Ann Math Artif Intell (2008) 52:143–168
DOI 10.1007/s10472-009-9121-1

The giving tree: constructing trees for efficient offline and online multi-robot coverage

Noa Agmon · Noam Hazon · Gal A. Kaminka · The MAVERICK Group

Published online: 18 March 2009

Abstract This paper discusses the problem of building efficient coverage paths for a team of robots. An efficient multi-robot coverage algorithm should result in a coverage path for every robot, such that the union of all paths generates a full coverage of the terrain and the total coverage time is minimized. A method underlying several coverage algorithms, suggests the use of spanning trees as base for creating coverage paths. However, overall performance of the coverage is heavily dependent on the given spanning tree. This paper focuses on the challenge of constructing a coverage spanning tree for both online and offline coverage that minimizes the time to complete coverage. Our general approach involves building a spanning tree by growing sub-trees from the initial location of the robots. This paper first describes a polynomial time tree-construction algorithm for offline coverage. The use of this algorithm is shown by extensive simulations to significantly improve the coverage time of the terrain even when used as a basis for a simple, inefficient, coverage algorithm. Second, this paper provides an algorithm for online coverage of a finite terrain based on spanning-trees, that is complete and guarantees linear time coverage with no redundancy in the coverage. In addition, the solutions proposed by this paper guarantee robustness to failing robots: the offline trees are used as base for robust multi-robot coverage algorithms, and the online algorithm is proven to be robust.

Keywords Efficient multi-robot coverage · Coverage algorithm

This work was funded in part by Israel's Ministry of Science and Technology.

N. Agmon (✉) · N. Hazon · G. A. Kaminka · The MAVERICK Group
Department of Computer Science, Bar Ilan University, Ramat Gan, Israel
e-mail: segaln@cs.biu.ac.il

N. Hazon
e-mail: hazonn@cs.biu.ac.il

G. A. Kaminka
e-mail: galk@cs.biu.ac.il

Mathematics Subject Classification (2000) 68T40 Robotics

"And the tree was happy..." (*Shel Silverstein*, The Giving Tree)

1 Introduction

The general problem of covering an area by single or multi robot systems is a fundamental problem in robotics. It has applications in various domains, from humanitarian missions such as search and rescue and de-mining, to agriculture applications such as seeding or harvesting, to, recently, household cleaning. The problem was extensively investigated in both single-robot domains (e.g. [6, 15, 16]) and multi-robot systems (e.g. [13, 20, 21, 24]).

This paper discusses the problem of building efficient coverage paths for a team of robots. In this problem, a team of robots, each equipped with some tool, for example a sensor, are required to jointly sweep the entire given terrain while minimizing the total coverage time. In our work, we assume that the area is divided into cells, and the robots travel through all cells of the terrain. Following the taxonomy presented in [5], the division of the area is an approximate cellular decomposition, and we handle both online and offline coverage. In offline coverage, the map of the area is given in advance, therefore the coverage paths of the robots can be determined prior to the execution of the coverage algorithm. In online coverage, the coverage has to be completed without the use of a map or any a-priori knowledge of the area, and the coverage paths of the robots are constructed during the execution.

Previous work has often pointed out that one advantage of using multiple robots for coverage is the potential for more efficient coverage [5]. Another potential advantage of using multiple robots is that they may offer greater *robustness*: Even if one robot fails catastrophically, others may take over its coverage subtask. In other words, as long as there exists one non-faulty robot, the coverage mission will be completed successfully. Unfortunately, this important capability has been neglected in most existing work on on-line algorithms.

Several methods are found in the literature for coverage by single and multi-robot systems. One basic method that has received considerable attention is the method presented by Gabriely and Rimon [10], where the authors describe a polynomial time *Spanning Tree Coverage* algorithm, better known as the STC algorithm. In this method, Gabriely and Rimon offer a method for finding a hamiltonian cycle covering a terrain that satisfies some assumptions. In particular, it is assumed that the robot is equipped with a square shaped tool of size D, hence the area was divided into N cells of size D placed on a grid. The grid was then made coarse such that each new cell is of size $2D\ X\ 2D$, and a spanning tree was built according to this new grid. After such a tree was built, the robot follows the tree around, creating a hamiltonian cycle visiting all cells of the original grid. The idea was first broadened for a multi-robot system in [13], by the family of MSTC algorithms. A different variation on this idea was introduced in [24].

When building the tree in a single robot system, the influence of the structure of the tree is theoretically irrelevant for the coverage time. Clearly, one might want to construct spanning trees with special characterizations, for example minimizing the number of turns of choosing some preferred directionality. Yet, the coverage time

guaranteed by the STC algorithms is linear in the size of the grid, since each cell except for the boundary cells is covered once, hence the total coverage time is N.

On the other hand, in multi-robot systems, the structure of the tree can have crucial consequences on the coverage time of the terrain. The choice of the spanning tree can change the robots' initial positions with respect to each other from being concentrated, i.e., placed as a bundle, to being scattered along the spanning tree path—all without actually changing the physical initial position of the robots. The structure of the tree itself can therefore substantially decrease the coverage time obtained by algorithms based upon it. Hence we concentrate on building appropriate coverage spanning trees. The general method we follow when building such trees in both scenarios—online and offline coverage, is to gradually grow subtrees from the initial position of the robots.

Hence, the first part of this paper specifically deals with constructing spanning trees for offline coverage that reduces the total coverage time of algorithms using these spanning trees as base for coverage. The coverage time of a terrain is determined by the robot traveling through the longest period of time. In a system with homogenous robots, this time corresponds to the longest distance traversed by a single robot. We try to minimize this distance by creating trees where the robots are placed as uniformly as possible around it. Therefore, when constructing the trees we try to minimize the maximal distance between every two consecutive robots along the spanning tree path. If such tree is obtained, we show that all versions of the MSTC algorithm ran on these trees achieves substantially better coverage time compared to their coverage time on other randomly generated trees. Note that these trees, along with decreasing the coverage time of the algorithms which use them as base for coverage, also enjoy the benefits of the algorithms themselves. Specifically, if used as base for the family of MSTC algorithms, it promises robustness. The algorithm we propose has a polynomial time complexity in the number of cells to be covered. This results in the surprising conclusion that as we add obstacles to the terrain, the complexity of the tree construction algorithm reduces, since the number of covered cells diminishes.

The second part of the paper deals with online coverage. We present a *guaranteed robust* multi-robot on-line coverage algorithm. The algorithm is based on the use of spanning tree coverage paths. It runs in a distributed fashion, using communications to alert robots to the positions of their peers. Each robot works within a dynamically-growing portion of the work-area, constructing a local spanning-tree covering this portion, as it moves. It maintains knowledge of where this spanning-tree can connect with those of others, and selects connections that will allow it to take over the local spanning trees of others, should they fail. We also address the challenge of using the robust on-line multi-robot coverage algorithm with physical vacuum cleaning robots. We present techniques useful in approximating the assumptions required by STC algorithms (e.g., known positions, within an agreed-upon coordinate system). We show the effectiveness of our implemented algorithm in extensive experiments.

2 Background

The challenge of covering a terrain by a team of mobile robots has received considerable attention in the literature. The growing interest in this area is first and foremost due to the fact that the coverage task is implementable in various domains.

Moreover, the concentration in multi-robot systems comes from the 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. Many approaches can be found in the literature for multi-robot coverage.

Choset [5] provides a survey of coverage algorithms, which distinguishes between *offline* algorithms, in which a map of the work-area is given to the robots in advance, 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 decomposed to a set of regions, whose union fills the entire area exactly. Following Choset's terminology, in this paper we focus on both online and offline coverage, based on approximate cell decomposition of the area.

We focus on spanning tree based coverage, first proposed by Gabriely and Rimon in [10]. They proposed the basic method of dividing the terrain into $2D \; X \; 2D$ cells, and described the polynomial time spanning tree coverage algorithm (STC) for complete offline and online coverage of the terrain by a single robot. In [11], they suggest two different algorithms for building an on-line tree, but the motivation comes from the desire to create a spanning tree with a specific scanning direction.

The generalization of the single-robot STC algorithm to offline multi-robot systems was first introduced by Hazon and Kaminka in [13]. They presented several offline algorithms for multi-robot coverage of a terrain by the MSTC algorithm, which guarantee robust, time-efficient and complete coverage. They describe two versions of the MSTC algorithm: non-backtracking MSTC, and backtracking MSTC, herein referred to as NB_MSTC and B_MSTC, respectively. In the NB_MSTC algorithm the robots simply move in counterclockwise direction along the spanning tree path until reaching the initial position of the following robot if no faults occur, or take over the coverage path of the consecutive robot otherwise. In the B_MSTC the robots can backtrack over parts of their coverage path, i.e., they can go both clockwise and counterclockwise. They have shown that if the robots backtrack, the worst case performs up to twice as faster as in the non-backtracking case, despite the redundancy. Other results by Hazon and Kaminka, described in [14], provide an optimal polynomial time coverage algorithm, herein referred to as Opt_MSTC. The algorithm is similar to the B_MSTC algorithm with modifications that assure the optimal coverage time given the initial locations of the robots and an initial spanning tree. The optimality is guaranteed only for the backtracking method, i.e., if the robots go back and forth *along the given spanning tree*. Hence they promise to make the most (optimal) out of the given tree and initial locations of the robots if the robots do not deviate from the path dictated by the structure of the tree. We focus here on generating good trees for such algorithms.

Work by Zheng et al. [24] proposed an additional offline multi-robot coverage algorithm, where their solution is based on dividing the *given* spanning tree into k subtrees, where there might exist path overlapping between robots. Their algorithm performs better compared to both NB_MSTC and B_MSTC algorithms, however their solution is not robust. In addition, they note that different choices of trees may result is different coverage time, but did not further discuss the issue.

There have been additional investigations of online multi-robot coverage, for example in the world of ant robotics. Wagner at. al. [23] propose a series of theoretical multi-robot ant-based algorithms which use approximate cellular decomposition.

The algorithms involve little or no direct communications, instead using simulated pheromones for communications or traces of robots. Some of these algorithms solve only the discrete coverage problem and some offer complete robust coverage, but not necessarily efficient. Recent work by Osherovich et al. [18] offer a robust coverage algorithm for ants in continuous domains. Svennebring and Koenig [22] offer a feasibility study for ant coverage. They perform experiments with real ant-robots and large-scale simulations. They show robustness, but provide no analytic guarantees for completeness or efficiency.

Acar and Choset [1] presented a robust on-line *single* robot coverage algorithm while their robustness quality is the ability to filter bad sensors readings.

Rekleitis et al. [19] uses two robots in online settings, using a visibility graph-like decomposition (sort of exact cellular decomposition). The algorithm uses 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. [21] extends the Boustrophedon approach [5] 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. Their solution, though, is not robust to failures, i.e., it could stop functioning if one of the key robots fails. In [17], Kong et al. provide an improved algorithm for multi-robot coverage with unbounded communication, where the algorithm is demonstrated to be robust to failures (yet this property is not theoretically proven to be complete).

Butler et al. [4] proposed a sensor-based multi-robot coverage, in a rectilinear environment, which based on the exact cellular decomposition. They do not prove their robustness, and the robots could cover the same area many times.

The recent Brick & Mortar algorithm suggested by Ferranti et al. [8] is an online coverage algorithm that assumes the robots communicate using miniature storage devices that are placed along the entire area, such that one device is placed in each cell. The use of these devices is their solution to the extensive communication assumption, made by other online coverage algorithm, which is partially made also in our work. Their work does not refer to the robustness of the coverage. In addition, their solution might result in redundancy of the coverage.

Other approaches, other than ones based on cellular decomposition of the terrain, can be found in the literature for multi-robot coverage. For example, in [3], Batalin and Sukhatme 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. Yet, similarly to their work, our offline tree construction algorithm uses the "spreading out" principle in building the coverage tree.

3 Constructing trees for offline coverage

In this section we describe our method for improving offline coverage by a team of robots. First, we show that the initial choice of the spanning tree has significant impact on the coverage time. We then describe our scheme for tree construction and some variants of the general method. Finally, we describe results from extensive simulations showing significant improvement in coverage time even when the trees

are used as base for the simplest NB_MSTC algorithm. Note that by combining our tree construction algorithm with the family of MSTC algorithms, we ensure efficient and robust multi robot coverage.

3.1 Motivation for building new spanning trees

In this section we describe the motivation behind our construction scheme of the trees. First, we show that the structure of the spanning tree has crucial role in the coverage time obtained by algorithms that use the tree as base for coverage. We prove that any coverage algorithm, even an optimal one, cannot achieve low coverage time as can be achieved by using a different tree. Second, we show that a spanning tree, which by itself obtains the optimal coverage time, does not necessarily exist, hence the theoretical optimal coverage time might remain unreachable in some cases. Last, we describe our definition of *optimal* spanning trees and explain the rationale behind this definition.

3.1.1 Importance of the spanning tree structure

An optimal time coverage algorithm for a system with k robots will (theoretically) result in total coverage time of $\lceil \frac{N}{k} \rceil$. Even the most basic multi-robot coverage algorithm will result in such a coverage time if the robots are uniformly placed along the spanning tree path, i.e., within distance of at most $\lceil \frac{N}{k} \rceil$ from one another.

We argue that the choice of spanning tree has crucial consequences on the coverage time obtained by algorithms using the spanning tree as base for coverage. This is more evidently seen in algorithms that do not diverge from the spanning tree path, such as the MSTC algorithms. Consider, for example, the Opt_MSTC algorithm. These algorithms create optimal paths along the spanning tree for the k robots, not allowing (nonfaulty) robots to bypass one another during the execution of the coverage algorithm. There, even in the worst initial distribution case in which all robots are bundled in their initial position, the best possible improvement will result in an improvement factor of approximately 2: from $N - k + 1$ to $\frac{N-k}{2} + 1$. On the other hand, the improvement by spreading the robots along the spanning tree can reach nearly a factor of k : from $N - k + 1$ to $\frac{N}{k}$.

An illustration of the importance of the right choice of spanning tree is given in Fig. 1. The figure presents an example for a terrain in which $N = 36$, $k = 3$ and two different trees are suggested as base for coverage. The spanning tree is described by the bold lines, and we use the different kinds of dashed lines to describe the spanning tree path, each dashed line represents the distance between two adjacent robots along the path. In order to clarify the example, the section between each two adjacent robots is given a different background as well. Note that in both grids the robots are initially located in the same positions. The tree in Fig. 1a places the robots uniformly along the tree path, thus a coverage time of $\lceil \frac{N}{k} \rceil$ is easily obtained if the robots simply follow the tree path in a counterclockwise direction. However, in Fig. 1b. the robots are placed arbitrarily along the tree path, thus any multi-robot coverage algorithm, based on the spanning tree, will find it hard to result in such coverage time.

A formal statement regarding the possible improvement in coverage time obtained by algorithms vs. improvement obtained by changing the tree is given by Theorem 1. First, let us introduce the following definition. Note that we distinguish

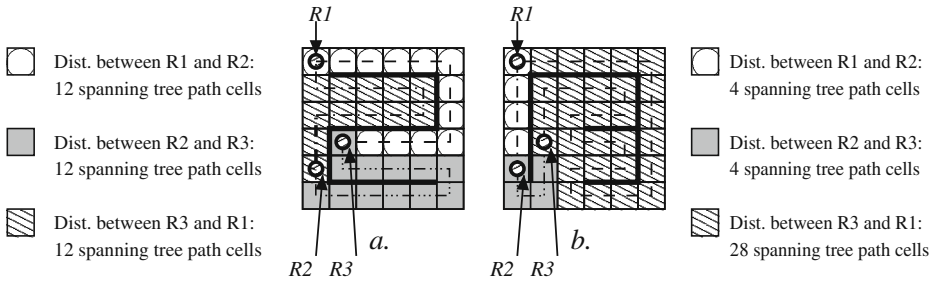


Fig. 1 Illustrating how different trees can influence coverage time (**a**, **b**)

between a *procedure* that is executed on the input to generate the tree, and *algorithm* which is the coverage algorithm executed given the input tree.

Definition Given the initial positions of k robots on a terrain with N cells, let M be the coverage time of the terrain obtained by the basic NB_MSTC algorithm. A procedure $\mathcal{P}$ or an algorithm $\mathcal{A}$ are said to ensure an *improvement factor* t, if the coverage time obtained by NB_MSTC after applying $\mathcal{P}$ on the input, or the coverage time obtained by $\mathcal{A}$ on the same input is $\frac{M}{t}$.

Theorem 1 *Any multi-robot coverage algorithm for homogenous robots based on a spanning tree which does not divert from the spanning tree path will result in a maximal improvement factor of at most* 2.

Proof Denote the distance between the initial location of robot R_i and R_{i+1} on the spanning tree path by D_i (also known as the segment D_i), and let $D_{max} = \max_{1 \le i \le N}\{D_i\}$. Clearly, the coverage time obtained by NB_MSTC is exactly D_{max}. Also, D_{max} determines the coverage time of any coverage algorithm $\mathcal{A}$ that does not divert from the spanning tree path. As the robots are homogenous and cannot bypass one another (assuming they are nonfaulty), an improvement in the coverage algorithm can reduce from D_{max} to $\lceil \frac{D_{max}}{2} \rceil$ if robots on the extremity of D_{max} should simply walk towards one another while covering the terrain. If there is some other segment D_j which requires coverage time of some $t' > \lceil \frac{D_{max}}{2} \rceil$, then the new coverage time is t'. Note that t' can be smaller than the distance D_j if an algorithm allowing backtracking is permitted. In other words, the improvement factor is

$$\frac{D_{max}}{\max\{\lceil \frac{D_{max}}{2} \rceil, t'\}} \le 2 \qquad \square$$

While the change of the coverage algorithm can result in an improvement factor of at most 2, the example described in Fig. 2 leads us to the conjecture that improvement factor due to a change in the tree can reach almost the value of k. As seen in Fig. 2b, the coverage time obtained by NB_MSTC is $N - k = 56 - 3 = 53$, while the coverage time obtained by the same algorithm on a spanning tree constructed in a way that places the robots in an equally scattered way along the tree (Fig. 2a.) is $\lceil \frac{N-k}{k} \rceil = 19$, hence the improvement factor obtained by changing the tree is $\frac{53}{19} \approx 2.8$, which is almost k.

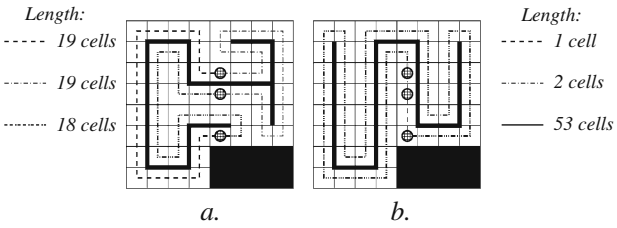


Fig. 2 An example of a case in which the improvement factor is almost k if the tree is appropriately constructed (**a**, **b**)

We have established the fact that the choice of a spanning tree can have far reaching consequences on the coverage time of the terrain, possibly more than the choice of the coverage algorithm. Moreover, a spanning tree that places all robots within distance of at most $\lceil \frac{N}{k} \rceil$ will, by itself, result in the optimal coverage time. Unfortunately, such a tree does not necessarily exist. For example, in Fig. 3, $N = 16$, $k = 2$ and all possible spanning trees are described. The minimal maximal distance between two consecutive robots over all possible spanning trees is 10 cells, where $\lceil \frac{N}{k} \rceil = \lceil \frac{16}{2} \rceil = 8$.

In our tree construction scheme we will try to approximate this optimal dispersion of robots along the spanning tree. We will do that by trying to satisfy the following objective, as much as possible. First, let $\tilde{G}$ be a grid with $N/4$ cells, possibly containing obstacles (the obstacles are not counted as cells). Let G be $\tilde{G}$'s fine grid after dividing each cell into four cells of size D.

Objective Given the initial locations of k robots on cells of G, find a spanning tree of $\tilde{G}$ that minimizes the maximal distance between every two consecutive robots along the spanning tree path.

The idea behind this objective is that it spreads the robots as uniformly as possible along the spanning tree path. The construction of an *optimal* tree, that will achieve exactly the objective, is believed to be $\mathcal{NP}$-hard [24]. Hence our tree construction algorithm can be considered as a heuristic algorithm for the problem of finding the optimal tree for the coverage task.

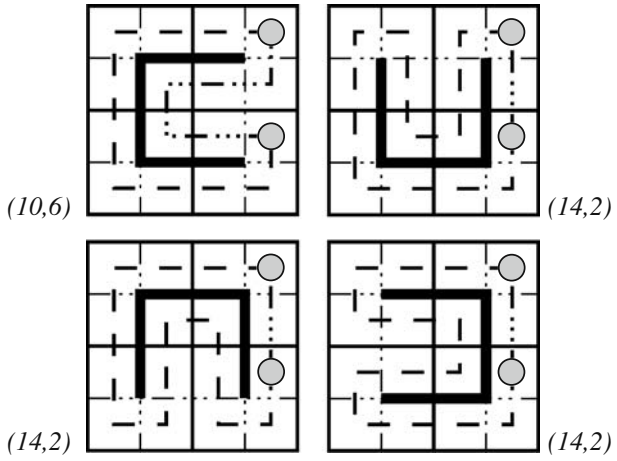


Fig. 3 An example of a case in which there is no spanning tree that has maximal distance of $\lceil \frac{N}{k} \rceil = \lceil \frac{16}{2} \rceil = 8$ between consecutive robots along the spanning tree path. The *numbers in parenthesis* describe the distance between two robots along the spanning tree path

3.2 Tree construction algorithm

In this subsection we describe a spanning tree construction algorithm, Create_Tree. This algorithm creates spanning trees while considering the initial location of all robots in the team and the objective described above, i.e., it tries to minimize the maximal distance between any two adjacent robots on the tree.

The general algorithm, described in Algorithm 1, is composed of two stages. In the first stage, a subtree is created gradually for each robot starting from the initial position of the robot, such that in each cycle either one or two cells are added to each subtree. Denote the subtree originated in R_i by T_{R_i}. The cells are chosen in a way that maximizes the distance from current expansion of all other trees. The algorithm tries to find the longest possible path for the tree. When it fails to continue, it tries to perform Hilling, in which it looks for ways to "stretch" the path as follows. It looks for two joint unoccupied cells adjacent to the path. If it found such cells, then it adds them to the path as demonstrated in Fig. 4. If the algorithm failed to find more hills, then it expands the tree, from both sides of the path, in a BFS (breadth-first-search) manner. It first attempts to add one cell near the origin of the tree (initial position of the robot), then it checks for a possible free adjacent cell of its sons, and so on, until the entire grid is covered by all k disjoint subtrees.

In the second stage of the algorithm, after such k subtrees are generated, it is only left to connect them (second stage). Denote an edge connecting two different trees T_{R_i} and T_{R_j} by $\mathrm{br}(T_{R_i}, T_{R_j})$. As we are given k subtrees to be connected to one tree covering the entire grid, it is required to find $k-1$ bridges. These bridges should be chosen in a way that the resulting tree does not contain cycles or, equivalently, cover the entire grid. For example, if $k=4$ then possible valid choice of bridges are $\{\mathrm{br}(T_{R_1}, T_{R_2}), \mathrm{br}(T_{R_1}, T_{R_3}), \mathrm{br}(T_{R_3}, T_{R_4})\}$, where $\{\mathrm{br}(T_{R_1}, T_{R_2}), \mathrm{br}(T_{R_2}, T_{R_3}), \mathrm{br}(T_{R_1}, T_{R_3})\}$ is invalid, as T_{R_4} remains disconnected. Create_Tree picks randomly a valid choice of $k-1$ bridges, and calculates the maximal distance between two adjacent robots on the tree according to the **fine** grid. It repeats the process k^α times, and reports the best tree it observed, according to the above criterion. The value of α is chosen empirically.

Clearly, the algorithm provides complete coverage of the terrain, as the first stage of constructing subtrees does not end before every cell is occupied by some subtree. The first stage terminates, as in each cycle at least one cell is added to at least

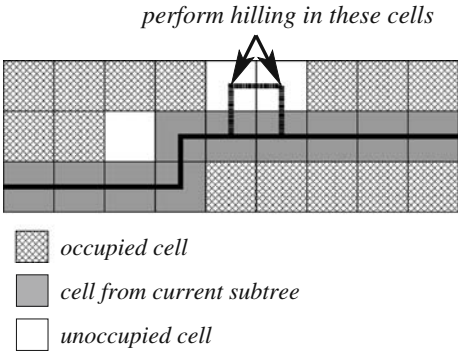


Fig. 4 Illustration of the Hilling procedure

one subtree, hence given a finite terrain the algorithm halts. A formal proof of the completeness of Create_Tree is given in Lemma 2.

Algorithm 1 Procedure Create_Tree

```
Build k subtrees as follows.
for every robot R_i, 1 ≤ i ≤ k do
    for each possible next cell (up, down, right, left) do
        Compute the Manhattan distance from the current location of all other robots.
    if more than one possible next move exists then
        pick the one whose minimal distance to any other robot is maximized.
    if there is no next possible move then
        perform Procedure Hilling from the last main branch.
    if failed to find an unoccupied cell in Hilling then
        Branch-out in a BFS manner from the main branch
Find all possible bridges between the k trees.
for i = 0 to max{k^α, N} do
    At random, find a valid set of bridges B_i between trees such that they create one tree with all N vertices.
    Compute the set S_i of distances between every two consecutive robots on the tree.
    Best_Result is initialized with S_0.
    if the maximal value in S_i is lower than the maximal value in Best_Result then
        Best_Result ← S_i.
Return the tree associated with Best_Result.
```

Lemma 2 *Procedure* Create_Tree *generates a tree that spans the entire graph G.*

Proof Assume, towards contradiction, that there exists one cell C that is not covered by any subtree T_i, $1 \leq i \leq k$. Since the map is finite, then there exists some cell C‘ adjacent to C that is connected to a sub-tree originated in the initial location of some robot R_a. If C was not covered by the algorithm, then T_a has necessarily finished all its phases - initial phase, Hilling and branching out. But if, while branching out, C' was passed through and C was empty, then it would have added C to T_a, leading to a contradiction. If both C and C' are empty, then there exists some $C'' \in T_b$, such that C is adjacent to C''. Similarly, either we get a contradiction, or C'' also is not in T_b. This continues until we get that either all cells are not in any tree, or all cells are (by contradiction). The former case is impossible, as at least the initial location of a robot belongs to its subtree, hence we are done. □

Theorem 3 *The time complexity of* Create_Tree *algorithm is* $\mathcal{O}(N^2 + k^\alpha N)$.

Proof In the stage where k subtrees are created, in the worst case when adding one cell to a subtree the algorithm runs over all current cells in the subtree (during Hilling or while branching out), hence the complexity is at most $\mathcal{O}(N^2)$. In the second stage, where the trees are connected, k^α different choices of trees are examined, each time the entire tree is traversed, thus the complexity of this stage is $\mathcal{O}(k^\alpha N)$. Hence

the entire complexity of the algorithm is $\mathcal{O}(N^2 + k^\alpha N)$. If the distance measure is shortest paths, then calculating all-pairs shortest paths is $\mathcal{O}(N^3)$ [7] □

3.2.1 Using different distance measures

Procedure Create_Tree, creates first k subtrees, and then connects them. The process of constructing the k subtrees is done while spreading each tree *away* from the other trees. The distance measure used to determine how distant the trees are was initially simply the Manhattan distance [2]. In this work, we have used three different distance measures: Manhattan distance, Euclidean distance and Shortest paths (following Floyd's all-pairs shortest paths algorithm [7]). Note that the time complexity of the shortest paths algorithm is $\mathcal{O}(N^3)$, where the other distance measures are calculated in $\mathcal{O}(1)$.

The theoretical advantage of using shortest paths is enormous: As shown in Fig. 5, using the shortest paths measure can decrease the coverage time from N to $N/2$. The tree in Fig. 5a was generated using the shortest paths measure, and the tree in Fig. 5b was generated using the Manhattan distance measure. Initially, there are twice as many cells below the horizontal corridor compared to the number of cells above this corridor. Note that when generating the subtrees using the shortest paths measure (Fig. 5a) there are two possible bridges between the trees—one near the initial positions of the robots and one at the endpoint of the subtree. The latter bridge will be chosen with high probability. In the second tree there is only one possible bridge connecting between the two subtrees. The distance between the two robots along the first spanning tree path (Fig. 5a) is $N/2$. The distance between the two robots along the second spanning tree (Fig. 5b, generated using Manhattan distance) is 8 fine grid cells. Therefore the distance changed from 8 to $N/2$ just by using a the shortest paths distance measure.

Yet, in the average case using this measure did not make a difference—the results of running MSTC algorithms on the trees generated using the three distance

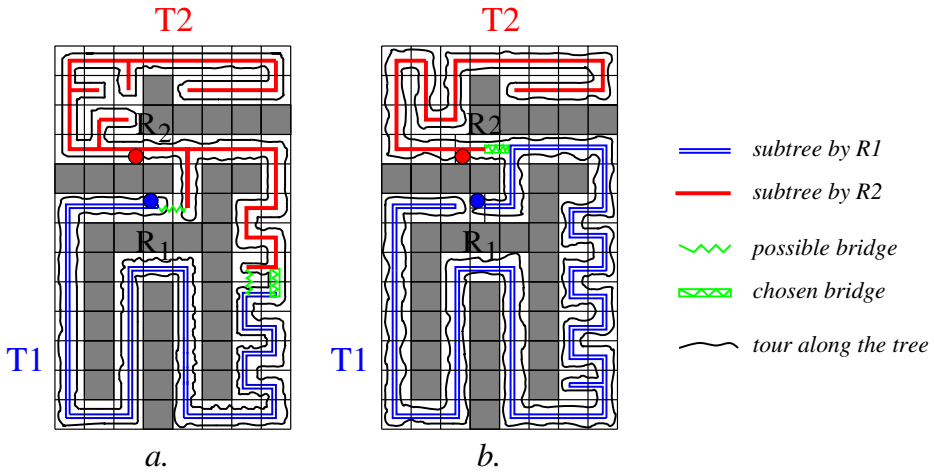


Fig. 5 Illustrating the trees created using different distance measures by Procedure Create_Tree: **a** Manhattan distance and **b** Shortest paths

measures converge. The reason, in our opinion, is that it requires a very specific structure of the terrain in order for the shortest paths measure to make a significant change. One terrain that will gain from using this measure is one with many corridors (see example in Fig. 5). In such a terrain, the difference between the Manhattan distance and the shortest path is significant. Hence we conclude that Create_Tree is adaptable in the sense that the distance measure can be changed in order to fit the terrain.

3.3 Evaluation

We have evaluated the effect of the tree construction algorithm Create_Tree on the coverage time obtained by NB_MSTC, B_MSTC and Opt_MSTC. First, we determine the α used by the algorithm. Then we describe extensive simulations of Create_Tree with our chosen α.

3.3.1 Determining α

When connecting the k subtrees, procedure Create_Tree chooses at random $\max\{k^\alpha, N\}$ times a set of bridges yielding a tree, and chooses the best option between them. We have chosen the value α empirically to be 2. We have seen that if $\alpha = 2$, then the coverage time obtained by the MSTC family of algorithms has decreased substantially. A further improvement was seen in case $\alpha = 3$, but the intensity of the improvement diminished, more evidently with the results of the Opt_MSTC algorithm (see Fig. 6).

Note that the time complexity is substantially higher if $\alpha = 3$ and rises from Nk^2 to Nk^3, i.e., an addition of $Nk^2(k-1)$ operations. This becomes critical for large N and k's.

3.3.2 Experimental results, $\alpha = 2$

The evaluation of Create_Tree on the coverage time obtained by the family of MSTC algorithms was done while taking two other parameters under consideration. First, the number of robots - from 3 to 30 robots. The second parameter is the *density* of obstacles in the terrain, i.e., the ratio between the number of obstacles and the area size.

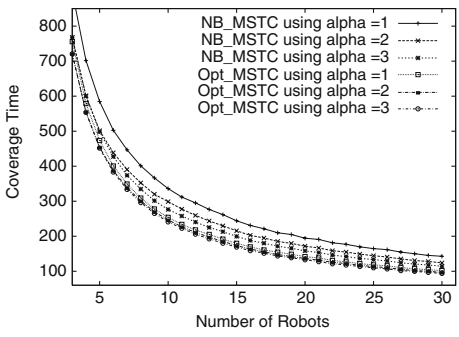


Fig. 6 Comparing $\alpha = 2$ to $\alpha = 3$ for 1 to 30 robots, with 13% of the area contains obstacles (not disconnecting the area)

The coverage time obtained by the above algorithms on the trees constructed by all three variants of Create_Tree was compared against coverage time obtained by the algorithms running on randomly generated spanning trees. The terrain over which the experiment was ran was a $20X30$ coarse grid (600 coarse cells, or 2400 fine cells). We have first performed the experiment on a grid with no obstacles ("clean" grid), then added at random 40 (6.6 %), 80 (13.3 %), 100 (16.7 %) and 160 (26.7 %) obstacles to the coarse grid.

Each trial was run for every number of robots (from 3 to 30) and for every density of obstacles in the terrain. First, we have created 300 input lines by each tree construction method: randomly generated trees and Create_Tree generated trees, where each input line represents a random initial distribution of the robots. These input lines were later given to the NB_MSTC, B_MSTC and Opt_MSTC algorithms and the coverage times obtained by these algorithms were compared.

The average coverage times obtained by the algorithms NB_MSTC and Opt_MSTC for are brought in Fig. 7. The results show clearly that the average coverage time obtained by running algorithms NB_MSTC and Opt_MSTC on trees constructed by algorithm Create_Tree are statistically significantly better (using paired two-tailed t-test, the p-value always less than 10^{-12}) than the average coverage time obtained by those algorithms when ran on randomly generated trees. Moreover, the coverage time obtained by running the simplest non-backtracking MSTC algorithm on the trees generated by Create_Tree is, in most cases, even lower than the *optimal* MSTC algorithm ran on randomly generated trees. These results repeated in both dimensions in which the experiment was conducted: number of robots and density of obstacles. The results from running the experiment on B_MSTC are omitted for clarity reasons of the display, but they are compatible with all other results.

An interesting result follows from comparing the improvement in coverage time obtained by the algorithms after performing Create_Tree with different density of obstacles in the terrain. While the improvement in the coverage time obtained by the algorithms after running Create_Tree remains statistically significant compared to randomly generated trees, as the obstacles become more dense the improvement lessens. For instance, the improvement ratio for 30 robots with no obstacles for the NB_MSTC and Opt_MSTC algorithms are 58% and 38% respectively. When the density of obstacles is 26% the improvement ratio decreases to 48% and 28% (respectively).

Figure 8 presents as an example the improvement ratio in coverage time between Create_Tree generated trees vs. randomly generated trees followed by the execution NB_MSTC algorithm. Note that the repetitiveness of the phenomenon is *not* absolute over all number of robots, but the trend is clear. Our initial explanation for the reason of this phenomenon was that the Manhattan distance does not capture the real distance between the robots when more and more obstacles are added to the terrain. However, as stated previously, even when changing the distance measure to shortest paths the results were not improved. We then deduce that as more and more robots are added and as more obstacles are added to the terrain, there is less freedom in constructing the k spanning trees, i.e., the possibility to spread the subtrees away from each other is limited.

An additional interesting results follows from comparing between the ratio of improvement of the results obtained by the Opt_MSTC and the NB_MSTC algorithms (Fig. 8). In both cases the improvement ratio from using the Create_Tree generated

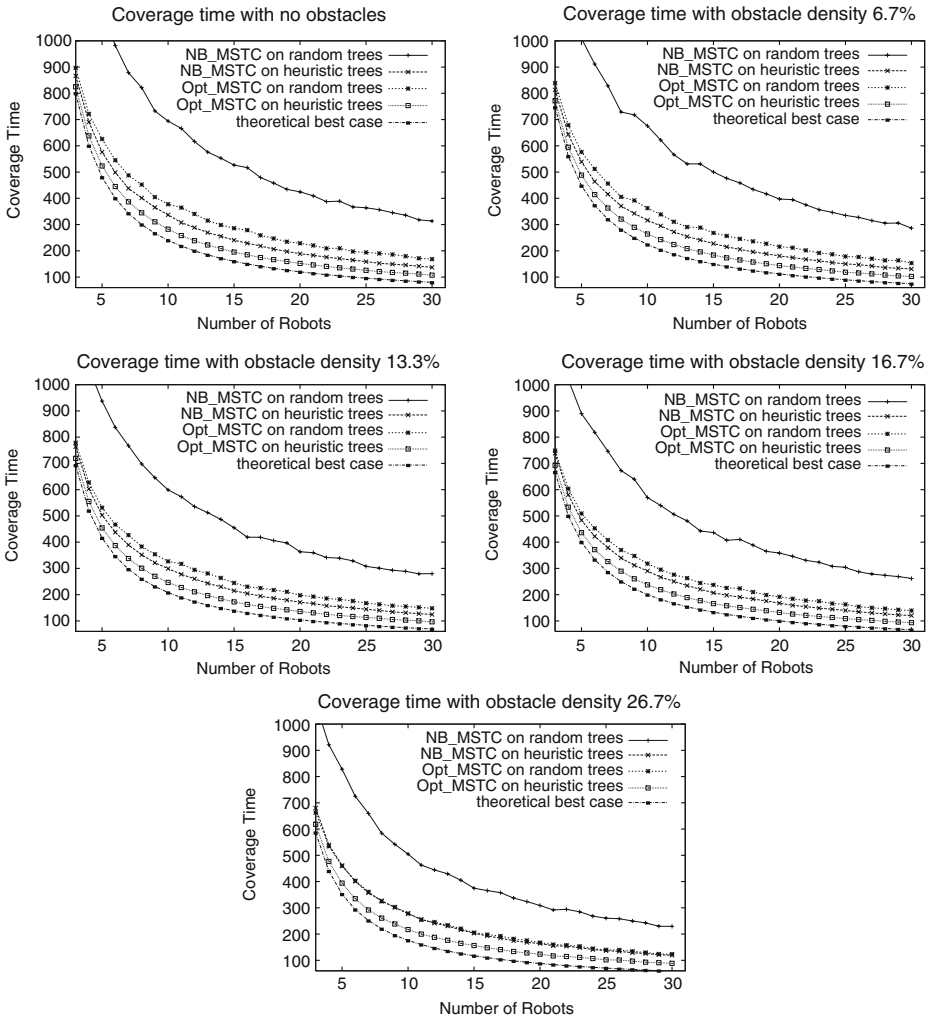


Fig. 7 Results from comparing coverage time when using random trees vs. trees generated using Create_Tree algorithm

trees is relatively high, although using the NB_MSTC coverage algorithm results in much higher improvement ratio. This change is originated in the fact that if using the simple NB_MSTC algorithm the change in coverage time is much more evident. The Opt_MSTC algorithm by itself performs some improvement in coverage time, so there is less to improve from that point.

4 Online spanning-tree based coverage

In this section we re-use the approach of growing and connecting local subtrees in the online coverage case. Here, the robots do not have a-priori knowledge of the

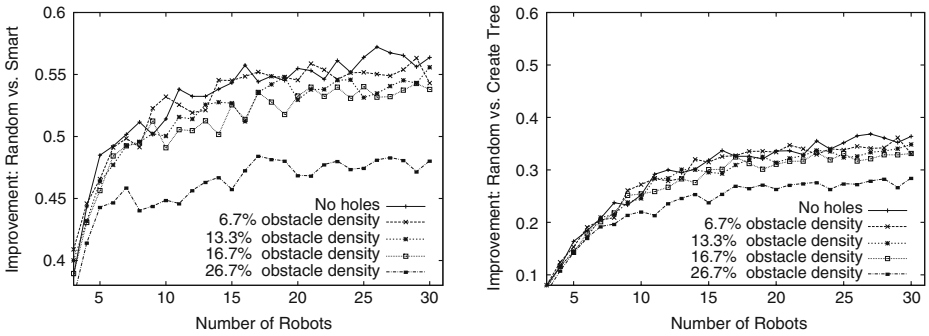


Fig. 8 Comparison between the improvement ratio in coverage time obtained by algorithm NB_MSTC (*left*) and Opt_MSTC (*right*) after generating trees randomly vs. using Create_Tree algorithm with different density of obstacles in the terrain

work-area, i.e., the exact work-area boundaries and all the obstacles locations (which are assumed to be static). We assume the robots know only their absolute initial positions, and that they are able to communicate explicitly with one another.

4.1 Description of the on-line MSTC algorithm

We herein describe in details the online spanning tree based coverage algorithm, and prove it is complete, non-redundant and robust to robot failures.

As mentioned previously, we divide the area into square cells of size $4D$, each one consists of four sub-cells of size D. Denote the number of cells in the grid by N, and denote the number of sub-cells by n, i.e., $n = 4N$. The area occupancy in the beginning is unknown so every cell is initially considered to be empty.

The starting point of the algorithm is the work-area and k robots with their absolute initial positions: $A_0, \ldots, A_{k-1}$. The initial position of every robot is assumed to be in an obstacle-free cell, and the robot should know its position. One assumption the algorithm makes (similar to assumptions made previously in [10, 13]) is that robots can locate themselves within an agreed-upon grid decomposition of the work area. In practice, of course, this assumption is not necessarily satisfied. Section 4.2 below discusses methods for approximating this assumption in practice, which we utilize in our work with physical robots.

We seek algorithms that are *complete*, *non-redundant*, and *robust*. An algorithm is complete if, for k robots, it produces paths for each robot, such that the union of all k paths complete covers the work area. By work area we mean to all the cells which are not occupied by obstacles, and are accessible from at least one of the robots' initial positions. An algorithm is non-redundant if it does not cover the same place more than one time. The robustness criteria ensures that as long as one robot is still alive, the coverage will be completed. Another advantage of this algorithm is that the robots return to their initial positions when the coverage is completed which can facilitate their collection or storage.

The algorithms below are run in a distributed fashion, and generate on-line coverage that is complete, non-redundant, and robust. Each robot runs the initialization algorithm first (Algorithm OMSTC_INIT), and then executes (in parallel to its peers)

an instance of the On-line Robust Multi-robot STC — ORMSTC (Algorithm 3). Each ORMSTC instance generates a path for its controlled robot on-line, one step at a time. It is the union of these paths by the execution sequence of the algorithm that guarantees to be complete, non-redundant, and robust.

We begin by describing Algorithm OMSTC_INIT (2). The initialization procedure constructs the agreed-upon coordinate system underlying the grid work area. It then allows each robot to locate itself within the grid, and update its peers on the initial position of each robot.

Algorithm 2 OMSTC_INIT()

1: Decompose the working area into $2D \times 2D$ cells (grid), agreed among all the robots.
2: Decompose each $2D \times 2D$ cell into 4 sub-cells (size D)
3: $i \leftarrow$ my robot ID
4: **if** $A_i \neq$ the middle of a sub-cell **then**
5: $\quad s_i \leftarrow$ the closest sub-cell
6: $\quad$ Move to s_i
7: **else**
8: $\quad s_i \leftarrow A_i$
9: $S_i \leftarrow$ the cell that contains s_i
10: Announce S_i as your starting location to the other robots
11: Receive S_j, where $j \neq i$, starting cells of other robots
12: Update map with $S_0, \ldots, S_{k-1}$
13: Initialize *connection*$[0 \ldots k-1][0, 1] \leftarrow$ *null*

Once the grid is constructed and robots know their initial positions, Algorithm ORMSTC (executed in a distributed fashion by all robots) carries out the coverage. This recursive algorithm receives two parameters: X, the new cell that the robot just entered, and W, the old cell from which the robot has arrived. We denote a cell with an obstacle in one or more of its four sub-cells, or one that contains the robot's own spanning tree edge, as *a blocking* cell. In the first recursive call to the algorithm, the argument X is the robot's starting cell S_i. W is chosen such that it is consistent with future calls to the algorithm—closest to s_i (Fig. 9).

The idea behind Algorithm ORMSTC is that each robot gradually builds a local spanning tree of uncovered cells that it discovers, while tracking the state of any of its peers whose path it has met. The spanning tree is built by a depth-first-like procedure: Scan for a non-occupied neighboring cell (Lines 1–2), build a tree edge to it (Line 15), enter it (Line 16) and continue recursively with this cell (Line 17). If there is no free cell, the robot goes back along its local spanning tree to the previous covered cell, exiting the recursion (Lines 18–20). See Fig. 10 for an illustration of an execution of ORMSTC.

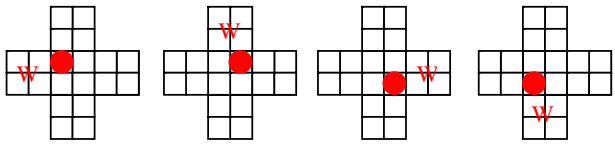
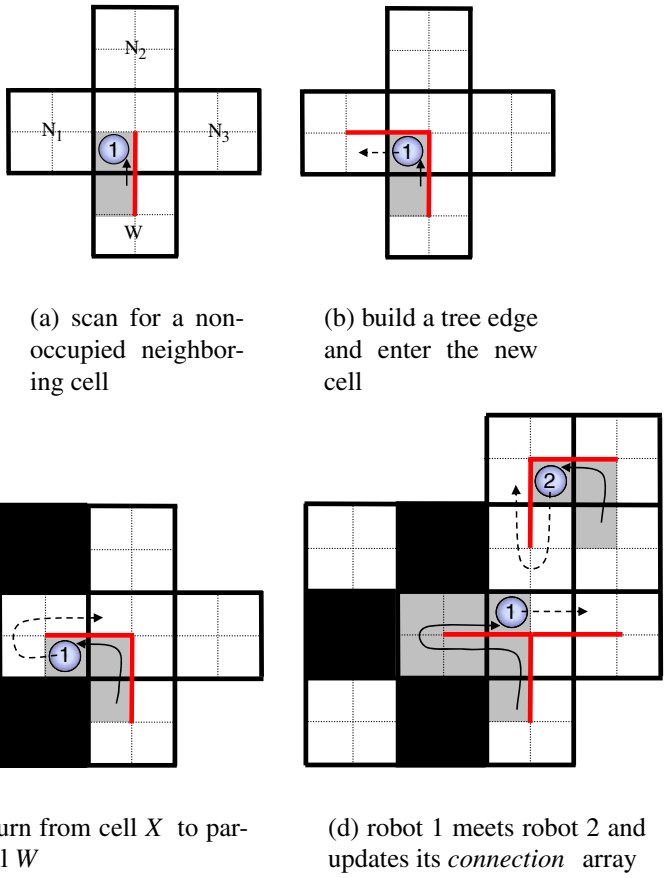


Fig. 9 The 4 possible initial positions (marked with a *dot*), and their respective recommended W

Algorithm 3 ORMSTC(W, X)

```
1: N_{1..4} ← X's neighboring cells in clockwise order, ending with W = N_4
2: for i ← 1 to 3 do
3:    if N_i = blocking cell then
4:       continue to the next i
5:    if N_i has a tree edge of robot j ≠ i then
6:       check whether robot j is alive
7:       if robot j is alive then
8:          if connection[j][0] = null then
9:             connection[j][0] ← the edge from X to N_i
10:         connection[j][1] ← the edge from X to N_i
11:         continue to the next i
12:      else {robot j is not alive}
13:         remove robot j from connections array and broadcast it
14:         mark j's cells as empty on the map and broadcast it
15:   construct a tree edge from X to N_i and broadcast it
16:   move to a sub-cell of N_i by following the right side of the tree edge
17:   execute ORMSTC(X, N_i)
18: if X ≠ S_i then
19:   move back from X to W along the right side of the tree edge
20:   return from recursive call
21: if W ≠ blocking cell then
22:   execute ORMSTC(X, W)
23: move to sub-cell s_i along the right side of the tree edge
24: broadcast completion of work
25: while not all the robots announced completion do
26:   if ∃j, s.t. connection[j][0] ≠ null and robot j is not alive then
27:      mark j's cells as empty on the map and broadcast it
28:      broadcast withdrawal of completion
29:      decide which connection: connection[j][0] or connection[j][1] is closer to s_i
         when moving in clockwise or counter-clockwise direction along your tree
         edges
30:      move to this connection in the appropriate direction
31:      X ← your connection cell
32:      Y ← robot j's connection cell
33:      remove robot j from connections array and broadcast it
34:      construct a tree edge from X to Y and broadcast it
35:      move to a sub-cell of Y by following the right-side of the tree edges
36:      execute ORMSTC(X, Y)
```

During this gradually-expanding coverage process, the first time a robot i meets a cell with robot j's tree-edge ($i \neq j$), it examines its peer's state (Lines 5–6). If robot j is still alive, robot i saves the edge which connects its tree to robot j's tree as $connection[j][0]$ (Lines 7–9). From this point on, robot i will update $connection[j][1]$ to save the last edge which connects its tree to robot j's tree, i.e., whenever robot i meets a cell with robot j's tree edge (Lines 10–11).



(a) scan for a non-occupied neighboring cell

(b) build a tree edge and enter the new cell

(c) return from cell X to parent cell W

(d) robot 1 meets robot 2 and updates its *connection* array

Fig. 10 Illustration of an execution of ORMSTC (**a**–**d**)

If, during this phase, robot i discovers that robot j is not alive anymore, it announces to the other robots that robot j is dead. Then all robots delete the entries for robot j from their *connection* arrays, and the cells which robot j was responsible for are marked empty (Lines 12–14). Robot i and the other robots can now build their spanning tree edges to these cells and cover them (see below for a discussion of the case where two robots want to enter the same cell).

When a robot has no neighboring cells to cover, and it is back in its initial position, it makes sure that W (this is the initial W given as input) is covered (Lines 21–22). Then the robot finishes covering its starting cell and announces to the other robots that it has completed its work (Lines 23–24).

However, the coverage process is not completed until all the robots announce completion of their work. Until then, a robot who finishes its work monitors the state of all the robots for whom it has a non-empty *connection* entry (Lines 25–26). If such a robot j is not alive, the robot sets all cells assigned to j in the map to empty, and updates the other robots (Line 27). It then turns to cover robot j's cells, thus withdrawing its previously-declared completion of its work (Line 28). The robot has

two possibilities to reach robot j's cells: Along the left side of its spanning tree edges, till it reaches *connection*[j][0], the first connection edge between it and robot j's path; or in an opposite direction along the right side of the spanning tree edges, till it reaches *connection*[j][1], the last connection edge between it and robot j. The robot chooses the best option and moves to the chosen connection edge (Lines 29–30). Now it can delete robot j from the connection array (line 31), and continue to construct the spanning tree edges for the new cells by recursively calling the algorithm (Lines 32–36).

Algorithm ORMSTC makes several assumptions about the robots' capabilities. First, in lines 1–2, each robot explores its three neighboring cells. To do this, each robot must have the ability to sense and determine if its three neighboring cells are free from obstacles. If the cell is partially occupied by an obstacle it will not be covered. Second, the algorithm requires reliable communication. Each informative message that a robot receives (a cell that is now occupied with a tree edge, a dead robot, etc.) updates the map and overall world state (in the memory of its peers). Obviously, there is also an assumption here that robots are cooperative, in that when a robot is asked if it is alive, it broadcasts truthfully if it can.

In lines 15 and 34 the robot constructs a spanning-tree edge. A synchronization problem could occur if more than one robot wants to construct a tree edge in the same cell. This can be solved by any synchronization protocol. For instance, we can require robots to notify the others whenever they wish to construct an edge to a cell Q. If a conflict over Q is detected, robots can use their distinct IDs to select who will construct the edge (e.g., highest ID), or they may allow the robot with the smallest number of covered cells to go first (intuitively, this is the most underutilized robot). The other robots treat Q as a cell with another robot's spanning tree edge and continue with the algorithm.

We prove the completeness of the ORMSTC algorithm (Theorem 4). Each robot constructs its own spanning tree and circumnavigates it to produce a closed curve which visits all the sub-cells of the tree cells. Completeness is achieved by ensuring that every cell (which is not occupied by obstacles and is accessible from at least one of the robots' initial positions) will have a tree edge connection from one of the trees.

Theorem 4 (Completeness) *Given a grid work-area WA, and k robots, Algorithm* ORMSTC *generates k paths k_i, such that $\bigcup_i k_i = WA$, i.e., the paths cover every cell within the the work-area.*

Proof By induction on the number of robots k.

Induction Base ($k = 1$) with only one robot, ORMSTC operates exactly like the On-line STC Algorithm which was proven to be complete for every work-area (Lemma 3.3 in [10]).

Induction Step Suppose it is known that $k - 1$ robots completely cover every work-area. We will prove it for k robots. Without loss of generality, let us consider robot i. Executing ORMSTC, i will build its local spanning tree edges, and generate a path to cover some cells. The other robots treat these cells as occupied, exactly as if they were filled with obstacles. Therefore all the other cells will be part of $k - 1$ paths and covered by the $k - 1$ robots, according the induction relaxation. Robot i treats all the

cells of the other $k-1$ robots as occupied cells, so it will completely cover its cells according to the induction assumption. □

We now turn to examining ORMSTC with respect to coverage optimality. Previous work has discussed several optimization criteria [13], one of which is *redundancy*, the number of times a sub-cell is visited.

ORMSTC can be shown to be non-redundant. Theorem 5 below guarantees that the robots visit all the cells only once (if no failure has occurred—see below for a discussion of robustness). This guarantee is in fact a feature of many spanning-tree coverage algorithms, as circumnavigating a tree produce a closed curve which visits all the sub-cells exactly one time [10]. The non-backtracking algorithm in [10], which is an off-line algorithm, divides this curve between the robots to achieve a complete non-redundant coverage.

Theorem 5 (Non-Redundancy) *If all robots use Algorithm* ORMSTC*, and no robot fails, no cell is visited more than once.*

Proof If no robot fails, then each robot only covers the cells for which it builds a tree edge. If there is already a tree edge to a cell, the robot will not enter it (Line 5). Thus every cell is covered only by a single robot. Since robots never backtrack, every point is only covered once. □

As key motivation for using multiple robots comes from robustness concerns, we prove that Algorithm ORMSTC above is robust to catastrophic failures, where robots fail and can no longer move. Lines 12–14 and 25–36 guarantee the robustness. If one robot fails, there is always at least one robot that will detect it and will take the responsibility to cover its section (see below for formal proof). Conflicts over empty cells are handled as described above.

Theorem 6 (Robustness) *Algorithm* ORMSTC *guarantees that the coverage will be completed in finite time even with up to* $k-1$ *robots failing.*

Proof Based on the completeness theorem (Theorem 4), any number of robots can cover the work area. Thus if one or more robots fail, all the cells that were not occupied by tree edges of the failing robots and are accessible to other live robots will be covered. So all we have to prove is that cells with tree edges of a dead robot, or cells which are accessible only to a robot that has died will be covered by another robot. Such cells may exist due to the structure of the work area, or because the dead robot covered a group of cells which blocks the access of other robots to free cells.

Cells with existing tree edges of a robot are treated by the other robots as cells with obstacles. According the completeness theorem, there is at least one robot that will cover a neighboring cell of one of these cells, thus will have a connection to this cell. There are two possible cases:

1. A robot failed before a robot that has a connection with it reached the connection. In this case, lines 13–14 ensures that the dead robot's covered cells will be declared free so they will be covered by other robots.

2. A robot fails after all the robots that have a connection with it reached the connection. In this case, lines 27 and 33 apply, to ensure that the robot's covered cells will be declared free so they will be covered by other robots.

In both possible cases, the freeing of cells previously-covered by the dead robot also makes any cell which was only accessible to the dead robot accessible to others. Based on the completeness theorem, at least one other robot is guaranteed to reach all these cells. Thus the algorithm is proved robust. □

4.2 From Theory to Practice

In real-world settings, some of the assumptions underlying ORMSTC can not be satisfied with certainty, and can only be approximated. This section examines methods useful for such approximations, and their instantiations with physical robots.

In particular, we have implemented the ORMSTC algorithm for controlling multiple vacuum cleaning robots, the RV-400 manufactured by Friendly Robotics [9]. Each commercial robot was modified to be controlled by an small Linux-running computer, sitting on top of it. A generic interface driver for the RV-400 robot was built in Player[12], and a client program was built to control it. Each robot has several forward-looking sonar distance sensors, as well as sideways sonars. One robot is shown Fig. 11.

The ORMSTC algorithm (indeed, many of the STC algorithms) make several assumptions. First, there are assumptions as to the work area being provided as input. ORMSTC assumes, for instance, that the work-area has known bounds, and that it is divided into a grid that is known by all robots (i.e., all robots have the same division). ORMSTC assumes robots can communicate reliably, and locate themselves within a global coordinate system. Finally, ORMSTC makes assumptions about the sensory information available to the robots. In particular, ORMSTC makes the assumption that each robot can sense obstacles within the front, left, and right $4D$ cells.

One challenging assumption is that of a global coordinate system that all robots can locate themselves within. In outdoor environments, a GPS signal may in principle be used for such purposes (note that the position only has to be known within the resolution of a sub-cell). However, in circumstances where a global location sensor (such as the GPS) is unavailable, a different approach is needed. In particular, this is true in the indoor environments in which the vacuum cleaning RV-400 is to operate.

Fig. 11 RV-400 robot used in initial experiments

For the purposes of the experiments, we have settled on letting the robots know their *initial* location on an arbitrary global coordinate system. Once the robots began to move, however, they relied solely on their odometry measurements to position themselves. In the future, we hope to experiments with alternative approaches.

One advantage of ORMSTC in this regard is that its movements are limited to turns of 90° left or right, and to moving forward a fixed distance. This offers an opportunity for both reducing errors by calibration for odometry errors specific to this limited range of movements, and by resetting after each step, thus avoiding accumulative errors. Indeed, this was the approach taken in the experiments (see next section).

Given a global coordinate system, ORMSTC also requires robots to agree on how to divide up the work-area into a grid. This agreement is critical: Differences in the division may cause grids created by different robots to be mis-aligned, or overlap. To do this, the bounds of the grid have to be known, in principle. Once the bounds are known, the robots only have to decide on the origin point for the approximate cell decomposition.

Here again a number of approximating solutions were found to be useful. First, one can have the robots use a dynamic work-area. During the initialization phase, the robots determines the maximal distances, X_{max} and Y_{max} (along the X- and Y-axes, respectively), over all pairs of robots. They then build a temporary rectangular work-area around them, with sides greater or equal to X_{max}, Y_{max}. As the robots move about, they will push the boundaries of the work-area into newly discovered empty cells that lie beyond the bounds, or they will encounter the real bounds of the work area, which will be regarded as obstacles. A related approximation is to provide the robots with an initial work-area that is known to be too big, and allow the robots to discover the actual bounds. This was the technique we utilized.

Robustness against collisions is an additional concern in real-world situations. Normally, as each robot only covers the path along its own tree, Theorem 5 guarantees that no collisions take place. This separation between the paths of different robots decreases the chance of collisions. In practice, localization, movement errors, and the way the grid is constructed may cause the robot to move away from its assigned path, and thus risk collision. We utilized our bumps sensors to cope with this problem as they are often used as a key signal in vacuum-cleaning robots. Our heuristic is to simply respond to a bump by moving back a little, waiting for a random (short) period of time trying again. If bumps occur three times in a row in the same location, the location is marked as a bound or obstacle. A more complicated solution which requires more communication is to coordinate between the robots that have adjacent tree edges when a collision is likely to occur.

A final challenge was offered by the robots' limited sensor range. The robot is equipped with ten sonar sensors which are not capable of sensing all three neighboring cells of the robot cell at the same time as described in the algorithm requirements before. We solved this problem by dividing the original sensing and movement phases to three steps. The robot first senses its first cell by turning its sensors towards it. If it is empty, it continues with the regular algorithm flow. If not, in moves forward to be as close as possible to the border of the next require-sensing cell and only then it turns to sense it and continues with the algorithm. The same procedure is performed to the third neighboring cell. Although it slowed down the algorithm performance this fix enabled us to run the algorithm in the simulation with the robots constraints,

so it can be applied also to run the algorithm on different real robots with limited sensors.

4.3 Experimental results

We conducted systematic experiments with our implementation of the ORMSTC algorithm, to measure its effectiveness in practice with the RV400 robot. The experiments were conducted using the Player/Stage software package [12], a popular and practical development tool for real robots. Initial experiments were carried out with physical RV400 robots, to test the accuracy of the simulation environment used. However, to measure the coverage results accurately, the experiments below were run in the simulation environment. Figure 12 shows a screen shot of running example with six robots in one of the simulated environments used in the experiments.

In the experiment, we focused on demonstrating that the ORMSTC algorithm—and our implementation of it for real robots—indeed manages to effectively use multiple robots in coverage. We ran our algorithm with 2,4,6,8 and 10 robots. Each team was tested on two different environments. The *Cave* environment had irregularly-shaped obstacles, but was relative open. The *Room* environment had many rectangular obstacles, and represents a typical indoor office room. For each team size and environment type, 10 trials were run. The initial positions were randomly selected.

The results are shown in Fig. 13. The X-axis measures the number of robots in the group. The Y-axis measures the coverage time. The two curves represent the two different environments. Every data point represents the average ten trials, and the horizontal line at each point shows the standard deviation in each direction.

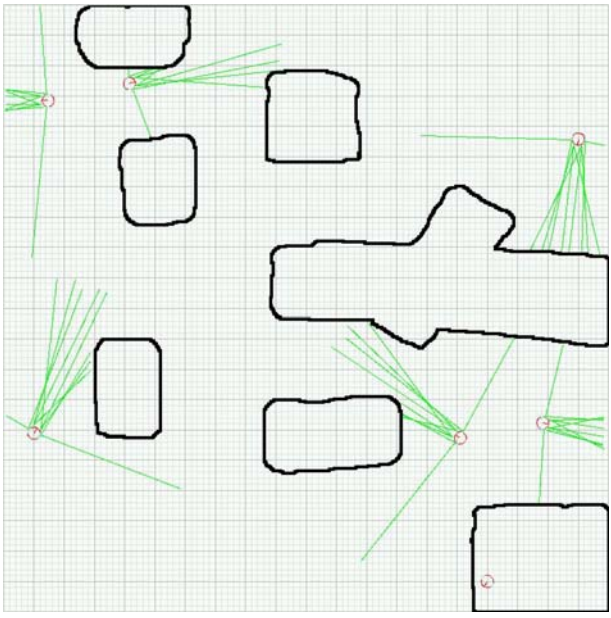


Fig. 12 Simulation screen shot of six robots covering the *Cave* environment

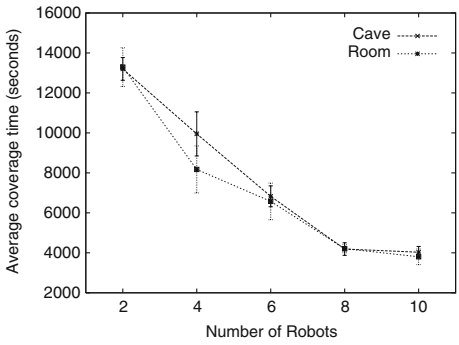


Fig. 13 Overall coverage time

The results show that in both environments, coverage time decreases in general when increasing the group size. However, we can also see that the marginal coverage decreases with the addition of new members. This is a well-known phenomenon (in economics, but also in robotics). It is due to the overhead imposed on a bigger group of robots, in collisions avoidance and communication load. The overhead cost can be also seen when comparing the two coverage times of the two environments. Although the indoor environment is smaller, the coverage time is almost the same because there are more obstacles and doors to pass and there is a greater chance of collision with walls or other robots.

5 Conclusions

Many real-world coverage applications require multiple robots to completely cover a given work-area, either with a given map of the area (offline coverage) or with no a-priori knowledge of the area (online coverage). A popular approach for such coverage rely on the spanning tree coverage method (initially introduced for single robot coverage by Gabrieli and Rimon in [10]). All of the spanning tree coverage algorithms initially depend on the selection of a spanning tree.

In this work we have discussed the importance of the structure of the spanning tree on the coverage time obtained by algorithms that use this tree as base for coverage. We used the same basic idea for tree generation in both online and offline scenarios: constructing local subtrees, and from them generating a spanning tree.

First, we have focused on offline coverage. We have shown that the structure of the tree can have crucial consequence on the coverage time. We have presented an algorithm for constructing trees that is motivated by the objective similar to the one defining an optimal tree, a problem that is thought to be $\mathcal{NP}$-hard. We have extensively tested the influence of the spanning tree structure on the coverage time obtained by existing algorithms while taking several parameters under consideration. In these simulations, we compared coverage time obtained by the family of MSTC algorithms on trees constructed by our heuristic procedure against random trees. Simulation results show that when using heuristic trees, then the resulted coverage time obtained by all algorithms were statistically significantly better than the results obtained by running the algorithms on randomly generated trees. Moreover, the average coverage time obtained by the simplest algorithm on spanning trees created

by our procedure were, in most cases, better than the average coverage time obtained by the best algorithm on randomly generated trees.

We then focused on online coverage. We reused the approach we took, to develop an online robust coverage algorithm, the ORMSTC, a multi-robot coverage algorithm which is able to cover an unknown environment. We analytically showed that ORMSTC algorithm is complete and robust in face of catastrophic robot failures. As there is always a gap between theory and practice, we analyzed the assumptions underlying the algorithmic requirements. We discuss various approximation techniques for these requirements, to allow the algorithm to work in real world situations. Based on early trials with real-robots, we conducted systematic experiments with our implementation, to measure the ORMSTC's effectiveness in practice. The results show that the algorithm works well in different environments and group sizes.

References

1. Acar, E.U., Choset, H.: Robust sensor-based coverage of unstructured environments. In: International Conference on Intelligent Robots and Systems, pp. 61–68, Maui, 29 October–3 November 2001
2. Agmon, N., Hazon, N., Kaminka, G.A.: Constructing spanning trees for efficient multi-robot coverage. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Orlando, 15–19 May 2006
3. Batalin, M.A., Sukhatme, G.S.: Spreading out: a local approach to multirobot coverage. In: Proc. of the 6th Internat. Symposium on Distributed Autonomous Robotic Systems, pp. 373–382, Fukuoka, 25–27 June 2002
4. Butler, Z.J., Rizzi, A., Hollis, R.L.: Complete distributed coverage of rectilinear environments. In: Workshop on the Algorithmic Foundations of Robotics, Hanover, March 2000
5. Choset, H.: Coverage for robotics—a survey of recent results. Ann. Math. Artif. Intell. **31**(1–4), 113–126 (2001)
6. Colegrave, J., Branch, A.: A case study of autonomous household vacuum cleaner. In: AIAA/NASA CIRFFSS Conference on Intelligent Robots for Factory Field, Service, and Space, Houston, 20–24 March 1994
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT, Cambridge (1990)
8. Ferranti, E., Trigoni, N., Levene, M.: Brick & mortar: an on-line multiagent exploration algorithm. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Rome, 10–14 April 2007
9. Friendly Robotics®, Ltd.: Friendly robotics vacuum cleaner. http://www.friendlyrobotics.com/friendly vac/ (2009)
10. Gabriely, Y., Rimon, E.: Spanning-tree based coverage of continuous areas by a mobile robot. Ann. Math. Artif. Intell. **31**(1–4), 77–98 (2001)
11. Gabriely, Y., Rimon, E.: Competitive on-line coverage of grid environments by a mobile robot. Comput. Geom. **24**, 197–224 (2003)
12. Gerkey, B.P., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the International Conference on Advanced Robotics, pp. 317–323, Coimbra, July 2003
13. Hazon, N., Kaminka, G.A.: Redundancy, efficiency and robustness in multi-robot coverage. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Barcelona, 18–22 April 2005
14. Hazon, N., Kaminka, G.A.: On redundancy, efficiency, and robustness in coverage for multiple robots. Robot. Autonom. Syst. **56**(12), 1102–1114 (2008)
15. Hedberg, S.: Robots cleaning up hazardous waste. AI Expert **5**, 20–24 (1995)
16. Huang, Y.Y., Cao, Z.L., Hall, E.L.: Region filling operations for mobile robot using computer graphics. In: Proceedings of the IEEE Conference on Robotics and Automation, pp. 1607–1614, San Francisco, April 1986

17. Kong, C.S., New, A.P., Rekleitis, I.: Distributed coverage with multi-robot system. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Orlando, 15–19 May 2006
18. Osherovich, E., Yanovski, V., Wagner I.A., Bruckstein, A.M.: Robust and efficient covering of unknown continuous domains with simple, ant-like a(ge)nts. Technical report, Technion, Israel (2007)
19. Rekleitis, I., Dudek, G., Milios, E.: Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In: International Joint Conference in Artificial Intelligence (IJCAI), vol 2, pp. 1340–1345. Morgan Kaufmann, Nagoya (1997)
20. Rekleitis, I., Dudek, G., Milios, E.: Multi-robot collaboration for robust exploration. Ann. Math. Artif. Intell. **31**, 7–40 (2001)
21. Rekleitis, I., Lee-Shue V., Peng New, A., Choset, H.: Limited communication, multi-robot team based coverage. InL IEEE International Conference on Robotics and Automation, pp. 3462–3468. IEEE, Piscataway (2004)
22. Svennebring, J., Koenig, S.: Building terrain-covering ant robots: a feasibility study. Auton. Robots **16**(3), 313–332 (2004)
23. Wagner, I.A., Lindenbaum, M., Bruckstein, A.M.: Mac vs. pc determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. Int. J. Rob. Res. **19**(1), 12–31 (2000)
24. Zheng, X., Jain, S., Koenig, S., Kempe, D.: Multi-robot forest coverage. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Edmonton, 2–6 August 2005