Ann Math Artif Intell (2009) 57:293–320
DOI 10.1007/s10472-010-9193-y

Multi-robot area patrol under frequency constraints

Yehuda Elmaliach · Noa Agmon · Gal A. Kaminka

Published online: 2 July 2010

Abstract Patrolling involves generating patrol paths for mobile robots such that every point on the paths is repeatedly covered. This paper focuses on patrolling in closed areas, where every point in the area is to be visited repeatedly by one or more robots. Previous work has often examined paths that allow for repeated coverage, but ignored the frequency in which points in the area are visited. In contrast, we first present formal frequency-based optimization criteria used for evaluation of patrol algorithms. Then, we present a patrol algorithm that guarantees maximal uniform frequency, i.e., each point in the target area is covered at the same optimal frequency. This solution is based on finding a circular path that visits all points in the area, while taking into account terrain directionality and velocity constraints. Robots are positioned uniformly along this path in minimal time, using a second algorithm. Moreover, the solution is guaranteed to be robust in the sense that uniform frequency of the patrol is achieved as long as at least one robot works properly. We then present a set of algorithms for handling events along the patrol path. The algorithms differ in the way they handle the event, as a function of the time constraints for handling them. However, all the algorithms handle events while maintaining the patrol path, and minimizing the disturbance to the system.

N. Agmon (✉) · G. A. Kaminka
The MAVERICK Group, Computer Science Department, Bar Ilan University,
Ramat Gan, 52900, Israel
e-mail: segaln@cs.biu.ac.il, noasag@gmail.com

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

Y. Elmaliach
Computer Science Department, College of Management Academic Studies,
Rishon LeZion, 75490, Israel
e-mail: elmaley@gmail.com

Keywords Multi-Robotics · Multi-robot path planning · Team planning

Mathematics Subject Classifications (2010) 93C85 Robotics · 93C85 Automated systems

1 Introduction

Robots can save human lives and costs by replacing humans in mundane, or dangerous tasks. For instance, robots may be used for cleaning [12], and hazardous waste removal [25]. Some specific applications of interest involve surveillance and patrolling along polylines (open-ended fences) [16], perimeters [4, 34], or sensitive areas [8].

This paper discusses the problem of patrolling within a target work area, enclosed within a closed polygon. Patrolling is defined as *"The act of walking or traveling around an area, at regular intervals, in order to protect or supervise it"* [1]. If the entire terrain cannot be monitored at all times, each location in the target area is monitored once every t time cycles. The frequency is, then, $f = 1/t$. Increased availability of multiple robots raises new opportunities for patrol missions. First and foremost, patrolling can be made more time-efficient in the sense that the frequency is potentially higher, i.e., t is smaller. In addition, robustness can be attained in the sense that if at least one robot is active, the patrol mission can still be accomplished.

Previous work has offered several approaches to patrolling of areas [6, 7, 9, 19, 28]. However, key challenges in surveillance have been left open. First, patrolling has mostly been done in ad-hoc fashion, without a formal analysis of the quality of the task in light of its principal goal. Second, the opportunity for increased robustness in the sense of overcoming robot failure has not been investigated theoretically. Third, handling non-uniform terrains in terms of velocity and directional constraints was not addressed.

Hence, this paper deals with constructing patrol paths for a group of mobile robots that are provably optimal (in terms of point-visit frequency) and robust (to robot death). We base our work on recent multi-robot coverage algorithms [2, 3, 23, 24], which cover all points in an area *once*. We extend those to patrolling of a non-uniform continuous target area (divided into a grid). The solution we present guarantees that every point will be visited with the same frequency by creating one cyclic patrol path visiting all points in the target area (a Hamiltonian cycle in the grid), and instructing all robots to move along this cycle while maintaining equidistant relative positions. Note that a Hamiltonian cycle necessarily exists in the terrains we consider (see Section 3.2).

Robots have velocity limitations, which depends on both the *terrain* in a given location, and *direction* in which they travel. For example, climbing up a hill is typically be done in a lower velocity compared to climbing down the same hill. Therefore a cost should be associated with each point (and direction) of the terrain, making the terrain grid directionally non-uniform.

We therefore consider directionally non-uniform terrains. We first provide an algorithm that finds the minimal cost cyclic path (minimal cost Hamiltonian cycle) given the terrain. We then find points along the path from which the patrol will start, and find an optimal assignment of robots to those locations in the sense that they will arrive at their starting points in minimal time. Finally, we evaluate our derived patrol algorithm using the frequency optimization criteria described in Section 3.1. By basing our solution on the choice of minimal cost Hamiltonian cycle, we guarantee maximal frequency and uniform frequency in the cycle, as traveling in a minimal cost path will guarantee minimal time duration between every two visits to each point. Moreover, guaranteeing uniform distance between the robots along the execution guarantees uniform frequency. Similar to the robustness of the multi-robot coverage described in [23], our solution is robust, and guarantees maximal uniform frequency for one or more non-faulty robots.

This paper also deals with the case in which while patrolling along the area, the robots are required to handle events. An event is a transient addition to the area in a specific location, such that it requires special attention by the robots as they pass through it for a limited period of time. We assume that each event has an associated urgency of handling it. This is expressed by a limit on the time during which this event should be handled. For example, in security applications an event could be a detection of an unauthorized personnel inside the area, and in cleaning missions it could correlate to cleaning a broken glass in addition to the regular cleaning mission.

We provide a set of algorithms that deal with a single event. The difference between the algorithms lies in the urgency of handling the event. If there is enough time to handle the event jointly by all robots, then it will be divided between the team of robots uniformly. This division can guarantee that the event will be handled along with minimizing the interference with the patrol path to other cells in the area. Specifically, we are interested in minimizing the time it takes the robots to regain their uniform distribution along the cyclic path. In addition, we would like to minimize the frequency disturbance to other cells during the period of time the event is handled. We show that the algorithms we provide guarantee these properties whenever possible.

This paper is organized as follows. The next section provides an overview of related work, and motivates our formulation of the area patrolling problem. Section 3 formulates the area patrol problem from the point of view of point-visit frequency optimization. It presents the spanning-tree patrolling (STP) algorithm, and the initialization algorithm required to show optimal patrolling. Section 4 extends the original STP algorithm to account for events which cause delays in patrolling. Section 5 concludes.

2 Background

The patrolling task, sometimes referred to as *repetitive sweeping* or *repetitive coverage*, is relatively a recent challenge area for multi-agent and multi-robot researchers. In general, three approaches can be found in the literature:

Cyclic paths A principled approach to patrolling, in which the algorithms plan cyclic paths through the work area. By nature, points on the cyclic path are visited repeatedly as long as the robots continue to move along the path.

Randomized movement A different approach to patrolling relies on continuously executing randomized movement in the work area. The nature of the randomization results in uniform (given sufficient time) distribution of visits to each point in the work area.

Hierarchical area division (partitioning) Not strictly a patrolling approach, but one used often in connection with multi-robot patrolling. In this approach, the work area is (recursively) partitioned into sub-areas. Each sub-area is allocated, using some task-allocation mechanism, to different robots. Each robot then patrols the sub-area using a single-robot patrol algorithm (Randomized Movement or Cyclic Path). Robots may switch or take over areas as needed.

Mechado et al. in [28] describe the problem in terms of movement in a general graph, argue for its novelty in terms of existing works at the time, and introduce a measure of patrolling quality, called *idleness*. Idleness (of a graph) measures the average number of time steps between visits to all vertices. They then utilize idleness in a number of heuristic patrolling algorithms for multiple robots, based on local or global path-planning, which they compare empirically in simulations. These are intended to balance average and worst-case idleness over time. In this work, we provide a formal treatment of patrolling and idleness, which we translate to point-visit frequency. We introduce three different optimization criteria that build on idleness (average frequency, uniformity of frequency, and worst-case frequency) and analytically show that these criteria can all converge to optimal in the case of area patrolling, using the algorithms we develop.

A survey by Almeida et al. [7] brings a discussion of different patrolling approaches, with respect to the idleness criteria. They compare between patrol paths created by the following different methods. (a) Machine learning methods. (b) Patrol paths generated by agents using negotiation mechanisms. (c) A heuristic patrol algorithms based on greedy local idleness (see above) criteria. (d) An approach based using an approximation to the Traveling Salesman Problem (TSP), i.e., finding a minimal cost cyclic path that visits the entire graph, where all agents visit all vertices in the graph (as opposed to the previous methods that divide the graph between the different agents). They empirically demonstrate significant advantages to the TSP-based approach in the idleness criteria. Our work addresses patrolling formally and analytically, in contrast to this work. We focus on a cyclic-path approach, and analytically show its optimality. However, rather than examining general graphs, we focus on patrolling in two-dimensional work-areas, in which we use approximate cellular decomposition.

Chevaleyre [9] and Chevaleyre et al. [10] offer a theoretical analysis of the patrol problem. They discuss two approaches to multi-robot patrolling in general graphs: One in which the problem is treated as finding a cyclic path through all nodes (i.e., the Traveling Salesman Problem TSP); and another in which the general graph is partitioned into k sub-graphs (where there are k robots), and each partition is patrolled independently. They examine the conditions under-which a cyclic approach would be preferable to a partition-based approach, in terms of the worst idleness criteria. While we examine the optimality of other criteria as well (e.g., uniformity of point-visit frequency), we specialize our techniques to specific graphs, grids that form approximate cell decompositions of two-dimensional work areas. However, we also provide guarantees on robustness and efficiency of multi-robot solutions.

Empirical investigations of patrolling have often utilized a partitioning approach, but often disregarded point-visit frequency, in favor of coordination and robustness concerns. For instance, Guo et al. divide the patrolling area between robots [21, 22]. This study focused on the robots' localization and sensorial capabilities. This paper's goals are complementary, in the sense that we assume perfect sensors and localization, but provide guaranteed frequency optimization.

Ahmadi and Stone [6] describe a negotiation-based approach for dividing the area between the robots, dealing with events such as addition and removal of robots from the system. We instead provide analytical treatment of robot removal and addition, and provide algorithms that guarantee optimal patrolling frequency, as well as a procedure to minimize the time for adjusting the patrolling to the removal or addition of robots.

Jung and Sukhatme describe in [26] a region based approach for tracking targets in a system with multiple robots and stationary sensors. They explicitly discuss patrolling frequency. We do not utilize stationary sensors in this work, and show that there are several different frequency criteria possible.

It is also possible, in principle, to carry out patrolling by repeated movements within the work area. Many swarm or ant-based coverage algorithms, when executed indefinitely, may in practice result in uniform distribution of point visits, though the frequency of their visits might not be easily guaranteed. One work that stands out among these is work by Yanovski et al. [35] who have shown that an ant-like exploration in a general graph, by multiple agents using simulated markings in the vertices, can result in point visit frequency which is uniform up to a factor of two (i.e., the number of visits to the most visited edge is no more than twice the number of visits to the least visited edge).

In this, Yanovski et al. build on earlier work on ant-robot coverage [31–33]. Indeed, the patrol problem is closely related to the area coverage problem, in the sense that both require the robot or group of robots to visit all points in the given terrain. However, while coverage seeks to minimize the number of visits to each point (ideally, visiting it only once), patrolling seeks to maximize it (while still visiting all points). Therefore solutions that are used for the coverage problem might be used as basis for patrolling.

Our own approach builds on earlier work in coverage; specifically, on Spanning Tree Coverage (STC), first introduced by Gabriely and Rimon [17] for single robots, and then extended to the multi-robot case by Hazon and Kaminka [23, 24] and by Agmon et al. [2, 3]. The key idea in this family of algorithms is to approximate a two-dimensional work area using a grid, such that a Hamiltonian cycle is guaranteed to exist through the grid, which can be found by generating a spanning tree in the grid graph. This Hamiltonian cyclic path is used as the basis for patrolling, as the next section shows. Although we build on MSTC, our work here differs from it in three important ways. First, we allow modeling non-uniform terrains, in the sense of velocity and direction constraints imposed on different locations within the area. Second, we provide an algorithm for placing the robots such that their patrolling can commence as quickly as possible, a stage only worthwhile in patrolling. Finally, we address here also events that cause delay in patrolling, where previous work ignored such events.

There are additional related investigations. First, the discussion above (and this paper) focused on patrolling areas, as a special instance of patrolling in general graphs. Recently, Elmaliach et al. have also tackled frequency-based patrolling in open polylines [16], which is a distinct acyclic special instance of the problem. While the optimization criteria we introduce here remain the same, it is possible to show that in polyline patrolling, it is impossible to achieve uniform visit frequency, in contrast to the case in patrolling areas.

Second, a key difference between our work and all works discussed above is that we address the allocation of patrolling robots to handle events. This allocation is an instance of the general task allocation problem for multiple robots. There are many approaches used for solving this problem in various domains; however, recently, market-based approaches seem to be of particular interest: Smith created a system called *Contract Net*, a distributed problem solving in which the nodes in the system negotiate and send their bid to the manager which allocates the task for the lowest coast bid [30]. Dias et al. [14, 15] first use the concept of market based for multiple robots that cooperate for achieving a common goal. Golfarelli et al. [20] proposed a negotiation protocol in environment that the only possible contract could be swapping the task between the agents.

In contrast to these general techniques, we take advantage of the cooperative nature of the patrolling task (as described in this paper), and specialize the task allocation mechanism such that the load of handling events is divided between the robots equally. Each event is handled by all robots using no negotiation, or other tools for determining which robot will handle which part of the event. Our solution is, therefore, easy to solve and is determined quickly.

We emphasize that patrolling, as studied in this paper, is investigated from the point of view of optimizing point visit frequency. There are alternative optimization criteria for patrolling. For instance, Paruchuri et al. [29] and Agmon et al. [4, 5] study patrolling in adversarial environments, in which the robots' goal is to maximize their rewards. These rewards are received if the robots manage to observe the adversary, which tries to evade the patrolling robots.

3 Frequency-based area patrolling

We now turn to introducing our approach for patrolling of a two-dimensional work area. Section 3.1 introduces the area patrol problem, and the criteria used to assess patrolling quality. Section 3.2 then introduces the basic ideas underlying spanning-tree patrolling, the key idea in the algorithms we present. It also shows how to account for velocity and directionality constraints in generating the patrolling paths, to guarantee optimal patrolling frequency. Section 3.3 addresses the bootstrapping stage in which robots find their initial positions along the patrolling path. Section 3.4 ties the generation of the patrol paths, and the assignment of initial positions to robots, to discuss the optimality and robustness guarantees on the algorithm.

3.1 The area patrol problem

We are given a polygon enclosing a continuous work area. We are given k robots, that are required to *repeatedly* visit every point within the area. The work area is decomposed into cells, where each cell is of the size of the robots' cover tool. The area may contain obstacles, through-which robots cannot move. We denote the number of cells in the grid that do not contain obstacles by N. If $k \geq N$, then all points can be visited at all times by the team simply by assigning a robot to each point. Formally, the time interval between visits is 0, since each point is visited with every passing time unit, by at least one robot. However, in the more common case, $k \ll N$. In this case, necessarily, at least some of the cells have a non-zero time interval between visits.

Thus the frequency in which points are visited are the focal point for the area patrol problem. There are several possible point visit frequency criteria according to which patrol algorithms can be evaluated:

Uniform frequency The goal is to decrease the variance between the frequencies in which each point is visited, i.e., all targets should ideally be visited with uniform frequency f.

Average frequency The goal is to increase the average frequency f in which targets are visited. Note that this is independent of achieving uniform frequency.

Under-bounded frequency The goal is to increase the minimal frequency in which any target is visited, such that every target is visited with frequency of at least f. In other words, all points should be visited at least once every $1/f$ cycles.

Given the goals above, the area patrol problem is to generate trajectories (velocity along paths) for each of the k robots, such that these achieve one or more of the goals above (or maximize some given trade-off between them). As in work on multi-robot coverage, different variations exist [11]: *Offline* (map of area and obstacles given a priori) or *online* (work area unknown), using *approximate* cell coverage (when using approximate cell decomposition methods, partial cells may be excluded from the coverage path), or *exact* (all points in the area visited). Patrolling may take into account directionality and velocity constraints (e.g., the velocity in one direction may be different than in another), priorities in patrolling, etc.

3.2 Spanning-tree patrolling

This paper addresses the area patrol problem as defined above, focusing on offline trajectory planning and approximate cell decomposition (utilizing a grid). We also take into account directional movement constraints, allowing the algorithms to model different robot velocity constraints in different directions. The algorithms achieve all three goals stated above: Uniformity of patrolling frequency, maximal average patrolling frequency, and maximal minimum frequency. We do this by generating a cyclic path visiting all target areas (a Hamiltonian cycle) and then place the robots uniformly along the cyclic path. If all robots move at the same direction then clearly each cell is visited at the same frequency (uniform frequency). Moreover, in uniform terrains each cell is visited at least once every $\lceil \frac{cycle\ length}{num\ robots} \rceil$ number of cycles, where cycle length is simply the number of nodes plus one.

We base our work on Multirobot Spanning Tree Coverage (MSTC) [3, 23]. A single robot is assumed to be equipped with a square-shaped tool of size D: Any point within the tool's area is taken to be visited. The robot moves by sliding the tool over the area in any of the four basic directions (North, South, East, West). The work area is approximately divided into a grid with cells of size D. The grid is then made coarse, such that each new cell is of size $2D \times 2D$. The center-points of all such cells are connected to those of their neighbors in the four basic directions (North,South,East,West), to form a graph. A spanning tree is then induced from the graph. The robots follow the tree around in a clockwise or counterclockwise direction, creating a Hamiltonian cycle visiting all cells of the original grid (see example in Fig. 1).

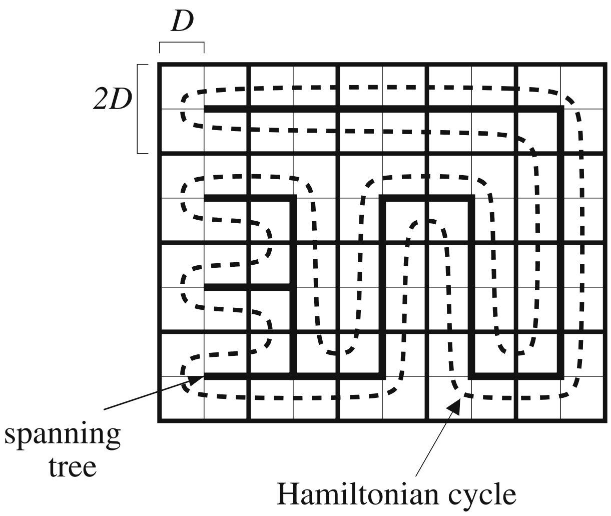

Fig. 1 An example of spanning tree based coverage. Coarse grid is in *bold*, and the spanning tree connects all coarse grid cells. The Hamiltonian cycle over the fine grid is the *dotted line* along the spanning tree

The key idea in Spanning-Tree Patrolling (STP) is to utilize the cyclic path for repeated patrolling. By placing robots in equidistant positions along the cycle induced by the spanning tree, synchronized movement by the robots will provide uniform, maximal frequency of visits to all points along the path. However, in non-uniform terrains, movement in the four different directions can occur in different velocities. Moreover, terrain directionality constraints may mean that clockwise movement in any given location may have different velocity than in the counterclockwise direction.

We assign a cost—signifying velocity—to a movement between any two adjacent cells in the fine grid. Different costs may be assigned to two edges connecting the vertices in opposite directions. Formally, we denote the grid as a directed graph $G = (V, E)$, with a cost function $C(e), \forall e \in E$. We define the *minimal Hamiltonian cycle* as follows.

Definition A *Minimal Hamiltonian cycle* (HC) of a graph $G = (V, E)$, is a Hamiltonian cycle that visits all vertices $v \in V$ and the sum of all edges in the HC is minimal.

Our objective is now to convert the directed edges of the fine grid to undirected edges in the coarse grid while preserving the properties of the edges such that a minimal spanning tree on the coarse grid will yield a minimal Hamiltonian cycle on the fine grid. Then, placing the robots in equidistant *spatio-temporal* positions along the cycle will guarantee uniform maximal frequency.

Since the patrol tour can be conducted in either clockwise (CW) or counterclockwise (CCW) directions along the spanning tree path, we divide our world to CW and CCW. In general, there are four directed edges entering and four leaving each cell in the fine grid. Since we follow some spanning tree path, the options decrease and each cell have up to two incoming and two outgoing edges in each world (CW and CCW), as described in Fig. 2. We find a minimal spanning tree in each of the worlds separately, and choose the minimal between both as base for the patrol path.

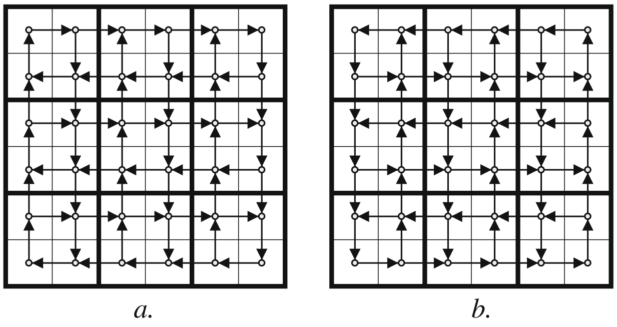

Fig. 2 Division of the area to clockwise (**a**) and counterclockwise (**b**) directions. The *graphs* are built such that the movement is suitable for traveling along a spanning tree. Union of the *two graphs* provide all possible movement options from each cell: up, down, right and left

Note that Fig. 2 clearly illustrates that the CW and CCW worlds are complementary in the following sense. First, the intersection between the worlds is empty, i.e., $\forall e \in E$ (where E is the set of G's edges), $e \in \mathsf{CW}(G)$ or $e \in \mathsf{CCW}(G)$. Second, together they provide all connections between adjacent edges in four possible directions: North, South, East, and West.

In the following, we describe a principled assignment of weights (Assignment Assign_Opt) to the undirected edges of the coarse grid based on the weights of the directed edges of the fine grid. We then argue that using this assignment, finding a minimal spanning tree on the coarse grid representation guarantees determining a minimal Hamiltonian cycle on the fine grid. In order to do that, we first prove that in our scenario, a Hamiltonian cycle is created by each spanning tree and vice versa, i.e., each Hamiltonian cycle in the fine grid is translated to a spanning tree in the coarse grid. Based on that, we then prove, in Lemma 2, that Assignment Assign_Opt yields the minimality property we seek.[1]

Assignment Assign_Opt The cost assigned to the undirected edge (u, v) in the coarse grid (see Fig. 3) is the sum of the directed edges in the fine grid, parallel to (u, v) from its two sides minus the sum of the directed edges perpendicular to (u, v) and intersecting it, or: $(a + b) - (c + d)$. Note that this can generate edges with negative cost. In this case we shift the cost of all edges by the minimal negative value, and use Kruskal's algorithm [13] for finding an MST. Note that Kruskal's algorithm finds the minimal spanning tree for a graph by adding the minimal weight edges to the spanning tree gradually, while assuring that the graph remains connected yet without cycles, hence it is appropriate also for graphs with negative weights.

Lemma 1 *Every spanning tree on the coarse grid can be translated to a Hamiltonian cycle on the fine grid and vice versa, i.e., every Hamiltonian cycle on the fine grid can be translated to a spanning tree on the coarse grid.*

[1]Gabriely and Rimon discuss edge weights in single-robot STC [18]. In their work, each edge in the coarse grid was given a different weight in order to favor movement in certain directions, and a *minimal* spanning tree (MST) was found. However, they do not discuss the correspondence of these weights to the fine grid, and minimality of the Hamiltonian cycle was not proven.

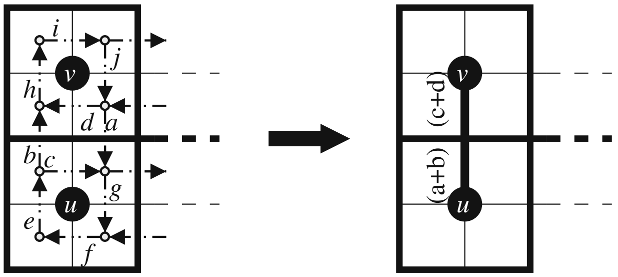

Fig. 3 The assignment of weights to the undirected edges of the coarse grid based on the directed edges of the fine grid (here in the CW direction)

Proof The first part of the lemma is shown in the initial algorithm of Gabriely and Rimon [17]. According to their algorithm, a Hamiltonian cycle is generated simply by moving along the spanning tree path in the fine grid.

In order to prove the second direction, we will first show that the existence of Hamiltonian cycle in the fine grid guarantees that only full edges are picked in the coarse grid. We choose, without loss of generality, the *CW* case. Figure 4 illustrates two adjacent vertices in the coarse grid, *u* and *v*, and their corresponding vertices in the fine grid. Denote the edge (u_2, v_1) by *a*, (v_3, v_4) by *b*, (u_2, u_4) by *d* and (v_3, v_1) by *c*. We must show that choosing edge *a* guarantees choosing also *b* and excludes *c*, *d*, and vice versa, i.e., choosing *c* forces choosing *d* and excludes *a*, *b*. Assume, towards contradiction, that a Hamiltonian cycle exists in the grid, but it uses only edge *a* and not *b*. Therefore in order to visit vertex u_4 *d* is chosen, contradicting the fact that they are all part in a Hamiltonian cycle, as u_2 cannot have two outgoing edges. The fact that *c* and *d* cannot be chosen along with *a* and *b* is proven similarly.

It is left to show that the Hamiltonian cycle on the fine grid creates a spanning tree on the coarse grid. Assume, towards contradiction, that there exists a Hamiltonian cycle that does not translate into a spanning tree on the coarse grid. This means that there exists a vertex in the coarse grid that is not covered by the spanning tree. This can happen only if not all fine vertices are visited, contradicting the fact that we have a Hamiltonian cycle. □

Corollary 2 *A Hamiltonian cycle on the fine grid can include edges from either the* CW *world or the* CCW *world, but not from both.*

Figure 5 shows an example illustrating the corollary. A closed path is a Hamiltonian cycle if it covers all vertices of the graph once, meaning that the in-degree and out-degree of vertices in this path is 1. In Fig. 5, all edges in the path

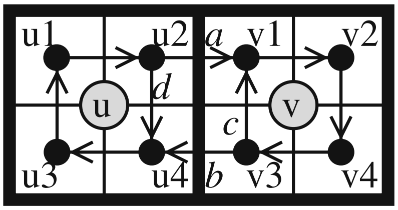

Fig. 4 Illustration of Lemma 1

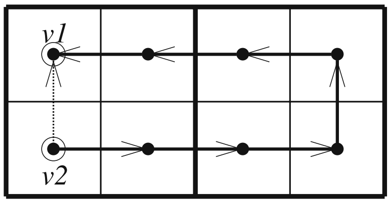

Fig. 5 Illustration of Corollary 1, demonstrating the problem of combining edges from the CW and the CCW world

are from the CCW world except for edge (v_1, v_2) (dashed), which is from the CW world. The resulting path is *not* a Hamiltonian cycle, as v_1 has in-degree 2 and v_2 has out-degree 2.

Lemma 3 *Using Assignment* Assign_Opt*, an* MST *on the coarse grid representation yields a minimal Hamiltonian cycle (*HC_Min*) on the fine grid.*

Proof Assume, towards contradiction, that there exists a Hamiltonian cycle, HC' with total weight smaller than HC_Min. This can happen in one of two scenarios.

Case 1 The spanning tree ST' (corresponding to HC') has lower cost than the MST. This contradicts the minimality of the MST, hence this case is impossible.

Case 2 The spanning tree ST' (corresponding to HC') has higher total weight than the MST's weight and still $\mathsf{HC}' < \mathsf{HC\_Min}$. Consider the case in which the trees differ by one edge, $e \in \mathsf{MST}, e \notin \mathsf{ST}'$ and $e' \in \mathsf{ST}', e' \notin \mathsf{MST}$. Denote the directed edges forming e by a, b, c, d and the directed edges forming e' by a', b', c', d' (as described in Assign_Opt). Since $\mathsf{ST}' > \mathsf{MST}$ and based on Lemma 1, it follows that weight$(e') >$ weight(e). Therefore, according to Assign_Opt, $a' + b' - (c' + d') > a + b - (c + d)$ $\Rightarrow a' + b' - (a + b) > c' + d' - (c + d)$. Since we assume that $\mathsf{HC}' < \mathsf{HC\_Min}$ and they differ only by e and e', if follows by the inclusion of e in HC' and exclusion in HC_Min that $a' + b' + c + d < a + b + c' + d' \Rightarrow a' + b' - (a + b) < c' + d' -$ $(c + d)$, leading to a contradiction.

It can be shown similarly for every spanning tree greater than the MST that this case is impossible. □

As a corollary of Lemmas 1 and 3, algorithm Generate_Cycle (Algorithm 1) finds the minimal Hamiltonian cycle over the work area. If robots move along this cycle such that their spatio-temporal positions are equidistant, they will visit all points along the path with uniform, maximal frequency. The next section examines an algorithm for moving the robots from their initial positions into such equidistant positions as quickly as possible. The algorithm relies on a call to Kruskal's algorithm for finding a minimal spanning tree [13].

Note that the construction of minimal Hamiltonian cycle for patrolling applies to the coverage problem as well. The minimal cycle as found here for non-uniform terrains can be used also for single- or multi- robot coverage, achieving minimal coverage time with respect to the constraints on the terrain, as long as the robots move in either CCW or in CW direction (as implied by the output).

Algorithm 1 Generate_Cycle

```
1: Divide the area into two CW and CCW scenarios.
2: for each scenario (CW and CCW) do
3:     Create a graph on the coarse grid by assigning weights according to Assignment Assign_Opt
4: Find a minimal spanning tree in the coarse grid using Kruskal's algorithm.
5: Calculate the total length of the Hamiltonian cycle generated by the minimal spanning tree.
6: Report scenario (CW or CCW) and cycle with shorter total length.
```

3.3 Allocating robots to initial positions

After establishing the minimal cyclic path for the patrol mission by the group of mobile robots, it is left to determine the position of the robots along the cycle from which they begin their patrol. Clearly, in order to achieve uniform frequency it is sufficient to spread the robots uniformly along the cyclic path. The distance between every two robots along the cyclic path should be the total weight of the cycle divided by the number of robots, yielding an equal distance between every two consecutive robots along the patrol path. Since there is more than one possible assignment of the robots to such positions, we want to find the assignment that requires minimal change from current positions of the robots. Therefore we describe herein the algorithm Initialization, which finds the locations from which the robots should start patrolling, while minimizing the maximal distance a robot should travel in order to arrive at its location. As the robots move simultaneously from their initial positions to their positions along the cycle, this corresponds to minimizing the time it takes all robots to be positioned and ready for the patrol mission.

We define the *Minimal Path Matching* (Min_Path_Match) problem as follows. Given a weighted graph $G = (V, E, W)$, a Hamiltonian cycle visiting all vertices in the graph, and a set of initial positions of k robots on vertices of G. Find an assignment of each robot to a position in the graph such that the following are fulfilled.

1. The distance between every two consecutive robots along the Hamiltonian cycle is equal.
2. The maximal distance traveled by a robot from its initial position to the assigned location is minimized.

We suggest the initialization algorithm Initialization (Algorithm 2) for solving the Min_Path_Match problem. The input to the algorithm includes: (1) The fine graph G; (2) the minimal Hamiltonian cycle HC found by Generate_Cycle; (3) the set of initial locations of the robots on the graph RI; and (4) BW, the smallest indivisible unit of velocity by which we measure the length of an edge. Generally, BW equals 1, unless some edges require scaling involving fractions. We define the length of a Hamiltonian cycle by len(HC).

The algorithm works as follows. First, it generates HC$'$ by separating the edges of the cyclic path into sub-edges, each of size BW (see Fig. 6. Each vertex in HC' represents an optional starting point. It then sets the initial positions of the robots

Algorithm 2 Initialization(G,HC,RI,BW)

```
1: $L \leftarrow \emptyset$ {output optimal match}
2: $min \leftarrow \infty$ {minimal match weight}
3: $\mathsf{HC}' \leftarrow$ separation of $\mathsf{HC}$ by $BW$.
4: $VL \leftarrow k$ vertices from $\mathsf{HC}'$ with distance of $\frac{\text{len}(\mathsf{HC}')}{k}$ between consecutive vertices along $\mathsf{HC}'$.
5: for all robots $r$ do
6:     Compute shortest path from $r$'s position to all vertices in $\mathsf{HC}'$.
7: for $i \leftarrow 1$ to $\frac{\text{len}(\mathsf{HC}')}{k}$ do
8:     Let $BG$ be a full bipartite graph of the two sets $RI$ and $VL$ {the weights will be based on the above shortest path calculation}
9:     $\langle ML, \text{MatchValue}\rangle \leftarrow \mathsf{PMPM}(BG)$
10:    if MatchValue $< min$ then
11:        $L \leftarrow ML$
12:        $min \leftarrow$ MatchValue
13:    $VL \leftarrow VL \oplus 1$ {update $VL$, see discussion}
14: return $L$
```

along the path such that the distance between them is $\frac{\text{len}(\mathsf{HC}')}{k}$. Then, it finds the assignment of robots to these locations such that the maximal distance traveled by a robot from its initial position to the assigned location is minimized. It does that by using procedure PMPM, and a subsequent check of the minimal maximal distance of all rotations of the positions along the cycle. It returns the positions yielding minimal maximal distance. We discuss these steps in detail below.

In step 3, algorithm Initialization creates optional starting points along the spanning tree path. In step 4 it arbitrarily selects a set of k starting points, one for each robot, with equal weights from one to the next. In steps 5 and 6, it computes the lengths of the shortest paths from each robot's current location to all other vertices in HC'. These lengths are used later on to weight the bi-partite graph used to compute the minimal time for robots to take their place.

In steps 7–13, algorithm Initialization goes through all $\frac{\text{len}(\mathsf{HC}')}{k}$ possible configurations of k starting points that are evenly spaced in HC'. For each configuration (beginning with the first configuration set in step 4), the algorithm

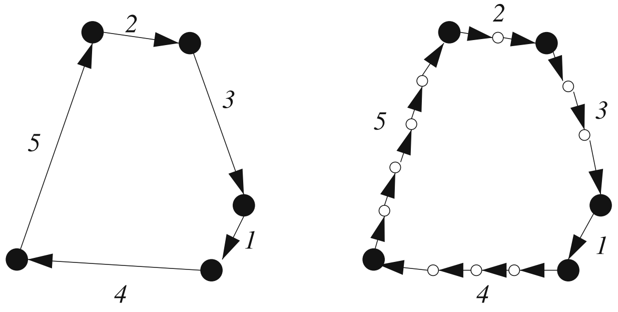

Fig. 6 On the *left*: the basic Hamiltonian cycle. On the *right*: the separated Hamiltonian cycle (in this example $BW = 1$)

Algorithm 3 Procedure PMPM(BG)

```
1: $ML \leftarrow \emptyset$
2: Let $V$ be $BG$ vertices
3: Sort the edges in $BG$
4: Construct $BG'$ from $BG$ with edge weights start from 1 and by the increased
   sorted order multiple by $\frac{|V|}{2}$ (see Fig. 7).
5: Run the Hungarian Algorithm($BG'$, $ML$)
6: Let $m$ be the largest edge weight from the match $ML$ in the corresponding $BG$
   edges.
7: Return $\langle ML, m \rangle$
```

creates a bipartite graph with k nodes on one partition (signifying the current locations of the robots in RI, and the k possible target points in VL in the other. The edges connecting the two partitions are weighted based on the lengths of the shortest paths computed in steps 5 and 6. The algorithm calls algorithm PMPM (Algorithm 3), which receives the weighted bipartite graph BG. Procedure PMPM returns a tuple: A set ML of optimal possible match considering BG, and the value the match, MatchValue. An optimal match is one in which the maximal weight of an edge in the bipartite graph is minimal, over all possible permutations of maximal edges. MatchValue is the weight of this maximal edge. The result is checked to see if it improves the current best match (steps 10–12). Then the next configuration of k points is selected in step 13. The notation $\oplus + 1$ is used here to denote the operation of updating the set VL such that each original vertex v in VL is replaced by the new vertex u, where the edge (v, u) exists in HC$'$. Note that because HC' is a circle, exactly one such edge must necessarily exist.

Procedure PMPM (Algorithm 3) uses the *Hungarian algorithm* [27] which finds a match in bipartite graphs with minimal sum of edges. As illustrated in Fig. 7, the Hungarian algorithm finds a match between r_1 and d_1 and between r_2 and d_2 since this is the minimal match sum. But in our application we would like to find the minimal largest edge from all the possible permutations. In this example, we want to match r_1 to d_2 and r_2 to d_1. In this match the maximal edge weight is 9 while in the previous it is 10. Step 4 in the PMPM algorithm construct BG' from BG. The BG' graph, by construction, causes the *Hungarian algorithm* call (in step 5) to prefer the permutation in which the maximal edge is minimal.

Lemma 4 *The construction of BG' in step* 3 *of* PMPM *assures that the Hungarian algorithm returns a match with minimal maximal edge in BG.*

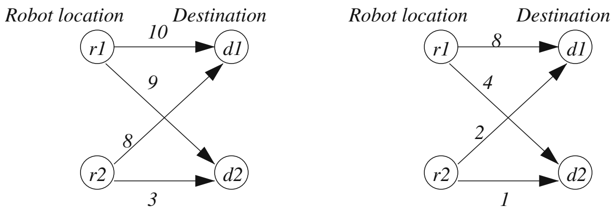

Fig. 7 Basic bipartite graph, and bipartite graph after conversion

Proof First, we prove that the construction assures the selection of a match with minimal maximal edge in BG'. For that, assume, towards contradiction, that the Hungarian algorithm returns a minimal match with sum of edges M and maximal edge m but there exists another match with sum of edges M' that has a maximal edge m' such that $m > m'$. By the construction of BG' in step 3 of PMPM it follows that any edge in BG' is greater than the sum of all edges smaller than it. Specifically, by our assumption that $m > m'$ it follows that $m > M' \Rightarrow M > M'$ contradicting the minimality of M returned by the Hungarian algorithm.

It is left to show that the match found on BG' yields a match on BG with minimal maximal edge as well. This follows directly from the fact that the order of edges remains through construction, hence minimal maximal edge in BG transforms to the minimal maximal edge in BG', and back. □

The time complexity of Procedure PMPM is as the Hungarian algorithm [27] k^3 and Algorithm Initialization runs it $\frac{|V|}{k}$ times. It also run shortest path algorithm (such as Dijkstra) k times. Thus the overall run-time complexity of Initialization is $O(KV(K + \log V))$.

3.4 Performance of STP

In this section we evaluate the performance of the patrol algorithm that is based on procedures Generate_Cycle (Algorithm 1) and Initialization (Algorithm 2). We first examine their combined performance according to the frequency optimization criteria described in Section 3.1, and then discuss their robustness to robot death failures.

3.4.1 Point-visit frequency

We prove that the combination of Procedures Generate_Cycle and Initialization guarantees optimality in *all* three point-visit frequency criteria.

Theorem 5 *The patrol algorithm which is derived by the combination of Procedures* Generate_Cycle *and* Initialization *guarantees:* (a) *Perfectly uniform frequency* (b) *Maximal average frequency* (c) *Optimal under-bounded frequency.*

Proof

(a) The first part of the Min_Path_Match problem requires the robots to be placed initially, i.e., before they begin their patrol, in uniform distance along the cyclic path. This requirement is clearly fulfilled by step 4 in Procedure Initialization, where the only positions considered are the ones where all robots are equidistant along the cyclic path. Since the robots are homogeneous and all target areas are covered by the cyclic path, their movement along the cyclic path yields a uniform frequency of $\frac{1}{\text{len}(\mathsf{HC})/k}$, where k is the number of robots and len(HC) is the total length of the minimal HC found by Generate_Cycle. Since all points are visited in this same frequency, the standard deviation is 0, and the point-visit frequency is perfectly uniform.

(b) and (c) The cyclic path found by Generate_Cycle was proven by Lemma 2 to be minimal. Therefore one robot traveling along this cycle has maximal frequency of $\frac{1}{\text{len}(\mathsf{HC})}$, hence the maximal possible frequency by k robots is $k \times \frac{1}{\text{len}(\mathsf{HC})}$, which is exactly the frequency guaranteed by our algorithm. All targets are monitored, then, exactly once every $\frac{\text{len}(\mathsf{HC})}{k}$ cycles, and by that optimal under-bounded frequency is guaranteed. Since we have proven uniform and under-bounded frequency, maximal average frequency is straightforward. □

3.4.2 Guaranteeing robustness

We use the circular path not only for assuring uniform frequency while patrolling, but for robustness as well. Specifically, we refer to robustness in the sense that as long as at least one robot remains intact, the patrol mission is guaranteed to be performed successfully (with all three optimality criteria maintained). This notion of robustness, and the basis for its existence in circular paths, has been discussed also in multi-robot coverage work [23].

Robustness is clearly guaranteed: If one robot fails the other robots simply divide the circular path again between them by re-running Procedure Initialization. Theorem 5 is, then, guaranteed for the new number of robots. In this statement we have a hidden assumption that the system is stable in the sense that the uniform, maximal-average, optimal under-bounded frequency is guaranteed as long as the system performs properly, and if a failure occurs it again guarantees the above properties after a short reorganization time. This reorganization time is the period of time necessary for the robots to execute the algorithm and arrive at their new initial positions. If all robots are to move along the cyclic path following the current direction, then this period of time will not exceed $\frac{\text{len}(\mathsf{HC})}{6}$ (see Lemma 6 for the proof). If the system is unstable, i.e., robots fail one after the other, then Theorem 5 is guaranteed for the final number of robots after stabilization.

Lemma 6 *The reorganization time required when decreasing the number of robots from* k *to* $k-1$ *is at worst the time required to travel the distance* $\frac{\text{len}(\mathsf{HC})}{6}$ *if robots follow the circular path on their way towards their new initial positions.*

Proof Consider the case in which there are three robots, and one fails. The length of the HC is divided by the three robots prior to the failure, and is divided by two after the failure. Therefore, if only one robot travels along the path, it has to travel from len $(\mathsf{HC})/3$ to len $(\mathsf{HC})/2$, which is exactly len $(\mathsf{HC})/6$. For any other k, the distance traveled is smaller: $\frac{\text{len}(\mathsf{HC})}{k-1} - \frac{\text{len}(\mathsf{HC})}{k} < \frac{\text{len}(\mathsf{HC})}{2} - \frac{\text{len}(\mathsf{HC})}{3}$ for any $k > 2$. Clearly, for $k = 2$ the remaining robot has no reorganization phase, as it simply patrols along the circular path alone. □

4 Handling events

In environments in which a team of robots is required to continuously visit target locations for various missions, it is likely that the team will need to handle events. We define an event as a change in the environment that requires special treatment by the robots for a limited period of time. For example, if the robots patrol in an area in order to clean it, then such an event could be cleaning a broken glass (in addition to the regular cleaning duties). These events are transient additions to the cost of traveling through one or more cells in the grid. Because of their transient nature, it is highly important that they will not effect the system in the following sense. (i), the frequency criteria should be maintained as much as possible for the entire area during the time the event is handled, (ii), the *recovery time*, i.e., the time it takes to stabilize the system to the situation in which the robots are dispersed with equivalent distance (in time) around the HC should be minimal.

Definition An *event* is a tuple $< l_i, t_i, T_i >$, where l_i is the location in which the event occurs, t_i is the time that a robot needs to attend and handle the event (duration of handling), and T_i is the overall time to finish handling the event (the deadline for handling the event). We assume that event can occur only in one cell, otherwise it is handled separately for each cell.

The key idea in the algorithm is to divide the time it takes to handle event e_i (t_i) uniformly between all the robots. Therefore the total time invested by each robot in handling the event is $\frac{t_i}{k}$. However, if each robot will spend $\frac{t_i}{k}$ the first time it reaches l_i, then it will starve the other cells along the cycle HC. Therefore, handling the event could be done along a number of rounds, depending on the overall time allocated for handling the event (T_i).

When an event occurs, the first step is to check if the system can handle it. Let Feasible(e_i) be a Boolean variable that represents the possibility of the system to handle event e_i, and let d_{min} be the shortest distance (in time) between any robot from the team to the event's location (l_i). Therefore,

$$\mathsf{Feasible}(e_i) = \begin{cases} \mathsf{True} & \text{if } d_{min} + t_i \leq T_i \\ \mathsf{False} & \text{otherwise} \end{cases} \tag{1}$$

In other words, if the minimal time it takes the robot closest to l_i to arrive to the event plus the time it takes to handle the event is greater than the time restriction on this event, then the system will fail to handle it.

As mentioned above, the main goal of our algorithm is to keep handling the frequency criteria of other points in the area by dividing the event between all the robots, and adding some extra cost to the cost of traveling along the HC in that location. However, there are cases in which the structure of the robots that travel along the HC is broken, i.e., one of the robots has to leave its course and travel directly to l_i. Denote the minimal distance (in time) between the location of the event (l_i) and the robot next to arrive at l_i *on its path* along the HC by d_{Nmin}. Note that always $d_{min} \leq d_{Nmin}$. Therefore we define the Boolean variable NoBreak that represents the possibility of handling the event while maintaining the movement of all robots along the original patrol path (the HC). NoBreak is defined as follows.

$$\mathsf{NoBreak}(e_i) = \begin{cases} \mathsf{True} & \text{if } d_{Nmin} + t_i \le T_i \\ \mathsf{False} & \text{otherwise} \end{cases} \quad (2)$$

The algorithm CooperateEvent (Algorithm 4) deals with an event e_i in case $\mathsf{NoBreak}(e_i) = \mathsf{True}$. The algorithm IsolatedEvent (Algorithm 6) deals with cases in which $\mathsf{NoBreak}(e_i) = \mathsf{False}$, yet $\mathsf{Feasible}(e_i) = \mathsf{True}$.

We define the success criteria for handling an event e_i as follows (priority in descending order):

a. t_i time units were invested in handling the event jointly by the team members within T_i time units, i.e., the event is handled on time.
b. Recovery time is minimized.
c. Frequency criteria is maintained throughout the patrolled area.

4.1 Handling events along the patrol path

In this section, we describe Procedures CooperateEvent and SingleRoundEvent that handles the simplest events from the robot team's perspective. These events do not require any of the robots to divert from their patrol path, and can be handled cooperatively between the robots (all of them or only part of them). This is possible for all events e_i where T_i is large enough (the event should not be handled urgently), i.e., if $\mathsf{NoBreak}(e_i) = \mathsf{True}$.

We divide this case into two sub-cases. In the first, T_i is large enough to allow t_i to be divided between the k robots. Here, Procedures CooperateEvent is executed. If this is not the case, then Procedure SingleRoundEvent is called. We first describe Procedure CooperateEvent and prove how it achieves the successful event handling criteria, and then turn to discuss Procedure SingleRoundEvent and its characteristics.

4.1.1 Procedure CooperateEvent

The key idea in algorithm CooperateEvent (Algorithm 4) is to divide the handling time of an event e_i (t_i) between all robots. First, the procedure will calculate the number of cycles along the HC that the robots will patrol while handling e_i, denoted by r_i. Then, it will find the amount of time each robot should attend the event during each cycle, denoted by x_i. Denote the case in which the event cannot be divided between all the robots by the Boolean variable NoDivision. The following formula describes the condition in which this case exists.

$$\mathsf{NoDivision}(e_i) = \begin{cases} \mathsf{True} & \text{if } \mathsf{HC}(1 - \frac{1}{k}) + d_{Nmin} + t_i > T_i \\ \mathsf{False} & \text{otherwise} \end{cases} \quad (3)$$

$\mathsf{NoDivision}(e_i)$ is true in case e_i cannot be divided between *all* the robots, and there is less than a single round to handle the event. In this case procedure SingleRoundEvent will be called.

For a given event e_i, if $\mathsf{NoDivision}(e_i)$ is false, then Procedure CooperateEvent will divide the handling time of the event (t_i) between all the robots. The procedure gives all the robots the same amount of time x_i to handle the event for each round along all r_i rounds.

Since r_i represents the number of rounds, it should be an integer. Therefore if the optimal solution requires a fraction of a round, then the number is rounded to the first integer below, and the number of time units invested by each robot during the cycles increases. This is done in order to maintain a fair division of t_i between the robots, which also leads to a recovery time of zero from handling e_i.

In case $x_i > \frac{\mathsf{HC}}{k}$, i.e., the time that each robot should invest in handling the event is greater than the distance between consecutive robots along the cycle, then a neighbor robot will arrive to the event location that currently handled by a robot. In this case, the period of time the robots will handle the event per round will be $\frac{\mathsf{HC}}{k}$ and not as calculated at the beginning of the procedure. The number of rounds will also change (see procedure CooperateEvent).

Algorithm 4 CooperateEvent($e_i < l_i, t_i, T_i >, k, HC$)

```
1: $r_i \leftarrow \lfloor \frac{T_i - t_i}{\mathsf{HC}} \rfloor$
2: $x_i \leftarrow \frac{t_i}{r_i k}$
3: if $x_i \leq \frac{\mathsf{HC}}{k}$ then
4:     for $j \leftarrow 1$ to $r_i$, for each robot $R_i$ do
5:         Move along HC until arrive at location $l_i$.
6:         Handle event for $x_i$ time units.
7: else
8:     $visitCounter = r_i x_i$
9:     while $visitCounter > 0$, for each robot $R_i$ do
10:        Move along HC until arriving at location $l_i$.
11:        $handle \leftarrow \min\left(\frac{\mathsf{HC}}{k}, visitCounter\right)$
12:        Handle event for $handle$ time frame.
13:        $visitCounter \leftarrow visitCounter - handle$
```

Procedure CooperateEvent first finds the value of r_i and x_i. Then (in line 6) it checks whether x_i is less (or equal) than the time it takes to neighbor robot (along the HC) to arrive the event location. If so, then the robot handles the event for x_i time units and proceeds its area patrol along the HC. This happens iteratively for r_i rounds. When the time x_i of handling event is greater than the time it takes a neighbor robot to arrive the event location, then at each round a robot will handle the event without interfering it neighbor robot, i.e., $\frac{\mathsf{HC}}{k}$ time units (the distance in time between two robots along the HC path). Now, the number of rounds will change to $r_i = \frac{t_i}{\mathsf{HC}_k}$.

Lemma 7 *For an event e_i, if* NoDivision(e_i) = False, *then* CooperateEvent (*Algorithm* 4) *guarantees that e_i will be handled within T_i time units, i.e., on time.*

Proof Since NoDivision(e_i) is false, then the robots can share the event and handle it during at least one round. If $x_i \leq \frac{\mathsf{HC}}{k}$ (the distance between two consecutive robots along the patrol path), then according the assignment to x_i by CooperateEvent , the robots will finish handling the event on time. If $x_i > \frac{\mathsf{HC}}{k}$ the algorithm changes x_i value to be exactly $\frac{\mathsf{HC}}{k}$. Now, a robot handles the event for a duration of $\frac{\mathsf{HC}}{k}$ per round, and the time that the event is handled per round (by all the robots) is HC: A robot handles the event, then its neighbor replaces it, and handles the event and so on. That means that the event is always being handled, for a total duration of $t_i + d_{Nmin}$. Where d_{Nmin} is the time it took to the closest robot (along the HC path) to arrive the event location and t_i is the time that should invest the event. Since NoDivision(e_i) is false, we can be sure that $t_i + d_{Nmin} \leq T_i$ which says that the robot will finish handling the event on time. □

Recall that we defined the *recovery time* from handling an event e_i as the time it takes to stabilize the system to the situation in which the robots are dispersed with equivalent distance (in time) around the HC after handling e_i. The following Lemma 8 ensures that using procedure CooperateEvent, the recovery time of the system will be zero.

Lemma 8 *The recovery time from handling event e_i using Procedure* CooperateEvent *is zero.*

Proof Since t_i is divided equally between all the robots, each robot invests the same period of time on handling e_i. Moreover, each robot handles the event for exactly the same amount of time during each round. This ensures that all robots will finish handling the event during the same round. Thus the robots will be again with the same distance (in time) between them after the last robot finishes handling the event. □

Lemma 9 *Procedure* CooperateEvent *guarantees minimal interference to the frequency of visits to other cells other than l_i, under the restriction that the recovery time is zero.*

Proof Assume, towards contradiction, that there exists a Procedure $\mathcal{B}$ different than CooperateEvent that handles the event on time and with recovery time zero, yet the robots visit a cell $c \neq l_i$ with higher frequency during the handling time of e_i. Denote by x_i^B the number of time units $\mathcal{B}$ instructs the robots to invest in handling the event, in each round. If $x_i^B > x_i$, then necessarily the frequency of visiting c decreases. Therefore $x_i^B \leq x_i$. Denote the number of rounds during which e_i is handled according to $\mathcal{B}$ by r_i^B. If $x_i^B < x_i$, then necessarily $r_i^B > r_i$. However, $r_i \leftarrow \lfloor \frac{T_i - t_i}{\mathsf{HC}} \rfloor$, which gives the maximal number of rounds possible to complete handling e_i within T_i time units if we assume all robots handles e_i uniformly. Hence if $x_i^B < x_i$, then handling e_i by $\mathcal{B}$ exceeds T_i, contradicting the assumption that $\mathcal{B}$ handles e_i on time. If $\mathcal{B}$ instructs some robots to work more than the others (for example in the last round), then the recovery time is not zero, again leading to a contradiction. Therefore $x_i = x_i^B$, i.e., $\mathcal{B} =$ CooperateEvent, leading again to a contradiction. □

By combining Lemmas 7, 8 and 9, it follows that Procedure CooperateEvent handles event e_i successfully. Therefore we turn to consider cases in which Procedure CooperateEvent cannot be used, i.e., if T_i is too small to allow cooperation between the robots.

4.1.2 Procedure SingleRoundEvent

The best possible case for the team of robots is if T_i is large enough to permit the division of t_i uniformly between the robots. However, this is not always possible. Therefore if $\mathsf{NoBreak}(e_i) = \mathsf{True}$ yet $\mathsf{Feasible} = \mathsf{True}$, then procedure SingleRoundEvent will be activated. Since $\mathsf{NoBreak}(e_i) = \mathsf{True}$, at least one robot that patrols along the HC can handle the event. If $\mathsf{NoDivision}(e_i)$ is true, then not all robots can share the event as describes in CooperateEvent procedure. Procedure SingleRoundEvent handles this situation.

In order to achieve the good frequency performance, procedure SingleRoundEvent (Algorithm 5) needs to share the event handling by maximal number of robots. Let n_i be the number of neighbor robots that will help the closest robot to the event location along the HC path to handle the event, and let d_{Nmin} be the time it takes the first robot (the closet one along the HC) to arrive the event location—while traveling along the cycle. Since $\mathsf{NoBreak}(e_i)$ is true, then also $d_{Nmin} \leq T_i - t_i$, where $T_i - t_i$ is the slack time available, allowing the event to be abandoned from the moment it occurs until its deadline.

$$d_{Nmin} + \frac{\mathsf{HC}}{k} n_i \leq T_i - t_i \tag{4}$$

$$\Rightarrow \text{maximal } n_i = \lfloor \frac{k(T_i - t_i - d_{Nmin})}{\mathsf{HC}} \rfloor$$

We would like to find the *maximal* number of neighbor robots n_i that can assist the event handling. This is formulated in Eq. 4. Now, the number of robots that will share the event will be $n_i + 1$ (the closest robot plus the n_i neighbors). The amount of time that each of the $n_i + 1$ robots will devote to the event will be $\frac{t_i}{n_i+1}$. If this calculated time is greater than $\frac{\mathsf{HC}}{k}$ then it will be adjusted to be exactly $\frac{\mathsf{HC}}{k}$ and the number of robot that will handle the event will grow to be $\frac{t_i k}{\mathsf{HC}}$ as shown in procedure SingleRoundEvent

Algorithm 5 SingleRoundEvent($e_i < l_i, t_i, T_i >, k, HC$)

```
1: $n_i \leftarrow \lfloor \frac{k(T_i - t_i - d_{Nmin})}{\mathsf{HC}} \rfloor$
2: $r_i \leftarrow n_i + 1$
3: $x_i \leftarrow \frac{t_i}{r_i}$
4: **if** $x_i > \frac{\mathsf{HC}}{k}$ **then**
5:     $x_i \leftarrow \frac{\mathsf{HC}}{k}$
6:     $r_i \leftarrow \frac{t_i k}{\mathsf{HC}}$
7: **for all** $r_i$ closest robots along HC (on the selected direction) **do**
8:     patrol along HC until arriving at location $l_i$.
9:     handle event for the duration of $x_i$.
```

Algorithm SingleRoundEvent ensures that the robots will not bump into each other, since the robots that handle the event will do so only in their path along the HC. They will not bump into their neighbor since the distance (in time) between neighbor is $\frac{\mathsf{HC}}{k}$ which is also the limit time to handle an event for a robot. The procedure also ensures that each point in the area will be patrolled optimally as before the event occurred, but with maximal delay of $\frac{\mathsf{HC}}{k}$ from the time they finish handling the event, as proven in the following lemma.

Lemma 10 *The maximal recovery time from event e_i using Procedure* SingleRoundEvent *is* $\frac{\mathsf{HC}}{k}$.

Proof Let x_i be the time that a robot handles an event e_i using procedure SingleRoundEvent. The robots that share the event are in distance (in time) x_i from the place they should be if the event did not occur. In order to recover from the event (when it is finished) all the robots that share the event should move x_i while the other robots should stop their movement before they could patrol again. In procedure SingleRoundEvent the maximal time that robot can handle event is $\frac{\mathsf{HC}}{k}$ thus the maximal recovery time is $\frac{\mathsf{HC}}{k}$. □

4.2 Handling events outside of the patrol path

Procedures SingleRoundEvent and CooperateEvent will be executed when the robots can attend the event while still patrolling along the HC path. In case where this situation is not possible, i.e., NoBreak = False, yet Feasible = True, then a robot can handle the event by breaking out of the HC path. In this case algorithm IsolatedEvent (Algorithm 6) is executed.

Algorithm IsolatedEvent finds the preferred robot to send to handle the event. This robot will move to event location, handle it and proceed the patrol when its neighbor along the HC will arrive. The neighbor will now handle the event until its neighbor arrives and so on. This procedure finishes when the event was handled for t_i time units, i.e., finished completely.

IsolatedEvent finds the preferred robot that should handle the event by calculating the minimal time it takes for robot to arrive the event location by crossing the HC. If there is more than one robot with the same minimal time to arrive at the event's location, the chosen robot will be the one with the longest shortest-path along the HC to the event's location.

Algorithm 6 Procedure IsolatedEvent($e_i < l_i, t_i, T_i >, k, HC$)

```
1:  $S \leftarrow \emptyset$
2:  **for all** robots $r$ **do**
3:      $d \leftarrow$ the shortest path from the current robot location to event location $l_i$
4:      $S \leftarrow S \cup \{\langle d, r\rangle\}$ {add the distance and associated robot to $S$}
5:  $D \leftarrow \min_d(S)$ {all tuples with minimum distance}
6:  **if** $| D |= 1$ **then**
7:      The corresponding robot $r$ will move on its (calculated) shorted path to $l_i$
8:  **else**
9:      Choose from $D$ the corresponding robot that has the longest path to move on
        its HC path.
10:     This robot will move to $l_i$
11: **repeat**
12:     The robot in $l_i$ handles the event until replaced or $t_i$ reached.
13: **until** event was handled $t_i$
```

The procedure IsolatedEvent ensures that the robots will not bump into each other when they are attending to the event. Where they are moving along the HC they cannot meet since they are in different locations. The only way that two robots can meet is when one of the robots crosses the HC to the event's location, but as shown in Lemma 11, this situation is impossible.

Lemma 11 *The robots will not bump into each other when the selected robot crosses the* HC *path to attend an event e_i using Procedure* IsolatedEvent

Proof Procedure IsolatedEvent computes for each robot the shortest path to the event location. Let R_a be the robot chosen to first handle the event by Procedure IsolatedEvent. We first assume that R_a has the minimal path to l_i (and shares this minimal value with no other robot). Assume, towards contradiction, that there is a robot R_b that will bump into R_a on its way to l_i, i.e., R_a and R_b lie at the same location along the HC at the same time. Therefore at that time the distance between R_a and l_i is equal to the distance between R_b and l_i, which leads to a contradiction (since R_a has minimal distance to l_i).

Assume now that there are several robots $R_a, R_b, \ldots \in D$ with the same shortest path length to l_i. If we choose arbitrarily some $R_a \in D$, then it could indeed possibly bump into some other member of D on its way to l_i. Thus, procedure IsolatedEvent selects the robot whose shortest path (with respect to the other robots in D) is mostly a subset of its regular path along the HC. This ensures that this robot will never bump any robot while it crossing the HC. □

Lemma 11 and algorithm IsolatedEvent deal with the robot that is sent to handle the event. However, a question remains as to what the robots do once the event is handled, to stabilize the system.

Algorithm IsolatedRecovery (Algorithm 7) describes the recovery stage, executed once the event handling procedure IsolatedEvent is triggered. Note that once a robot moves outside of the HC then necessarily, a segment of length $\frac{\mathsf{HC}}{k}$ is left without

Algorithm 7 Procedure IsolatedRecovery($e_i < l_i, t_i, T_i >, k, HC$)

```
1:  Let $r_0$ be the robot at the event location.
2:  Let $r_1$ be $r_0$'s nearest neighbour along the path HC.
3:  Set $d$ to be the distance between $r_0$ and $r_1$.
4:  if $d > \frac{\mathsf{HC}}{k}$ then
5:      move $r_0$ a distance of $d - \frac{\mathsf{HC}}{k}$.
6:  else
7:      Let $R$ be the set of robots along the HC, starting with $r_0$, such that the distance
        between them is equal or smaller than $\frac{\mathsf{HC}}{k}$.
8:      for all $r \in R$ do
9:          if $r = r_0$ then
10:             move $r$ a distance of $d$.
11:         else
12:             move $r$ a distance of $\frac{\mathsf{HC}}{k}$.
```

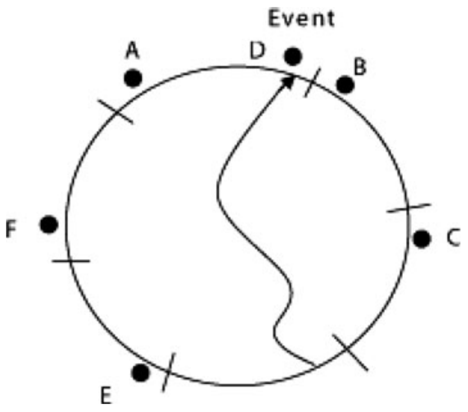

Fig. 8 The robots patrol along the HC. An event occurred, and one robot (D) crosses the HC and handles the event

an associated robot (Fig. 8). We call this segment the *empty segment*. The recovery procedure moves some of the robots along the HC to re-fill this empty segment such that again one robot is assigned to each of the k segments.

The algorithm works as follows. First, it computes the distance d between the event location (and the robot currently handling it, r_0), to its nearest neighbour r_1 along the path HC. If this distance d is greater than $\frac{\mathsf{HC}}{k}$, then the empty segment is between r_0 and r_1. In this case, r_0 needs to move until it is exactly at a distance of $\frac{\mathsf{HC}}{k}$ from r_1; it needs to move $d - \frac{\mathsf{HC}}{k}$. However, if d is smaller than or equal to $\frac{\mathsf{HC}}{k}$, the this means that there are more robots between r_0 and the empty segment. All of these robots (except for r_0 should move exactly one segment (i.e., $\frac{\mathsf{HC}}{k}$), and then r_0 should move a distance of d forward along HC, to reposition itself. Then the recovery is complete.

Lemma 12 *The maximal recovery time from procedure* IsolatedEvent *is* $\frac{\mathsf{HC}}{k}$.

The algorithm's operation is illustrated in Figs. 8, 9 and 10. Figure 8 shows the state of the system after a robot D crosses the HC path and handles the event, a decision

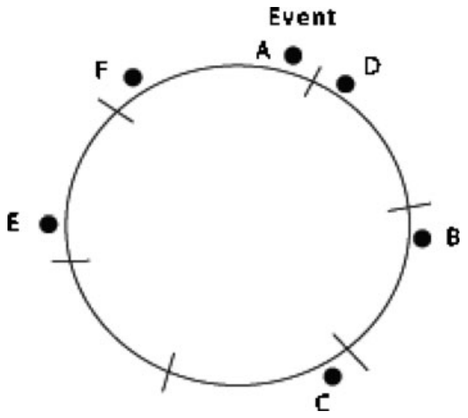

Fig. 9 The robots' location after $\frac{\mathsf{HC}}{k}$ time units. In order to recover from this event, only robots A, D, B and C will need to move, while the others wait

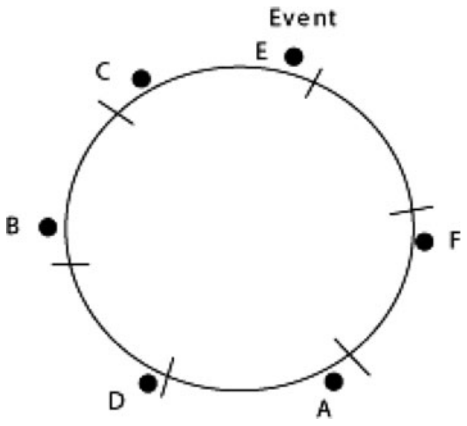

Fig. 10 The robots' location after three additional cycles, each of duration $\frac{\mathsf{HC}}{k}$. The empty segment follows the robot that handles the event. In order to recover from the event, robot E will only move to the empty space

made by IsolatedEvent. Suppose, first, some time passes, the robots move clockwise, and now handling the event is taken over by A, which completes handling the event, i.e., now recovery can commence (Fig. 9). Since the distance d here (between A and D) is smaller than $\frac{\mathsf{HC}}{k}$, A, D, B, C are in the set R (line 7). Robots D, B, and C should move ahead a distance of $\frac{\mathsf{HC}}{k}$, while robot A should move the distance of d ending up where D used to be. Then the system is stabilized and can patrol as usual.

Now suppose that A did not finish handling the event. In this case, the empty segment continues to shift around HC, until we reach the situation in Fig. 10. If E completes handling the event, then the distance d (between E and F) will be greater than $\frac{\mathsf{HC}}{k}$, which means that only E needs to move forward, a distance of $d - \frac{\mathsf{HC}}{k}$, to stabilize the system.

Given an event handled by algorithms IsolatedEvent and IsolatedRecovery, we can bound the recovery times for the system, following the decision to handle an event.

Algorithm 8 PatrolEvent($e_i < l_i, t_i, T_i >, k, HC$)

```
1: $D \leftarrow \emptyset$
2: for all robots do
3:     Compute Dijkstra's shortest path where the source is the current robot location and the destination is the event location $l_i$
4:     Add the distance to $D$
5: if $\min(D) + t_i > T_i$ then
6:     return "cannot handle event"
7: else if $d_{Nmin} + t_i > T_i$ then
8:     execute IsolatedEvent($e_i < l_i, t_i, T_i >, k, HC$)
9: else if $\mathsf{HC}(1 - \frac{1}{k}) + d_{Nmin} + t_i > T_i$ then
10:    execute SingleRoundEvent($e_i < l_i, t_i, T_i >, k, HC$)
11: else
12:    execute CooperateEvent($e_i < l_i, t_i, T_i >, k, HC$)
```

Proof When robot attend event and cross the HC there could maximum one empty segment (as shown above). The robot that attend event is far from its neighbor (along the HC) maximal $\frac{\mathsf{HC}}{k}$ or the empty segment is the neighbor. If the empty segment is the neighbor it take maximal $\frac{\mathsf{HC}}{k}$ to arrive the relative location of the other robots in the other segments as shown above. If there is neighbor robot in the next segment, the maximal distance is $\frac{\mathsf{HC}}{k}$. Since the robot that behind the robot that handle the event location should keep the same $\frac{\mathsf{HC}}{k}$ distance from the robot located in the neighbor segment. Which insure that the maximal distance for recovery is $\frac{\mathsf{HC}}{k}$. □

Summary Algorithm PatrolEvent (Algorithm 8) ties all the different conditions together, to fully address a new event and stabilize the system following its handling. The algorithm checks the values of Feasible(e_i) and NoDivision(e_i), and decides which procedure to execute for a new event. Be aware that after running procedures IsolatedEvent and SingleRoundEvent the robots needed to be ordered again along the HC, before patrolling can be resumed completely.

5 Conclusions

In this work we have formalized the area patrol problem and its frequency optimization goals. We have discussed point-visit frequency criteria according to which a patrol mission can be evaluated. We then described an spanning-tree patrolling (STP) approach for area patrolling. STP is based on finding a minimal Hamiltonian cyclic path in a non-uniform, directional, terrain. Based on this cyclic path, we have analytically demonstrated that an algorithm that assigns locations to the robots along the path such that the time necessary to arrive to those locations is minimal, and patrolling from those locations create a uniform maximal-frequency patrol. Last, we have shown that this algorithm is robust in the sense that it guarantees patrol at uniform frequency as long as at least one robot works properly.

We then turn to discuss the problem of allocation robots to handling events along the patrol path. Given the projected duration of handling an event, and the deadline by which handling the event must complete, we investigated different conditions and different methods for allocating robots to events. We described a set of algorithms for dividing the time it takes to handle the event between the robots, depending on the time constraint for finishing handling the event.

There are still several areas we plan to pursue in future work. First, we would like to take into consideration at the cycle generation phase also other aspects, for example minimizing number of turns. Second, we would like to examine the case in which the robots are heterogeneous. In addition, we would like to consider task allocation during patrol missions where the time constraint is not known, and a probability distribution is provided. Last, we wish to consider also non-uniform terrains having also prioritized requirements.

Acknowledgements We thank the anonymous reviewers for useful comments. The research in this paper was funded in part by the Israel Science Foundation, Grant #1357/07. As always, thanks to K. Ushi.

References

1. Abate, F.R.: The Oxford Dictionary and Thesaurus: the Ultimate Language Reference for American Readers. Oxford Univ. Press, Oxford (1996)
2. Agmon, N., Hazon, N., Kaminka, G.A.: Constructing spanning trees for efficient multi-robot coverage. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06) (2006)
3. Agmon, N., Hazon, N., Kaminka, G.A.: The giving tree: constructing trees for efficient offline and online multi-robot coverage. Ann. Math. Artif. Intell. **52**, 143–168 (2008)
4. Agmon, N., Kraus, S., Kaminka, G.A.: Multi-robot perimeter patrol in adversarial settings. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA-08) (2008)
5. Agmon, N., Sadov, V., Kaminka, G.A., Kraus, S.: The impact of adversarial knowledge on adversarial planning in perimeter patrol. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08), vol. 1, pp. 55–62 (2008)
6. Ahmadi, M., Stone, P.: A multi-robot system for continuous area sweeping tasks. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06) (2006)
7. Almeida, A., Ramalho, G.L., Santana, H.P., Tedesco, P., Menezes, T.R., Corruble, V., Chevaleyre, Y.: Recent advances on multi-agent patrolling. In: Advances in Artificial Intelligence SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence. Lecture Notes in Computer Science, vol. 3171, pp. 474–483. Springer, Berlin (2004)
8. Carrolla, D., Nguyena, C., Everetta, H., Frederickb, B.: Development and testing for physical security robots. In: SPIE, Orlando (2005)
9. Chevaleyre, Y.: Theoretical analysis of the multi-agent patrolling problem. In: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT) (2004)
10. Chevaleyre, Y., F. Sempé, Ramalho, G.L.: A theoretical analysis of multi-agent patrolling strategies. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04) Short Paper (2004)
11. Choset, H.: Coverage for robotics—a survey of recent results. Ann. Math. Artif. Intell. **31**, 113–126 (2001)
12. Colegrave, J., Branch, A.: A case study of autonomous household vacuum cleaner. In: AIAA/NASA CIRFFSS (1994)
13. Corman, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT, Cambridge (1990)
14. Dias, M.B., Stentz, A.: A free market architecture for distributed control of a multirobot system. In: Proceedings of the Sixth Conference on Intelligent Autonomous Systems (IAS-6), pp. 115–122 (2000)
15. Dias, M.B., Zlot, R.M., Kalra, N., Stentz, A.: Market-based multirobot coordination: a survey and analysis. Proc. IEEE **94**(7), 1257–1270 (2006)
16. Elmaliach, Y., Shiloni, A., Kaminka, G.A.: A realistic model of frequency-based multi-robot fence patrolling. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08), vol. 1, pp. 63–70 (2008)
17. Gabriely, Y., Rimon, E.: Spanning-tree based coverage of continuous areas by a mobile robot. Ann. Math. Artif. Intell. **31**, 77–98 (2001)
18. Gabriely, Y., Rimon, E.: Competitive on-line coverage of grid environments by a mobile robot. Comp. Geometry. **24**, 197–224 (2003)
19. Gage, D.W.: Command control for many-robot systems. In: The Nineteenth Annual AUVS Technical Symposium (AUVS-92) (1992)
20. Golfarelli, M., Maio, D., Rizzi, S.: A task-swap negotiation protocol based on the contract net paradigm. Technical Report 005-97, CSITE (1997)
21. Guo, Y., Parker, L., Madhavan, R.: Towards collaborative robots for infrastructure security applications. In: Proceedings of the 2004 International Symposium on Collaborative Technologies and Systems (CTS-04), pp. 235–240 (2004)
22. Guo, Y., Qu, Z.: Coverage control for a mobile robot patrolling a dynamic and uncertain environment. In: Proceedings of the Fifth World Congress on Intelligent Control and Automation (WCICA-04), vol. 6, pp. 4899–4903 (2004)
23. Hazon, N., Kaminka, G.: On redundancy, efficiency, and robustness in coverage for multiple robots. Robot. Auton. Syst. **56**, 1102–1114 (2008)
24. Hazon, N., Kaminka, G.A.: Redundancy, efficiency, and robustness in multi-robot coverage. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA-05) (2005)

25. Hedberg, S.: Robots cleaning up hazardous waste. In: AI Expert, pp. 20–24 (1995)
26. Jung, B., Sukhatme, G.: Tracking targets using multiple robots: the effect of environment occlusion. Auton. Robots. **13**(3), 191–205 (2002)
27. Kuhn, H.W.: The Hungarian method for the assignment problem. In: Naval Research Logistics Quarterly, vol. 2, pp. 83–97 (1995)
28. Machado, A., Ramalho, G., Zucker, J.-D., Drogoul, A.: Multi-agent patrolling: an empirical analysis of alternative architectures. In: Third International Workshop on Multi-Agent Based Simulation (MABS-02). Lecture Notes in Computer Science (2002)
29. Paruchuri, P., Pearce, J.P., Tambe, M., Ordonez, F., Kraus, S.: An efficient heuristic approach for security against multiple adversaries. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07) (2007)
30. Smith, R.G.: The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Trans. Comput. **C-29**(12), 1104–1113 (1981)
31. Wagner, I.A., Lindenbaum, M., Bruckstein, A.M.: Efficiently searching a graph by a smell-oriented vertex process. Ann. Math. Artif. Intell. **24**, 211–223 (1998)
32. Wagner, I.A., Lindenbaum, M., Bruckstein, A.M.: Distributed covering by ant-robots using evaporating traces. IEEE Trans. Robot. Autom. **15**(5), 918–933 (1999)
33. 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)
34. Williams, K., Burdick, J.: Multi-robot boundary coverage with plan revision. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06) (2006)
35. Yanovski, V.M., Wagner, I.A., and Bruckstein, A.M.: A distributed ant algorithm for efficiently patrolling a network. Algorithmica **37**, 165–186 (2003)