Cooperative *"Swarm Cleaning"* of Stationary Domains

1 Introduction

In this chapter we discuss the problem concerning a given (yet potentially unknown) region that needs to be *"cleaned"* by a swarm of autonomous robotic agents, namely – to have all of its 'tiles', or 'pixels' visited at least once by at least a single member of the swarm, and that this "cleaning" would be guaranteed to be completed within a finite (and as short as possible) time.

We assume that the swarm is collaborative, namely – that the various members of it are equipped with the same (and properly synchronized) software, and that each robot can acquire only the information which is available in its immediate vicinity, with the only way of inter-robot communication is by leaving traces on the common ground and sensing the traces left by other robots. Similar works can be found at [2, 21, 28, 29, 31].

In the spirit of [17], we consider simple robots with only a bounded amount of memory (i.e. *finite-state-machines*). We present a protocol for cleaning a dirty area that guarantees task completion (unless *all* robots die) and prove an upper bound on the time complexity of this protocol. Generalizing an idea from computer graphics (see [24]), we preserve the connectivity of the "dirty" region by allowing an agent to clean only so called *non-critical* points, points that do not disconnect the graph of dirty grid points. This ensures that the robots will stop only upon completing their mission.

An important advantage of this approach, in addition to the simplicity of the agents, is its fault-tolerance — even if almost all the agents cease to work before completion, the remaining ones will eventually complete the mission. We prove the correctness of the protocol as well as an upper bound on its running time, and show how our protocol can be extended for regions with obstacles. We also show simulation results of the protocol on several types of regions. These simulations indicate that

This chapter is based on work previously published in parts in [3, 9, 12].

Y. Altshuler et al., *Swarms and Network Intelligence in Search*,
Studies in Computational Intelligence 729, DOI 10.1007/978-3-319-63604-7_2

the precise cleaning time depends on the number of robots, their initial locations, and the shape of the dirty region.

This chapter is organized as follows: in Sect. 2 the problem is formally defined, as well as the aims and assumptions involved. Related work is presented in Sect. 3. The cleaning protocol is presented in Sect. 4, while an analysis of its time-complexity appears in Sect. 5. Section 6 is devoted to the problem of exploring/cleaning regions with obstacles, and is followed by several experimental examples in Sect. 7. We conclude this chapter with a discussion and by pointing out several connections to related work, presented in Sect. 8.

2 The Cooperative Cleaners Problem

Throughout this work, we shall assume that the time is discrete.

Definition 1 Let G denote a two dimensional integer grid $\mathbf{Z}^2$, whose vertices have a binary property called '*dirtiness*'. Let $dirt_t(v)$ state the dirtiness state of the vertex v at time t, taking either the value "*on*" or "*off*".

Definition 2 Let F_t be the dirty sub-graph of G at time t, i.e. $F_t = \{v \in G \mid dirt_t(v) = on\}$.

We assume that F_0 is a single simply-connected component (the actions of the agents will be so designed that this property will be preserved).

Let a group of k agents that can move on the grid G (moving from a tile to its neighbor in one time step) be placed at time t_0 on F_0, at point $p_0 \in F_t$.

Each agent is equipped with a sensor capable of telling the *dirtiness* status of all tiles in the digital sphere of diameter 7 (using the Manhattan distance function) which surrounds the agent. An agent is also aware of other agents which are located in these tiles, and all the agents agree on a common direction. Each tile may contain any number of agents simultaneously. This information will later be required by the agents' cleaning protocol. Each agent is equipped with a memory of size $O(1)$ bits w.r.t the size of the region.[1]

The agents are indistinguishable. Namely, they can be counted, but they do not contain any unique ID.

When an agent moves to a tile v, it has the possibility of cleaning this tile (i.e. causing $dirt_{(v)}$ to become *off*). The agents do not have any prior knowledge of the shape or size of the sub-graph F_0 except that it is a single and simply connected component.

The agents' goal is to clean G by eliminating the dirtiness entirely, meaning that the agents must ensure that:

$$\exists t_{success} \; s.t \; F_{t_{success}} = \emptyset$$

In addition, it is desired that time $t_{success}$ will be minimal.

[1] It should be stated that although the memory size of the agents is completely independent w.r.t to size of the problem, it is still $O(\log k)$ bits w.r.t to the number of agents (used for counting purposes).

In this work we impose the restriction of no central control and full 'decentralization', i.e. all agents are identical and no explicit communication between the agents is allowed. An important advantage of this approach, in addition to the simplicity of the agents, is fault-tolerance — even if almost all the agents cease to work before completion, the remaining ones will eventually complete the mission, if possible.

3 Related Work

As mentioned in previous sections, the cooperative cleaners problem has significant similarity to other types of multi-agents problems, such as cooperative coverage, or cooperative de-mining problems. In recent years, a lot of effort went to designing systems and algorithms for handling such tasks. In this process, various models and assumptions concerning the agents and their capabilities were used.

In general, most of the techniques used for the task of a distributed coverage use some sort of cellular decomposition. For example, in [33] the area to be covered is divided between the agents based on their relative locations. In [18] a different decomposition method is being used, which is analytically shown to guarantee a complete coverage of the area. Another interesting work is presented in [1], discussing two methods for cooperative coverage (one probabilistic and the other based on an exact cellular decomposition). All of the works mentioned above, however, rely on the assumption that the cellular decomposition of the area is possible. This in turn, requires the use of memory resources, used for storing the dynamic map generated, the boundaries of the cells, etc. As the initial size and geometric features of the area are generally not assumed to be known in advance, agents equipped with merely a constant amount of memory will most likely not be able to use such algorithms. In this work, we are interested in a multi-agents system which could perform such a cooperative coverage with a use of minimal amount of memory, unrelated to the size and geometry of the covered (or cleaned) area (this requirement is presented in Sect. 2). Such a system is presented in the later part of this work, and its ability to guarantee a completion of the task is shown.

Surprisingly, while many existing works concerning distributed (and decentralized) coverage present analytic proofs for the completion of the task (e.g. [1, 16, 18]), unfortunately, most of them lack analytic bounds for the completion time (although in many cases an extensive amount of empirical results of this nature are made available by extensive simulations). Although a proof for the coverage completion is an essential key in the design of a multi-agents system, analytic indicators for its efficiency should also be sought for. In this work, such results, bounding the cleaning time of the agents, are presented in Sect. 5. An interesting work to mention in this scope is this of [26, 35], where a swarm of ant-like robots is used for repeatedly covering an unknown area, using a real time search method called *node counting*. By using this method, the robots are shown to be able to efficiently perform such a coverage mission, and analytic bounds for the coverage time are discussed.

A different approach to a similar problem is discussed in [14], where a swarm of collaborative drones uses a network-centric approach in order to maximize the interception probability of mobile targets, that are bound to move only on a predefined network of roads.

As most coverage problems focus on achieving a complete coverage (and sometimes — in the minimal time possible), it is worth mentioning in this scope the work of [19], in which an interesting additional constraint is added. As this work was designed for an autonomous painting system, the uniformity of the coverage was demanded (in order to maintain the same thickness of the paint layer). In addition, this work examined the problem of 3-D coverage, in contrary to most search/cover systems, usually discussing the 2-D version. However, the main focus of this work was the use of an efficient single agent for this mission, instead of a cooperative multi-agents one.

4 The Cleaning Protocol

In order to solve the *Cooperative Cleaners* problem we propose a cleaning protocol, called **SWEEP**. Generalizing an idea from computer graphics (presented in [24]), this protocol preserves the connectivity of the *dirty* region by preventing the agents from cleaning *critical points* — points which when cleaned disconnect the dirty region (see Sect. 4.1). This ensures that the agents stop only upon completing their mission.

At each time step, each agent cleans its current location (assuming it is not a critical point), and moves according to a local movement rule, creating the effect of a clockwise traversal along the boundary of the dirty region. As a result, the agents "peel" layers from the region, while preserving its connectivity, until the region is cleaned entirely. An illustration of two agents working according to the protocol can be seen in Fig. 1.

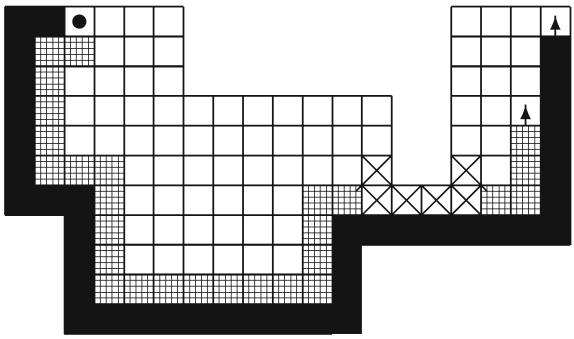

Fig. 1 An example of two agents using the **SWEEP** protocol, at time step 40. All the tiles presented were dirty at time 0. The *black dot* denotes the starting point of the agents. The X's mark the *critical points* which are not cleaned. The *black tiles* are the tiles cleaned by the first agent. The second layer of marked tiles represent the tiles cleaned by the second agent

To the basic description of the protocol given above, there are several exceptions. As the agents are equipped with very limited sensing and storage capabilities, the basic structure of the protocol must be enhanced with a set of local rules, designed for producing a pseudo-synchronization between the agents. Those rules generate *resting* and *waiting* commands for the agents, capable of delaying their actions, either within the time step (until certain agents complete their cleaning process for this time step), or causing them to pause for a single time step, resuming their operation at the next time step only. A detailed description concerning the need for these additions appears in Sects. 4.6 and 4.7. Note however that the rules in charge of the *resting* and *waiting* still obey the basic paradigm of this work, namely — they are local, use no prior knowledge of the agents, and can be implemented using extremely small amount of memory resources (namely — $O(\log k)$, k being the number of agents). A schematic flowchart of the protocol is presented in Fig. 3.

4.1 Cleaning Protocol — Definitions and Requirements

Definition 3 Let $\tau(t) = \big(\tau_1(t), \tau_2(t), \ldots, \tau_k(t)\big)$ denote the locations of the k agents at time t.

Definition 4 For a tile v, let $Neighborhood(v)$ denote the dirtiness states of v and its $8Neighbors$.

Let $\mathcal{M}_i$ denote some finite amount of memory contained in agent i, storing information needed for the protocol (e.g. the last moves of agent i). The requested cleaning protocol is therefore a rule f such that for every agent i:

$$f\big(\tau_i(t), Neighborhood(\tau_i(t)), \mathcal{M}_i(t)\big) \in \mathcal{D}$$

where $\mathcal{D} = \{\text{`}left\text{'}, \text{`}right\text{'}, \text{`}up\text{'}, \text{`}down\text{'}\}$.

Definition 5 Let ∂F denote the boundary of F. A tile is on the boundary if and only if at least one of its $8Neighbors$ is not in F, meaning:

$$\partial F = \{v \mid v \in F \ \wedge \ 8Neighbors(v) \ \cap \ (G \ \setminus \ F) \ \neq \ \emptyset\}$$

The requested rule f should meet the following goals:

- **Successful Termination**: $(\exists t_{success} \ s.t \ F_{t_{success}} = \emptyset)$.
- **Agreement on Completion**: within a finite time after completing the mission, all the agents must halt.
- **Efficiency**: the cleaning process should be efficient at time and in agents' memory resources.
- **Fault Tolerance**: if one or several stop working ("die") the rest of the agents will continue the cleaning process as efficiently as their number allows them.

4.2 The SWEEP Cleaning Protocol

The **SWEEP** protocol is implemented by each agent a_i, located at time t at $\tau_i(t) = (x, y)$. We define below several terms we use while discussing the **SWEEP** protocol. We stress the fact that this is indeed a *myopic* protocol, relying on neighborhood information, a *senile* protocol (the memory needed is constant + one counter whose size is upper bounded by $O(\log k)$, where k is the number of agents) and relies on *implicit local communication* only.

Definition 6 Let $\tilde{\tau}_i(t)$ denote the previous location of agent i with respect to $\tau_i(t)$, such that $\tilde{\tau}_i(t) \neq \tau_i(t)$, defined as:

$$\tilde{\tau}_i(t) \triangleq \tau_i(x) \; s.t. \; x = \max\{j \in \mathbb{N} \mid j < t \text{ and } \tau_i(j) \neq \tau_i(t)\}$$

Definition 7 The term '*rightmost*' means:

Starting from $\tilde{\tau}_i(t)$ (namely, the previous boundary tile that agent a_i had been in) scan the *four neighbors* of $\tau_i(t)$ in a clockwise order until a boundary tile (excluding $\tilde{\tau}_i(t)$) is found. Sometimes, $\tilde{\tau}_i(t)$ might not be a boundary tile (this may occasionally happen after a contamination spread, for dirty regions of specific geometric features. In those cases, an agent may find itself in an *internal point* — a dirty tile which is not located on the boundary. From this tile the agent must move to the adjacent boundary tile, and resume its traversal along the region's boundary). In this case, if $\tau_i(t)$ is a boundary point, then starting from $\tilde{\tau}_i(t)$ scan the *four neighbors* of $\tau_i(t)$ in a clockwise order until the second boundary tile is found. In case $t = 0$ select the tile as instructed in Fig. 2.

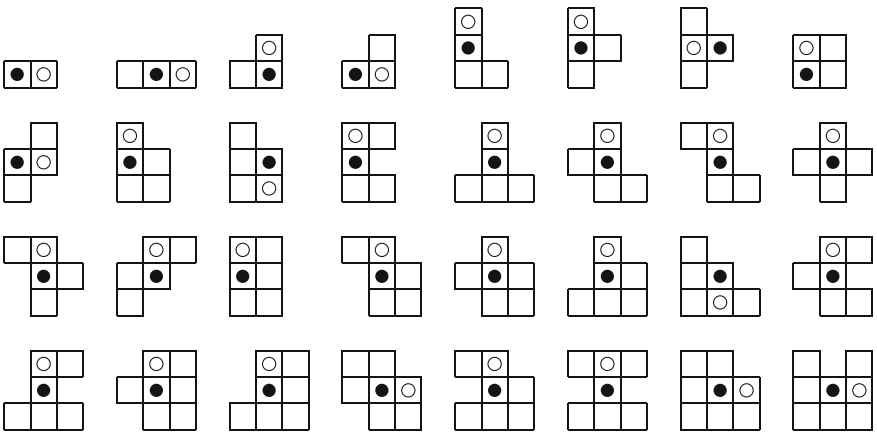

Fig. 2 When $t = 0$ the first movement of an agent located in (x, y) should be decided according to initial contamination status of the neighbors of (x, y), as appears in these charts — the agent's initial location is marked with a *filled circle* while the destination is marked with an *empty one*. All configurations which do not appear in these charts can be obtained by using rotations. This definition is needed in order to initialize the traversal behavior of the agents in the correct direction

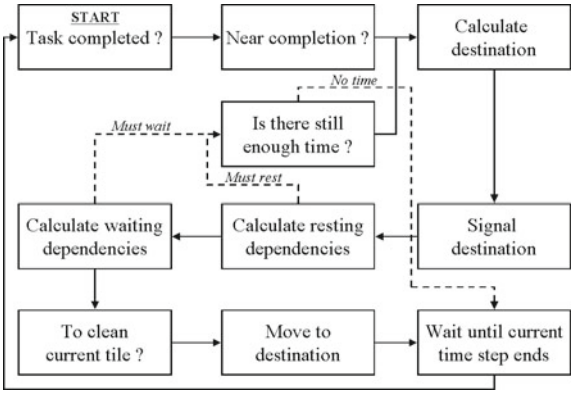

Fig. 3 A schematic flow chart of the **SWEEP** protocol. The *smooth lines* represent the basic flow of the protocol while the *dashed lines* represent cases in which the flow is interrupted. Such interruptions occur when an agent calculates that it must not move until other agents do so (either as a result of *waiting* or *resting* dependencies — see Lines 6 and 19 of **SWEEP** for more details)

The reference to *contamination spread* used in Definition 7 is given with consideration to the use of this term in the next chapters.

The additional information needed for the protocol and its sub-routines is contained in $\mathcal{M}_i$ and $Neighborhood(x, y)$.

A schematic flowchart of the protocol, describing its major components and procedures is presented in Fig. 3. The complete pseudo-code of the protocol and its sub-routines appears in Figs. 4 and 5. Upon initialization of the system, the *System Initialization* procedure is called (defined in Fig. 4). This procedure sets various initial values of the agents, and call the protocol's main procedure — *SWEEP* (defined in Fig. 5). This procedure in turn, uses various sub-routines and functions, all defined in Fig. 4.

4.3 Pivot Point

The initial location of the agents (the *pivot point*, denoted as p_0) is artificially set to be critical during the execution, hence it is also guaranteed to be the last point cleaned. Completion of the mission can therefore be concluded from the fact that all (working) agents are back at p_0 with no contaminated neighbors to go to, thereby reporting on completion of their individual missions. Note that this artificial preservation of the criticalness of the pivot point is not necessary for the algorithm to work. Rather, it just makes life easier for the user, as one can now know where to find the agents after the cleaning has been completed. If we do not start with all agents at the pivot and force p_0 to be critical, the location of the agents upon completion will generally not be known in advance.

System Initialization
    Arbitrarily choose a *pivot point* p_0 in ∂F_0, and mark it as *critical point*
    Place all the agents on p_0
    For $(i = 1; i \leq k; i++)$ do
        Call **Agent Reset** for agent i
        Call **SWEEP** for agent i
        Wait two time steps
    End for
End procedure

Agent Reset
    resting $\leftarrow$ *false*
    dest $\leftarrow$ null /* destination */
    near completion $\leftarrow$ *false*
    saturated perimeter $\leftarrow$ *false*
    waiting $\leftarrow \emptyset$
End procedure

Priority
    /* Assuming the agent moved from (x_0, y_0) to (x_1, y_1) */
    priority $\leftarrow 2(x_1 - x_0) + (y_1 - y_0)$
End procedure

Check Completion of Mission
    If $((x, y) = p_0)$ and (x, y) has no dirty neighbors then
        If (x, y) is dirty then
            Clean (x, y)
        STOP
End procedure

Check "Near Completion" of Mission
    /* Cases where every tile in F_t contains at least a single agent */
    near completion $\leftarrow$ *false*
    If each of the dirty neighbors of (x, y) contains at least one agent then
        near completion $\leftarrow$ *true*
    If each of the dirty neighbors of (x, y) has *near completion* = *true* then
        Clean (x, y) and **STOP**
    /* Cases where every non-critical tile in ∂F_t contains at least 2 agents */
    saturated perimeter $\leftarrow$ *false*
    If $((x, y) \in \partial F_t)$ and both (x, y) and all of its non-critical neighbors in ∂F_t contain at least two agents then
        saturated perimeter $\leftarrow$ *true*
    If $((x, y) \in \partial F_t)$ and both (x, y) and all of its neighbors in ∂F_t has *saturated perimeter* = *true* then
        Ignore *resting* commands for this time step
End procedure

Fig. 4 The first part of the **SWEEP** cleaning protocol. The terms ∂F and $\tau_i = (x, y)$ are defined in Sects. 4.1 and 4.2

4.4 Signaling

Since by assumption the agent's sensors can detect the status of all tiles which are contained within a digital sphere of radius four placed around the current location of the agent, each agent can artificially calculate the desired destination of all the agents

1: **SWEEP Protocol** /* Controls agent i after **Agent Reset** */
2: **Check Completion of Mission**
3: **Check "Near Completion" of Mission**
4: *dest* $\leftarrow$ *rightmost neighbor* of (x,y) /* Calculate destination */
5: *destination signal bits* $\leftarrow$ *dest* /* Signaling the desired destination */
6: /* Calculate resting dependencies (solves agents' clustering problem) */
7: Let all agents in (x,y) except agent i be divided to the following groups :
8: A_1 : Agents signaling towards any direction different than *dest*
9: A_2 : Agents signaling towards *dest* which entered (x,y) before agent i
10: A_3 : Agents signaling towards *dest* which entered (x,y) after agent i
11: A_4 : Agents signaling towards *dest* which entered (x,y) with agent i
12: Let group A_4 be divided into the following two groups :
13: A_{4a} : Agents with lower *priority* than this of agent i
14: A_{4b} : Agents with higher *priority* than this of agent i
15: *resting* $\leftarrow$ *false*
16: **If** ($A_2 \neq \emptyset$) or ($A_{4b} \neq \emptyset$) then
17: *resting* $\leftarrow$ *true*
18: **If** (current time-step $\mathcal{T}$ did not end yet) then jump to 4 **Else** jump to 35
19: *waiting* $\leftarrow \emptyset$ /* Waiting dependencies (agents synchronization) */
20: Let *active agent* denote a *non-resting* agent which didn't move in $\mathcal{T}$ yet
21: **If** (x-1,y) $\in F_t$ contains an active agent then *waiting* $\leftarrow$ *waiting* $\cup \{left\}$
22: **If** (x,y-1) $\in F_t$ contains an active agent then *waiting* $\leftarrow$ *waiting* $\cup \{down\}$
23: **If** (x-1,y-1) $\in F_t$ contains an active agent then *waiting* $\leftarrow$ *waiting* $\cup \{l\text{-}d\}$
24: **If** (x+1,y-1) $\in F_t$ contains an active agent then *waiting* $\leftarrow$ *waiting* $\cup \{r\text{-}d\}$
25: **If** *dest* = *right* and (x+1,y) contains an active agent j, and $dest_j \neq left$, and there are no other agents delayed by agent i (i.e. (x-1,y) does not contain active agent l with $dest_l$ =*right* and no active agents in (x,y+1),(x+1,y+1), (x-1,y+1), and (x+1,y) does not contain active agent n with $dest_n$ = *left*), then (*waiting* $\leftarrow$ *waiting* $\cup \{right\}$) and $\left(waiting_j \leftarrow waiting_j \setminus \{left\}\right)$
26: **If** *dest* = *up* and (x,y+1) contains an active agent j, and $dest_j \neq down$, and there are no other agents delayed by agent i (i.e. (x,y-1) does not contain active agent l with $dest_l$ =*up* and no active agents in (x+1,y),(x+1,y+1), (x-1,y+1), and (x,y+1) does not contain active agent n with $dest_n$ = *down*), then (*waiting* $\leftarrow$ *waiting* $\cup \{up\}$) and $\left(waiting_j \leftarrow waiting_j \setminus \{down\}\right)$
27: **If** (*waiting* $\neq \emptyset$) then
28: **If** ($\mathcal{T}$ has not ended yet) then jump to 4 **Else** jump to 35
29: /* Decide whether or not (x,y) should be cleaned */
30: **If** $\neg$ ((x,y) $\in \partial F_t$) or ((x,y) $\equiv p_0$) or (x,y) has 2 dirty tiles in its $4Neighbors$ which are not connected via a path of dirty tiles from its $8Neighbors$ then
31: (x,y) is an *internal point* or a *critical point* and should not be cleaned
32: **Else**
33: Clean (x,y) if and only if it does not still contain other agents
34: Move to *dest*
35: Wait until $\mathcal{T}$ ends.
36: Return to 2

Fig. 5 The **SWEEP** cleaning protocol. The term *rightmost neighbor* is defined in Sect. 4.2. *l-d* and *r-d* are left-down and right-down, respectively

which are located in one of its $4Neighbors$ tiles (see Fig. 6). Thus, the *signaling* action of each agent can be simulated by the other agents near him, and hence an explicit signaling by the agents is not actually required. However, the signaling action is kept in the description and flowchart of the protocol (in procedure 5) for the sake of simplicity and understandability.

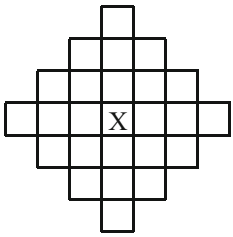

Fig. 6 Digital sphere of diameter 7, placed around the agent

4.5 Connectivity Preservation

The connectivity of the region yet to be cleaned, F, is preserved by allowing the agents to clean only *non-critical* points. This guarantees both successful termination and agreement of completion (since having no dirty neighbors implies that $F = \emptyset$). Also, should several agents malfunction and halt, as long as there are still functioning agents, the mission will still be carried out, albeit slower.

Note that a tile which was originally clean, or which was cleaned by the agents, is guaranteed to remain clean. As a tile can be cleaned by the agents only when it is in ∂F, it is easy to see that the simple connectivity of F is maintained throughout the cleaning process (as the creation of "holes" — clean tiles which are surrounded by tiles of F — is impossible).

Note that when several agents are located in the same tile, only the last one who exits cleans it (see Line 33 in **SWEEP**), in order to prevent agents from being 'thrown' out of the dirty region (meaning, cleaning a tile in which another agent is located and thus preventing this agent from being able to continue the execution of the cleaning protocol). An alternative method for ensuring the agents are always capable of executing the cleaning protocol would have been the implementation of a "dirtiness seeking mechanism" (i.e. by applying methods such as suggested in [15]). Such a mechanism would have allowed an agent to clean its current location, even if other agents had also been present there. That solution, however, would have been far less elegant and would have added additional difficulties to the analysis process.

4.6 Agents Synchronization

Note that agents operating in the described environment must have some mean of synchronization, which is mandatory in order to prevent agents from operating in the same time — risking to cut the dirty region into several connected components, as shown in Fig. 7.

To ensure such scenarios will not occur, an order between the operating agents must be implemented. Note — throughout the next paragraphs agents which are signaling a *resting* status (see Sect. 4.7 for more details) are being disregarded while calculating the dependencies of the agents' movements. The creation of the following order is implemented in Procedure 19 of **SWEEP**:

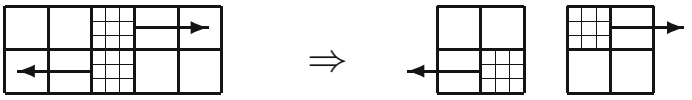

Fig. 7 When the agents do not possess a synchronization mechanism, they may, among others, damage the region's connectivity. In this example, two agents clean their current locations, and move according to the **SWEEP** protocol. Since they are not synchronized, the tiles which they are located in are not treated as critical at the time of cleaning. However, the region's connectivity is not preserved. Should one of the agents had waited for its neighbor to complete executing the protocol's steps before resuming its actions, while deciding whether to clean its current location, it would have treated this tile as critical, and therefore avoid cleaning it. In this case, the connectivity of the region would have been maintained

Definition 8 For agent i, let $P_i \subseteq \{up, down, left, right, right\text{-}down, left\text{-}down\}$ be a set of directions of tiles, in which there are currently agents, which agent i is delayed by (meaning, agent i will not start moving until the agents in these tiles move). Unless stated otherwise, $P_i = \emptyset$.

For agent i which is located in tile (x, y), if $(x-1, y)$ is a tile in F, which contains an agent, then $P_i \leftarrow P_i \cup \{left\}$. If $(x, y-1)$ is a tile in F, which contains an agent, then $P_i \leftarrow P_i \cup \{down\}$, and similarly for $(x+1, y-1)$ and *right-down* and for $(x-1, y-1)$ and *left-down*.

Definition 9 Let $dest_i \in \{up, down, left, right\}$ be the destination agent i might be interested in moving to, after leaving its current location.

Let each agent be equipped with a two bit flag (that may be implemented for example by two small light-bulbs). This flag states the desired destination of the agent. Alternatively, each tile can be treated as a physical tile, in which the agent can move. Thus, the agent can move towards the top side of the tile, which will be equivalent for using the flag in order to signal that it intends to move up.

Let each time step be divided into two phases. In phase 1, every agent "signals" the destination it intends to move towards, either by moving to the appropriate side of the tile, or by using the destination flag.

As we defined an artificial rule which states the superiority of $left$ and $down$ over $right$ and up (and internally, of $down$ over $left$), there are several specific scenarios in which this asymmetry should be reversed in order to ensure a proper operation of the agents. Following is such a "dependencies switching" rule: For agent i, which is located in (x, y), if $dest_i = right$ and tile $(x+1, y)$ contains an agent j, and $dest_j \neq left$, and there are no other agents which are delayed by agent i (i.e. tile $(x-1, y)$ does not contain an agent l where $dest_l = right$ and tiles $(x, y+1)$, $(x+1, y+1)$, $(x-1, y+1)$ do not contain any agent and tile $(x+1, y)$ does not contain an agent n where $dest_n = left$), then $P_i \leftarrow P_i \cup \{right\}$ and $P_j \leftarrow P_j \setminus \{left\}$. Also, if $dest_i = up$ and tile $(x, y+1)$ contains an agent k, and $dest_k \neq down$, and there are no other agents which are delayed by agent i (i.e. tile $(x, y-1)$ does not contain

an agent m where $dest_m = up$ and tiles $(x+1, y)$, $(x+1, y+1)$, $(x-1, y+1)$ do not contain any agent and tile $(x, y+1)$ does not contain an agent q where $dest_q = down$), then $P_i \leftarrow P_i \cup \{up\}$ and $P_k \leftarrow P_k \setminus \{down\}$.

At phase 2 of each time step, the agents start to operate in turns, according to the order implied by P_i. This guarantees that the connectivity of the region is kept, since the simultaneous movement of two neighboring agents is prevented.

Notice that deadlocks are impossible — since the basic rule is that every agent is delayed by the agents in its $left$ and $down$ neighbor tiles. Therefore, at any given time, and for every possible group of agents, there exist an agent with the minimal x and y coordinates (which by definition, is not delayed by any other agent of this group). After this agent moves, all the agents which are delayed by it, can now move, and so on. As to the "dependencies switching rule" — let agent i located in tile (x, y) have the minimal x and y values among the agents who had not moved yet, and let $dest_i = up$, and let tile $(x, y+1)$ contain an agent j such that $dest_j \neq down$. Then although agent i is located below agent j, it will be delayed by it (i.e. $(up \in P_i)$ and $\neq (down \in P_j)$) as long as agent i is not delaying any other agent (as this is the requirement of the "dependencies switching rule"). In this case, we should show that there can not be a cycle of dependencies, which starts at agent j, ends at agent i, and is closed by the dependency of agent i on agent j. Such a circle can not exist since for it to end in agent i, it means that agent i is delaying another agent k. However, this is impossible since agent i is known not to delay any agent (specifically agent k). Hence, circular dependencies are prevented and no deadlock are possible.

Note that while phase 2 is in process, F_t may change due to cleaning by the agents. As a result, the desired destinations of the agents as well as their dependencies forest, must also be dynamic. This is achieved through the repeated recalculation of these values, for every $waiting$ agent. For example, assume agent i to be located in (x, y), and $dest_i = down$ without loss of generality, and let agent j located in tile $(x, y-1)$ moves out of this tile and cleans it. Then, $dest_i$ naturally change (as the tile (x, y) does no longer belong to F_t, and thus is not a legitimate destination for agent i), and thus, P_i may also change. In this case, agent i should change its "destination signal" and act according to its new $dest_i$ and P_i. This is implemented in **SWEEP** by calculating the waiting agents' destinations and dependencies lists repeatedly, until either all agents have moved or until the time step has ended (meaning that some agents had to change their status to $resting$, and pause until the next time step — see Sect. 4.7 for more details). Note that every $waiting$ agent is guaranteed either to complete its movement in the current time step, or to be forced to wait for the next time step, by switching its status to $resting$.

Notice that the "dependencies switching rule" is not required in order to ensure a proper completion of the mission, but rather to improve the agents' performance, by preventing a bug demonstrated in Fig. 8.

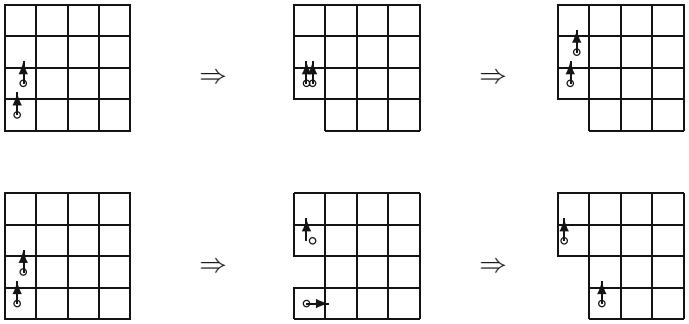

Fig. 8 The *upper charts* demonstrate a performance bug which may be caused due to local dependencies. The agents are advancing according to the **SWEEP** protocol, but their cleaning performance is decreased. The *lower charts* demonstrates the cleaning operation after adding the dependencies switching mechanism

4.7 Clustering Problem

Since we are interested in preventing the agents from moving together and functioning as a single agent (due to a resulting decrease in the system's performance), we would like to impose a "resting policy" which will ensure that the agents do not group into clusters which move together. This is done by artificially delaying agents in their current locations, should several agents placed in the same tile wish to move to the same destination. However, we would like the resting time of the agents to be minimal, for performance and analysis reasons.

The following resting policy is implemented by Line 6 of **SWEEP**. Using its sensors, an agent intending to move into tile v is aware of other agents which intend to move into v as well. Thus, when an agent enters a tile which already contains other agents, it knows whether these agents entered this tile at the same time step it did, or whether they had occupied this tile before the current time step had started.

Note that in phase 1 of each time step all the agents are signaling their desired destinations. Thus, an agent can identify which ones of the agents which are located in its current tile intend to move to the same destination as its. Only those agents may cause this agent to delay its actions and rest.

From the agents which intend to move to the same direction as agent i, the agent can distinguish between three groups of agents — the agents which entered this tile before the time step agent i did (group A_2), those which entered the tile in the same time step as i (group A_4), and those who entered it after agent i did (group A_3). If $A_2 \neq \emptyset$, agent i waits, change its status to *resting agent*, signals this status to the other agents in its vicinity, and does not move. As this rule is kept by every other agent, agent i is guaranteed that the agents of group A_3 in their turn will wait for agent i moves before being free to do so as well.

As to the agents of A_4, the problem is solved by induction over the time steps, namely — that at any given time, two agents can not leave the same tile towards the same direction at the same time step. The base of this induction holds for small values of t as the agents are periodically released by the initialization procedure of **SWEEP**, while no two agents are released at the same time step. Later, as we are assured that all the agents of group A_4 arrived to v from different tiles (which are also different than the tile agent i had entered v from), then since all the agents in group A_4 know the previous locations of each other, a consensus over a local ordering of A_4 is established (note that no explicit communication is needed to form this ordering). An example for such an order is the **Priority** function of the **SWEEP** protocol. As a result, the agents are able to exit the tile they are currently located in, in an orderly fashion, according to a well defined order. Thus, the following invariant holds: "*at any given time t, for any two tiles v, u, there can only be a single agent which moves from v to u at time step t*". Thus, the clustering problem is solved.

Notice that there is a single exception to this mechanism, in which the resting commands are overruled. This happens when all non-critical perimeter tiles contain at least two agents. In this scenario, following the resting commands would have created a situation in which no tile can be cleaned, as for every agent which leaves a certain tile, there are still "resting" agents located in the same tile. Therefore, once this scenario is detected by the *check near completion of mission* procedure of **SWEEP**, the agents ignore their resting commands momentarily, in order to be able to clean the non-critical perimeter tiles. The order and internal prioritization of the agents are maintained, for calculating the new *resting* commands in the following time step.

4.8 Mission Termination

The termination of the protocol is done in one of two cases — either an agent finds itself in the pivot point, while all of its neighbors are clean (which means that this is the last dirty tile and therefore should be cleaned), or when each dirty tile contains at least one agent (which is a generalization of the previous scenario). The second case is implemented by allowing the agents to signal to their four neighbors whether all of their dirty neighbors contain at least a single agent.

5 Analysis

The introduction of the notion of critical points makes time-complexity analysis of the **SWEEP** protocol significantly harder since a critical point may be visited several times before it is cleaned. We conjecture that the protocol proposed is efficient, and additional agents will speed up the cleaning process only up to a limit. In order to give this argument a rigorous form, some definitions are required.

5.1 Definitions

Definition 10 Let S_t denote the size of the dirty region F at time t, namely the number of grid points (or tiles) in F_t.

Actually, F defines a dichotomy of $\mathbf{Z}^2$ into F and $\overline{F} = \mathbf{Z}^2 \setminus F$.

Recalling Definition 5, the boundary of the dirty region F is denoted as ∂F and defined as:

$$\partial F = \{(x, y) \mid (x, y) \in F \ \wedge \ (x, y) \text{ has an } 8Neighbor \text{ in } (G \setminus F)\}$$

Definition 11 A *path* in F is defined to be a sequence $(v_0, v_1, \ldots, v_n)$ of tiles in F such that every two consecutive tiles are 4 *connected* (the *Manhattan* distance between them is 1). The *length* of a *path* is defined to be the number of tiles in it.

Definition 12 Let tile v be called a *critical point* if there exist $v_1, v_2 \in 4$ $Neighbors(v)$ for which all paths connecting v_1 and v_2, included in $8Neighbors(v)$, necessarily pass through v (where v, v_1, v_2 and all said paths are in F).

Definition 13 We shall denote by c_t the circumference of F at time t, defined as follows: let v_0 and v_n be two 4 *connected* tiles in ∂F_t, and let $C_t = (v_0, v_1, \ldots, v_n)$ be a shortest *path* connecting v_0 and v_n, which contains all the tiles of ∂F_t and only such tiles. Notice that C_t may contain several instances of the same tile, if this tile is a *critical point* (meaning that C_t is an ordering of the tiles of ∂F_t, in which multiple instances of tiles that are *critical points* are allowed). c_t will be defined as the length of C_t. An example appears in Fig. 9.

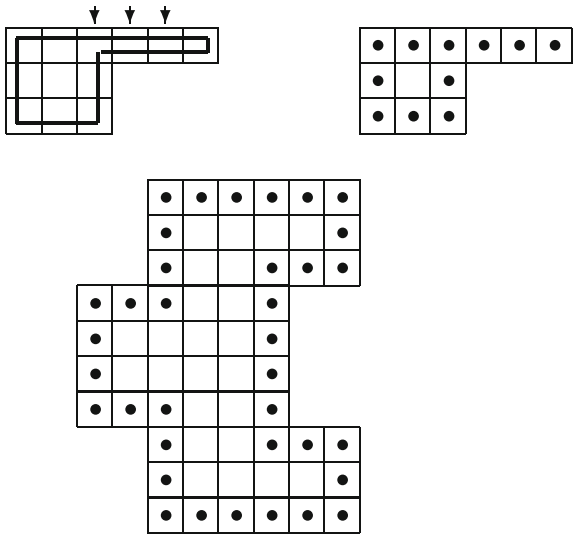

Fig. 9 The *line* in the *left upper chart* goes through the tiles of C_t where the 3 *arrows* denote tiles that are included twice. The *circles* in the *right upper chart* denote the tiles of ∂F_t. Note that while ∂F_t contains 11 tiles, C_t contains 14. In the *lower chart*, there are no *critical points* in ∂F_t and therefore $c_t = |\partial F_t| = |C_t| = 36$

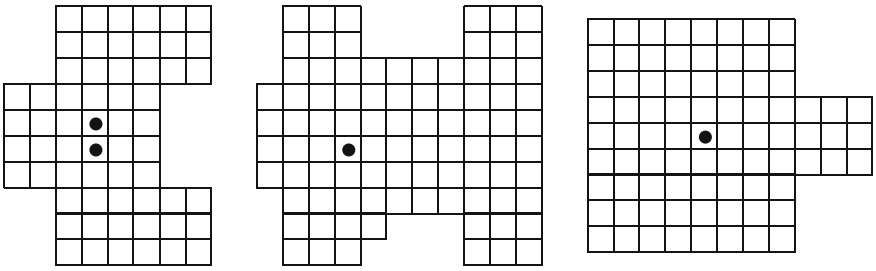

Fig. 10 The chart contains three regions. The *black circles* denote the deepest tiles within the regions. Notice that while in the left region there are two tiles whose depth is the maximal, in the other regions there is only one tile of maximal depth. The widths of the regions equal the depths of the deepest tiles, and are (from *left* to *right*) 2, 3, and 4

Definition 14 For some $v \in F_t$ let $Strings_t(v)$ denote the set of all *paths* in F_t that begin in v and end at any *non-critical point* in ∂F_t, and let $w(F_t, v)$ denote the *depth* of v — the length of the shortest *path* in $Strings_t(v)$ (unless v is a *critical point* in which case its *depth* is defined to be zero).

Definition 15 Let $W(F_t)$ denote the *width* of F_t, defined as the maximal depth of all the tiles in F_t, i.e.:

$$W(F_t) = \max\{w(F_t, v) \mid v \in F_t\}$$

An example appears in Fig. 10.

Definition 16 Let F_t' denote the region one could get, having one agent traverse F_t once, using the **SWEEP** protocol (i.e. when $k = 1$, $F_t' = F_{t+c_t}$). For the sake of simplicity, F_t'' will be used instead of $(F_t')'$, and so on.

Definition 17 The longest in-region distance between p_0 and any other point in F_t will be referred to as $l(F_t)$, the *length* of F_t:

$$l(F_t) = \max_{v \in F_t} \left\{ d_{F_t}(p_0, v) \right\}$$

where $d_{F_t}(x, y)$ is the distance (shortest path within F_t) between x and y.

5.2 Correctness

Lemma 1 *If $S_t > 0$ then $W(F_t) \geq 1$, that is: any finite simple region has at least two non-critical points on its boundary.*

Proof The boundary ∂F_t is a connected graph, hence it has a spanning tree. In such a tree there are at least two vertices with degree 1. These vertices are necessarily not critical in F_t, since any boundary point which is critical in F_t is also critical in ∂F_t.

Lemma 2 *During the course of the cleaning process, if all the agents are using the* ***SWEEP*** *protocol the agents are never going outside the dirty region. Namely, the only time an agent is located on a clean tile is when* $F = \emptyset$.

Proof The only movements allowed by the **SWEEP** protocol are towards a dirty tile. The only times an agent is allowed to clean a tile are when it is the last agent leaving this tile, or when cleaning this tile means that the mission is completed (for example, when this is the only dirty tile left). In addition, the various preemption which are in charge of agents synchronization prevent an agent to move to a dirty tile, which is getting cleaned during its movement. As a result, an agent is guaranteed to be located in dirty tiles throughout the entire cleaning process.

Theorem 1 *A group of agents executing the* ***SWEEP*** *protocol will eventually clean a simply connected region and stop together at the pivot point* p_0.

Proof While F_t has not yet been cleaned, $S_t > 0$ and hence, by Lemma 1 there is a non-critical point on ∂F_t. Since the agents obey the **SWEEP** protocol, while $F_t \neq \emptyset$, there is at least a single agent still traversing F_t. As this agent does not change the direction of its traversal, it is guaranteed to arrive to (and clean) a non-critical point at least once during the course of its traversal (thus decreasing S_t by at least 1). As $c_t \leq 4S_t$ (since during a traversal around F_t any tile can be entered to from each one of its 4 neighbors at most once), the maximal length of this traversal is $4S_t$. As agents may indeed delay one another, it is shown in Lemma 5 that the actual time required for an agent to complete its traversal around F_t is at most four times the length of this traversal, namely — $16S_t$. As $\forall t$, $S_t \leq S_0$, it can be seen that after no more than $16s_0^2$ time steps we shall have $S_t = 0$. The fact that all agents will meet at the same point is implied by the following two rules that are implemented in the **SWEEP** protocol:

- **Rule 1**: F_t is always being kept connected (we never clean a tile that has no clean neighbor and never clean a critical tile either).
- **Rule 2**: The pivot p_0 is cleaned only at the last time step of the cleaning process.

5.3 Detailed Analysis

In this section, we shall discuss an upper bound for the cleaning time of k agents employing the **SWEEP** cleaning protocol, for some dirty region F_0. From an upper bound for the cleaning time of a swarm comprising k agents, an upper bound for the number of agents needed to ensure a successful cleaning of a given region in a given number of time-steps can be derived.

The following Lemma states the change in the cardinality of a region's circumference after being traversed by an agent using the **SWEEP** protocol.

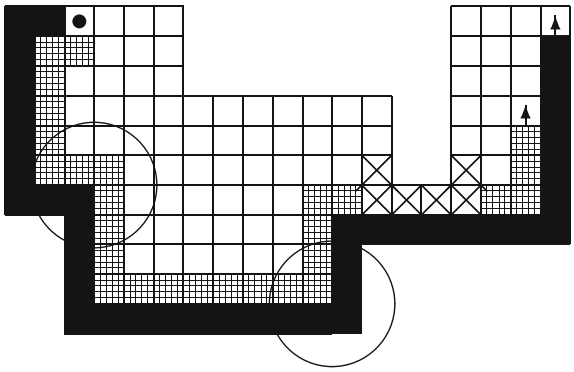

Fig. 11 An illustration of Lemma 3 and of the **SWEEP** protocol. The *black dot* denotes the starting point of the agents cleaning F (which is the entire region presented). The X's mark the *critical points* which are not cleaned. The *black tiles* are some of the tiles of ∂F, cleaned by the first agent. The second layer of marked tiles represent some of the tiles of $\partial F'$, cleaned by the second agent. The effect of corners on the traversal cardinality is either an increase (marked by the *left circle*) or a decrease (marked by the *right circle*)

Lemma 3 *The cardinality of the region's circumference always decreases after being traversed by an agent applying the* ***SWEEP*** *protocol (assuming no spread occurs), namely:*

$$|\partial F'| \leq \max\{1, |\partial F| - 8\}$$

Proof For any region F such that the number of tiles in F' is at least 2, note that a traversal around F is a closed, simple, rectilinear polygon. $\partial F'$ was obtained after deleting all the *non-critical* points of ∂F. Traversing such a polygon, an agent either goes straight, makes an internal turn ("left" turn, if we assume a clockwise movement), or makes an external turn ("right" turn). Suppose without loss of generality that an agent is moving up and making an internal turn left. Assume that there are no critical points. The path around this turn, along the newly created boundary tiles is now longer by two, as it contains an additional movement up, and another one towards the left. Similarly, an external turn, creates a new path which is shorter by two tile (as a single horizontal movement and a single vertical movement are no longer necessary). Since ∂F is a simple rectilinear polygon, it always has four "right" turns more than "left" ones.[2] When *critical* points are met, they are not being cleaned, which means that they exist in both ∂F and $\partial F'$. Re-visiting these points, however, does not change the overall size of the set of tiles (see an example in Fig. 11).

This proof, though, does not always hold for regions that, after being traversed once, produce either $\emptyset$ or a single tile. The reason is that the proof of the Lemma that appears in [36] relies on the fact that for each "turn" in F, there is a corresponding

[2]This is a simple consequence of the "rotation index" Theorem (see e.g. [20] p. 396): If $\alpha : [0, 1] \to R^2$ is a plane, regular, simple, closed curve, then $\int_0^1 k(s)ds = 2\pi$, where $k(s)$ is the curvature of $\alpha(s)$ and the curve is traversed in the positive direction.

"turn" in F'. However, the concept of "turn" requires the existence of a possible movement of an agent from one tile to another time. This obviously cannot be done for $F' = \emptyset$ or for $F' = \{v\}$. For such regions it can easily be seen the Lemma holds, as $|\partial\emptyset| = 0$ and as $|\partial\{v\}| = 1$.

While producing a bound for the cleaning time of the agents we shall demonstrate that while being traversed by the agents using the **SWEEP** protocol, the width of the initial region F_0 decreases monotonically. The main component of this proof is presented in the following Lemma.

Lemma 4 *Every time a region is traversed by an agent using the **SWEEP** protocol, its width is decreased by at least one, namely:*

$$W(F'_t) \leq W(F_t) - 1$$

A corollary is that the number of tours around a region F that k agents must accomplish before $F = \emptyset$ is at most $(\frac{W(F)}{k} + 1)$.

Proof By the definition of the depth of a tile, it is the shortest path to a *non-critical* tile in ∂F. According to the **SWEEP** protocol, after an agent had traversed F, all of the *non-critical points* in ∂F were cleaned. This is true as the agent is instructed to clean all the *non-critical points* it is going through. The only exception are tiles which an agent does not clean upon exit, as there are other agents currently located in it. However, in this case, this tile will be cleaned by the last agent leaving it, at the following time steps. Therefore, after an agent traverses the region, the depth of every internal tile was decreased by (at least) one, meaning that the total width of the region was decreased by (at least) one. As a result, as all agents are identical, when k agents complete a traversal of F, the width of F is decreased by at least k.

When examining the cleaning time of the agents, one must remember that merely calculating the length of the paths the agents must move along is not enough. Since in various scenarios the **SWEEP** protocol may direct an agent to stop and wait (for example, when several agents are located in the same tile), a way to bound the actual time it takes an agent to move along a specific path must be found. The following Lemma establishes the relation between the length of a path and the maximal time it takes an agent using the **SWEEP** protocol to move along it.

Lemma 5 *The time it takes an agent which uses the **SWEEP** protocol to move along a path of length c_t (including delays caused by other agents located in the same tiles) is at most $4 \cdot c_t$.*

Proof According to the problem's specifications, each agent moves along the dirty region F at a pace of one tile per time step. The only exception may occur when several agents are delayed after entering the same tile.

According to the **SWEEP** protocol, although several agents can enter the same tile at the same time step, agents located in the same tile can leave it at the same time step only if they do so towards different directions. As a result, there arises the question whether more and more agents can enter a certain tile without leaving it, causing a decrease in the swarm's performance.

Let v be an empty tile. Let us assume without loss of generality that at some time step t, 4 agents enter v (this is of course the maximal number of agents which can enter v at the same time step, due to the invariant stating that no two agents can exit u to v at the same time step, for u, some neighbor of v, and since v has at most 4 neighbors). Thus, in time step $t-1$ v had exactly 4 neighbors in ∂F. Let us assume that in time step $t+1$ v still has 4 neighbors in ∂F. Thus, all the 4 agents which entered v intends to move to 4 different directions (since according to the **SWEEP** protocol, each of them has different *rightmost* neighbor). Thus, none of them will be delayed. Alternatively, let us assume that at time step $t+1$ v has less than 4 neighbors in ∂F. Let α denote the number of neighbors of v in ∂F. Then, since the 4 agents which entered v in time t did so from different directions, according to the **SWEEP** protocol, α of them will be able to leave v towards the α neighbors of v, and $(4-\alpha)$ will stay in v. However, in this time step, only α new agents can enter v, since v has only α neighbors in ∂F. Thus, in time step $t+2$ there are at most 4 agents in v. This is true of course for each time steps $T > t+2$. Since the number of agents in each tile is at most 4, and since each tile with agents in it has at least one neighbor of ∂F (since the agents had to enter it from a neighbor in ∂F), then there are at least $\frac{k}{4}$ agents which are able to move. Thus, even if the agents collide with each other continuously, the time it takes k agents to traverse a region of circumference c_t is at most $4 \cdot k \cdot c_t$ (and the average traversal time of an agent, amortized for the entire group of agents, is at most 4 times its minimal traversal time).

Notice that this still holds for the rare occasions in which the *check near completion of mission* procedure detects that every non-critical perimeter tile of F_t contains at least two agents. In his case, the *resting* commands are indeed overruled, allowing temporarily more than 4 agents to be located into the same tile, however — as the internal prioritization of the agents is maintained, this is corrected at the following time steps, as the "redundant" agents will wait (sometimes for more than 4 time steps). Notice though that this "long resting" was already compensated by the "untimely movement" those agents had performed while entering this tile.

It should be stated that in reality, the agents' traversal time is only slightly more than c_t, although an analytic proof is yet to be found.

When an agent using the **SWEEP** protocol is traversing F_t it does so by moving along a path which contains all the tiles of ∂F_t. Since by doing that, the agent may pass through the same tile more than once, the length of this path may be larger than the number of tiles in ∂F_t. A bound for the ratio between the two is described in the following Lemma.

Lemma 6 *c_t, the length of the circumference of F_t never exceeds twice its cardinality, namely:*

$$c_t \leq 2 \cdot |\partial F_t| - 2$$

```
1: Add (x, y) to L and mark it as OLD
2: Let v denote the rightmost neighbor of (x, y)

3: If v ≡ p_s then
4:       Delete all the marked tiles from L
5:       STOP
6: End if

7: If (v is marked as OLD) and (no circle was formed) then
8:       /* The traversal repeats this tile as well */
9:       Call SPAN-DFS for v
10:      Upon return — STOP
11: End if

12: If (v is marked as OLD) and (a circle was formed) then
13:      /* Continue, and clean the redundant tiles */
14:      Add to L the sequence of tiles from L, starting with (x, y)
         and backwards to v
15:      Mark these tiles (excluding (x, y)) to be deleted
16:      Call SPAN-DFS for v
17:      Upon return — STOP
18: End if

19: /* The rightmost neighbor was not OLD */
20: Call SPAN-DFS for v
21: Upon return — STOP
```

Fig. 12 The recursive spanning tree construction algorithm. The term *rightmost* has the same meaning as in the **SWEEP** protocol. By checking L we can find out whether continuing to an **OLD** neighbor is a "clean return" (e.g. (A,B,C,B)) or whether it completes a circle (e.g. (A,B,C,A))

Proof ∂F_t is a connected graph and thus it has spanning trees. An algorithm for constructing a spanning tree for ∂F_t and a *Depth First Search* (DFS) scan of it was constructed, such that the path generated by using the DFS algorithm contains a traversal around F_t (meaning that the DFS scan generates a path which after having several of its tiles removed, equals a path which traverses F_t).

This recursive spanning tree construction algorithm appears in Fig. 12. The algorithm receives p_s, a *non-critical* tile as its starting point and an empty list of tiles, L.[3] The algorithm constructs a spanning tree of ∂F_t as well as a DFS scan for this tree, by adding tiles to the list, while optionally marking some of them as tiles which should later be deleted. After the deletion of these tiles the remaining tiles form a path which traverses F_t. An example of the above appears in Fig. 13.

A DFS scan of a tree can go through each edge at most twice. A tree of $|V|$ vertices contains exactly $(|V| - 1)$ edges. Thus, a DFS scan of a spanning tree of ∂F_t contains at most $2 \cdot (|\partial F_t| - 1)$ transitions of edges. Since there exists such a spanning tree that contains a 'tour' around F_t then: $c_t \leq 2 \cdot |\partial F_t| - 2$.

Definition 18 Let $W_{REMOVED}(t)$ denote the decrease in F's width due to the agents cleaning activity until time t.

[3]The existence of a *non-critical* point is guaranteed since ∂F_t is a connected graph and thus has a spanning tree, in which at least two tiles has a degree of 1, which makes them *non-critical* tiles.

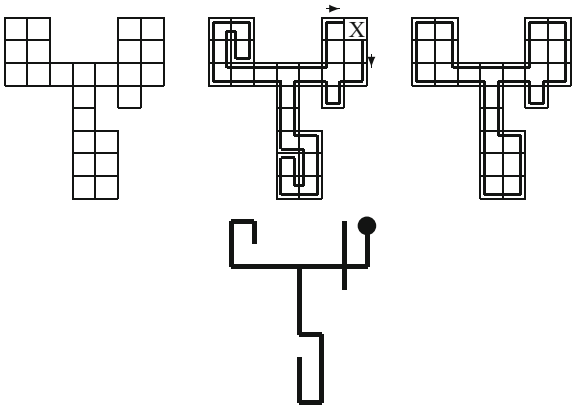

Fig. 13 An example of the spanning tree protocol. The *left chart* represents the dirty region F_t. The *middle chart* shows the DFS scan of ∂F_t according to the **SPAN-DFS** protocol, when the big X marks p_s, the *non-critical* starting point. The *right chart* shows the traversal path which is contained in this DFS scan. The drawing on the *bottom* shows the spanning tree that is created by the protocol and is searched by the DFS algorithm

Lemma 7

$$\text{If } \left\lfloor \sum_{i=1}^{t} \frac{k}{4 \cdot c_i} \right\rfloor \geq W(F_0) \text{ then } W_{REMOVED}(t) \geq W(F_0)$$

Proof $W_{REMOVED}(t)$ can be defined as the number of completed traversals around the dirty region, multiplied by the number of the agents k. Note that at time step t, each agent completes $\frac{1}{c_i}$ of the circumference. Since we know the value of c_t for every t, and since we know that the time it takes an agent to move along a path is at most 4 times the length of this path (see Lemma 5), then the decrease in the original width of the region caused by the cleaning process of the agents is bounded as described.

Note that as the expression above does not define between a multitude of agents working for a short while, and a single agent which is working for a long period of time, it may not hold for smaller values of t (in which the accumulated partial peelings is smaller than $W(F_0)$). Therefore, while merely writing $W_{REMOVED}(t) \geq \left\lfloor \sum_{i=1}^{t} \frac{k}{4 \cdot c_i} \right\rfloor$ might not always hold, the expression as appears above does.

A bound over the cleaning time of a given region F_0 can then be produced as follows:

Theorem 2 *Assume that k agents start cleaning a simple connected region* F_0 *at some boundary point* p_0 *and work according to the* ***SWEEP*** *protocol, and denote by* $t_{success}(k)$ *the time needed for this group to clean* F_0*. Then it holds that:*

$$\max \left\{ 2k, \left\lceil \frac{S_0}{k} \right\rceil, 2l(F_0) \right\} \leq t_{success}(k) \leq \frac{8(|\partial F_0| - 1) \cdot (W(F_0) + k)}{k} + 2k$$

Proof The lower bound is quite obvious — the left term $2k$ is the time needed to release the k agents from the pivot point, $\frac{S_0}{k}$ is a lower bound on the time necessary to cover the region if the agents were optimally located at the beginning. The right term, $2l(F_0)$, comes from the observation that at least one agent should visit (and return from) any point of F_0, including the one farthest from p_0, to which the distance is $l(F_0)$.

Considering the upper bound, in order for F_0 to be cleaned, there should exist a time $t_{success}$ in which the width of the region will be 0. Note that once the dirty region was reduced to a 'skeleton' comprises only of tiles of depth zero, an additional traversal must be done in order to clean it entirely.

By combining Lemmas 3 and 6 we know that for $c_j(i)$ and $\partial F_j(i)$ be the length and cardinality respectively of the i-th traversal of agent j:

$$c_j(i) \leq 2(|\partial F_j(i)| - 1) \leq 2(|\partial F_0| - 1)$$

Using Lemma 7 we see that:

$$W_{REMOVED}(t) \geq \left\lfloor \sum_{i=1}^{t} \frac{k}{4 \cdot c_i} \right\rfloor \geq \left\lfloor \frac{k \cdot t}{8(|\partial F_0| - 1)} \right\rfloor \tag{1}$$

Thus, we are interested in:

$$\left\lfloor \frac{k \cdot t_{success}(k)}{8(|\partial F_0| - 1)} \right\rfloor \geq W(F_0) + k \tag{2}$$

adding the "$+k$" for the additional traversal needed for cleaning the dirty 'skeleton'.

Since releasing the agents requires an additional $2k$ time steps, the final bound for this case is:

$$t_{success}(k) \geq \frac{8(|\partial F_0| - 1) \cdot (W(F_0) + k)}{k} + 2k$$

Using the above Theorem we can bound the *efficiency ratio*, defined as $\frac{t_{success}(k)}{S_0}$ which expresses the benefit of using k agents for a cleaning mission:

Corollary 1

$$\max\left\{\frac{2k}{S_0}, \frac{1}{k}, \frac{2l(F_0)}{S_0}\right\} \leq \frac{t_{success}(k)}{S_0} \leq \frac{8\,(|\partial F_0| - 1) \cdot (W(F_0) + k)}{k \cdot S_0} + \frac{2k}{S_0} \tag{3}$$

An interesting result of Corollary 1 is that when $W(F_0) \ll k \ll S_0$, i.e. the number of agents is large relative to the width but small compared to the area, then the efficiency ratio is bounded below by twice the ratio of the length and area, and bounded above by $8\frac{|\partial F_0|}{S_0}$. Note here the similarity to the ratio $\frac{c_0}{\sqrt{S_0}}$, known as the *shape-factor*.

Another conclusion is that when we scale up the region by a factor of n, the area increases as n^2 but the width, length and circumference all increase as n so we get the following:

Corollary 2

$$as\ n \to \infty, \qquad L_\infty \leq \frac{t_{success}(k)}{S} \leq U_\infty$$

where

$$L_\infty = \frac{1}{k}, \qquad U_\infty = 8\frac{\partial F \cdot W}{k \cdot S} = 8\frac{|\partial F_0| \cdot W(F_0)}{k \cdot S_0}$$

and $S = S_0 n^2$, $\partial F = |\partial F_0| n$, $W = W(F_0)n$ *are the scaled area, circumference cardinality and width, respectively.*

5.4 Robustness

The issue of an algorithm's robustness and fault-tolerance is a major aspect of robotics systems, as the environments in which the agents operate may often include unpredicted changed from the original specification of the system, such as various kinds of noises, robots' malfunctions, etc.

While analyzing the performance of the cleaning protocol, the robotic agents were assumed to work perfectly. In real robotic system, however, this is not always the case. In simulation, however, it was quite easy to examine what happens when one or more agents disappear. The disappearance of an agent which dies is necessary in our framework in order for the system to work properly.

As discussed in Sect. 4.1, one of the requirements of the **SWEEP** protocol is a complete fault-tolerance to malfunctions in the agents, meaning — even if one or several agents stop working ("die" and disappear) the rest of the agents will continue the cleaning process as efficiently as their number allows them. The basic idea enabling this robustness is the complete independence between the agents, due to the full decentralized nature of the system (as the agents do not share the cleaning mission between them, nor do they rely on one another in the performance of their cleaning process).

A different aspect of the algorithm's robustness is the performance of the agents in noisy environments. In general, random noise can appear and influence various aspects of the system — in sensing whether a tile should be cleaned, in sensing the presence of other agents in an agent's vicinity, in the agent's movement, etc. As in almost any system, it is easy to see that severe noise may significantly affect the algorithm's efficiency, and may even prevent the agents from completion the mission altogether. For example, difficulties in sensing whether a tile should be cleaned or not, may cause an agent to clean a critical point, thus separating the region

into several connected components. In addition, noises in the agents' movement system may cause the agents to detach from the contaminated region, thus delaying, or even preventing the completion of the cleaning process. Whereas a situation in which the noise level is kept at a reasonable level (i.e. correct identification of clean and contaminated tiles, and movement noises), it is the authors' opinion that the completion of the algorithm may still be achieved, although the cleaning will increase. When the agent has difficulties in the sensors in charge of detecting other agents in its vicinity, and interpreting their signaling (see relevant sections for details about signaling and synchronization) several problematic scenarios (which these signaling mechanisms were designed to avoid), may occur. However, simulations show that in such cases often only the cleaning time is affected, while the completion of the cleaning mission is preserved. Nevertheless, as this issue was not fully investigated yet, a perfect neighbors detection is still one of the algorithm's requirements.

One example for the above described problem is presented Fig. 7, in which a failure in the sensing mechanism may damage the simple-connectivity of the contaminated region. Note that in such scenarios, there is least a single agent in each of the new contaminated components. Therefore, each component is an instance of the cooperative cleaners problem, for which the upper bound for the cleaning time may be applied.

Another example is demonstrated in Fig. 8, where a bug in the *waiting* mechanism (see Sect. 4.6 for more details) may prevent some of the agents from cleaning contaminated tiles. The same holds for possible bugs in the *resting* mechanism (see Sect. 4.7 for more details) which may cause several agents to move together, acting as a single cleaning agent.

It is also interesting to examine the scenario in which all the robots are picked up, and re-placed on some arbitrary positions onboard the dirty region. Thanks to the definition of the SWEEP protocol, the robots will navigate to the boundary of the region, and resume their cleaning. Naturally, the overall cleaning time might be affected, however — the completion of the cleaning is still guaranteed.

5.5 *Comparison to Existing Work*

Looking at the existing work that has been done with respect to multi robotic systems designed for a cooperative cleaning/coverage/etc., several analytic results concerning the completion time of the mission can be found.

An interesting result is presented in [35]. In this work, a group of simple robots is used for periodically covering an unknown area, using the nodes counting method. Based on a more general result for undirected domains shown in [27], the following bound is given:

The cover time of teams of ant robots (of a given size) that use node counting on strongly connected undirected graphs can be exponential in the square root of the number of vertices.

Namely:

$$f(k) = O(2^{\sqrt{S_0}}) \tag{4}$$

denoting the covering time of k robots by $f(k)$.

Recalling Theorem 2 in which:

$$t_{success}(k) \leq \frac{8(|\partial F_0| - 1) \cdot (W(F_0) + k)}{k} + 2k$$

as $|\partial F_0| = O(S_0)$ and as $W(F_0) = O(\sqrt{S_0})$ we see that:

$$t_{success}(k) = O\left(\frac{1}{k} S_0^{1.5} + S_0 + k\right)$$

As for practical reasons we assume that $k < S_0$ we can see that:

$$t_{success}(k) = O\left(\frac{1}{k} S_0^{1.5} + S_0\right)$$

and when the number of robots is independent in the size of the region, the time complexity can be reduced to:

$$t_{success}(k) = O\left(S_0^{1.5}\right)$$

Namely:

Corollary 3

$$\left(\frac{8(|\partial F_0| - 1) \cdot (W(F_0) + k)}{k} + 2k < d\right) \Rightarrow \left(t_{success}(k) = O\left(S_0^{1.5}\right)\right)$$

Comparing this to the bound of Eq. 4, we see that the time complexity of the **SWEEP** protocol is much smaller than this of the nodes counting technique.

It should be mentioned though, that in [35] the authors clearly state that it is their belief that the coverage time of robots using nodes counting in grids is much smaller. This estimation is also demonstrated experimentally. However, it should also be pointed out that the methods described in [35] require the robots to have some sort of *marking* capability (namely, leaving trails along the region's tiles). Such a requirement is not necessary for the **SWEEP** protocol.

6 Regions with Obstacles

So far we have only dealt with simply connected regions, i.e. — regions with no "holes". In the case of a (connected) region with obstacles (i.e. holes) the simple **SWEEP**ing protocol will not work, due to the following "cycle" problem: eventually, each obstacle will be surrounded by critical points, and there will be a time when *all* boundary points of F will be critical (we shall call such a situation *useless*, as opposed to the *useful* state when some points are cleaned during a tour). As a cure to this problem, we suggest to add an "odoring" feature to our cleaners, that is — an agent will be able to label each tile it is going through by a small amount of "perfume" (this action may remind one of the pheromones left by ants during their walks). These labels will designate the directions according to which agents traveled through the labeled tiles. Upon getting to the useless state (detected by each agent due to no cleaning between two consecutive visits from the same entrance to the pivot point) an agent will continue to traverse the boundary, but will now look for a point which has a single label — that is, one that has odor pointing only on one direction of movement. If this tile does not have a neighbor which is entirely unlabeled, the agent will clean this point (despite its "criticality", as it is not really critical since it is necessarily part of a cycle around an obstacle) and then continue as in a useful state (see Fig. 14 for an example). This will open one cycle, hence, if there are m obstacles, we will need $\frac{m}{k}$ such tours before the region is completely clean. On the other hand, Lemma 3 no longer holds (see Fig. 16) since the boundary area can increase with time. However the boundary is always bounded above by the area. We now make the following conjecture:

Conjecture 1 Assume that k agents start at some boundary point of a non-simple connected region F_0 with s obstacles in it, and work according to the **SWEEP-WITH-OBSTACLES** protocol, and denote by $t_{success}(k)$ the time needed for this group to clean F_0. Then it holds that:

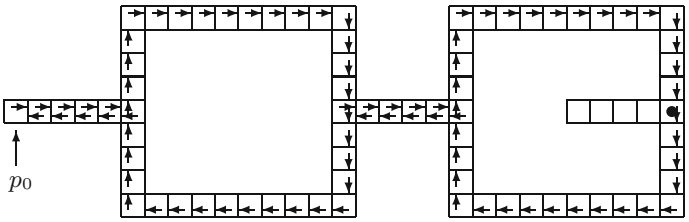

Fig. 14 An example of agents' odoring used in Sect. 6 for cleaning regions with obstacles. The figure demonstrates the labels on the region's tiles after an agent reached the *useless* state, in which it did not clean any tile in its last tour. Note that any of the tiles labeled with one direction only, can be cleaned while still preserving the region's connectivity. Note that from those tiles, only tiles with a neighbor which is not labeled with any direction can not be cleaned, since it is indeed a critical point. Such a point is marked in the figure by a *black circle*

$$\max\left\{2k, \left\lceil \frac{S_0}{k} \right\rceil, 2l(F_0)\right\} \leq t_{success}(k) \leq 8 \cdot S_0 \left(\frac{W(F_0)}{k} + \left\lceil \frac{s}{k} \right\rceil\right) + 2k \qquad (5)$$

where S_0,$l(F_0)$ and $W(F_0)$ denote the free area, length and width of F_0, respectively, and s is the number of obstacles in F_0.

Notice though, that the completion of the cleaning process is still guaranteed.

7 Experimental Results

A computer simulation, implementing the **SWEEP** protocol, was constructed. The protocol was run on several shapes of regions and for numbers of agents varying from 1 to 20. See Figs. 15 and 16 for some examples of the evolution of the layout with time. The gray level of each pixel designates the index of the agent that actually cleaned this point. The right side of Fig. 15 shows the same region and number of agents as in the left side of this Figure, but with randomly chosen initial locations at the corners. It can be seen that the dirty region is cleaned in a similar way. It should be said here that all the theory we developed in the previous sections (up to

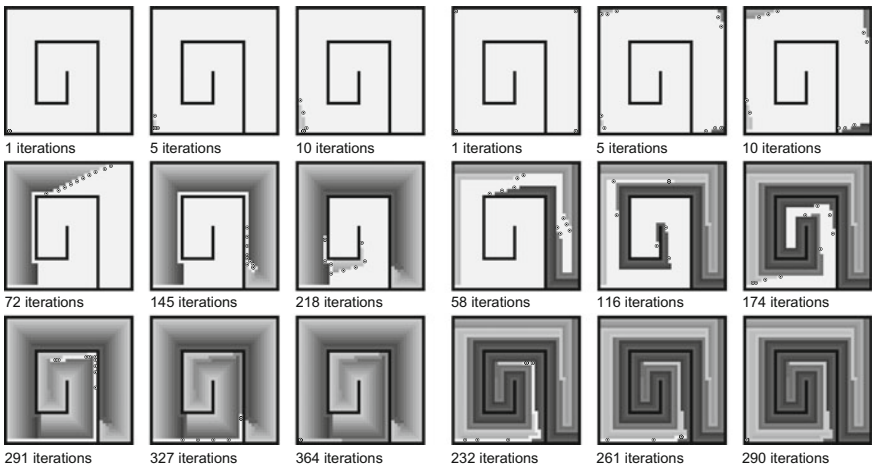

Fig. 15 Cooperative cleaners: maze, 10 agents (*left*), and 10 agents with random initial locations on the boundary (*right*)

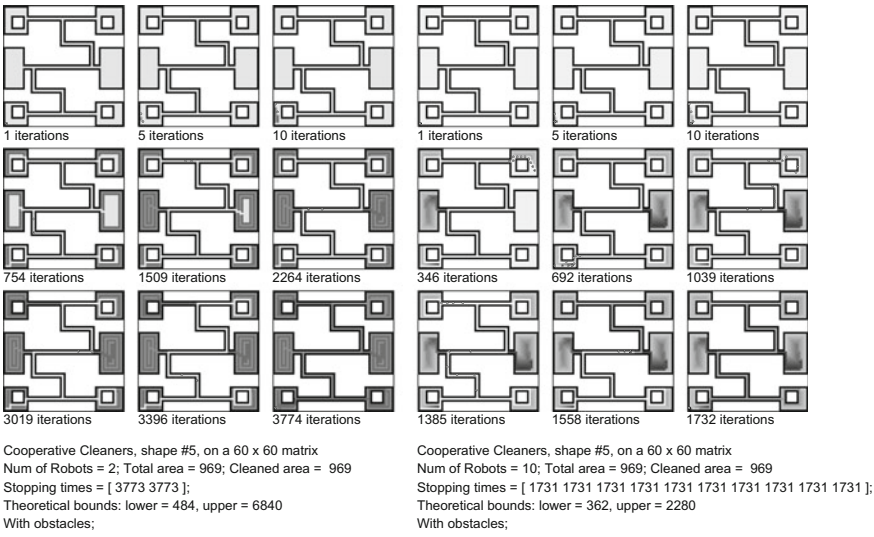

Fig. 16 Cooperative cleaners: 2 agents, 6 rooms with obstacles (*left*), 10 agents, 6 rooms with obstacles (*right*)

a small additive constant) applies to the case where the agents are initially located in randomly selected points on the boundary of F_0 (rather than starting from p_0). Figure 16 shows the evolution of the **SWEEP-WITH-OBSTACLES** protocol for the same regions with four additional obstacles in each, with 2 agents (left) and 10 agents (right).

Figure 17 depicts the timing results (as produced by computer simulations) describing the cleaning time as a function of the number of agents, compared to the theoretical bounds presented in previous sections (the cleaning time is normalized by the initial area of the dirty region). In Fig. 18 we show the results for the same figures with additional obstacles, together with the conjectured theoretical bounds. All the figures include their appropriate statistical values.

Additional result are presented in Fig. 19, comparing the various efficiency ratios obtained for initial dirty regions of various sizes. This is examined even further by Fig. 20, which shows the behavior of $\frac{t_{success}(50)}{t_{success}(1)}$ namely, the ratio between the cleaning time of 50 agents and the cleaning time required for a single agent. The change in this ratio can be seen, hinting the fact that as the initial region is larger, the addition of more agents can more efficiently decrease the cleaning time of the agents.

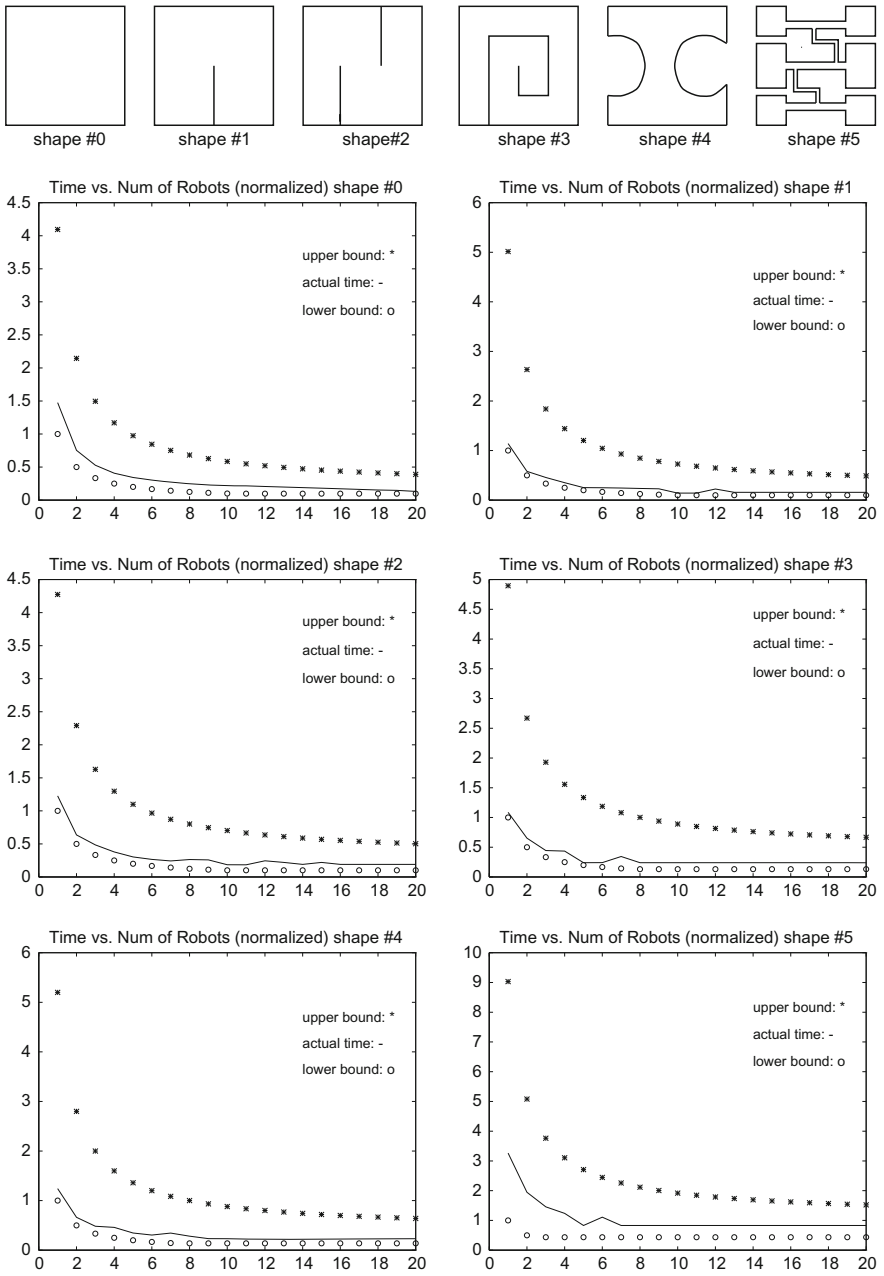

Fig. 17 Various shapes tested in **SWEEP** simulations and their normalized cleaning time $t(k) = \frac{t_{success}(k)}{S_0}$ as a function of the number of agents, k, for each region, together with the lower and upper bounds according to Theorem 2. For a confidence level of 95%, based on at least 10 simulation instances per each dirty region, the appropriate confidence interval is less than $\pm 6\%$

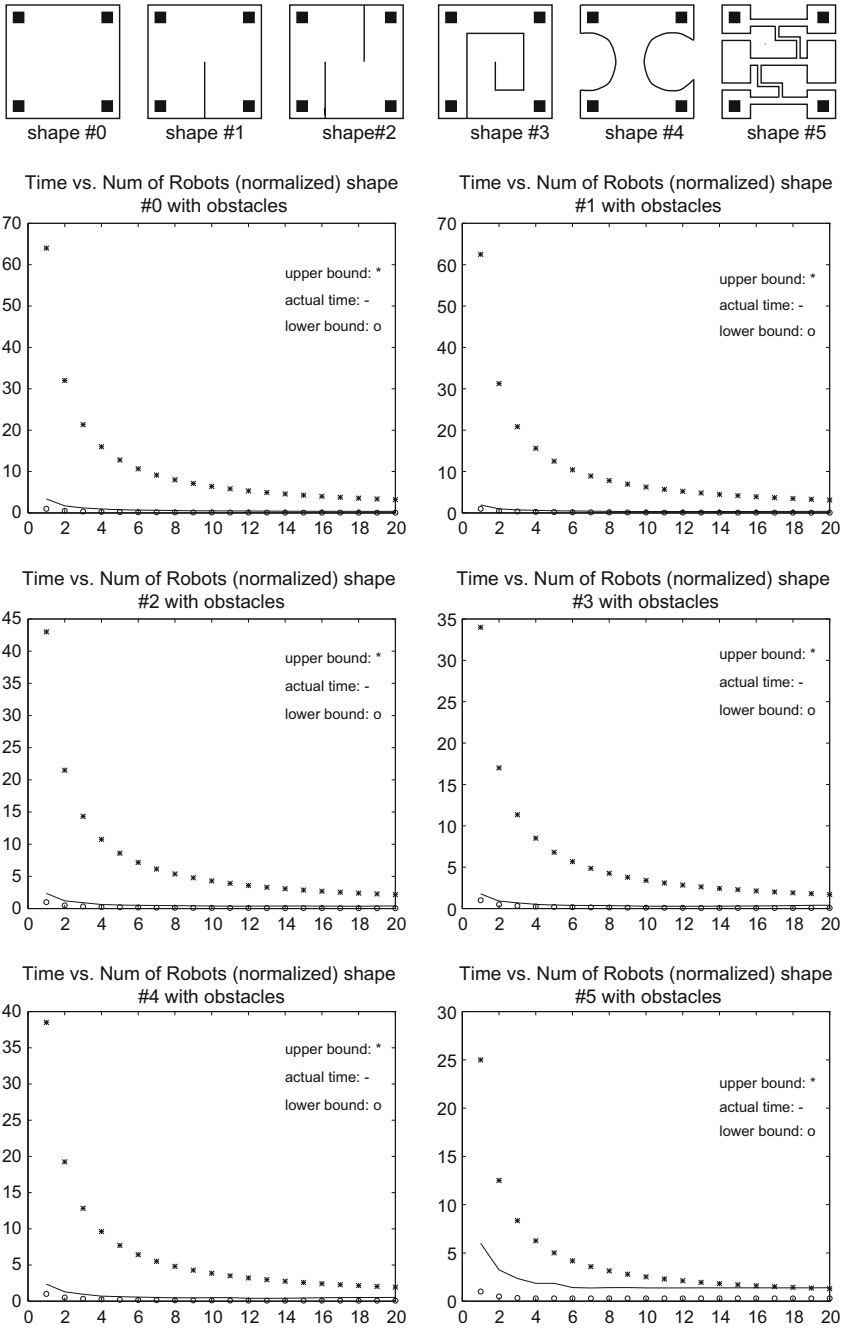

Fig. 18 Various shapes tested in **SWEEP-WITH-OBSTACLES** simulations and their normalized cleaning time $t(k) = \frac{t_{success}(k)}{S_0}$ as a function of the number of agents, k, together with the lower and upper bounds according to Theorem 2. For a confidence level of 95%, based on at least 10 simulation instances per each dirty region, the appropriate confidence interval is less than $\pm 10\%$

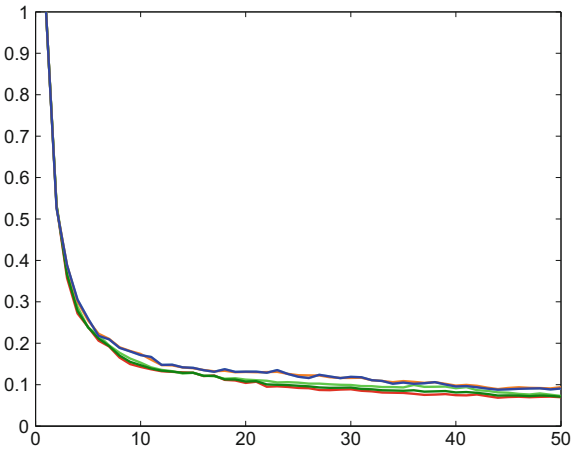

Fig. 19 Cooperative cleaners: from 1 to 50 agents, dirty regions of sizes 600 to 1500 tiles, 10 simulations per region's size, cleaning time normalized by the cleaning time of a single agent. Note that as the initial dirty area is larger, the ratio between $t_{success}(k)$ and $t_{success}(1)$ increases. In fact, comparing the time ratios of regions of sizes 600 and 1500 tiles, the difference reaches 30%. For a confidence level of 95% the confidence interval is less than $\pm 15\%$

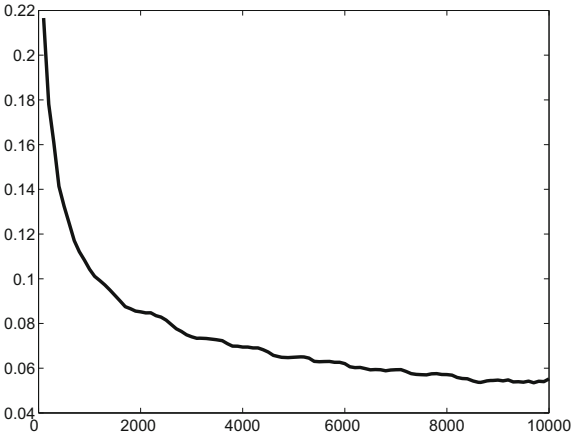

Fig. 20 Cooperative cleaners: graph demonstrates the behavior of $\frac{t_{success}(50)}{t_{success}(1)}$ for dirty regions of sizes 100 to 10000 tiles. 10 simulations were done per region's size. Note how as the initial dirty area gets larger $\frac{t_{success}(50)}{t_{success}(1)}$ decreases. The minimal value possible is of course $\frac{1}{50} = 0.02$. For a confidence level of 90% the confidence interval is less than $\pm 10\%$

8 Conclusion

Our cooperative **SWEEP**ing protocol can be considered as a case of social behavior in the sense of [34], where one induces multi-agent cooperation by forcing the agents to obey some simple "social" guidelines. This raises the question what happens if

one agent malfunctions. We have shown that if less than k agents stop, the others will take over their responsibilities. But what if some agents start to cheat? Such adversaries will have catastrophic consequences, since a crazy agent may clean a critical point and disconnect the dirty region.

Another question of interest is the resolution of collisions between agents — the formation of clusters of agents located in the same tile, and delaying each other. In this work we resolve such a problem by artificially producing an implicitly agreed order among the agents, depending on their previous locations and movements. However, it is an interesting open question whether some random based method may achieve better results.

As can be seen throughout the analysis of our proposed collaborative protocol, the ability of a collaborative swarm of highly limited drones to efficiently scan a pre-defined region is strongly associated to the geometric features of this region. This topic was thoroughly analyzed in works such as [8, 11, 12], with a complexity result presented in [4, 5].

The cleaning problem discussed is also related to the geometric problem of pocket-machining, see [23] for details. An interesting problem of cleaning and maintenance of a system of pipes by an autonomous agent is discussed in [30]. The importance of cleaning hazardous waste by agents is described in [22].

Another question of interest is how to guarantee covering, at least with high probability, without using any external signs, using only the inter-agent collisions as an indicator for a good direction to proceed.

Considering the question of how fast can k agents clean a pre-defined contaminated region, it is of interest to notice here that an off-line version of the problem, that is: finding the shortest path that visits all grid points in F, where F is completely known in advance, is *NP-hard* even for a single agent. It is a corollary of the fact that Hamilton path in a non-simple grid-graph is *NP-complete* [25].

Finally, it is interesting to examine cases where the swarm have to operate not such under completion (e.g. ability to guarantee complete scan) or efficiency (e.g. scan time) constraints, but also under adversarial reactive behavior of the search region, namely – the ability of parts of the search space that were previously scanned to become "unscanned" after a certain portion of time. Such property of the problem is analyzed in the next chapters of this book, and was analyzed in works such as [6, 7, 10, 13, 32].

References

1. E.U. Acar, Y. Zhang, H. Choset, M. Schervish, A.G. Costa, R. Melamud, D.C. Lean, A. Gravelin, Path planning for robotic demining and development of a test platform, in *International Conference on Field and Service Robotics* (2001) pp. 161–168
2. M. Ahmadi, P. Stone, A multi-robot system for continuous area sweeping tasks, in *IEEE International Conference on Robotics and Automation, 2006 (ICRA 2006)* (2006)
3. Y. Altshuler, *Multi Agents Robotics in Dynamic Environments*. Ph.D. thesis, Israeli Institute of Technology, May 2010

4. Y. Altshuler, A.M. Bruckstein, The complexity of grid coverage by swarm robotics, in *ANTS 2010* (LNCS, 2010), pp. 536–543
5. Y. Altshuler, A.M. Bruckstein, Static and expanding grid coverage with ant robots: complexity results. Theoret. Comput. Sci. **412**(35), 4661–4674 (2011)
6. Y. Altshuler, A.M. Bruckstein, I.A. Wagner, Swarm robotics for a dynamic cleaning problem, in *IEEE Swarm Intelligence Symposium* (2005), pp. 209–216
7. Y. Altshuler, V. Yanovski, I.A. Wagner, A.M. Bruckstein, The cooperative hunters - efficient cooperative search for smart targets using uav swarms, in *Second International Conference on Informatics in Control, Automation and Robotics (ICINCO), the First International Workshop on Multi-Agent Robotic Systems (MARS)* (2005), pp. 165–170
8. Y. Altshuler, I.A. Wagner, A.M. Bruckstein, Shape factor's effect on a dynamic cleaners swarm, in *Third International Conference on Informatics in Control, Automation and Robotics (ICINCO), the Second International Workshop on Multi-Agent Robotic Systems (MARS)* (2006), pp. 13–21
9. Y. Altshuler, V. Yanovsky, I. Wagner, A. Bruckstein, Swarm intelligencesearchers, cleaners and hunters. *Swarm Intelligent Systems* (2006), pp. 93–132
10. Y. Altshuler, V. Yanovsky, A.M. Bruckstein, I.A. Wagner, Efficient cooperative search of smart targets using uav swarms. ROBOTICA **26**, 551–557 (2008)
11. Y. Altshuler, I.A. Wagner, A.M. Bruckstein, On swarm optimality in dynamic and symmetric environments. Economics **7**, 11 (2008)
12. Y. Altshuler, I.A. Wagner, A.M. Bruckstein, Collaborative exploration in grid domains, in *Sixth International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (2009)
13. Y. Altshuler, I.A. Wagner, V. Yanovski, A.M. Bruckstein, Multi-agent cooperative cleaning of expanding domains. Int. J. Robot. Res. **30**, 1037–1071 (2010)
14. Y. Altshuler, A. Pentland, S. Bekhor, Y. Shiftan, A. Bruckstein, Optimal dynamic coverage infrastructure for large-scale fleets of reconnaissance uavs (2016), arXiv:1611.05735
15. R. Baeza-Yates, R. Schott, Parallel searching in the plane. Comput. Geom. **5**, 143–154 (1995)
16. M.A. Batalin, G.S. Sukhatme, Spreading out: a local approach to multi-robot coverage, in *6th International Symposium on Distributed Autonomous Robotics Systems* (2002)
17. V. Braitenberg, *Vehicles* (MIT Press, Cambridge, 1984)
18. Z. Butler, A. Rizzi, R. Hollis, Distributed coverage of rectilinear environments, in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics* (2001)
19. D.C. Conner, A. Greenfield, P.N. Atkar, A.A. Rizzi, H. Choset, Paint deposition modeling for trajectory planning on automotive surfaces. IEEE Trans. Autom. Sci. Eng. **2**(4), 381–392 (2005)
20. M.P. Do-Carmo, *Differential Geometry of Curves and Surfaces* (Prentice-Hall, New-Jersey, 1976)
21. Y. Gabriely, E. Rimon, Competitive on-line coverage of grid environments by a mobile robot. Comput. Geom. **24**, 197–224 (2003)
22. S. Hedberg, Robots cleaning up hazardous waste, in *AI Expert* (1995), pp. 20–24
23. M. Held, On the computational geometry of pocket machining. *Lecture Notes in Computer Science* (1991)
24. D. Henrich, Space efficient region filling in raster graphics. Vis. Comput. **10**, 205–215 (1994)
25. A. Itai, C.H. Papadimitriou, J.L. Szwarefiter, Hamilton paths in grid graphs. SIAM J. Comput. **11**, 676–686 (1982)
26. S. Koenig, Y. Liu, Terrain coverage with ant robots: a simulation study, in *AGENTS'01* (2001)
27. S. Koenig, B. Szymanski, Y. Liu, Efficient and inefficient ant coverage methods. Ann. Math. Artif. Intell. **31**, 41–76 (2001)
28. D. Latimer, S. Srinivasa, V. Lee-Shue, S. Sonne, H. Choset, A. Hurst, Toward sensor based coverage with robot teams, in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation* (2002)
29. T.W. Min, H.K. Yin, A decentralized approach for cooperative sweeping by multiple mobile robots, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (1998), pp. 380–385

30. W. Neubauer, Locomotion with articulated legs in pipes or ducts. Robot. Auton. Syst. **11**, 163–169 (1993)
31. M. Polycarpou, Y. Yang, K. Passino, A cooperative search framework for distributed agents, in *IEEE International Symposium on Intelligent Control* (2001), pp. 1–6
32. E. Regev, Y. Altshuler, A.M. Bruckstein, The cooperative cleaners problem in stochastic dynamic environments (2012), arXiv:1201.6322
33. I. Rekleitis, V. Lee-Shuey, A. Peng Newz, H. Choset, Limited communication, multi-robot team based coverage, in *IEEE International Conference on Robotics and Automation* (2004)
34. Y. Shoham, M. Tennenholtz, On social laws for artificial agent societies: off line design. AI J. **73**(1–2), 231–252 (1995)
35. J. Svennebring, S. Koenig, Building terrain-covering ant robots: a feasibility study. Auton. Robots **16**(3), 313–332 (2004)
36. I.A. Wagner, Y. Altshuler, V. Yanovski, A.M. Bruckstein, Cooperative cleaners: a study in ant robotics. Int. J. Robot. Res. (IJRR) **27**(1), 127–151 (2008)

http://www.springer.com/978-3-319-63602-3

Swarms and Network Intelligence in Search

Altshuler, Y.; Pentland, A.; Bruckstein, A.M.

2018, IX, 238 p. 116 illus., 53 illus. in color., Hardcover
ISBN: 978-3-319-63602-3