backslash T_S^{D_i}|.$$

For the optimal offline algorithm OPT, in level-i, we have that

$$B'_i \cdot X_{OPT,i} \geq |T_P \backslash T_S^{D_i}|.$$

That means $X_{ALG,i} \leq c_i \cdot X_{OPT,i}$, where $c_i \leq 2$. □

Theorem 6.5. OnlineCPPAlg *achieves $O(\log B)$-approximation for both performance metrics (the number of paths and the total lengths of the paths).*

Proof. We have from Lemma 4.4 that there are $\log_{\frac{1}{1-\gamma}} \left(\frac{B}{2}\right)$ levels in T_P. We have from Lemma 6.4 that for any level-i, the number of paths of r is within a constant c_i factor of the number of paths by an optimal offline algorithm OPT. Therefore, combining these two results, we have that the approximation for algorithm OnlineCPPAlg is $c_{\max} \cdot \log_{\frac{1}{1-\gamma}} \left(\frac{B}{2}\right)$, where $c_{\max} = \max\{c_1, c_2, \ldots\}$, the maximum among the c_i of all $\log_{\frac{1}{1-\gamma}} \left(\frac{B}{2}\right)$ levels in T_P and $c_{\max} \leq 2.35$. □

Proof of Theorem 1.1: Theorem 6.5 proves Theorem 1.1 for $L = 1$. Since the cells are decomposed proportional to the size of the robot $L \times L$, with a simple adaptation of the analysis we obtain $O(\log(B/L))$-approximation on both performance metrics. The correctness analysis stays the same. □

7 SIMULATIONS

We have implemented the proposed algorithm OnlineCPPAlg using Java programming language on a desktop computer with an Intel i7-7700 CPU and 16GB RAM. We have created two different environments with both convex and concave obstacles in them. The charging station is placed in the bottom-left corner. Both environments are of the same dimension – 8×8 ($l = 8$). The budget B is set to 32 (unless otherwise mentioned), i.e., four times the size of each side of the environment. These tested environments are shown in Fig. 6 (video of simulation: `https://youtu.be/r16J7V5juSQ`).

In environment 1 (Fig. 6(a)), it takes our algorithm a negligible amount of time 24.59 milliseconds (ms) to plan the set of paths to cover the free cells in the environment while correctly mapping the obstacles in it. Using OnlineCPPAlg, the robot needs to calculate 19 paths to cover the whole environment, i.e., the robot needed to recharge its battery 19 times. The number of free cells ($\mathcal{F}$) in this particular environment is 54. Therefore, no algorithm can cover this environment in less than $\frac{2\mathcal{F}}{B}$ number of paths. With $B = 32$, this minimum number of paths boils down to 3.375. The ratio of the number of paths required by OnlineCPPAlg to the minimum number of possible paths is 5.63, denoted by Rt. The sum of the path lengths for environment 1 is 372.

For environment 2 (Fig. 6(b)), as there are more obstacles in it, $\mathcal{F} = 48$. We observe an effect of this lower value of $\mathcal{F}$ on the properties of paths generated by OnlineCPPAlg. The sum of lengths of all paths reduces to 314 from 372 – a reduction of 15.59% in path length when the number of free cells is reduced by 11%. The theoretical minimum number of paths possible to cover this particular environment under the energy constraint is 3 whereas OnlineCPPAlg uses 16 paths. Thus, Rt drops to 5.33. Runtime of OnlineCPPAlg in this case is 24.68 ms.

To see the effect of the maximum energy budget available to the robot (B) on the number of paths and the corresponding Rt, we also vary the value of B between $5l$ and $6l$. With $B = 5l$ and $B = 6l$, the run times to cover the environment 1 using OnlineCPPAlg are comparable – 20.49 and 21.72 ms. respectively. As the budget was higher, the robot could explore more cells in one path rather than coming back to the charging station frequently to recharge itself. The total number of paths and the sum of path lengths also see significant drops to 14 and 319 with $5l$, and 10 and 246 with $6l$ respectively – reductions of 26% and 47% respectively in number of paths from $B = 4l$ and reductions of 14.24% and 33.87% respectively in total path lengths from $B = 4l$. Rt also reduces from 5.63 ($B = 4l$) to 5.18 ($B = 5l$) and 4.44 ($B = 6l$) respectively.

In environment 2, the effect of increasing B has similar effect to the environment 1. For example, with both $B = 5l$ and $6l$, the number of paths reduces significantly – 12 and 9 respectively. Although the robot has more budget to spend, due to the less number of free cells in the environment and the shape of the obstacles, the robot cannot leverage it to its maximum potential. Therefore, we notice improvements in all the performance metrics, but the changes are comparable to environment 1. Rt drops from 5.33 with $B = 4l$ to 5 and 4.5 respectively with $B = 5l$ and $6l$. The sum of path lengths also reduces to 270 and 215 respectively with $B = 5l$ and $6l$ from 314 with $B = 4l$.

8 CONCLUDING REMARKS

We have presented the *first* optimal algorithm for solving OnlineCPP by a robot with energy constraints and showed that the cost of our algorithm is always within a factor of $O(\log B)$ of the optimal solution for both performance metrics, the number and the lengths of the paths. Having the robot and cells of size $L \times L$, $L > 1$, our analysis gives $O(\log(B/L))$-approximation. Our algorithm significantly improves the $O(B/L)$ approximation of the best previously known algorithm of Shnaps and Rimon [15]. A promising direction for future work is to generalize our result to continuous robot motion. We also plan to test our algorithm in a real-world setting.

REFERENCES

[1] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. 2007. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA.

[2] Peter Brass, Andrea Gasparri, Flavio Cabrera-Mora, and Jizhong Xiao. 2009. Multi-robot Tree and Graph Exploration. In *ICRA*. 495–500.

[3] Young-Ho Choi, Tae-Kyeong Lee, Sanghoon Baek, and Se-Young Oh. 2009. Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform. In *IROS*. IEEE, 5788–5793.

[4] Howie Choset. 2000. Coverage of Known Spaces: The Boustrophedon Cellular Decomposition. *Auton. Robots* 9, 3 (Dec. 2000), 247–253.

[5] Pierre Fraigniaud, Leszek Gąsieniec, Dariusz R. Kowalski, and Andrzej Pelc. 2006. Collective Tree Exploration. *Netw.* 48, 3 (Oct. 2006), 166–177.

[6] Yoav Gabriely and Elon Rimon. 2001. Spanning-tree Based Coverage of Continuous Areas by a Mobile Robot. *Annals of Mathematics and Artificial Intelligence* 31, 1-4 (May 2001), 77–98.

[7] Enric Galceran and Marc Carreras. 2013. A Survey on Coverage Path Planning for Robotics. *Robot. Auton. Syst.* 61, 12 (Dec. 2013), 1258–1276.

[8] Enrique González, Oscar Álvarez, Yul Díaz, Carlos Parra, and César Bustacara. 2005. BSA: A Complete Coverage Algorithm. *ICRA* (2005), 2040–2044.

[9] Gilbert Laporte. 1992. The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 3 (1992), 345–358.

[10] Chung-Lun Li, David Simchi-Levi, and Martin Desrochers. 1992. On the Distance Constrained Vehicle Routing Problem. *Oper. Res.* 40, 4 (Aug. 1992), 790–799.

[11] Raphael Mannadiar and Ioannis M. Rekleitis. 2010. Optimal coverage of a known arbitrary environment. *ICRA* (2010), 5525–5530.

[12] Saurabh Mishra, Samuel Rodríguez, Marco Morales, and Nancy M. Amato. 2016. Battery-constrained coverage. In *CASE*. IEEE, 695–700.

[13] Viswanath Nagarajan and R. Ravi. 2012. Approximation Algorithms for Distance Constrained Vehicle Routing Problems. *Netw.* 59, 2 (March 2012), 209–214.

[14] Iddo Shnaps and Elon Rimon. 2014. Online Coverage by a Tethered Autonomous Mobile Robot in Planar Unknown Environments. *IEEE Trans. Robotics* 30, 4 (2014), 966–974.

[15] Iddo Shnaps and Elon Rimon. 2016. Online Coverage of Planar Environments by a Battery Powered Autonomous Mobile Robot. *IEEE Trans. Automation Science and Engineering* 13, 2 (2016), 425–436.

[16] Grant P. Strimel and Manuela M. Veloso. 2014. Coverage planning with finite resources. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), 2950–2956.

[17] Minghan Wei and Volkan Isler. 2018. Coverage Path Planning Under the Energy Constraint. In *ICRA*. 368–373.

[18] Minghan Wei and Volkan Isler. 2018. A Log-Approximation for Coverage Path Planning with the Energy Constraint. In *ICAPS*. 532–539.