Redundancy, Efficiency and Robustness in Multi-Robot Coverage

Noam Hazon and Gal A. Kaminka*
The MAVERICK Group, Computer Science Department
Bar Ilan University, Israel
{haoznn,galk}@cs.biu.ac.il

Abstract— **Area coverage is an important task for mobile robots, with many real-world applications. Motivated by potential efficiency and robustness improvements, there is growing interest in the use of multiple robots in coverage. Previous investigations of multi-robot coverage focuses on completeness and eliminating redundancy, but does not formally address robustness, nor examine the impact of the initial positions of robots on the coverage time. Indeed, a common assumption is that non-redundancy leads to improved coverage time. We address robustness and efficiency in a family of multi-robot coverage algorithms, based on spanning-tree coverage of approximate cell decomposition. We analytically show that the algorithms are robust, in that as long as a single robot is able to move, the coverage will be completed. We also show that non-redundant (non-backtracking) versions of the algorithms have a worst-case coverage time virtually identical to that of a single robot—thus no performance gain is guaranteed in non-redundant coverage. Moreover, this worst-case is in fact common in real-world applications. Surprisingly, however, redundant coverage algorithms lead to guaranteed performance which halves the coverage time even in the worst case.**

I. Introduction

Area coverage is an important task for mobile robots, with many real-world applications such as floor cleaning [2], de-mining [5], and harvesting. In these, a robot is given a bounded work-area, possibly containing obstacles. The robot is assumed to have an associated *tool* of a given shape [3]—often corresponding to the robot's relevant sensor or actuator range—that must visit every point within the work-area. Since the tool size is typically much smaller than the work-area, the robot's task consists of finding and moving along a path that will take the tool over the entire work-area. This is sometimes referred to as exhaustive geographical search [9], or sweeping.

In recent years, there is growing interest in the use of multiple robots in coverage, motivated by efficiency and robustness. First, multiple robots may complete the task more quickly than a single robot, by dividing the work-area between them. Second, multi-robot algorithms may succeed in face of failures, since even if a robot fails, its peers might still be able to cover its assigned area. Formally, a coverage algorithm is said to be *complete* if, for any work-area, it produces a path that completely covers the work-area. We want multi-robot algorithms to be not only complete, but also *efficient* (in that they minimize the time it takes to cover the area), *non-backtracking* (in that any portion of the work area is covered only once), and *robust* (in that they can handle catastrophic robot failures).

Previous investigations that examine the use of multiple robots in coverage mostly focus on completeness and non-backtracking. However, much of previous work does not formally consider robustness. Moreover, while completeness and non-backtracking properties are sufficient to show that a single-robot coverage algorithm is also efficient (in coverage time), it turns out that this is not true in the general case. Surprisingly, in multi-robot coverage, non-backtracking and efficiency are independent optimization criteria: Non-backtracking algorithms may be inefficient, and efficient algorithms may be backtracking. Finally, the initial position of robots in the work-area significantly affects the completion time of the coverage, both in backtracking and non-backtracking algorithms. Yet no bounds are known for the coverage completion time, as a function of the number of robots and their initial placement.

This paper examines robustness and efficiency in multi-robot coverage. We focus on coverage using a map of the work-area (known as *off-line coverage* [1]). We assume the tool to be a square of size D. The work-area is then approximately decomposed into cells, where each cell is a square of size $4D$, i.e., a square of four tool-size sub-cells. As with other approximate cell-decomposition approaches ([1]), cells that are partially covered—by obstacles or the bounds of the work-area—are discarded from consideration. We use an algorithm based on a spanning-tree to extract a path that visits all sub-cells. Previous work on generating such a path (called *STC* for Spanning-Tree Coverage) have shown it to be complete and non-backtracking [3].

We present a family of novel algorithms, called MSTC (*Multirobot Spanning-Tree Coverage*) that address robustness and efficiency. First, we construct a non-backtracking MSTC algorithm that is guaranteed to be *robust*: It guarantees that the work-area will be completely covered in finite time, as long as at least a single robot is functioning correctly. We

*This work is partially supported by ISF Grant #1211/04

analyze the best-case and worst-case completion times for this algorithm, and find that in the worst-case, the coverage time for k robots is essentially equal to that of a single robot. Unfortunately, this worst-case scenario is common in coverage applications: This is where all robots start from approximately the same position (e.g., doorway to the work-area). We further prove that this result holds for any non-backtracking algorithm that uses STC paths.

We then present a second MSTC algorithm, which allows for some backtracking: It may have a robot visit a cell twice, but no more. We show that surprisingly, even though this algorithm involves backtracking, its worst-case coverage time for $k > 2$ robots is half that of a single robot. These results show that coverage algorithms must distinguish between non-backtracking and efficiency properties. These two criteria converge only in the single-robot case, but are distinct (and may be mutually exclusive) in the general k-robot case.

II. BACKGROUND

Recent years are seeing much interest in multi-robot coverage algorithms, thanks to two key features made possible by using multiple robots: (i) robustness in face of single-robot catastrophic failures, and (ii) enhanced productivity, thanks to the parallelization of sub-tasks.

Choset [1] provides a recent survey of coverage algorithms, which distinguishes between *offline* algorithms, in which a map of the work-area is given to the robots, and *online* algorithms, in which no map is given. The survey further distinguishes between *Approximate cellular decomposition*, where the free space is approximately covered by a grid of equally-shaped cells, and *exact decomposition*, where the free space is exactly partitioned.

Our algorithms build on the single-robot off-line STC (Spanning-Tree Coverage) algorithm [3] that is based on an approximate cellular decomposition. A different approach to extending the STC algorithm to multiple robots can be found in [6], but does not carry the robustness and performance guarantees we provide below.

Spires and Goldsmith [9] show an off-line multi-robot algorithm based on an approximate cellular decomposition. The algorithm uses a Hilbert space-filling curve which guarantees a robust coverage path. Unfortunately, this works only in obstacle-free work-areas. Also, they argue that the initial positions of the robots within the work-area significantly affect the coverage time, but do not provide guarantees on the performance of their algorithm. In contrast, we provide an algorithm that handle obstacles, and is guaranteed to reduce the coverage time (compared to the single-robot case) regardless of initial positions.

Other investigations of multi-robot coverage also ignore the initial positions of the robots. Wagner et al. [11] proposes ant-based algorithms which use approximate cellular decomposition. The algorithms involve little or no direct communications, instead using simulated pheromones. These algorithms solve only the discrete coverage problem, without guaranteeing efficiency and robustness. Instead, heuristics are used to overcome bad initial starting points.

Kurbayashi et al. [4] suggest an off-line centralized multi-robots coverage algorithm based on an exact cellular decomposition. However, no guarantees on robustness are provided. Our coverage algorithms are distributed and robust.

Rekleitis et al. [7] uses two robots to cover an unknown environment, using a visibility graph-like decomposition (sort of exact cellular decomposition). The algorithm use the robots as beacons to eliminate odometry errors, but does not address catastrophic failures (i.e., when a robot dies). In a more recent article, Rekleitis et al. [8] extends the Boustrophedon approach [1] to a multi-robot version. Their algorithm also operates under the restriction that communication between two robots is available only when they are within line of sight of each other.

III. NON-BACKTRACKING MSTC

We focus in this paper on the off-line coverage case[1], [3], where the robots have a-priori knowledge of the work-area, i.e. they have a complete map of the work-area, its boundaries and all the obstacles (which are assumed to be static). Each robot has an associated tool shaped as a square of size D. The objective is to cover the work-area using this tool. In real-world applications, the tool may correspond to sensors that must be swept through the work-area to detect a feature of interest, and the size D may be determined by the effective range of the sensors. Or, in vacuum cleaning application, the tool may correspond to the opening of the vacuum itself, typically underneath the robot. As with previous work [3], we assume robots can move continuously, in the four basic directions (up/down, left/right), and can locate themselves within the work-area to within a sub-cell of size D.

We divide the area into square cells of size $4D$ (each one consists of 4 sub-cells of size D), while discarding cells which are partially covered by obstacles. We define a graph structure, G(V,E). V is the nodes set, which are the center points of each cell, and E is the edges set, which are the line segments connecting centers of adjacent cells. Then we build a spanning tree for G using any spanning-tree construction algorithm. We can affect the shape of the covering path by adding weights to the edges and building a minimum spanning tree [10]. This can be used, for instance, to reduce the number of turns, by assigning horizontal edges greater weights than those of vertical edges [3].

We can now define the MSTC problem: We are given an STC path for a given work area, and a set of k robots. We assume that the robots have initial positions $S_0, \ldots, S_{k-1}$ within the cell decomposition of the work-area. In this, we depart from previous work on multi-robot coverage which does not take into account the initial positions of the k robots.

The challenge is to assign k portions of the STC path to the different robots, such that when all the robots complete their assigned sub-paths, the entire work-area is covered.

We begin by examining an instance of this problem, where robots are assumed to be homogeneous in their same speed and tool size D. We use N to denote the number of cells in the grid, and n to denote the number of sub-cells. We further assume that the work-area is contiguous, i.e., all cells of the work-area are accessible from any starting position.

The coverage works in two phases. First, Algorithm 1 builds an STC path using the method in [3] (briefly described above). Then, to carry out the coverage, each robot uses its copy of this STC path, and its initial position on the path, to follow a sub-path that is assigned to it (Algorithm 2). This is done while making sure that robots make up for catastrophic failures of their peers. Note that the execution of Algorithm 2 is complete decentralized, as each robot executes its own independent copy.

Starting from S_0, Algorithm 1 constructs a spanning tree for G. Moving along a path which circumnavigates the spanning tree along a counterclockwise direction orders the starting points as shown in Fig. 1. The construction of the spanning-tree in this pre-process phase can be done by one robot and broadcast to the others, or it can be done by every robot independently while they use the same algorithm for the building of the tree.

Once the path has been constructed and divided into sections, Algorithm 2 is executed in a distributed fashion by all robots. After the initialization phase (lines 1–2), each robot starts to cover its section $[S_i, \ldots, S_j)$, from its current location S_i to the initial position S_j of the next robot, along the STC in a counterclockwise direction (lines 3–4, see Fig. 1). Lines 6–11 guarantee the robustness: If one robot fails, the robot behind it takes the responsibility to cover its section (see below for formal proof).

Algorithm 1 MSTC Path Plan(work-area W, robots' initial positions $S_0, \ldots, S_{k-1}$)

```
1: Arbitrarily pick the starting point $S_0$
2: Starting from $S_0$, construct $P$, an STC path of $W$ (as
   described above).
3: Order the positions $S_0, \ldots, S_{k-1}$ along the STC, starting
   from $S_0$ and moving in a counter-clockwise direction.
4: Return $P$, ordered list of positions $S_0, \ldots, S_{k-1}$.
```

Note that the algorithm addresses communication requirements in general form. In practice, communications can be implemented in many different ways. For example, the status of liveness (lines 6, 8) can be determined by the robots' sending of an "I am alive" message every period of time. When a message is not received by a robot after a defined timeout period, it is considered dead. Alternatively, liveness can be checked when reaching the initial position of another robot. Similarly, the announcement of section completion (line 5) can be communicated in various ways.

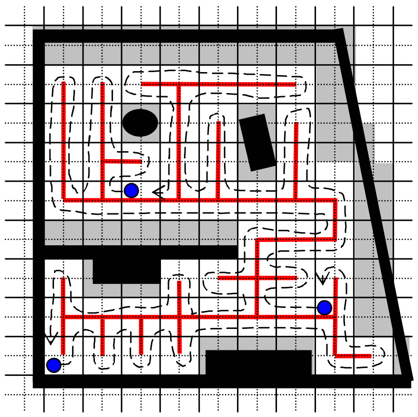
Fig. 1. The grid, the spanning tree and the paths for three robots.

Algorithm 2 non-backtracking MSTC(STC path P, ordered positions $S_0, \ldots, S_{k-1}$)

```
1: Let $i \leftarrow$ my own id (in the range $0 \ldots k-1$)
2: Let $t \leftarrow (i+1)$ $mod$ $k$ // next robot's position, cyclically
3: while current position $\neq S_t - 1$ do
4:     Move towards $S_t$ along STC, in counter-clockwise
       direction // this changes current position
5: Announce completion of $[S_i, S_t)$
6: while $R_t$ is alive and $[S_0, \ldots, S_{k-1}, S_0]$ incomplete do
7:     Wait
8: if $R_t$ is not alive and $[S_0, \ldots, S_{k-1}, S_0]$ incomplete then
9:     $i \leftarrow t$
10:    $t \leftarrow (t+1)$ $mod$ $k$
11:    Go to 3 // Take over role of failing robot
12: Stop
```

We analyze these algorithms. First, to prove completeness and optimality we remind the reader that circumnavigating the spanning tree produce a closed curve which visits all the sub-cells exactly one time [3]. In Algorithm 1 the STC curve is partitioned into k sections whose union is the whole path. That leads to the completeness theorem below.

Theorem 3.1 (Completeness): Algorithm 1 generates k paths that together cover every cell accessible from the starting cell S_0.

Proof: Previous work has shown that step 2 produces a path that covers all cells (Lemma 3.3 in [3]). Step 3 partitions this path into k sections. Therefore, the union of the k sections covers every cell accessible from S_0. □

Given the set of paths produced, Algorithm 2 makes sure the robots visit all these cells only once (if no failure has occurred). The following theorem applies.

Theorem 3.2 (Non-Backtracking): If all robots use Algorithm 2, and no robot fails, no cell is visited more than once.

Proof: If no robot fails, then each robot i only covers the section $[S_i, S_{i+1})$ of the STC path (where if $i = k$, then cyclically $i + 1 = 0$). Thus every cell is covered only by a single robot. Since robots never backtrack, every point is only covered once. □

Robustness. As key motivation for using multiple robots comes from robustness concerns, we prove that Algorithm 2 above is robust to catastrophic failures, where robots fail and can no longer move. This result relies on an assumption that robots which fail do not block live robots.

Theorem 3.3 (Robustness): Algorithm 2 guarantees that the coverage will be completed in finite time even with up to $k-1$ robots failing.

Proof: The path is divided to k sections. We will prove that each section will be covered. Due to the nature of the path generated, all the robots are topologically moving in a circle, so the robot that is responsible to cover a section has $k-1$ robots behind it. This is correct for any section i. We will prove that this section i will be covered, by induction on the number of robots k.

Induction Base ($k = 3$). If robot R_i that is responsible to cover this section is not dead before the completion of the cover of this section, then this section is covered. Else, $R_{(i-1)\ mod\ k}$ or $R_{(i-2)\ mod\ k}$ is alive. If $R_{(i-1)\ mod\ k}$ is alive, according to line 6 in the algorithm it will return to step 3 and cover this section. If only $R_{(i-2)\ mod\ k}$ is alive, according to line 6 in the algorithm it will return to step 3 and cover section $i-1$ (because $R_{(i-1)\ mod\ k}$ is not alive). Then the condition will be true again because R_i is dead, and $R_{(i-2)\ mod\ k}$ will cover also section i.

Induction Step. Suppose it is known that if at least one of k robots is alive section i will be covered. We will prove it for $k+1$ robots.

If robot R_i that is responsible to cover this section is not dead before the completion of the cover of this section, then this section is covered. Otherwise, there is at least one of k robots behind it that is alive. According the induction step, every section within k sections behind R_i will be covered, including the section behind it. The robot that will cover this section will cover also section i (according to line 6 in the algorithm, because R_i is not alive). □

Robustness is guaranteed with a simple mechanism. There is no need to reconfigure the group after a robot failed. It also does not matter which robot fails or how many robots failed at the same time.

Robustness against collisions is an additional concern with multiple robots. Normally, as each robot only covers its own section, theorem 3.2 also guarantees that no collisions take place, as the STC path never crosses itself. In practice, localization and movement errors may cause the robot to move away from its assigned path, and thus risk collision. Despite this, the separation between the paths of different robots decreases the chance of collisions.

Efficiency. Additional important motivation for using multiple robots is the possibility of reducing the coverage time by parallelizing portions of the coverage. In single-robot settings, guarantees of completeness and non-backtracking are sufficient to show (in combination) optimality of coverage time, since every cell is visited, but only once (the minimum). Thus n cells are covered in n steps.

To analyze the number of steps required to complete the coverage, we have to take into account the initial configuration. We define the *running time* as the maximum over the steps that each robot has to go, $\max_{i \in k} step(i)$, where $step(i)$ is the total number of steps taken by robot i.

Using multiple robots, the hope is to reduce the coverage time to approximately n/k. Indeed, the following theorem shows this to be a best-case scenario for Algorithm 2.

Theorem 3.4 (MSTC Best Case): The best running time for the algorithm is $\lceil \frac{n}{k} - 1 \rceil$

Proof: The best-case scenario is when the starting positions $S_0, \ldots, S_{k-1}$ place the robots at equal distance from each other, thus partitioning the STC path into k sections, each of size n/k. □

Unfortunately, it turns out that the running time is critically dependent on the initial positions of the robots. Indeed as the following theorem shows, the worst case scenario for Algorithm 2 has a running time that is almost equivalent to that of a single robot.

Theorem 3.5 (MSTC Non-Backtracking Worst Case): The worst running time for the non-backtracking algorithm is $n - k - 1$

Proof: The worst-case scenario is where all the robots start next to each other, on adjacent cells. Since all robots move in the same direction, all but one robot will only cover the cell they are on before reaching the end of their assigned section. One robot will have a section assigned to that contains all $n - k$ remaining sub-cells (Fig. 2). □

The result demonstrates that the initial position of the robot within the work-area can adversely affect the coverage time. Unfortunately, the worst-case scenario is common in real-world applications, e.g., vacuuming (all robots start from a single doorway), de-mining (all robots start from a single entry point to the mine field), or lawn mowing (all robots start at the mower storage area).

This worst-case scenario may appear deceptively simple to address. One may reason that by allowing another robot to head in the opposite direction, two robots may cover the $n - k$ section in parallel, thus completing the coverage in approximately $n/2$ (Fig. 3). However, it turns out that this is

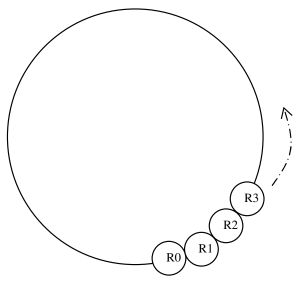

Fig. 2. The worst case scenario in the non-backtracking algorithm.

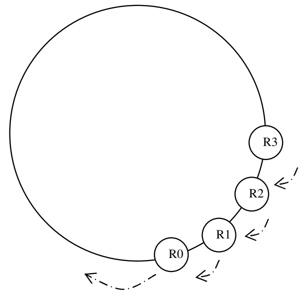

Fig. 4. The worst case scenario for any non-backtracking algorithm.

incorrect.

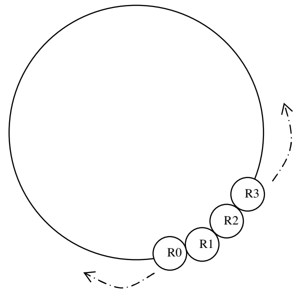

Fig. 3. Naive solution for the non-backtracking worst case scenario.

A more general result is proven below, and shows that the worst-case scenario is in fact much more general than for Algorithm 2. Indeed, it is applicable to any STC-based algorithm that is non-backtracking.

Theorem 3.6 (General Non-Backtracking Worst Case): Any non-backtracking covering algorithm based on partitioning the spanning tree path to sections, has a worst-case running time of $n - 2(k-1) - 1$.

Proof: Consider the case where robots are positioned such that a single empty sub-cell separates each pair (Fig. 4). Because no backtracking is allowed, only one of the extreme robots can cover the big part of the path. The others, including the extreme robot from the other side, can cover only the empty sub-cell next to them, regardless the method that the algorithm chooses for deciding on a direction for movement. So we get $k-1$ robots that cover two squares (their square, and the square next to them), and one robot that has to cover the rest of the path $n - 2(k-1) - 1$. □

In other words, there is no non-backtracking algorithm for setting the coverage direction of the coverage for different robots such that the worst case above is eliminated. We remind the reader that the requirement for non-backtracking movement is inherited from the single-robot STC algorithm [3], where it also leads to optimality in coverage time. The next section examines what happens when we remove the requirement of non-backtracking movement.

IV. Backtracking MSTC

Let us examine an instance of the worst-case scenario of a non-backtracking algorithm, with only two robots that are positioned such that there is a single empty sub-cell between them. Without backtracking, one of the robots would have to commit to covering the single sub-cell, while the other would then be forced to cover the remaining $n-3$ sub-cells. However, if we allow robots to backtrack, then the robot that covers the single sub-cell would be able to cover it, then backtrack, and head in the other direction. The two robots would then meet approximately in the middle of the $n-3$ section, thus halving the coverage time.

Naturally, a new worst-case scenario can be found for this back-tracking case. In this scenario, the initial positions of the two robots separated by are a third of the STC path. One robot thus covers $2/3$ of the path, while the other robot goes a $1/3$ of the path in one direction and then backtracks, but it can't help the first one in its section. The overall coverage time will be $2n/3$.

To define a general back-tracking algorithm, let us first define a few helpful notations. sec_i is the section that robot R_i is responsible to cover. Unlike in the non- backtracking algorithm sometimes $sec_i \neq [S_i, S_{i+1})$. $sec_i \leftarrow [S_1, S_2)$ means that the section starts at S_1 and ends just before S_2. The point $S_2 \leftarrow S_1 + L$, is the point in a distance of L from S_1 when moving in a counterclockwise direction along that STC path. $direction1_i$ is the initial direction of movement for robot i, while $direction2_i$ is the direction of movement for robot i if it has to backtrack.

We now turn to describing the MSTC backtracking algorithm. The first phase of building the STC and ordering the starting point is the same as in the non-backtracking case (Algorithm 1). During run-time, the robots re-divide the sections if backtracking is needed (Algorithm 3). They then follow the backtracking algorithm (Algorithm 4). We present here the general case for $k > 2$ robots (the case of $k = 2$ robots is somewhat different, and we skip it for lack of space).

The idea of the initialization phase is to allocate sections and directions of movement to the robots. If there is no part of the path that is longer than half of the entire STC path, all the sections and directions of movement are the same as in the non-backtracking algorithm (Algorithm 3, lines 1–4). Otherwise, the two robots that have this section between them share its coverage. One of them will have to go in a clockwise direction, leaving to the robot next to it (from the other side) to also cover the distance between them. To avoid the case that this robot will have to cover more than half of the path because of the backtracking, this robot gets help from one of its neighbors—the one closest to it. They both cover half of the distance between them and return to cover their original part of the path. See Fig. 5 for example of this situation.

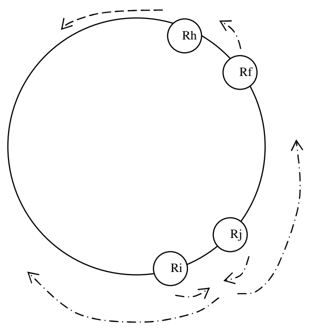

Fig. 5. An execution example of Algorithm 4. The indexes used are the same as in the initialization phase.

The backtracking algorithm (Algorithm 4) follows the re-divided sections generated in the initialization phase, similarly to the way the non-backtracking algorithm does. Algorithm 4 also ensures that only after a robot finishes to cover its section, even if it includes going in one direction and then backtrack, it covers sections of dead robots. Thus this algorithm is also robust.

The algorithm's completeness and robustness can be proven similarly to the completeness and robustness of the non-backtracking Algorithm 2. With respect to its backtracking, it can be shown that any point that is covered more than once, is covered by the same robot, and that there is no point that is covered more than twice. We skip these proofs for reasons of space.

The best-case coverage time for the backtracking MSTC algorithm is the same as for the non-backtracking version, i.e., $n/k - 1$. This is because in the best case, the initial positions of the robots are equidistant, and the robots can cover their sections without backtracking. The worst-case coverage time is analyzed below:

Theorem 4.1 (MSTC Backtracking Worst Case): The worst-case running time for the backtracking algorithm is $n/2 - 1$ when $k > 2$, and $\lceil 2n/3 - 1 \rceil$ when $k = 2$.

Proof: There are two cases, depending on the value of k. For lack of space, we only show the case $k > 2$. If there is no section that is longer than half of the path, then when every robot covers its section, no robot covers more than half of the path. On the other hand, if there is a section longer than half the path, then necessarily it is the only one. We denote it as $[S_h, S_i)$ (as in the algorithm). There are three possible cases:

Algorithm 3 initialization phase(STC path P, ordered positions $S_0, \ldots, S_{k-1}$)

```
1:  for all $i$ such that $0 \le i \le k-1$ do
2:      Let $sec_i \leftarrow [S_i, S_{(i+1)\ mod\ k})$
3:      Let $direction1_i \leftarrow$ counterclockwise
4:      Let $direction2_i \leftarrow$ null
5:  if there is $h$ such that $sec_h > \frac{1}{2}(\sum_0^k [S_i, S_{(i+1)\ mod\ k}))$ then
6:      $i \leftarrow (h+1)\ mod\ k$
7:      $j \leftarrow (i+1)\ mod\ k$
8:      $f \leftarrow (j+1)\ mod\ k$
9:      if $[S_i, S_j) < [S_j, S_f)$ then
10:         $sec_h \leftarrow [S_h, S_h + \lceil \frac{[S_h,S_i)-[S_i,s_j)}{2} \rceil)$
11:         $sec_i \leftarrow [\lceil \frac{[S_h,S_i)-[S_i,S_j)}{2} \rceil, \lceil \frac{[S_i,S_j)}{2} \rceil)$
12:         $direction1_i \leftarrow counterclockwise$
13:         $direction2_i \leftarrow clockwise$
14:         $sec_j \leftarrow [\lceil \frac{[S_i,S_j)}{2} \rceil, S_f)$
15:         $direction1_j \leftarrow clockwise$
16:         $direction2_j \leftarrow counterclockwise$
17:     else
18:         $direction1_j \leftarrow counterclockwise$
19:         $direction2_j \leftarrow clockwise$
20:         if $h = f$ then
21:             $sec_h \leftarrow [\lceil \frac{[S_j,S_h)}{2} \rceil, \lceil \frac{[S_h,S_i)}{2} \rceil)$
22:             $direction1_h \leftarrow clockwise$
23:             $direction2_h \leftarrow counterclockwise$
24:             $sec_i \leftarrow [\lceil \frac{[S_h,S_i)}{2} \rceil, S_i)$
25:             $direction1_i \leftarrow clockwise$
26:             $sec_j \leftarrow [S_i, \lceil \frac{[S_j,S_h)}{2} \rceil)$
27:         else
28:             $sec_h \leftarrow [S_h, \lceil \frac{[S_h,S_i)}{2} \rceil)$
29:             $sec_i \leftarrow [\lceil \frac{[S_h,S_i)}{2} \rceil, S_i)$
30:             $direction1_i \leftarrow clockwise$
31:             $sec_j \leftarrow [S_i, \lceil \frac{[S_j,S_f)}{2} \rceil)$
32:             $sec_f \leftarrow [\lceil \frac{[S_j,S_f)}{2} \rceil, S_{(f+1)\ mod\ k})$
33:             $direction1_f \leftarrow clockwise$
34:             $direction2_f \leftarrow counterclockwise$
```

- $[S_i, S_j) < [S_j, S_f)$. $[S_i, S_j) + [S_j, S_f) <$ half of the entire path $\Rightarrow [S_i, S_j) < 1/4$ of the entire path. R_j passes twice over half of $[S_i, S_j)$ and over $[S_j, S_f)$ so the its total path is: $[S_i, S_j) + [S_j, S_f) <$ half of the entire path. R_i passes twice over half of $[S_i, S_j)$ and over $[S_h, S_i)$ until it meets R_h. In the time that

Algorithm 4 backtracking MSTC(STC path P, ordered positions $S_0, \ldots, S_{k-1}$)

```
Require: initialization phase
 1: Let $s \leftarrow$ my own id (in the range $0 \ldots k-1$)
 2: Let $t \leftarrow (s+1)\ mod\ k$ // next robot's position, cyclically
 3: while current position $\neq$ one edge of your $sec$ do
 4:     Move towards edge of your $sec$ along STC, according
        your $direction1$ argument
 5: if your $direction2 \neq null$ then
 6:     your $direction1 \leftarrow$ your $direction2$
 7:     your $direction2 \leftarrow null$
 8:     Go to 3
 9: else
10:     Announce completion of your $sec$
11:     while $R_t$ is alive and there is $i$ such that $sec_i$ incom-
        plete do
12:         Wait
13:     if $R_t$ is not alive and and there is $i$ such that $sec_i$
        incomplete then
14:         $s \leftarrow t$
15:         $t \leftarrow (t+1)\ mod\ k$
16:         Go to 3 // Take over role of failing robot
17:     Stop
```

R_j passes half of $[S_i, S_j)$ and backtracks, R_h passes this distance to $R_j \Rightarrow$ The remaining area to cover is $([S_h, S_i)+[S_i, S_j))/2 \Rightarrow$ The number of steps for every one of them is: $\frac{[S_h,S_i)+[S_i,S_j)}{2}+[S_i, S_j) = [S_h, S_i)/2+[S_i, S_j)/2$. $[S_h, S_i) \leq n - ([S_i, S_j) + [S_j, S_f)) \Rightarrow [S_h, S_i)/2 + [S_i, S_j)/2 \leq n - [S_j, S_f)/2 \leq n/2 - 1$

- $[S_i, S_j) \geq [S_j, S_f)$ **and** $h = f$. This case could only happen with three robots, and the proof is similar to the proof in the previous case.
- $[S_i, S_j) \geq [S_j, S_f)$ **while** $h \neq f$. R_f passes twice over half of $[S_j, S_f)$ and over $[S_f, S_{(f+1)\ mod\ k})$, so its total path is $[S_j, S_f) + [S_f, S_{(f+1)\ mod\ k})$. R_j passes twice over half of $[S_j, S_f)$ and over $[S_i, S_j)$, so its total path is $[S_i, S_j) + [S_j, S_f)$. $[S_h, S_i) >$ half of the entire path $\Rightarrow [S_i, S_j) + [S_j, S_f) + [S_f, S_{(f+1)\ mod\ k}) \leq$ half of the entire path $\Rightarrow R_j$ and R_f covered less than half of the entire path. R_h and R_i passes half of $[S_h, S_i)$. $[S_h, S_i) <$ length of the entire path $\Rightarrow R_h$ and R_i covered less than half of the entire path.

Thus in all cases, three or more robots take no more than $n/2 - 1$ to complete coverage. □

The key insight offered by these results is that non-backtracking, the property that no portion of the work-area is covered more than once, is a distinct performance criteria from that of efficiency. These converge in the single robot case, but not in general. Indeed, it can be shown that *only* utilizing some backtracking can we guarantee improved coverage time. To see this, consider a case where the two robots are behind each other, in a corridor leading into the work area. Unless the second robot covers at least a portion of the area covered by the first robot, there is no way for the robots to split the covering task between them. Without some redundancy, the first robot will necessarily have to cover almost all of the work area by itself.

V. Conclusion and Future Work

We presented algorithms for multi-robot coverage, that are *complete* and *robust* in face of catastrophic robot failures. We examined the efficiency of these algorithms in terms of coverage time, and have shown that the initial positions of the robots have significant impact on the coverage time. In particular, while all algorithms carry the potential for best-case coverage in time n/k (where n is the number of cells, and k the number of robots), non-backtracking coverage has a worst-case time essentially equal to that of a single robot. Unfortunately, this is the common case where robots start right next to each other. In contrast, the backtracking algorithm is guaranteed to halve the coverage time of a single robot. The results shed new light on multi-robot coverage problems, and show that we must distinguish between redundancy and efficiency, as these are application-dependent optimization criteria. While the backtracking algorithm is guaranteed to perform better than single-robot coverage, it is not necessarily optimal. We intend to explore optimal efficiency in the future.

Acknowledgments. We thank Moshe Lewenstein for useful discussions. K. Ushi and Shira deserve special thanks.

References

[1] H. Choset. Coverage for robotics—a survey of recent results. *Ann. Math. and AI*, 31:113–126, 2001.
[2] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA CIRFFSS*, 1994.
[3] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Ann. Math. and AI*, 31:77–98, 2001.
[4] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by multiple mobile robots. In *ICRA-96*, 1996.
[5] J. Nicoud and M. Habib. The pemex autonomous demining robot: Perception and navigation strategies. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robot Systems*, pages 1:419–424, 1995.
[6] A. M. Nikolaus Correll, Kjerstin Easton and J. Burdick. Distributed exploration and coverage. www.coro.caltech.edu.
[7] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *IJCAI-97*, volume 2, pages 1340–1345, Nagoya, Japan, August 1997. Morgan Kaufmann Publishers, Inc.
[8] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team based coverage. In *ICRA-04*, pages 3462–3468, New Orleasn, LA, April 2004.
[9] S. V. Spires and S. Y. Goldsmith. Exhaustive geographic search with mobile robots along space-filling curves. In *Proceedings of the First International Workshop on Collective Robotics*, pages 1–12. Springer-Verlag, 1998.
[10] R. E. Tarjan. *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, 1983.
[11] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Trans. Robotics Autom.*, 15(5):918–933, 1999.