Taking Turns in Complete Coverage for Multiple Robots

Lee-or Alon, Noa Agmon and Gal A. Kaminka

Bar-Ilan University, Ramat Gan 5290002, Israel,
{(✉) alonlee1, agmon, galk}@cs.biu.ac.il

Abstract. *Coverage* is a canonical task where a robot or a group of robots are required to visit every point in a given work area, typically within the shortest possible time. Previous work on offline coverage highlighted the benefits of determining a circular coverage path, divided into segments for different robots (if more than one). This paper contributes a number of significant improvements to the planning and utilization of circular coverage paths with single and multiple robots. We focus on circular paths that exactly decompose the environment into cells, where each obstacle-free cell is covered in a back-and-forth movement. We show that locally changing the coverage direction (alignment) in each cell can improve coverage time, and that this allows for merging bordering cells into larger cells, significantly reducing the number of turns taken by the robots. We additionally present a novel data structure to compactly represent all possible coverage and non-coverage paths between cells in the work area. Finally, we discuss the complexity of global multi-robot assignment of path segments, and present greedy polynomial-time approximations which provide excellent results in practice.

Keywords: multi-robot systems, coverage

1 Introduction

Coverage is a canonical robotic task where a robot or a group of robots are required to visit every point in a given work area, typically within the shortest possible time [6, 7]. The *offline coverage* problem assumes a static and accurate work area is given as input, alongside the current robot location(s). The task is then to produce a path (or set of paths) for the robots, such that the entire work area is covered. This planning problem is believed to be *NP-Hard* [7], even when an approximation of the area is covered [1].

Previous work on offline coverage highlighted the benefits of determining a circular coverage path, divided into segments for different robots (if more than one). For a single robot, this guarantees the robot comes back to its starting location, and reduces coverage redundancy [5, 14]. For multiple robots, it also adds potential for robustness, as robots can take over for failing teammates, simply by staying on the circular joint path [9, 1, 12].

This paper contributes a number of significant improvements to the planning and utilization of circular coverage paths with single and multiple robots. We

focus on circular paths that exactly decompose the environment into cells, where each obstacle-free cell is covered via the seed-spreading method [13]. Specifically, (i) we show that locally changing the coverage direction (alignment) in each cell can improve coverage time, and (ii) that this allows for merging bordering cells into larger-cells, significantly reducing the number of turns taken by the robots. We additionally (iii) present a novel data structure to compactly represent all possible coverage and non-coverage paths between cells in the work area. This allows some robots to plan paths that take shortcuts through the work area, improving the coverage time even further. Finally, (iv) we discuss the complexity of global multi-robot assignment of path segments, and present greedy polynomial-time approximations which provide excellent results in practice. The techniques developed in this paper are guaranteed analytically where possible, and evaluated in simulation in hundreds of experimental trials.

2 Background

The literature on coverage is very extensive, and we can do it no justice in the limited space available here; for a recent survey see [7]. In this paper, we focus on *offline robot coverage*, where a map of the work area is given, which is assumed to be complete and correct (i.e., the environment is static and is accurately modeled).

One approach divides the work area into a regular grid. A *circular* path through all obstacle-free grid cells covers *approximately* the work area. Gabriely and Rimon discuss the single-robot case [5]; Hazon et al. [9] and Agmon et al. [1] discuss the multi-robot case. In contrast, we focus on an *exact work-area decomposition*, where the work-area is decomposed into irregularly-sized obstacle-free cells, each to be covered by simple zigzag motions [13]. The objective is to find a circular path through these cells.

In general, in exact cellular decompositions, the free space is divided into non-overlapping cells whose union is exactly the given free area and a path is planned to take the robot to cover each. There are several methods (see [7]). We use the boustrophedon decomposition [2], where a virtual sweep line is used to determine the cells' division. It is moved across the work area at a given orientation. The points on cell borders where the free-space along the sweep line changes, are called *critical points*.

Xu et al. have considered cyclic coverage paths in such exact work-area decompositions [14]. The key to their technique is to represent each critical point as a vertex, with edges connecting critical points, i.e., edges represent cells. A *Chinese Postman Problem* algorithm (e.g.,[8]) is then used to create an Eulerian circuit path through the graph, duplicating some edges as necessary (representing the splitting of their associated cells in half). This reduces the redundancy of the coverage. We follow the same approach, but after the creation of the Eulerian circuit we convert it to a simple circle. In addition, we present a novel technique that allows the coverage direction (the salient angle of the zigzag motion in each cell) to be changed locally, per cell. Furthermore, we present an algorithm for

merging the coverage patterns of nearby cells, so as to reduce the number of turns, the principal factor in coverage time.

The idea of changing the local coverage direction to reduce the number of turns was introduced by Huang et al. [10]. They discuss a combinatorial dynamic-programming algorithm which examines all possible angles, and all possible cell divisions and mergers. In contrast, we restrict ourselves to two angles (parallel and orthogonal to the decomposition sweep line), and provide a polynomial time algorithm for this task.

Karapetyan et al. [12] extend Xu et al. [14] to multi-robot coverage, minimizing overall coverage time (*makespan*). Under the assumption that all robots start with the same position, and the requirement that all robots finish there, they utilize a heuristic algorithm that assigns robots to different segments in the path. A key idea in their work (which we extend and improve on) is that robots can move quickly through a cell when they do not cover it, in order to reach another cell. However, even when moving quickly, the robots must follow the circular path. The techniques we present here do not make the same assumptions, and utilize a novel data structure to compactly represent *shortcut paths* between cells, even when they do not follow each other directly in the circular path. This, coupled with the local change in coverage direction and the merging procedure, improve the *makespan* significantly.

3 Efficient Coverage With a Single Robot

In this section we first discuss two factors that influence the efficiency of a robot coverage: the direction of the coverage, and the consolidation of cells to reduce turns. The techniques improve both single- and multi-robot coverage. We then present (in Section 3.2) a polynomial-time algorithms for utilizing these factors to improve coverage time—*guaranteed!*.

3.1 Factors Influencing Coverage Time

Consider the case of a single robot covering the work area. If it must visit every point in the area, it would seem that the only factor in the coverage time is any redundancy in the coverage path. But this is not the case. The number of turns taken by the robot is an important factor affecting coverage time, even when the robot visits each point exactly once. This is because turning takes time [7, 14]. Thus reducing the number of turns can reduce the coverage time considerably.

Coverage Direction (Alignment). Work area exact decompositions results in a set of cells to be covered individually. A common approach for covering an individual cell is by using back and forth motions [13]. Most previous work assumes such motions are parallel to the decomposition sweep lines. However, this assumption is unnecessary, and in fact may result in greater coverage time.

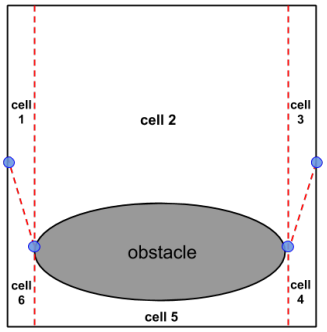
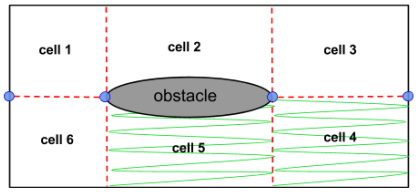


Fig. 1. Red lines designate cells boundaries. Critical points in blue. Grey areas mark obstacles; white areas are obstacle-free.

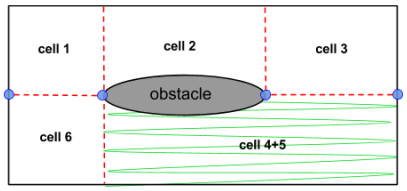
Consider, for example, the area illustrated in Figure 1. Assume that the chosen coverage direction is horizontal. This is obviously the optimal motion for cell 5 with respect to number of turns that the robot preforms. However, in cell 3 the robot would perform more rotations than if the coverage direction were vertical. Therefore, this is not the optimal coverage direction for cell 3. If we instead assume that the coverage direction is vertical—optimal for cell 3—then the coverage direction would be non-optimal for cell 5 since the robot will perform more rotations than if the direction of the coverage was to be horizontal.

Therefore, we should formulate each cell differently according to its shape. There are many methods for determining the optimal coverage direction of a cell. In this paper we determine the optimal coverage direction by calculating the number of turns in the circumscribed rectangle of the cell.

Cell Consolidation The decomposition of the work area leaves neighboring cells that may end up being covered in the same direction. Consolidating such cells reduces the number of turns. Figure 2 shows an example; note how the number of turns is significantly reduced.



(a) Unmerged cells.



(b) Merged cells.

Fig. 2. The designation of the same area prior and after the consolidation of cell 4 with cell 5. The critical points, cell divisions and obstacles are shown as above. The green curved lines show the path of the robot covering cells 4 and 5.

3.2 Consolidating and Deciding on Direction

To optimize the coverage time we present Algorithm *Optimal_Cell_Merger* (Alg. 1). The algorithm checks two possible coverage directions for each cell, while simultaneously attempting to consolidate neighboring cells. It uses dynamic programming, and is based on the algorithm for optimal matrix chain multiplication [4].

Algorithm 1 creates a matrix m that contains, in cell $[i, j]$, the minimal number of turns it takes to cover the area from cell i to cell j (and all the cells between i and j in the cells' list that is given as input). First, for each cell $[i, i]$ within matrix m, the algorithm inserts the minimal number of turns it takes to cover cell i in the map (lines 4-6). This is done by calling the function $minimum\_number\_of\_turns()$, which plots zig-zag coverage paths in the two possible directions, and returns the minimal number of turns. In lines 13-23, the algorithm computes cell $m[i, j]$. This calculation is contingent on cells $m[i, k]$ and $m[k + 1, j]$. These cells have been already computed by the algorithm. The solution—the exact configuration of the cells consolidation—will be stored in matrix s (the solution matrix). Algorithm 1 *guarantees* the optimal consolidation of cells (Theorem 1).

Algorithm 1: Optimal_Cell_Merger (cells, robot's size)

```
 1: n = length(cells)
 2: s = [n][n]
 3: m = [n][n]
 4: for i = 1 to n do
 5:    m[i][i] ← minimum_number_of_turns(cells[i])
 6:    s[i][i] ← cells[i]
 7: end for
 8: for h = 2 to n − 1 do
 9:    for i = 0 to n − h − 1 do
10:       j ← i + h
11:       min_value ← ∞
12:       for k = i to j − 1 do
13:          current ← m[i][k] + m[k + 1][j]
14:          if current ≤ min_value then
15:             min_value ← current
16:             s[i][j] ← s[i][k] ∪ s[k + 1][j]
17:          end if
18:       end for
19:       number_of_turns_after_merger ←
          minimum_number_of_turns(cell_i ⊕ ... ⊕ cell_j)
20:       if number_of_turns_after_merger ≤ min_value then
21:          min_value ← number_of_turns_after_merger
22:       end if
23:       m[i][j] ← min_value
24:    end for
25: end for
```

Theorem 1. *Given an acyclic ordered list of cells, Algorithm Optimal_Cell_Merger returns the cell merger that minimizes the number of turns for a robot covering the cells in the ordered list.*

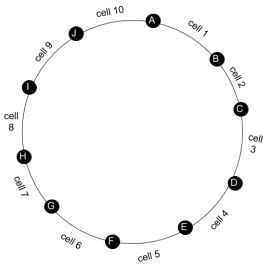
Proof. (Sketched, for lack of space.) We prove by induction on the number of cells n. When $n = 1$, there is a single cell. For $n > 1$, and assuming true for $n-1$, we prove by contradiction. We consider two adjacent cells in a non-optimal solution, that should have been merged in an optimal solution (and were not), or were separated (optimally) (and were merged). In both cases, we show a contradiction to the induction step assumption.

Complexity of Algorithm 1. Let M be the complexity of minimum_number_of_turns function. Then the complexity of minimum_number_of_turns is $O(Mn^3)$. Note that in principle, there exists an improvement to the optimal matrix chain multiplication algorithm, which is the basis for Algorithm 1, that reduces the time complexity from $O(n^3)$ to $O(n^{2.376})$ [3]. Thus there is potential for even better run-time.

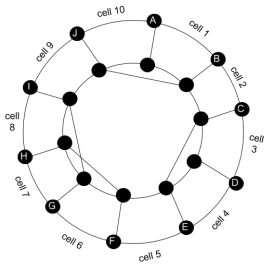
4 Efficient Coverage for a Multi-Robot System

The algorithm and methods discussed above can be used for both single-robot and multi-robot systems. While in a single robot system the robot can simply trace the Eulerian circuit as shown by Xu et al. [14], in multi-robot systems more complex algorithms need to be considered. In this section we suggest four algorithms that solve the mentioned problem and examine each one of them.

The simple circuit that was discussed in Section 2 is denoted as graph G.



(a) An Eulerian graph representing possible coverage paths that begin and end in the same cell, e.g., as used in [14]. Edges correspond to cells. Weights of edges correspond to the coverage time of the respective cells.



(b) Novel representation: Outer circle as in Fig. 3(a). Inner circle edges explicitly represent shortcut paths through cells. Shortcut edge weights are set based on minimal time for moving between cell entry and exit points, without covering it. Edges between outer circle vertices and inner circle vertices have 0 weight.

Fig. 3. Graph representations of cells and motion possibilities between them. Vertices represent cell entry/exit points.

We suggest the following double-circle data-structure: the outer circle is the simple circle G and it represents coverage of cells. The inner circle represents the shortcuts which a robot can preform (see figure 3) between cells. The following assumption should be highlighted in this context: Each robot starts at one of G's vertices. If not, the robot will go to the closest vertex.

The naive solution for this problem places all robots on the graph according to their initial position and each robot covers the cells in a clockwise direction along the circle until it reaches to the initial position of the robot next to it in G. Clearly, this algorithm does not guarantee optimal temporal efficiency.

The optimal scheduling for a given map (without cells consolidation), can be found by creating all possible scheduling options (i.e. create all options for robots' assignments) and choosing the one with minimal *makespan*. This algorithms is denoted as global. Despite the fact that this algorithm finds the optimal scheduling without cell merger, this algorithm's time complexity is exponential in the number of robots. The problem of finding the optimal scheduling (i.e. schedule with minimal *makespan*) is NP-hard [11]. Hence, we suggest two suboptimal greedy algorithms, Sequential_Robots_Tree (SeRT) and Alternate_Robot_Cell_Tree (ARCT), that have better performance than the naive algorithm that was described above.

For the first algorithm, SeRT, we present the following tree structure: Level i represents robot i and each edge represents an uncovered cell as illustrated in Figure 4. Namely, this tree represents one assignment for each robot, that is a cell to cover. If there are more robots than uncovered cells, some robots will not be assigned with a job in this round.

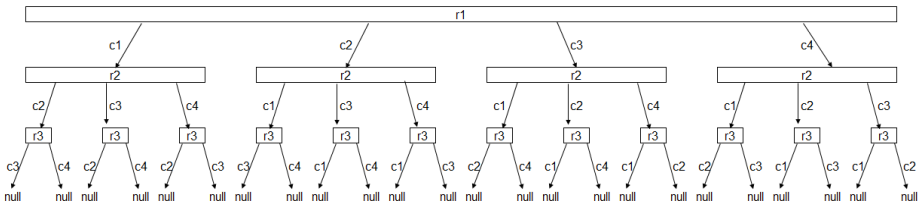


Fig. 4. In this example there are four cells to be covered (c1, c2, c3, c4) and three robots (r1, r2, r3). The first level represents r1, the second level represents r2 and the third level represents r3.

The algorithm's flow is as follows: After constructing a tree as described above, use the DFS algorithm and choose the branch with the minimal *makespan*. Assign each robot its job (robot's job is the edge that comes out of the vertex that represents this robot in the chosen branch). Finally, the cells that were covered are removed from the uncovered list and the algorithm is been called again until all cells are covered. Let n denote the number of cells to be covered, and k denotes the number of robots. Therefore, the time complexity of SeRT is

$O(\frac{n}{k}n^k)$. This algorithm guarantees complete coverage. Nonetheless, it does not guarantee optimality. For example, consider the scenario shown in Figure 5.

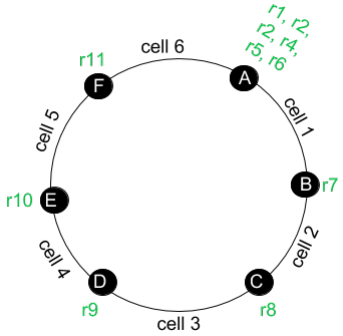


Fig. 5. In this example there are six cells to be covered and eleven robots (r1 - r11). Robots r1 - r6 are at vertex A, r7 is at vertex B, r8 is at vertex C, r9 is at vertex D, r10 is at vertex E and r11 is at vertex F.

In this example, the algorithm assigns robot r1 - r6 while robots r7 - r11 are idle. Clearly, the optimal schedule would be to assign r6 - r11, yet in this algorithm r7 - r11 can not be assigned.

As seen in this example, the order of the robots might affect the working time of each robot. Therefore, we suggest an improvement for this algorithm by Algorithm ARCT. Instead of using a tree structure such as the i-th level represents robot i, we use the following tree structure: Each level (except of the first level which represents the tree's root) represents cells to be covered and robot to cover a cell alternately. Like before, this tree represents single assignment for each robot. Let us denote the level of the tree's root as 0. Level

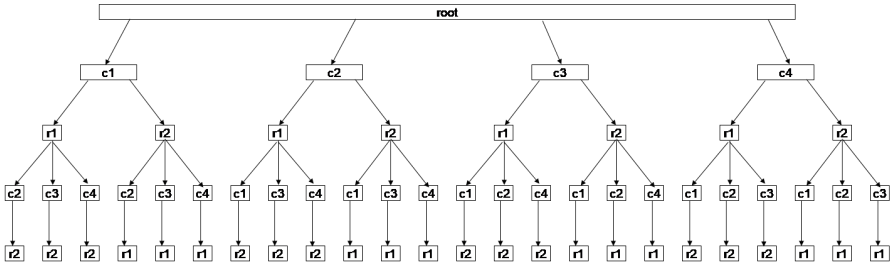


Fig. 6. In this example there are four cells to be covered (c1, c2, c3, c4) and two robots (r1 and r2).

1 represents all cells that were not covered yet. Cell i's children in this tree represent all available robots that can cover it, i.e. all robots that do not have an assignment yet in this branch. Their children represent all cells that were not covered yet in this path and so on until each robot as an assignment or all cells were covered. Figure 6 illustrates a single tree as described above. This algorithm assures complete coverage, however this algorithm does not guarantee optimal time efficiency. The time complexity of ARCT is $O(\frac{n}{k}n^k k^k)$ where k is the number of robots, and n is the number of cells to be covered.

5 Experiments

To evaluate the contributions in this paper, beyond their theoretically-proven properties, we examined coverage of a single robot and multiple robots in five different maps. For each map we randomly chose up to fifteen different initial robot locations. On each map configuration we tested our algorithms using one, three and five homogeneous robots. We assume that after consolidation of cells the robot must start cover the new cell either at the first sub-cell or at the last sub-cell. Furthermore, if there are two different coverage paths with the same *makespan*, the algorithm will choose the path that was created first. All the results were statistically analyzed using the One-Way ANOVA test, p-values are reported herein.

5.1 Single Robot

We examined five maps as mentioned above. For each map we examined all fifteen possible robot initial locations. The following diagram (Figure 7) presents the results. The x-axis represents the different map and the y-axis represents the average *makespan*. The blue columns represent the robot's coverage *makespan* before cells' merger, while the green columns represent the robot's coverage after the merger. The results have proven that shortcuts do not change the robot's *makespan* in case of a single robot. That occurs because single robot traces the circle and does not need to come around other robots. Therefore, it does not need to travel through a cell without covering it.

5.2 Multiple Robots

In this sub-section we will compare the algorithms that have been presented in Section 4, and will discuss their results.

Table 1 presents the results of the average *makespan* of Algorithm global in our experiments. The best value of each row is written in bold.

As the table shows, the merger of cells either improves or does not change the *makespan*. Moreover, the average results with the inner shortcuts without cells consolidation have better *makespan* then the average results without inner shortcuts before consolidation.

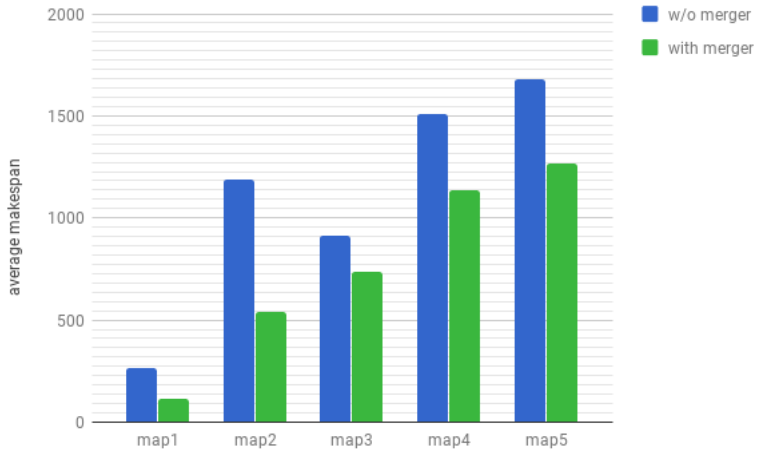


Fig. 7. Single robot chart. The x-axis represents the different map and the y-axis represents the average *makespan*.

Table 1. This table presents the average *makespan* of each category we tested. The best value of each row is written in bold font.

map #	robots #	w/o shortcuts, w/o merge	w/o shortcuts, with merge	with shortcuts, w/o merge	with shortcuts, with merge
1	3	134.00	96.67	107.20	**88.80**
	5	88.00	**55.33**	88.00	56.13
2	3	573.87	399.07	513.33	**369.47**
	5	416.53	304.00	364.20	**290.07**
3	3	470.87	422.73	428.33	**372.27**
	5	320.47	302.40	293.33	**280.40**
4	3	826.93	660.80	645.73	**532.13**
	5	519.07	489.07	382.33	**378.20**
5	3	943.93	792.53	745.20	**628.33**
	5	723.13	718.53	514.07	**508.27**

Figure 8 presents both the best value of each algorithm and the value of each greedy algorithm in the category of the global algorithm's best value from table 1. In addition, as shown in the columns of map 3 with 3 robots, the improved greedy algorithm yields better results than the global algorithm. However, as mentioned before, in case of more than one path with the same *makespan*, the first generated path was chosen. Therefore, there might be another coverage path with the same *makespan* that results a better *makespan* after consolidation.

In average, consolidation of cells significantly improves the *makespan* (p-value of the global algorithm is 4.86E-04, the p-value of ARCT is 2.16E-07 and the p-value of SeRT is 4.89E-05).

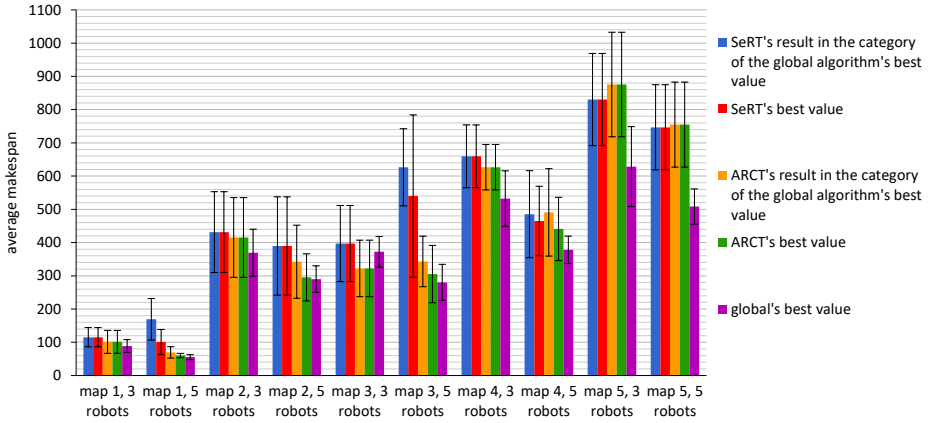


Fig. 8. The best results for all suggested algorithms compared to the result of each greedy algorithm, for three and five robots on each map.

6 Conclusions and Future Work

In this paper we address the problem of efficient coverage of a known environment. We have shown that in order to increase the temporal efficiency, it is necessary to both choose the coverage angle locally and merge cells. Furthermore, we presented an efficient method for cell consolidation that can be used for both single and multi-robot systems. Moreover, algorithms for complete coverage were presented. Both theoretic and empiric analysis addressed the efficiency of our algorithms.

Much work is left for the future. For example, the question of what is the optimal number of robots to cover a given area remains open. Moreover, in the presented experiments the cells were consolidated after the coverage path had been chosen. Thus, all the algorithms that were mentioned above can be improved. This can be achieved by choosing the coverage path with minimal *makespan* before cell merger and after it. However, this would be less efficient. In addition, one can first merge the map's cells and then divide the consolidated cells to paths. Nevertheless, it requires a different cell division method.

Acknowledgements

This research was supported by ISF grant #2306/18. As always, thanks to K. Ushi.

References

1. Agmon, N., Hazon, N., Kaminka, G.A.: The giving tree: Constructing trees for efficient offline and online multi-robot coverage. Annals of Math and Artificial Intelligence 52(2–4), 143–168 (2008)

2. Choset, H., Pignon, P.: Coverage path planning: The boustrophedon decomposition. In: International Conference on Field and Service Robotics. Canberra, Australia (December 1997)
3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9(3), 251–280 (1987)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press (1990)
5. Gabriely, Y., Rimon, E.: Spanning-tree based coverage of continuous areas by a mobile robot. Annals of Mathematics and Artificial Intelligence 31(1-4), 77–98 (2001)
6. Gage, D.W.: Command control for many-robot systems. In: The nineteenth annual AUVS Technical Symposium (AUVS-92) (1992)
7. Galceran, E., Carreras, M.: A survey on coverage path planning for robotics. Robotics and Autonomous Systems 61(12), 1258–1276 (Dec 2013), http://www.sciencedirect.com/science/article/pii/S092188901300167X, 00000
8. Guan, M.K.: Graphic programming using odd or even points. Chinese Mathematics 1(3), 273–277 (1962)
9. Hazon, N., Kaminka, G.: On redundancy, efficiency, and robustness in coverage for multiple robots. Robotics and Autonomous Systems 56(12), 1102–1114 (2008)
10. Huang, W.H.: Optimal line-sweep-based decompositions for coverage algorithms. In: Proceedings the IEEE International Conference on Robotics and Automation. pp. 27–32 (2001)
11. I. Rekleitis, A. New, E.S.R., Choset, H.: Efficient boustrophedon multi-robot coverage: an algorithmic approach. Annals of Mathematics and Artificial Intelligence 52(2–4), 109–142 (2008)
12. Karapetyan, N., Benson, K., McKinney, C., Taslakian, P., Rekleitis, I.: Efficient multi-robot coverage of a known environment. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1846 – 1852 (2017)
13. Lumelsky, V.J., Mukhopadhyay, S., Sun, K.: Dynamic path planning in sensor-based terrain acquisition. IEEE Transactions on Robotics and Automation 6(4), 462–472 (1990)
14. Xu, A., Viriyasuthee, C., Rekleitis, I.: Efficient complete coverage of a known arbitrary environment with applications to aerial operations. Autonomous Robots 36(4), 365–381 (Apr 2014)