On Redundancy, Efficiency, and Robustness in Coverage for Multiple Robots

Noam Hazon and Gal A. Kaminka

The MAVERICK Group

Computer Science Department

Bar Ilan University, Israel

{*hazonn,galk*}*@cs.biu.ac.il*

Abstract

Motivated by potential efficiency and robustness gains, there is growing interest in the use of multiple robots for *coverage*. In coverage, robots visit every point in a target area, at least once. Previous investigations of multi-robot coverage focus on completeness of the coverage, and on eliminating redundancy, but do not formally address robustness. Moreover, a common assumption is that elimination of redundancy leads to improved efficiency (coverage time). We address robustness and efficiency in a novel family of multi-robot coverage algorithms, based on spanning-tree coverage of approximate cell decomposition of the work area. 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. Surprisingly, however, redundant coverage algorithms lead to guaranteed performance which halves the coverage time even in the worst case. We present a polynomial-time redundant-coverage algorithm, whose coverage time is optimal,

Preprint submitted to Elsevier *25 February 2008*

and which is able to address robots heterogeneous in speed and fuel. We compare the performance of all algorithms empirically and show that use of the optimal algorithm leads to significant improvements in coverage time.

Key words: multi-robot systems, coverage, path-planning

1 Introduction

Area coverage is an important task for mobile robots, with many real-world applications such as floor cleaning [5], de-mining [15], and surveillance by unmanned aerial vehicles (UAVs) [3, 10]. 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 [8]—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 [19], or sweeping [9].

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), and *robust* (in that

they can handle catastrophic robot failures). We may additionally want the algorithm to be *non-redundant* (*non-backtracking*), in that any portion of the work area is covered only once.

Previous investigations that examine the use of multiple robots in coverage focus on guaranteeing completeness and non-redundancy. 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-redundancy and efficiency are independent optimization criteria: Non-backtracking algorithms may be inefficient, and efficient algorithms may use 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 a static work-area (known as *off-line coverage* [4]). 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 ([4]), 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 [8].

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 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 set of MSTC algorithms, which allows for some backtracking: They may have a robot visit a cell twice, but no more. We show that surprisingly, even though backtracking algorithms inherently involve redundancy, their worst-case coverage time for $k > 2$ robots is half that of a single robot. Two backtracking algorithms are shown. The first (simple) is linear-time. However, in experiments, this algorithm had approximately the same coverage-time as the non-backtracking algorithm.

We thus introduce a polynomial-time *optimal* backtracking MSTC algorithm, and show empirically that its coverage time is significantly better than the simple algorithm. These analytical and empiric results show that coverage algorithms must distinguish between redundancy and efficiency. These two criteria converge only in the single-robot case, but are distinct (and may be mutually exclusive) in the general k-robot case.

The paper is organized as follows. The next section presents related work and

motivation. Section 3 defines the MSTC problem, and the non-backtracking algorithm. Section 4 presents the simple and optimal backtracking algorithms, and their extensions to heterogeneous robots. The next section (Section 5) presents experimental results evaluating the coverage time of the different algorithms. We conclude with a summary and brief overview of future work.

2 Background

Recent years are seeing much interest in multi-robot coverage algorithms, motivated by two opportunities 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 [4] provides a 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. The work presented in this paper focuses on offline, approximate-decomposition algorithms.

Our algorithms build on the single-robot off-line STC (Spanning-Tree Coverage) algorithm [8] that is an approximate cellular-decomposition technique. A different approach to extending the STC algorithm to multiple robots can be found in [6, 7]. This approach does not carry the robustness and performance guarantees our algorithms provide.

Zheng et al. [27] describe Multi-Robot Forest Coverage (MRFC), another

multi-robot variant of the spanning-tree coverage framework. In their algorithm, the work-area is divided into a set of spanning trees (a spanning forest), which are then traversed separately by robots (i.e., each robot covers a single tree in the forest). They show empirically that the MRFC algorithms provide better coverage time than the simple backtracking and non-backtracking algorithms we present; a comparison against the optimal backtracking algorithm presented in this paper has not been carried out. The efficiency gains of MRFC come at a cost: MRFC relies on an assumption that more than one robot can occupy a single cell at the same time, in contrast to our work. Moreover, MRFC does not guarantee robustness.

Wagner at. al. [23–25] and later Osherovich et. al. [16] propose a family of robust multi-robot ant-based algorithms which use approximate cellular decomposition. The algorithms involve little or no direct communications, and rely on no map memory, instead using simulated pheromones for communications. The algorithms are provably robust, but are not necessarily efficient: In contrast to the minimal capabilities of ant robots (other than pheromone use), our algorithms make use of the robots' memory and broadcast communication to carry out redundant coverage (backtracking over areas already covered) only when strictly necessary to improve efficiency.

Svennebring and Koenig [21] offer a feasibility study of ant-based online coverage. They perform experiments with real ant-robots and large-scale simulations. They show that the algorithms result in robust coverage, but provide no analytic guarantees of completeness or efficiency.

In contrast to our approximate cellular-decomposition approach, a number of multi-robot coverage algorithms have focused on exact decomposition, which

provides improved coverage area. Kurbayashi et al. [14] 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. Batalin and Sukhatme [2] offer two coverage algorithms by a multi-robot system in which the robots spread out in the terrain, and move away from each other while covering the area and minimizing the interaction between the robots. In their work, they aim to achieve optimal coverage *area*, and do not prove any formal statement regarding optimality of coverage time.

Rekleitis et al. developed a family of important coverage algorithms. In [17], they report on using two robots to cover an unknown environment, using a visibility graph-like decomposition (a type of exact cellular-decomposition). The algorithm use the robots as beacons to eliminate odometry errors, and is thus robust to positioning errors. However, it does not address catastrophic failures (i.e., when a robot dies). Our work almost perfectly complements this work: MSTC algorithms assume good positioning, but are robust to catastrophic failures. In [18] and [13], they report on a multi-robot version of a coverage method known as the Boustrophedon. The algorithm utilizes communications, even under the restriction that communication between two robots is available only when they are within line of sight of each other, a restriction not addressed in our work. However, there are no guarantees of robustness to catastrophic failures.

While most multi-robot coverage algorithms focus on efficiency, Spires and Goldsmith [19] present a robust off-line multi-robot algorithm based on an approximate cellular decomposition, which uses a Hilbert space-filling curve. Unfortunately, this works only in obstacle-free work-areas. Also, they do not

provide bounds on the performance of their algorithm, given the initial positions of the robots within the work-area. 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.

There are methods in operations research [20, 26], which tackle related problems. However, the goals of these methods is to optimize the the efforts to be spent in searching for static or moving targets in a work-area. This task is different from coverage in that (1) special attention has to be paid to the possible movement trajectories of the targets, and (2) the robots task terminates as soon as a target is found, while coverage terminates only when the entire area is covered.

3 Non-Backtracking MSTC

We focus in this paper on the off-line coverage case [4, 8], 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 [8], we assume robots can move continuously, in the four basic directions (back/forth, left/right), and can locate themselves within the work-area to within a sub-cell of size D. Note that this specifically

restricts our attention to cases where movement involves only two cells of size D: The cell in which the movement originates and an adjacent target cell with whom it shares an edge.

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 [1, 22]. This can be used, for instance, to reduce the number of turns, by assigning horizontal edges greater weights than those of vertical edges [8].

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 [8] (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.

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 $S_0, \ldots, S_{k-1}$.

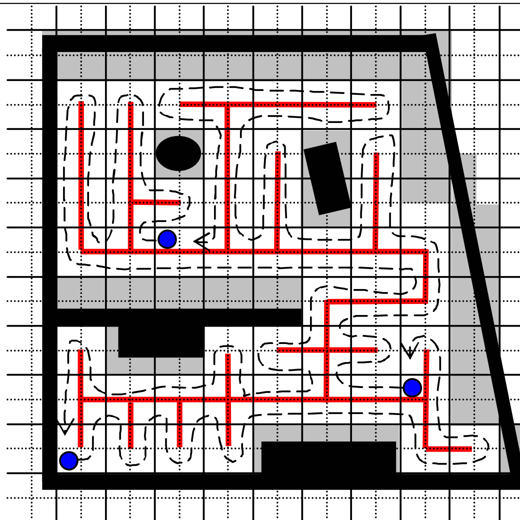


Fig. 1. The grid, the spanning tree and the paths for three robots.

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 5–11 guarantee the robustness: If one robot fails, the robot behind it takes the responsibility to cover its section (see below for formal proof). To ease the notation, we denote $(a+b)\texttt{mod}k$ as $a \oplus b$, and $(a-b)\texttt{mod}k$ as $a \ominus b$, where k is the number of 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 $t \leftarrow i \oplus 1$ // 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: $t \leftarrow t \oplus 1$
11: Goto 3 // Take over role of failing robot
12: Stop.

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.

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 [8]. 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 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 [8]). 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 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 $[S_i, S_{i\oplus 1})$ 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 one 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. While this is an

obvious concern with unmanned ground vehicles (UGVs), it is actually an assumption that is satisfied in applications of coverage to unmanned aerial vehicles [3, 10].

Theorem 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, $R_{i\ominus 1}$ or $R_{i\ominus 2}$ is alive. If $R_{i\ominus 1}$ is alive, according to line 6 in the algorithm it will return to step 3 and cover this section. If only $R_{i\ominus 2}$ is alive, according to line 6 in the algorithm it will return to step 3 and cover section $i-1$ (because $R_{i\ominus 1}$ is not alive). Then the condition will be true again because R_i is dead, and $R_{i\ominus 2}$ 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 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 sce-

nario for Algorithm 2.

Theorem 4 (MSTC Non-Backtracking Best Case) *The best running time for Algorithm 2 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.

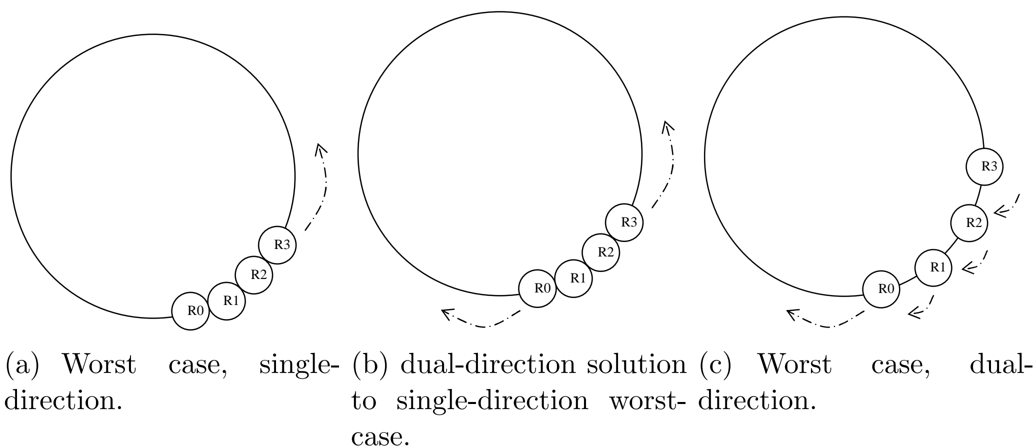
Theorem 5 (Uni-Directional Non-Backtracking Worst Case) *The worst running time for Algorithm 2* $n - k - 1$.

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(a)). □

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 may readily occur 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. 2(b)). However, it turns out that this is incorrect.



(a) Worst case, single-direction. (b) dual-direction solution to single-direction worst-case. (c) Worst case, dual-direction.

Fig. 2. Non-backtracking worst-cases, single and dual directions.

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

Theorem 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. 2(c)). 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 $n-2(k-1)-1$. $\square$

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 [8], where it also leads to optimality in coverage time. The next section examines what happens when we remove the requirement of non-backtracking movement.

4 Backtracking MSTC

We begin by examining the simplest case of backtracking MSTC algorithm, and show that backtracking can lead to a lower worst-case coverage time (Section 4.1). We then develop an optimal algorithm for the backtracking case (Section 4.2).

4.1 Simple 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 $sec_i \neq [S_i, S_{i\oplus 1})$ (The section which starts at S_i and ends just before $S_{i\oplus 1}$ when moving in a counterclockwise direction along the STC path, as defined before). We use $|[S_l, S_j]|$ to denote the length of the section $[S_l, S_j]$, taken along the shortest path along the STC cycle. The point $S_i + L$ is the point in a distance of L from S_i when moving in a counterclockwise direction along the STC path. $D1_i$ is the initial direction of movement for robot i, while $D2_i$ 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). We add another initialization phase where 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.

Fig. 3 illustrates the case where $|[S_i, S_j)| < |[S_j, S_f)|$. In this case, R_i and R_j cover together the distance between them and then backtrack. R_i returns to help R_h to cover the path between them and R_j move to cover its original pre-allocated section (Algorithm 3, lines 9–16). If R_j is closer to R_f than to R_i, R_f and R_j cover together the distance between them and then backtrack. R_j returns to cover R_i's original section (because R_i helps R_h to cover its section), and R_f move to cover its original pre-allocated section (Algorithm 3, lines 17–34). In this situation Algorithm 3 needs to distinguish between the case of only three robots (lines 20–26) to the case of more than three robots (lines 27–34).

The backtracking algorithm (Algorithm 4) follows the re-divided sections generated in the initialization phase, similarly to the way the non-backtracking

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

1: **for all** i such that $0 \leq i \leq k-1$ **do**
2: Let $sec_i \leftarrow [S_i, S_{i \oplus 1})$
3: Let $D1_i \leftarrow$ counterclockwise
4: Let $D2_i \leftarrow$ null
5: **if** there is h such that $sec_h > \frac{1}{2}(\sum_0^k |[S_i, S_{i \oplus 1})|)$ **then**
6: $i \leftarrow h \oplus 1$
7: $j \leftarrow i \oplus 1$
8: $f \leftarrow j \oplus 1$
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: $sec_i \leftarrow [S_h + \lceil \frac{|[S_h,S_i)| + |[S_i,S_j)|}{2} \rceil, S_i + \lceil \frac{|[S_i,S_j)|}{2} \rceil)$
12: $D1_i \leftarrow counterclockwise$
13: $D2_i \leftarrow clockwise$
14: $sec_j \leftarrow [S_i + \lceil \frac{|[S_i,S_j)|}{2} \rceil, S_f)$
15: $D1_j \leftarrow clockwise$
16: $D2_j \leftarrow counterclockwise$
17: **else**
18: $D1_j \leftarrow counterclockwise$
19: $D2_j \leftarrow clockwise$
20: **if** $h = f$ **then**
21: $sec_h \leftarrow [S_j + \lceil \frac{|[S_j,S_h)|}{2} \rceil, S_h + \lceil \frac{|[S_h,S_i)| + |[S_j,S_h)|}{2} \rceil)$
22: $D1_h \leftarrow clockwise$
23: $D2_h \leftarrow counterclockwise$
24: $sec_i \leftarrow [S_h + \lceil \frac{|[S_h,S_i)| + |[S_j,S_h)|}{2} \rceil, S_i)$
25: $D1_i \leftarrow clockwise$
26: $sec_j \leftarrow [S_i, \lceil \frac{|[S_j,S_h)|}{2} \rceil)$
27: **else**
28: $sec_h \leftarrow [S_h, S_h + \lceil \frac{|[S_h,S_i)|}{2} \rceil)$
29: $sec_i \leftarrow [S_h + \lceil \frac{|[S_h,S_i)|}{2} \rceil, S_i)$
30: $D1_i \leftarrow clockwise$
31: $sec_j \leftarrow [S_i, S_j + \lceil \frac{|[S_j,S_f)|}{2} \rceil)$
32: $sec_f \leftarrow [S_j + \lceil \frac{|[S_j,S_f)|}{2} \rceil, S_{f \oplus 1})$
33: $D1_f \leftarrow clockwise$
34: $D2_f \leftarrow counterclockwise$

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

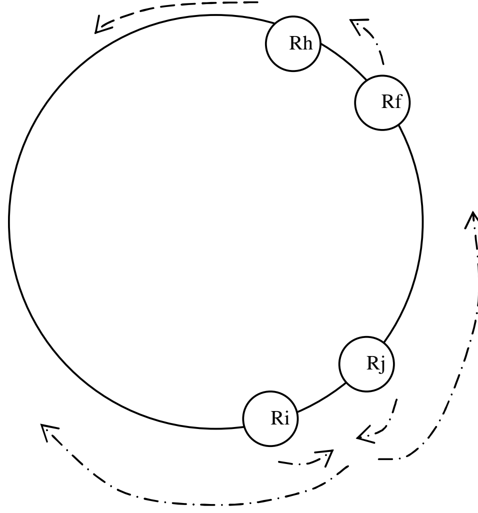


Fig. 3. An execution example of Algorithm 4. The indexes used are the same as in the initialization phase.

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 $t \leftarrow s \oplus 1$ // next robot's position, cyclically
3: **while** current position $\neq$ one end of your *sec* **do**
4: Move towards the end of your *sec* along STC, according your *direction*1 argument
5: **if** your *direction*2 $\neq$ *null* **then**
6: your *direction*1 $\leftarrow$ your *direction*2
7: your *direction*2 $\leftarrow$ *null*
8: Go to 3 // backtrack
9: **else**
10: Announce completion of your *sec*
11: **while** R_t is alive and there is i such that sec_i incomplete **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: $t \leftarrow t \oplus 1$
16: Go to 3 // take over role of failing robot
17: Stop

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., $\frac{n}{k}-1$. 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 7 (Backtracking Worst Case) *The worst-case running time for Algorithm 4 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.

Case 1 ($k = 2$). In the worst case, one of the robots has a section x that is less than or equal to half the path. If x is longer than a third (1/3) of the entire path, the other robot covers a section less than 2/3 of the path, and we are done. If x is equal to a 1/3 of the path, then the other robot covers 2/3 of the path, i.e., $\lceil 2n/3 - 1 \rceil$, and we are done. Otherwise, x is shorter than a 1/3 of the path, i.e., $x = n/3 - y, y > 0$. The robot that covers x backtracks over it. In this time the other robot passes twice that length, i.e., $2(n/3 - y) = 2n/3 - 2y$. At this point, the portion of the path remaining uncovered is $n - (n/3 - y) - (2n/3 - 2y) = 3y$. The two robots cover it together so each of them covers half of it. Hence, the total time taken by each is $2n/3 - 2y + 1.5y = 2n/3 - y/2$. If y is even, then, this is at most $2n/3 - 1$. If y is odd, then one robot covers $\lfloor y/2 \rfloor$ and the other $\lfloor y/2 \rfloor + 1$; i.e., the worst time in this case is $2n/3 - \lfloor y/2 \rfloor - 1 = 2n/3 - 1$.

Case 2 ($k > 2$). 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:

- $|[S_i,S_j)| < |[S_j,S_f)|$. $|[S_i,S_j)| + |[S_j,S_f)| <$ 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 R_j passes half of $[S_i,S_j)$ and backtracks, R_h passes this distance $([S_i,S_j))$ 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. The proof is analogous to that of 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 $[S_f,S_{f\oplus 1})$, so its total path is $|[S_j,S_f)| + |[S_f,S_{f\oplus 1})|$. 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 $\Rightarrow |[S_i,S_j)| + |[S_j,S_f)| + |[S_f,S_{f\oplus 1})| \leq$ 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. $\square$

4.2 Optimal Backtracking MSTC

Algorithm 4 allows backtracking for only two robots and only in the case where one robot has to cover more than half of the entire working area. It does not generate an optimal allocation of robots to assigned sections and directions,

and thus, while it guarantees a better worst-case coverage time than the non-backtracking algorithm of the previous section, its average performance can be improved.

The optimal backtracking MSTC initialization algorithm below (Algorithm 5) allows all robots to backtrack over any number of steps, in order to achieve the best time for the given initial configuration, *and given an STC path*[1]. Algorithm 5 is intended as a drop-in replacement for Algorithm 3, initializing tasks for each robot. While intuitively it may seem that the run-time complexity will grow combinatorially, with the number of possible allocations, it turns out that given the discretization of the problem, a polynomial-time solution exists.

The algorithm is described below. It assigns a *solution* to each robot, where this solution is a tuple $\langle R, L_1, L_2, D_1, D_2 \rangle$. R is the index of the robot in question, $0 \leq R \leq k-1$. L_1 is the length of the section to take before switching directions. L_2 is the length of the section to take after switching directions. It is measured from the robot's initial position after it has backtrack to it. If no switching is needed, its value will be zero. D_1 is the first direction to take; D_2 is the direction to take after traveling length L_1 along the STC, in direction D_1. D_1, D_2 can therefore be *cw* (clockwise), *ccw* (counterclockwise), or *null* (not switching direction).

The key to the algorithm resides in the discretization of the problem. Because we divide the working area to cells and sub-cells, each robot can move a finite number of steps. The algorithm calls algorithm Optimal (Algorithm 6) to check all options for an optimal assignment of paths for one of the robots (Lines 1–

[1] The problem of determining an optimal spanning-tree path is addressed in [1].

Algorithm 5 Optimal initialization phase(STC path P, ordered positions $S_0, \dots, S_{k-1}$)

```
 1: ⟨i, l1, l2, d1, d2⟩ ← Optimal(P, S0, ..., S(k−1))
 2: assign ⟨i, l1, l2, d1, d2⟩ to robot i
 3: time ← l1 · 2 + l2
 4: if d1 = ccw then
 5:     area ← |[Si, S(i⊕1)]| − l1
 6: else
 7:     area ← |[Si, S(i⊕1)]| − l2
 8: for r ← [i ⊕ 1, i ⊕ 2, ..., i ⊖ 1] do
 9:     if area · 2 ≥ time then  // no time for backtracking
10:         assign ⟨r, area, 0, cw, null⟩ to robot r
11:         area ← |[Sr, S(r⊕1)]|
12:     else
13:         if time − 2 · area ≥ (time−area)/2 then  // backtrack the cw direction
14:             assign ⟨r, area, min(time − 2 · area, |[Sr, S(r⊕1)]|), cw, ccw⟩ to robot r
15:             area ← max(|[Sr, S(r⊕1)]| − (time − 2 · area), |[Sr, S(r⊕1)]|)
16:         else  // backtrack the ccw direction
17:             assign ⟨r, min((time−area)/2, |[Sr, S(r⊕1)]|), area, ccw, cw⟩ to robot r
18:             area ← max(|[Sr, S(r⊕1)]| − (time−area)/2, |[Sr, S(r⊕1)]|)
19: if d2 = cw AND area < l2 then  // fix the last assignment
20:     assign ⟨i, l1, area, d1, d2⟩ to robot i
21: else if d1 = cw AND area < l1 then
22:     assign ⟨i, area, l2, d1, d2⟩ to robot i
```

2). Given an optimal assignment for one robot the algorithm calculate the other robots assignment of paths. First, robot i's coverage time is calculated (Line 3). We then check what is the remaining distance for the next robot, r, to cover (Lines 4–7). Given this distance and the time frame, there are 3 possible options: r can only cover the remaining distance (Lines 9–11), it can cover the remaining distance and backtrack to also cover part of the section between it and robot $r \oplus 1$ (Lines 12–15) or it can cover part of the section between it and $r \oplus 1$ and backtrack to cover the remaining distance between it and i (Lines 16–18). For each of the above options we calculate robot r assignment and the remaining area between it to the next robot, $r \oplus 1$. This assignment is done cyclically for all the robots (Line 8).

For each robot (line 2), Algorithm Optimal (Algorithm 6) checks all the pos-

sible steps that the robot can move in a counterclockwise direction along the spanning tree path, until it reaches the robot next to it (line 3). For each possible step, we check whether the robot should backtrack and move in the opposite direction. We search for a length to move in the opposite direction using a binary search (Lines 4–11). The total movement duration is the value of that solution. For each solution we first check that it is valid, meaning that within the solution duration time all the robots can complete to cover the rest of the area (the procedure Check in lines 4,6 and in the function Search). We then store this solution if it is the best so far (the procedure Solution in lines 7 and 11). We similarly test the other side, i.e. check all possible steps in the clockwise direction along the spanning tree path, until it reaches the robot next to it (lines 12–20).

Algorithm 6 Optimal(STC path P, ordered positions $S_0, \ldots, S_{k-1}$)

```
1: t ← ∅
2: for i ← 0 to k − 1 do
3:    for l ← 0 to |[S_i, S_{i⊕1}]| do
4:       if not Check(i, l, |[S_i, S_{i⊖1}]|) then
5:          continue to next l
6:       else if Check(i, l, 0) then
7:          t ← Solution(i, l, 0, t)
8:          break inner loop
9:       else
10:         r ← Search(0, |[S_i, S_{i⊖1}]|, i, l, 'right search')
11:         t ← Solution(i, l, r, t)
12:   for r ← 0 to |[S_i, S_{i⊖1}]| do
13:      if not Check(i, |[S_i, S_{i⊕1}]|, r) then
14:         continue to next r
15:      else if Check(i, 0, r) then
16:         t ← Solution(i, 0, r, t)
17:         break inner loop
18:      else
19:         l ← Search(0, |[S_i, S_{i⊕1}]|, i, r, 'left search')
20:         t ← Solution(i, l, r, t)
21: return t
```

The *Check* procedure works as follows. We get a configuration of robot, i, and its movements, and have to calculate if the other robots can complete the

coverage in the same time it takes to robot i to complete its sections. The idea is that when fixing the movement of one robot to determine overall coverage time, all other robots have only one opportunity for movement within the same time frame, so the check for the validity of the solution is linear in the number of robots. First, robot i's coverage time is calculated. To minimize the total coverage time, if robot i has to backtrack, it will do so in the smaller section (line 1). We then check what is the distance that the next robot can cover in the same time. If there is a remaining area between the robots which both did not cover it is not a valid solution (line 4). If not, this robot has a remaining area between it and the robot next to it (lines 7,10) that has to be covered in the same time frame, so we repeat the check between them. The check is done cyclically for all the robots (line 3), and if the total area can be covered within the time frame which was determined by the given configuration, the solution is valid.

Algorithm 7 Check(robot i, ccw movement amount $left$, cw movement amount $right$)

```
1: time ← min(left, right) · 2 + max(left, right)
2: area ← |[S_i, S_{i⊕1}]| − left
3: for r ← [i ⊕ 1, i ⊕ 2, . . . , i ⊖ 1] do
4:     if area > time then
5:         return false
6:     else if area · 2 ≥ time then
7:         area ← |[S_r, S_{r⊕1}]|
8:     else
9:         best ← max(time − 2 · area, (time−area)/2)
10:        area ← max(|[S_r, S_{r⊕1}]| − best, 0)
11: if area > right then
12:     return false
13: else
14:     return true
```

The *Solution* procedure (Algorithm 8) creates a new solution tuple, if it is better than the existing best solution t. First, in lines 1–3, the new solution length is compared to the existing solution length. If the existing solution is

better or equal to the new solution, the existing solution is returned (line 4). Otherwise, the new solution tuple is calculated. If the amount of CCW movement $left$ is zero, then the only movement is in the CW direction. The length of that movement L_1 is equal to the amount of movement in the CW direction, $right$, and the first direction to take is in the CW (clockwise) direction. This is done in lines 5–8. A similar check is done for the opposite direction (lines 9–12). However, if the solution calls for moving bi-directionally, than the algorithm first checks which direction involves less movement, CCW (lines 13–17) or CW (18–22). In either case, the direction involving less movement is taken first, since it will be backtracked-over in the counter direction.

Algorithm 8 Solution(robot i, ccw movement $left$, cw movement $right$, current best solution t)

1: $VAL \leftarrow min(left, right) \cdot 2 + max(left, right)$
2: $t_{VAL} \leftarrow min(t_{left}, t_{right}) \cdot 2 + max(t_{left}, t_{right})$
3: **if** $t_{VAL} \leq VAL$ **then**
4: return t
5: **if** $left = 0$ **then**
6: $L_1 \leftarrow right$
7: $D_1 \leftarrow clockwise$
8: $D_2 \leftarrow null$
9: **else if** $right = 0$ **then**
10: $L_1 \leftarrow left$
11: $D_1 \leftarrow counterclockwise$
12: $D_2 \leftarrow null$
13: **else if** $left \leq right$ **then**
14: $L_1 \leftarrow left$
15: $L_2 \leftarrow right$
16: $D_1 \leftarrow counterclockwise$
17: $D_2 \leftarrow clockwise$
18: **else**
19: $L_1 \leftarrow right$
20: $L_2 \leftarrow left$
21: $D_1 \leftarrow clockwise$
22: $D_2 \leftarrow counterclockwise$
23: return $\langle i, L_1, L_2, D_1, D_2 \rangle$

The *Search* procedure (Algorithm 9) performs a binary search over the length of a section between one robot to another. It gets a fixed movement length in

one direction for a specific robot, and searches for the shortest length that this robot can move in the other direction, while still enabling the other robots to complete the coverage of the remaining area in the same time it takes to this robot to complete its two direction movement. This check is done with *Check* procedure (lines 6, 11), recursively, until the length is found (lines 1–2). The direction of search is determined by the 'type' argument: a *right search* means that the 'movement' argument is a fixed counterclockwise movement along the spanning tree, with a length of *movement*, so the search is done for the length of the clockwise movement. A *left search* means the opposite. Before the call to this procedure we examine that there is a valid solution with this robot with the maximal possible movement (lines 4–5 in Algorithm 6) to guarantee that the procedure will not be stuck in an endless loop.

Algorithm 9 Search(low border *low*, high border *high*, robot index i, one side movement *movement*, type of search *type*)

```
1: if low = high then
2:     return low
3: else
4:     half ← ⌊(low+high)/2⌋
5:     if type = 'right search' then
6:         if Check(i, movement, half) then
7:             Search(low, (low+high)/2, i, movement, 'right search')
8:         else
9:             Search(half + 1, high, i, movement, 'right search')
10:    else
11:        if Check(i, half, movement) then
12:            Search(low, (low+high)/2, i, movement, 'left search')
13:        else
14:            Search(half + 1, high, i, movement, 'left search')
```

Algorithm 6 generates the optimal allocation of robots to sections (and their backtracking, if necessary), such that coverage is achieved at the best possible time. This is proven in Theorem 8 below. The optimality is according to the MSTC movement rules which we introduced before: All the robots move along the same spanning tree without crossing it, and every robot backtracks only

on its own steps. The only case where a robot has to cover another robot's cell is where the latter failed and its entire allocated section has to be covered by another robot.

Theorem 8 (Optimal MSTC Optimality) *Algorithm 6 generates an optimal allocation of sections for a given STC path, such that the overall coverage time is minimal.*

PROOF. The value of an solution for a given initial configuration is its overall coverage time. If this is optimal, then for each robot, its individual coverage time is less than or equal to this value; and there exists at least one robot whose coverage time is exactly that (otherwise this is not an optimal solution). By choosing the best valid solution of each robot and comparing it to the other robots' best solutions, we guarantee finding the optimal solution. □

The run-time of the allocation itself is polynomial in n and k. This is shown in Theorem 9 below.

Theorem 9 (Optimal Backtracking Run-Time) *Algorithm 6 runs in time $\mathcal{O}(nk^2 \log n)$.*

PROOF. The main loop is executed k times. In each phase there are 2 loops, both executed at most $O(n)$ times because this is the maximum number of possible steps. In each loop the function Check is executed twice and then the function Search and Solution (in the worst case). In the function Check there is only one loop which runs $k-1$ times thus its running time complexity is $O(k)$. The function Search runs a binary search on one section of the spanning tree

path, and uses Check function in each phase so its running time complexity is $O(k \log n)$. The function Solution uses only a constant number of check so its running time complexity is $O(1)$. So, the overall running time complexity is $\mathcal{O}(nk^2 \log n)$. $\square$

Typically the number of robots is much smaller than the number of cells in the area to be covered, i.e., $k << n$. Thus in applications, we expect the run-time to be mostly affected by the $n \log n$ factor.

The actual running time can be further improved if the algorithm is modified to skip checking values which are bigger than the largest initial section. However, while this is a useful implementation note, it does not affect theoretical run-time complexity.

4.3 Heterogeneous Robots

So far we assumed homogeneous robots, with equal speeds and fuel capacities. Thus, the coverage time for a given section, between neighboring robots, was considered equal regardless of which robot was chosen to traverse it. We now briefly describe how heterogeneous robots, in speed and/or fuel capacity, can be taken into account.

The optimal backtracking algorithm (Algorithm 6) works can be easily extended to address robots with heterogeneous speeds. The extension needed involves modifying the way coverage time is calculated in the Check procedure (Algorithm 7) and in the Solution procedure (Algorithm 8). Instead of calculating coverage time based on the length of the section, it should be cal-

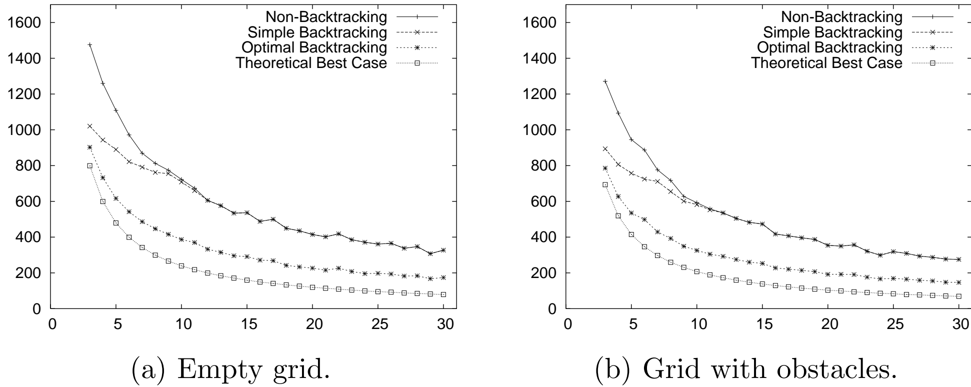
culated based on the length given the maximum speed of the robot in question (lines 7,9-10 in Algorithm 7). With this modification, Algorithm 6 is guaranteed to return a solution that is optimal in coverage time, even taking into account heterogeneous speed limits.

We will also want to consider the case where the robots are equipped with different amount of fuel or different battery capacity. The simple algorithms (Algorithms 2, 4) do not address this case, and may return a planned path that cannot be executed by the robots, given their fuel capacity.

However, the optimal backtracking algorithm provides a solution in this case as well. Algorithm 7 requires a modification in the calculation of the distance that a robot can cover within a given time frame, such that the calculation also takes into account the fuel available. With this change, the optimal backtracking algorithm is guaranteed to produce a solution that is feasible given the robots' fuel or battery constraints. However, this solution still minimizes coverage time, rather than fuel consumption.

5 Experiments

We empirically evaluated the performance of the different algorithms presented in this paper. In a first set of experiments, 3 to 30 robots were assigned for covering a grid of size 30×20 cells, i.e., 2400 D-size sub-cells. In each trial of the experiments, the number of robots was fixed, and their initial positions were randomly generated. Then the different coverage algorithms were run to calculate the coverage time. Each such trial was repeated 100 times. We repeated these experiments for both an empty grid, as well as grid with 80 $4D$



(a) Empty grid. (b) Grid with obstacles.

Fig. 4. Results of experiments with different MSTC coverage algorithms. Each data point is the average of 100 trials. The results in (a) were tested for significance using a paired two-tailed t-test, with $p = 2.5 \times 10^{-17}$. The results in (b) had $p = 8.8 \times 10^{-18}$ in the same test.

obstacle cells, whose position was randomly generated.

Figure 4-a shows the results of these experiments, in the empty grid case. In the figure, the X-axis shows the number of robots, while the Y-axis shows the running time. The figure shows several curves. The worst-case curves were calculated analytically, and show the worst-case coverage-times for the backtracking and non-backtracking algorithms. The best-case curve was also calculated analytically, and is shown so as to provide a benchmark against which to interpret the actual algorithms running times. Figure 4-b shows similar results, but for the grid with obstacles.

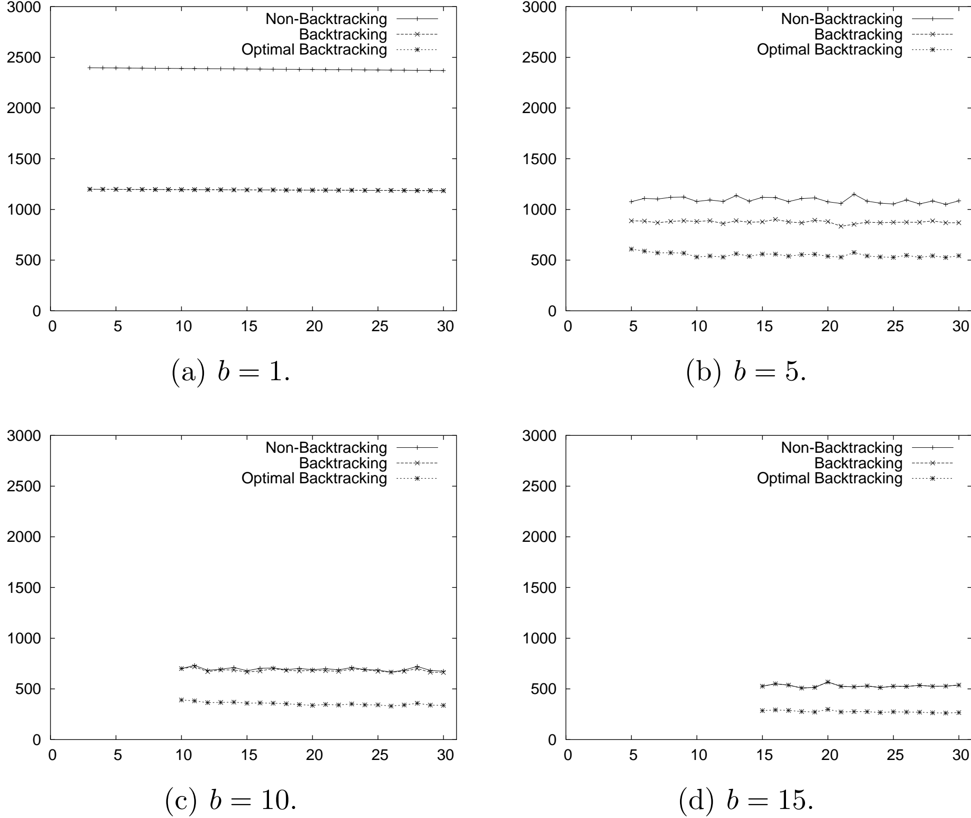
The figure shows that the simple non-backtracking and backtracking algorithms have a difference in performance for small teams, but converge and show the same run-time for larger teams. However, the optimal backtracking algorithm is clearly superior to the two techniques (note the statistical significance test results in the caption). This shows that the run-time can be significantly improved by carefully considering how the initial positions of the robots affect their planned coverage paths.

We thus wanted to explore further the affect of the initial positions of robots on their performance. In the experiments above, the initial positions were randomly generated, and thus in the limit, their average position would have been a distance of $\frac{n}{k}$ from each other, i.e., the best case. However, real-world settings typically do not have the flexibility of landing robots in their perfect initial positions.

To better simulate real-world conditions, we ran a second set of experiments, where 3–30 robots were assigned to bases, and the number of bases (and their positions) were controlled. For a team of k robots, we allowed for b bases, where $1 \leq b \leq k$. We then split the k robots into the b bases, and randomly selected the position of each base. When $b = 1$, it is the worst case for the non-backtracking case (or close to it), where all robots start from the same position. When $b = k$, it is the case of the experiments above.

Figure 5 shows a subset of the results of these experiments. In all sub-figures, the Y axis shows the total number of robots in the bases, and the Y axis shows coverage time. In Figure 5-a, all robots leave from a single base. The two backtracking algorithms converge to a value much below that of the non-backtracking algorithm, whose faced with its worst case (approximately). In Figure 5-b, the performance of the three robots is clearly differentiated, yet in Figures 5-c,d the simple backtracking and non-backtracking algorithms converge (as we saw in the first set of results, above). In all figures, however, the optimal algorithm significantly outperforms its competitors.

The experiments described above evaluate the performance of the different MSTC algorithms, but do not contrast them with closely-related offline coverage algorithms, such as MRFC [27], or robust ant-robotics approaches [23–25].



(a) $b = 1$. (b) $b = 5$. (c) $b = 10$. (d) $b = 15$.

Fig. 5. Coverage run-time when k robots begin coverage from b bases. Each base holds $\frac{k}{b}$ robots. Each data point is an average of 100 trials.

.

Such empirical comparison is inherently incomplete, as the underlying assumptions and guarantees are different. For instance, MRFC [27] has been shown to outperform the simple backtracking and non-backtracking algorithms presented above, in terms of coverage time. The authors observe empirically that robots also tend to come back to the originating point in MRFC, which they do not necessarily do in MSTC algorithms. However, MRFC achieves this at a cost of assuming two or more robots can occupy the same physical cell at the same time, and without guarantee of robustness. The ant-based coverage algorithms presented in [23–25] can guarantee complete, robust, coverage. But they do so at a cost of performance. We believe that a general framework, synthesizing the different approaches and assumptions, is necessary in order to systematically compare the different approaches to coverage.

6 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. We have also introduced a novel polynomial-time optimal backtracking coverage algorithm, capable of handling heterogeneous robots, given an initial configuration and STC path. We have shown in systematic experiments that its performance is a significant improvement over the simpler backtracking and non-backtracking algorithms.

These 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. For instance, vacuum cleaning applications may want to reduce time, yet painting applications may need to avoid having the robot re-visit its steps. This result is not intuitive; reduction in redundancy may cause an *increase* in coverage time, and thus reduce performance. The deployment of multiple robots must take this into account, and balance redundancy and efficiency as required.

Much remains for future work. In particular, we plan to tackle the question of

the efficiency of the spanning-tree underlying the planned paths, and examine the MSTC algorithms in online settings. See [1, 12] for initial steps in these directions.

Acknowledgments

This paper is based in part on a conference paper by the authors [11]. We thank Moshe Lewenstein, Noa Agmon, Sven Koenig, Sonal Jain, and Xiaoming Zheng for useful discussions. K. Ushi and Shira deserve special thanks. This work was partially supported by Israel Science Foundation.

References

[1] N. Agmon, N. Hazon, and G. A. Kaminka. Constructing spanning trees for efficient multi-robot coverage. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06)*, 2006.

[2] M. Batalin and G. Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Proc. of the 6th Internat. Symposium on Distributed Autonomous Robotic Systems*, page 373382, 2002.

[3] D. W. Casbeer, D. B. Kingston, R. W. Beard, and T. W. McLain. Cooperative forest fire surveillance using a team of small unmanned air vehicles. *International Journal of Systems Science*, 37(6):351–360, May 2006. Special Issue on Cooperative Control Approaches for Multiple Autonomous Vehicles.

[4] H. Choset. Coverage for robotics—a survey of recent results. *Annals of Math and Artificial Intelligence*, 31:113–126, 2001.

[5] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA CIRFFSS*, 1994.

[6] N. R. Correl. Collaborative exploration and coverage. Master's thesis, Collective Robotics Group, California Institute of Technology and Automatic Control Laboratory, Swiss Federal Institute of Technology, Zurich, 2003.

[7] N. Correll, K. Easton, A. Martinoli, and J. Burdick. Distributed exploration and coverage. http://www.cnse.caltech.edu/Research/reports/correll-full.html.

[8] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Math and Artificial Intelligence*, 31:77–98, 2001.

[9] D. W. Gage. Command control for many-robot systems. In *The nineteenth annual AUVS Technical Symposium (AUVS-92)*, 1992.

[10] A. Girard, A. Howell, and J. Hedrick. Border patrol and surveillance missions using multiple unmanned air vehicles. In *IEEE Conference on Decision and Control*, volume 1, pages 620–625, 2004.

[11] N. Hazon and G. A. Kaminka. Redundancy, efficiency, and robustness in multi-robot coverage. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-05)*, 2005.

[12] N. Hazon, F. Mieli, and G. A. Kaminka. Towards robust on-line multi-robot coverage. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06)*, 2006.

[13] C. S. Kong, A. P. New, and I. Rekleitis. Distributed coverage with multi-robot system. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 2423 – 2429, Orlando, Florida, May 2006.

[14] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-96)*, 1996.

[15] 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.

[16] E. Osherovich, V. Yanovski, W. I. A, and A. M. Bruckstein. Robust and efficient covering of unknown continuous domains with simple, ant-like a(ge)nts. Technical report, Technion, Israel, 2007.

[17] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 2, pages 1340–1345, Nagoya, Japan, August 1997. Morgan Kaufmann Publishers, Inc.

[18] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team based coverage. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-04)*, pages 3462–3468, New Orleasn, LA, April 2004.

[19] 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.

[20] L. D. Stone. *Theory of Optimal Search.* Military Applications Society, 2nd edition, 2004.

[21] J. Svennebring and S. Koenig. Building terrain-covering ant robots: A feasibility study. *Autonomous Robots*, 16(3):313–332, 2004.

[22] R. E. Tarjan. *Data structures and network algorithms.* Society for Industrial and Applied Mathematics, 1983.

[23] I. Wagner, M. Lindenbaum, and A. Bruckstein. Efficiently searching a graph by a smell-oriented vertex process. *Annals of Math and Artificial Intelligence*,

24:211–223, 1998.

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

[25] I. Wagner, M. Lindenbaum, and A. Bruckstein. Mac vs. pc determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of Robotics Research*, 19(1):12–31, 2000.

[26] A. R. Washburn. *Search and Detection*. Military Applications Society, 4th edition, 2002.

[27] X. Zheng, S. Jain, S. Koenig, and D. Kempe. Multi-robot forest coverage. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS-05)*, 2005.