Israel A. Wagner
Computer Science Department
Technion, Haifa 32000, Israel
wagner@cs.technion.ac.il
and
IBM Haifa Labs,
MATAM, Haifa 31905, Israel
wagner@il.ibm.com

Yaniv Altshuler
Vladimir Yanovski
Alfred M. Bruckstein
Computer Science Department
Technion, Haifa 32000, Israel

Cooperative Cleaners: A Study in Ant Robotics

Abstract

In the world of living creatures, simple-minded animals often cooperate to achieve common goals with amazing performance. One can consider this idea in the context of robotics, and suggest models for programming goal-oriented behavior into the members of a group of simple robots lacking global supervision. This can be done by controlling the local interactions between the robot agents, to have them jointly carry out a given mission. As a test case we analyze the problem of many simple robots cooperating to clean the dirty floor of a non-convex region in $\mathbf{Z}^2$*, using the dirt on the floor as the main means of inter-robot communication.*

KEY WORDS—robotics, covering, multirobotics, robustness, cooperative robotics

1. Introduction

In the world of living creatures, simple-minded animals such as ants or birds cooperate to achieve common goals with surprising performance. It seems that these animals are 'programmed' to interact locally in such a way that the desired global behavior is likely to emerge even if some individuals of the colony die or fail to carry out their task for some other reasons. We consider a similar approach to coordinate a group of robots without a central supervisor, using only local interactions between the robots. When this decentralized approach is used, much of the communication overheads (characteristic of centralized systems) are saved, the hardware of the robots can be fairly simple and better modularity is achieved. A properly designed system should achieve reliability through redundancy.

Significant research effort has been invested during the last few years in design and simulation of multi-agent systems (Brooks 1990; Steels 1990; Levi 1992; Sen et al. 1994; Arkin and Balch 1998; Wagner et al. 1998; Wagner and Bruckstein 2001). Such designs are often inspired by biology (see Brooks 1986; Arkin 1990; Arkin and Balch 1997 for behavior-based control models, Deneubourg et al. 1990; Drogoul and Ferber 1992; Mataric 1992 for flocking and dispersing models or Benda et al. 1985; Haynes and Sen 1986 for predator–prey approach). Inspiration has also been found in physics (e.g. Chevallier and Payandeh 2000) and the application of economics (see e.g. Smith 1980; Golfarelli et al. 1997; Wellman and Wurman 1998; Rabideau et al. 1999; Thayer et al. 2000; Gerkey and Mataric 2002; Zlot et al. 2002).

Tasks that have been of particular interest to researchers in recent years include synergetic mission planning (Alami et al. 1997), fault tolerance (Parker 1998), swarm control (Mataric 1994), human design of mission plans (MacKenzie et al. 1997), role assignment (Stone and Veloso 1999; Candea et al. 2001; Pagello et al. 2003), multi-robot path planning (Lumelsky and Harinarayan 1997; Ferrari et al. 1998; Svestka and Overmars 1998; Yamashita et al. 2000), formation generation (Arai et al. 1989; Fredslund and Mataric 2001; Gordon et al. 2003), traffic control (Premvuti and Yuta 1990), forma-

The International Journal of Robotics Research
Vol. 27, No. 1, January 2008, pp. 127–151
DOI: 10.1177/0278364907085789

Figure 16 appears in color online: http://ijr.sagepub.com

tion keeping (Wang 1989; Balch and Arkin 1998), exploration and mapping (Rekleitis et al. 2003), target tracking (Parker and Touzet 2000) and target search (LaValle et al. 1997).

Unfortunately, the geometrical theory of such multi-agent systems is far from being satisfactory, as has been pointed out in Beni and Wang (1991) and many other articles. A short discussion regarding the conceptual framework of intelligent swarms (or *swarm intelligence systems*) and the motivation behind the use of such appears in Section 2.

Our interest is focused on developing the mathematical tools necessary for the design and analysis of such systems. In a recent report (Wagner and Bruckstein 1994) we have shown that a number of agents can arrange themselves equidistantly in a row via a sequence of linear adjustments, based on a simple 'local' interaction. The configuration convergence is exponentially fast. A different way of cooperation between agents is inspired by the behavior of ant colonies, and is described by Bruckstein (1993). There it was proved that a sequence of ants engaged in deterministic chain pursuit will find the shortest (i.e. straight) path from the anthill to the food source, using only local interactions. In Bruckstein et al. (1994) we investigate the behavior of a group of agents on $\mathbf{Z}^2$, where each ant-like agent is pursuing his predecessor according to a discrete biased random-walk model of pursuit on the integer grid. We proved that the average paths of such a sequence of a(ge)nts engaged in a chain of probabilistic pursuit converge to the straight line between the origin and destination, and this happens exponentially fast.

In this paper we investigate a more complicated question concerning the behavior of many agents cooperating to explore an unknown area (for purposes of cleaning, painting, etc.), where each robot/agent can only see its near neighborhood. The only method of inter-robot communication is by leaving traces on the common ground and sensing the traces left by other robots. See Min and Yin (1998); Acar et al. (2001); Butler et al. (2001); Koenig and Liu (2001); Polycarpou et al. (2001); Batalin and Sukhatme (2002); Latimer et al. (2002); Rekleitisy et al. (2004) for similar research.

We present an algorithm for cleaning a dirty area that guarantees task completion (unless *all* robots die) and prove an upper bound on the time complexity of this algorithm. We also show simulation results of the algorithm on several types of regions. These simulations indicate that the precise time depends on the number of robots, their initial locations and the shape of the region.

In the spirit of Braitenberg (1984), we consider simple robots with only a bounded amount of memory (i.e. finite-state machines). Generalizing an idea from computer graphics (see Henrich 1994), 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 fault-tolerance. Even if almost all the agents cease to work before completion, the remaining agents will eventually complete the mission. We prove the correctness of the algorithm as well as an upper bound on its running time, and show how our algorithm can be extended for regions with obstacles.

The rest of the paper is organized as follows. In Section 3 the formal problem is defined as well as the aims and assumptions involved. The algorithm is presented in Section 4, and an analysis of its time-complexity is described in Section 5. Section 6 is devoted to the problem of exploring/cleaning regions with obstacles, and is followed by several experimental examples (Section 7). We conclude the paper with a discussion and by highlighting several connections to related work, presented in Section 8.

This paper is a revised and extended version of the work presented in Wagner and Bruckstein (1997).

2. Swarm Intelligence: Framework and Motivation

2.1. Motivation

Robotics solutions to increasingly complex and varied challenges are growing in demand. Experience has dictated that a single robot is no longer the best solution for many application domains. Instead, teams of robots must coordinate intelligently for successful task execution.

Dias and Stentz (2001) present a detailed description of multi-robot application domains and demonstrate how multi-robot systems can be more effective than a single robot in many of these domains. However, when designing such systems it should be noticed that simply increasing the number of robots assigned to a task does not necessarily improve the system's performance. Multiple robots must cooperate intelligently to avoid disturbing each other's activity and achieve efficiency.

There are several key advantages of the use of such *intelligent swarm robotics*. First, such systems inherently enjoy the benefit of parallelism. In task-decomposable application domains, robot teams can accomplish a given task more quickly than a single robot by dividing the task into sub-tasks and executing them concurrently. In certain domains, a single robot may simply be unable to accomplish the task on its own (e.g. carrying a large and heavy object). Second, decentralized systems tend to be, by their very nature, much more robust than centralized systems (or systems comprised of a single complex unit). Generally speaking, a team of robots may provide a more robust solution by introducing redundancy and by eliminating any single point of failure. While considering the alternative of using a single sophisticated robot, we should note that even the most complex and reliable robot may suffer an unexpected malfunction which will prevent it from completing its task. When using a multi-agents system, on the other hand, even if a large number of the agents stop working for some reason the entire group will often be able to complete its task (albeit slower). For example, for exploring a hazardous region (such as a minefield or the surface of Mars), the benefit of redundancy and robustness offered by a multi-agents system is highly noticeable.

Another advantage of the decentralized swarm approach is the ability of dynamically reallocating sub-tasks between the swarm's units, thus adapting to unexpected changes in the environment. Furthermore, since the system is decentralized it can respond relatively quickly to such changes due to the benefit of locality. This implies it has the ability to swiftly respond to changes without the need to notify a hierarchical chain of command. Note that as the swarm becomes larger, this advantage becomes increasingly noticeable.

In addition to the ability of quick response to changes, the decentralized nature of such systems also increases their scalability. The scalability of multi-agents systems is derived from the low overhead (both in communication and computation) such system possess. As the tasks assigned nowadays to multi-agents based systems become increasingly complex, so does the importance of the high scalability of the systems.

Finally, by using heterogeneous swarms, even more efficient systems could be designed thanks to the utilization of *specialists* or agents whose physical properties enable them to perform much more efficiently at certain well-defined tasks.

2.2. Simplicity of the Agents

A key principle in the notion of swarms, or multi-agent robotics, is the simplicity of the agents. The notion of 'simplicity' here means that the agents should be *significantly* simpler than a single sophisticated agent which can be constructed for the same purpose. As a result, the resources of such simple agents are assumed to be very limited, with respect to the following aspects.

- Memory resources: Basic agents should be assumed to contain only $O(1)$ memory resources (i.e. the size of memory is independent of the size of the problem or the number of agents). This usually imposes many interesting limitations on the agents. For example, agents can remember only a limited history of their activities so far. Thus, protocols designed for agents with such limited memory resources are usually very simple and attempt to solve a given problem by defining some (necessarily local) basic patterns. The task is completed by a repetition of these patterns by a large number of agents.

- Sensing capabilities: Defined according to the specific nature of the problem. For example, for agents moving along a 100×100 grid, a reasonable sensing radius may be 3 or 4 but certainly not 40.

- Computational resources: Although agents are assumed to employ only limited computational resources, a formal definition of this constraint is hard to define. In general, most of the time-polynomial algorithms may be used.

Another aspect of the simplicity of swarm algorithms is the extremely sparse use of communication. The issue of communication in multi-agents systems has been extensively studied in recent years. Distinctions between implicit and explicit communication are usually made. Implicit communication occurs as a side effect of other actions, or 'through the world' (see, for example Pagello et al. 1999), whereas explicit communication is a specific act intended only to convey information to other robots on the team.

Explicit communication can be performed in several ways, such as a short range point-to-point communication, a global broadcast or by using some sort of distributed shared memory. Such memory is often treated as a *pheromone*, used to convey small amounts of information between the agents (Wagner and Bruckstein 2000; Yanovski et al. 2001, 2003). This approach is inspired from the coordination and communication methods used by many social insects. Studies on ants (e.g. Adler and Gordon 1992; Gordon 1995) show that the pheromone-based search strategies used by ants in foraging for food in unknown terrains tend to be very efficient.

In the spirit of creating a system which uses agents of as simple a design possible, we aspire that the agents will have few communication capabilities. With respect to the taxonomy of multi-agents discussed in Dudek et al. (1996), we would be interested in using agents of the types *COM-NONE* or if necessary of type *COM-NEAR* with respect to their communication distances, and of types *BAND-MOTION*, *BAND-LOW* or even *BAND-NONE* (if possible) with respect to their communication bandwidth.

Although a certain amount of implicit communication can hardly be avoided (due to the simple fact that by changing the environment, the agents are constantly generating some kind of implicit information), explicit communication should be strongly limited or avoided altogether in order to fit our paradigm. Note that this is not the case in many works in this field and communication, as well as memory resources, are often being used in order to create complex cooperative systems.

3. The Cooperative Cleaners Problem

3.1. Definition of Problem

We shall assume that time is discrete. Let G denote a two-dimensional integer grid $\mathbf{Z}^2$, whose vertices have a binary property referred to as *dirtiness*. Let $dirt_t(v)$ represent the dirtiness state of the vertex v at time t, taking either the value *on* or *off*. 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 connected component. Our algorithm will preserve this property along its evolution.

Let a group of k agents that can move across the grid G (moving from a vertex to its neighbor in one time step) be placed at time t_0 on F_0.

Each agent is equipped with a sensor capable of determining the *dirtiness* status of all tiles in the digital sphere of diameter 5, surrounding 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.

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

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

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

In this work we impose the restriction of no central control and full 'de-centralization', 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 agents will eventually complete the mission if possible.

3.2. Related Work

The cooperative cleaners problem has significant similarity to other types of multi-agent problems such as cooperative coverage or cooperative de-mining problems. In recent years, much work has been done designing systems and algorithms for handling these tasks. In this process, various models were used and assumptions made concerning the agents and their capabilities.

In general, most of the techniques used for the task of a distributed coverage use some sort of cellular decomposition. For example, in Rekleitisy et al. (2004) the area to be covered is divided between the agents based on their relative locations. In Butler et al. (2001) 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 Acar et al. (2001), who discuss two methods for cooperative coverage: probabilistic and complete (using yet again 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 requires the use of memory resources to store the dynamic map generated and 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 with 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, defined in advance, and unrelated to the size and geometry of the covered (or cleaned) area (Section 3.1). 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. Acar et al. 2001; Butler et al. 2001; Batalin and Sukhatme 2002), most of them lack analytic bounds for the completion time (although in many cases an extensive amount of empirical data of this nature is available). 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. In this work, results bounding the cleaning time of the agents are presented in Section 5. Interesting research to mention in this scope is this of Koenig et al. (2001) and Svennebring and Koenig (2004), 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; analytic bounds for the coverage time are discussed.

As most coverage problems focus on achieving a complete coverage (sometimes in the minimal time) it is worth mentioning in this scope the work of Conner et al. (2005), in which an interesting additional constraint is added. As this work was designed for an autonomous painting system, 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 3D coverage, in contrast to most search/cover systems usually discussing two dimensions. However, the main focus of this work was the use of an efficient single agent for this mission as opposed to cooperative multi-agents.

4. The Cleaning Protocol

In order to solve the Cooperative Cleaners problem we propose a cleaning protocol referred to as CLEAN. Generalizing a computer graphics idea (presented in Henrich 1994), 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 Section 5.1). 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 shape. As a result, the agents

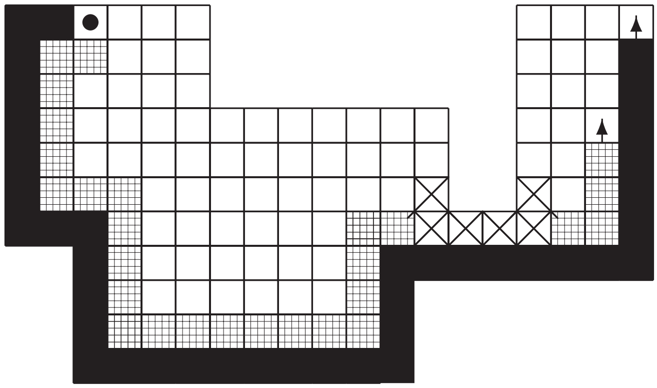

Fig. 1. An example of two agents using the CLEAN protocol.

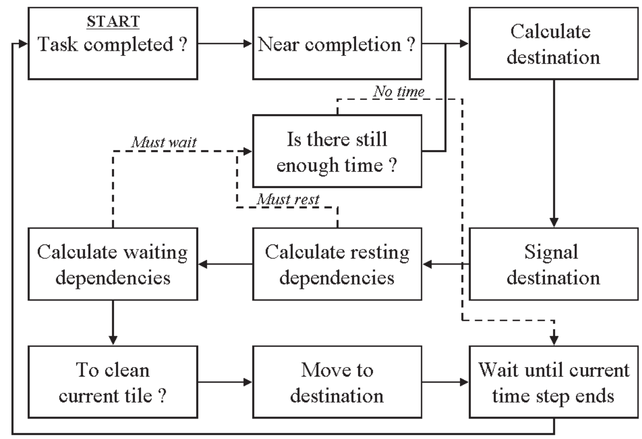

Fig. 2. A schematic flow chart of the CLEAN protocol.

'peel' layers from the shape while preserving its connectivity until the shape is cleaned entirely. An example of the protocol can be seen in Figure 1. Two agents using the CLEAN protocol are depicted at time step 40. All the tiles presented were dirty at time 0. The black dot denotes the starting point of the agents and the X's mark the *critical points* which are not cleaned. The black tiles are the tiles cleaned by the first agent, while the second layer of marked tiles represent the tiles cleaned by the second agent.

There are several exceptions to the basic description of the protocol given above. As the agents are equipped with very limited sensing and storage capabilities, the basic structure of the protocol is 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 other specific agents complete their cleaning process for this time step) or causing some agents to pause for a single time step. A detailed description concerning the need for these additions appears in Sections 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 problem, do not require explicit communication between the agents and can be implemented using a constant amount of memory resources. A schematic flowchart of the protocol is presented in Figure 2. 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 parts (5) and (6) of Section 4.2.2).

4.1. Cleaning Protocol: Definitions and Requirements

Let $r(t) = (\tau_1(t), \tau_2(t), \ldots, \tau_k(t))$ denote the locations of the k agents at time t. For a tile v, let Neighborhood(v) denote the dirtiness states of v and its 8 Neighbors. Let $\mathcal{M}_i$ denote some finite amount of memory contained in agent i, storing information needed for the protocol (e.g. the last move of agent i). The requested cleaning protocol is therefore a rule f such that for every agent i

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

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

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

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

The requested rule f should meet the following conditions.

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

4.2. The CLEAN Cleaning Protocol

The protocol is being used by each agent a_i, which at time t is located at $\tau_i(t) = (x, y)$. Definitions of several terms related to the CLEAN protocol follow.

The term 'rightmost' can be defined by considering the following. Starting from $\tau_i(t-1) = (x, y)$ (which is the previous boundary tile that agent a_i had been in) scan the neighbors of

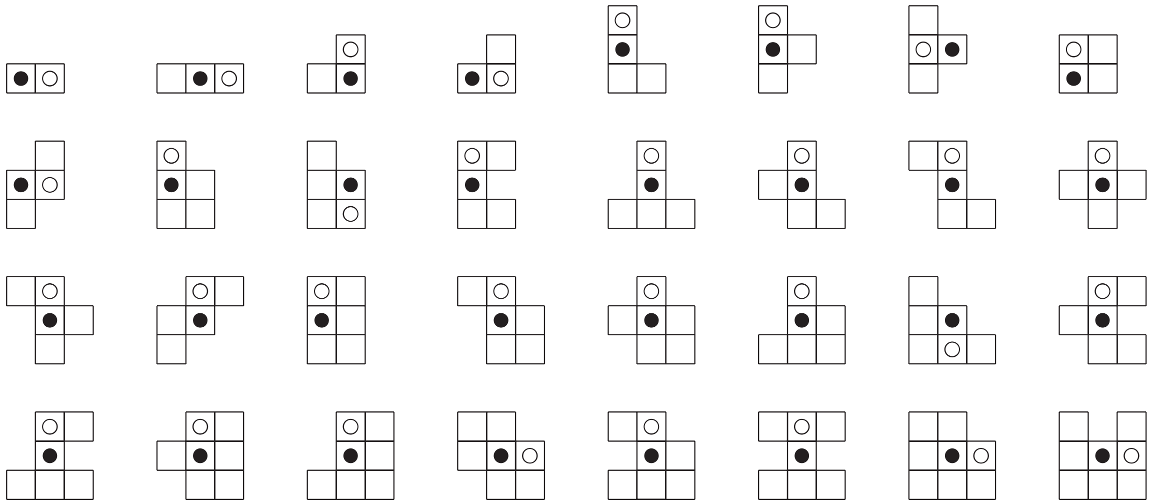

Fig. 3. When $t = 0$ the first movement of an agent located in (x, y) should be decided according to initial dirtiness status of the neighbors of (x, y), as appears in these charts. The agent's initial location is marked with a filled circle while the destination is marked with an empty one. All configurations that do not appear in these charts can be obtained by using rotations and mirroring.

(x, y) in a clockwise order until another boundary tile is found. In case $t = 0$ select the tile as instructed in Figure 3.

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 Figure 2. The complete pseudocode of the protocol and its sub-routines appears in Sections 4.2.1 and 4.2.2. Upon initialization of the system, the *System Initialization* procedure is called (defined in Section 4.2.1). This procedure sets various initial values of the agents and calls the protocol's main procedure *CLEAN* (Section 4.2.2). This procedure uses various sub-routines and functions defined in Section 4.2.1.

4.2.1. Initial part of the CLEAN cleaning protocol

The terms ∂F and $\tau_i = (x, y)$ are defined in Sections 4.1 and 4.2.

- **System Initialization** – initializing the system at the beginning of the agents operation.
  1. Arbitrarily choose the initial *pivot point* p_0 from all the points of ∂F_0.
  2. Artificially mark p_0 as a *critical point*.
  3. Place all the agents on p_0.
  4. For $(i = 1; i \leq k; i++)$ do
     (a) Call **Agent Reset** for agent i.
     (b) Call **CLEAN** for agent i.
     (c) Wait two time steps.
  5. End **System Initialization**.
- **Agent Reset** – resetting counters and flags.
  1. *resting* $\leftarrow$ *false*.
  2. *destination* $\leftarrow$ null.
  3. *cleaned in last tour* $\leftarrow$ *false*.
  4. *waiting* $\leftarrow \emptyset$.
  5. End **Agent Reset**.
- **Priority**.
  1. Assume the agent moved from tile (x_0, y_0) to tile (x_1, y_1) then
     (a) *priority* $\leftarrow 2(x_1 - x0) + (y_1 - y0)$.
  2. End **Priority**.
- **Check Completion of Mission**.
  1. If $((x, y) = p_0)$ and (x, y) has no dirty neighbors then
     (a) If (x, y) is dirty then
        i. Clean (x, y).
     (b) **STOP**.
  2. End **Check Completion of Mission**.
- **Check 'Near Completion' of Mission**: identify scenarios in which there are too many agents blocking each other, preventing the cleaning of the last few dirty tiles.

1. If (this is the first time this function is called for this agent) then
   (a) Jump to 3.
2. If ($(x, y) = p_0$) and (this is the end of a tour) then
   (a) If (*cleaned in last tour* = *false*) then
      i. *resting* ← *true*.
      ii. **STOP**.
   (b) Else
      i. *cleaned in last tour* ← *false*.
3. End **Check 'Near Completion' of Mission**.

4.2.2. The CLEAN cleaning protocol

The term *rightmost neighbor* is defined in section 4.2.

- **CLEAN Protocol** – controls agent number i actions after **Agent Reset** and until the mission is completed.
  1. **Check Completion of Mission**.
  2. **Check 'Near Completion' of Mission**.
  3. Calculate destination of movement.
     (a) *destination* ← *rightmost neighbor* of (x, y).
  4. Signaling the desired destination.
     (a) *destination signal bits* ← *destination*.
  5. Calculate resting dependencies – solves the agents clustering problem.
     (a) Let all agents in (x, y) except agent i be divided into the following four groups:
        – A_1: Agents signaling towards any direction different than *destination*.
        – A_2: Agents signaling towards *destination* which entered (x, y) before agent i.
        – A_3: Agents signaling towards *destination* which entered (x, y) after agent i.
        – A_4: Agents signaling towards *destination* which entered (x, y) in the same time step as agent i.
     (b) Let group A_4 be divided into the following two groups:
        – A_{4a}: Agents with lower *priority* than of agent i.
        – A_{4b}: Agents with higher *priority* than of agent i.
     (c) *resting* ← *false*.
     (d) If ($A_2 \neq \emptyset$) or ($A_{4b} \neq \emptyset$) then
        i. *resting* ← *true*.
        ii. If (current time step did not end yet) then jump to 3 else jump to 9.
  6. Calculate waiting dependencies – takes care of agents synchronization.
     (a) *waiting* ← $\emptyset$.
     (b) If $(x - 1, y) \in F_t$ and contains a *non-resting* agent which didn't move in the current time step yet then *waiting* ← *waiting* ∪{*left*}.
     (c) If $(x, y - 1) \in F_t$ and contains a *non-resting* agent which didn't move in the current time step yet then *waiting* ← *waiting* ∪ {*down*}.
     (d) If $(x - 1, y - 1) \in F_t$ and contains a *non-resting* agent which didn't move in the current time step yet then *waiting* ← *waiting* ∪{*left-down*}.
     (e) If $(x + 1, y - 1) \in F_t$ and contains a *non-resting* agent which didn't move in the current time step yet then *waiting* ← *waiting* ∪{*right-down*}.
     (f) If *destination* = *right* and tile $(x + 1, y)$ contains an agent j, and $destination_j$ ≠*left*, and there are no other agents delayed by the operating agent (i.e. tile $(x-1, y)$ does not contain an agent l where $destination_l$ = *right* and tile $(x, y+1)$ does not contain any agents and tile $(x + 1, y)$ does not contain an agent n where $destination_n$ = *left*), then (*waiting* ← *waiting* ∪ {*right*}) and ($waiting_j$ ← $waiting_j$ \ {*left*}).
     (g) If *destination* = *up* and tile $(x, y + 1)$ contains an agent j, and $destination_j$ ≠ *down*, and there are no other agents delayed by the operating agent (i.e. tile $(x, y - 1)$ does not contain an agent l where $destination_l$ = *up* and tile $(x+1, y)$ does not contain any agents and tile $(x, y + 1)$ does not contain an agent n where $destination_n$ = *down*), then (*waiting* ← *waiting* ∪ {*up*}) and ($waiting_j$ ← $waiting_j$ \ {*down*}).
     (h) If (*waiting* ≠ $\emptyset$) then
        i. If (current time step did not end yet) then jump to 3 else jump to 9.
  7. Decide whether to clean (x, y) (meaning: whether or not the tile is a *critical point*).
     (a) If (x, y) has two dirty tiles in its 4 Neighbors which are not connected via a sequence of dirty tiles from its 8 Neighbors then
        i. (x, y) is a *critical point* and should not be cleaned.
     (b) Else
        i. If (x, y) still has other agents in it then Do not clean (x, y).

ii. Else

A. Clean (x, y).

B. *cleaned in last tour* $\leftarrow$ *true*.

8. Move to *destination*.
9. Wait until the current time step ends.
10. Return to 1.

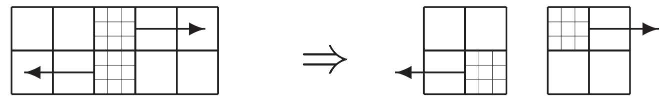

Fig. 4. When the agents do not possess a synchronization mechanism, they may damage the region's connectivity. In this example, two agents clean their current locations and move according to the CLEAN protocol.

4.3. Pivot Point

This initial location of the agents (denoted 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 dirty neighbors to go to, thereby reporting on completion of their individual missions. Note that this point (also called *pivot point*) is not necessary for the algorithm to work well; it simply makes life easier for the user since one knows where to find the agents after cleaning has been completed. If the agents are not placed on the pivot point at $t = 0$ and the pivot p_0 is not forced to be critical, then the location of the agents upon completion will generally not be known in advance.

4.4. Signaling

Since each agent's sensors can determine the status of all tiles which are contained within a digital sphere of radius 4 located around the current location of the agent, each agent can artificially calculate the desired destination of all the agents which are located in one of its 4 Neighbors tiles. 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 for the sake of simplicity and ease of understanding.

4.5. Connectivity Preservation

The connectivity of the region yet to be cleaned, F, is preserved by allowing an agent to clean only *non-critical* points. This guarantees both successful termination and agreement of completion (since having no dirty neighbors implies that $F = \emptyset$). 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. 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 agent to exit cleans it (see part (7(b)iA) of CLEAN) in order to prevent agents from being 'thrown' out of the dirty region (i.e. cleaning a tile in which another agent is located, 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 Baeza-Yates and Schott 1995). 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 method of synchronization which is mandatory in order to prevent agents from operating at the same time, damaging the connectivity of the dirty region as shown in Figure 4. Since the agents in the example of Figure 4 are not synchronized, the tiles in which they are located are not treated as critical at the time of cleaning, but the region's connectivity is not preserved. If one agent had waited for the other to complete its actions, it would not have cleaned its current tile and the region's connectivity would have been maintained.

To ensure such scenarios will not occur, an order between the operating agents must be implemented. Throughout the following paragraphs, agents signalling a *resting* status (Section 4.7) are disregarded while calculating the dependencies of the agents' movements. The following suggested order is implemented in part (6) of CLEAN.

For agent i, let $P_i \subseteq \{up, down, left, right\}$ be a set of tiles in which there are currently delayed agents i.e. 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 $P_i \leftarrow P_i \cup \{down\}$.

Let $\text{dest}_i \in \{up, down, left, right\}$ be the destination that agent i is interested in moving to after cleaning its current location.

Let each agent be equipped with a two bits flag (implemented by two small light bulbs, for example). This flag will state 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 to 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*, there are several specific scenarios in which this asymmetry should be reversed in order to ensure a proper operation of the agents. Such a 'dependencies-switching' rule is described as follows.

For agent i which is located in (x, y), if $\text{dest}_i = right$ and tile $(x+1, y)$ contains an agent j, $\text{dest}_j \neq left$ 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 $\text{dest}_l = right$, tile $(x, y + 1)$ does not contain any agent and tile $(x + 1, y)$ does not contain an agent n where $\text{dest}_n = left$), then $P_i \leftarrow P_i \cup \{right\}$ and $P_j \leftarrow P_j \setminus \{left\}$. Also, if $\text{dest}_i = up$ and tile $(x, y+1)$ contains an agent k, $\text{dest}_k \neq down$ 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 $\text{dest}_m = up$, tile $(x + 1, y)$ does not contain any agent and tile $(x, y + 1)$ does not contain an agent q where $\text{dest}_q = down$), then $P_i \leftarrow P_i \cup \{up\}$ and $P_k \leftarrow P_k \setminus \{down\}$.

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

Note that deadlocks are impossible. Since the basic rule is that every agent is delayed by the agents in its *left* and *down* neighbor tile, at any given time among the agents who have not yet moved in the current time step, there is always the agent with the minimal x and y coordinates which is able to move. After this agent moves, all the agents which have been delayed by it can move and so on. Regarding the 'dependencies-switching rule', let agent i located in tile (x, y) have the minimal x and y values among the agents who have not yet moved, let $\text{dest}_i = up$ and let tile $(x, y + 1)$ contain an agent j such that $\text{dest}_j \neq down$.

Although agent i is located below agent j, it will be delayed by it ($up \in P_i$ and $\neq (down \in P_j)$) as long as agent i is not delaying any other agents since this is the requirement of the 'dependencies-switching rule'. In this case, we should show that there cannot be a circle 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 cannot 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 not delaying any other agents (specifically agent k) was a strict requirement of the 'dependencies-switching rule' which allowed agent i to delay agent j. Thus, a dependencies circle is prevented and no deadlock is possible.

Note that while phase 2 is in process, F_t may change due to the cleaning of 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 constant recalculation of these values, for every *waiting* agent. For example, assume agent i to be located in (x, y) and $\text{dest}_i = down$ without loss of generality, and that agent j located in tile $(x, y - 1)$ moves out of this tile and cleans it. Then, dest_i naturally changes (as the tile (x, y) no longer belongs 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 CLEAN by calculating the waiting agents' destinations and dependencies lists repeatedly, until either all agents have moved or until the time step has ended implying that some agents had to change their status to *resting* and pause until the next time step: see Section 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 as demonstrated in Figure 5.

4.7. Clustering Problem

Since we are interested in preventing the agents from moving together and functioning as a single agent (due to the 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 reasons of performance and analysis.

The following resting policy is implemented by part (5) of CLEAN. Using its sensors, an agent intending to move into tile v is aware of other agents which also intend to move into v. 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 or if they had occupied this tile before then.

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

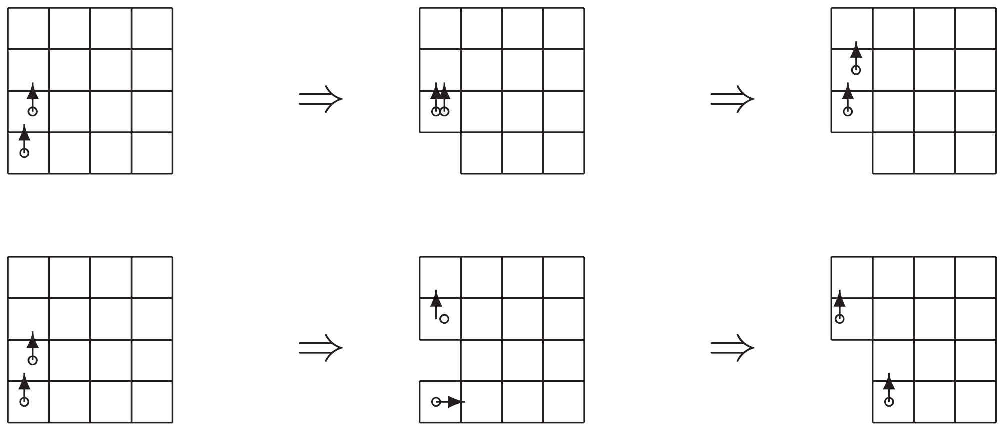

Fig. 5. The upper charts demonstrate a performance bug which may be caused due to the local dependencies. The agents are advancing according to the CLEAN protocol, but their cleaning performance is decreased. The lower charts demonstrate the cleaning operation after adding the dependencies-switching mechanism.

From the agents which intend to move in the same direction as agent i, the agent can distinguish between three groups of agents: the agents which entered the tile before the time step agent i (group A_2), those which entered the tile during the same time step as i (group A_4) and those who entered it after agent i (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 will wait for agent i to move before being free to do so as well.

Regarding the agents in A_4, the problem is solved by induction over the time steps using the assumption that two agents cannot leave the same tile towards the same direction in the same time step. This of course holds for the first time steps, as the agents are released manually by the CLEAN protocol's initialization procedure. Later, as we are assured that all the agents of group A_4 arrived at v from different tiles (which are also different from the tile agent i had entered v from), a consensus over a local ordering of A_4 is established (note that no explicit communication is needed to form this ordering) since all the agents in group A_4 know the previous locations of each other. An example of such an order is the *priority* function of the CLEAN 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.

4.8. Mission Termination

When $k > 1$ there may evolve scenarios in which k will be greater then $|F_t|$, the number of dirty tiles. In such a case, the agents may also be arranged in such a way that at any given time there is at least one agent in each tile. From such a scenario, agents may switch places without being able to clean a single tile since only the agent which is the last to exit a tile may clean it. The result of this situation would be a group of agents running around indefinitely without ever completing their cleaning mission. In order to prevent this scenario, the CLEAN protocol instructs any agent which reaches the pivot point and has not cleaned a single tile in its last traversal of F to halt (see *near completion of mission* procedure of CLEAN).

If an agent has not cleaned a tile during its last traversal of the shape, it means that either the shape is a 'skeleton' comprising of only critical points (see the proof of Theorem 9 in Section 5.3) or that $k' > |F_t|$ (k' denoting the currently active agents). In either case, halting this agent does not slow the completion of the cleaning task. It is easy to see that if the number of active agents outside p_0 is always kept smaller than $|F_t|$ then the size of the shape will always be decreased until the shape is entirely cleaned.

5. Analysis

The introduction of the notion of critical points makes time-complexity analysis of the CLEAN protocol significantly more difficult since a critical point may be visited several times before its cleaning. We propose that the algorithm is efficient, and that additional agents will only speed up the cleaning process to a certain limit. In order to give this argument a rigorous form, some definitions are required.

5.1. Definitions

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

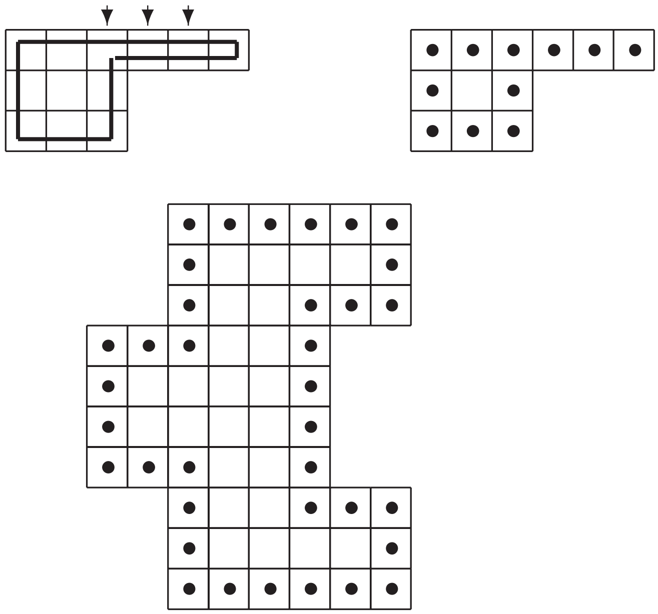

Fig. 6. The line in the upper left chart goes through the tiles of C_t where the three arrows denote tiles that are included twice. The circles in the upper right chart denote the tiles of ∂F_t. Note that while ∂F_t contains 11 tiles, C_t contains 14. In the lower chart, there are no *critical points* in ∂F_t and therefore $c_t = |\partial F_t| = |C_t| = 36$.

The boundary of the dirty region F is denoted as ∂F and defined as

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

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.

Let tile v be 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 8 Neighbors(v) necessarily pass through v (where v, v_1, v_2 and all said paths are from F).

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 the shortest path connecting v_0 and v_n which contains all the tiles of ∂F_t and only such tiles.

If there are several different shortest paths then let C_t be one of them. Note 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 is defined as the length of C_t. An example appears in Figure 6.

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, i.e. 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).

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

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

Figure 7 depicts three different shapes.

Let F'_t denote the region which could be accessed if one agent traversed F_t once using the CLEAN 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.

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, i.e.

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

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 CLEAN 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 CLEAN 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 this is the only dirty tile left. In addition, the various preemptions which are in charge of agent synchronization prevent an agent moving 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 3. *A group of agents executing the CLEAN protocol will eventually clean a simply connected region and stop together at the pivot point p_0.*

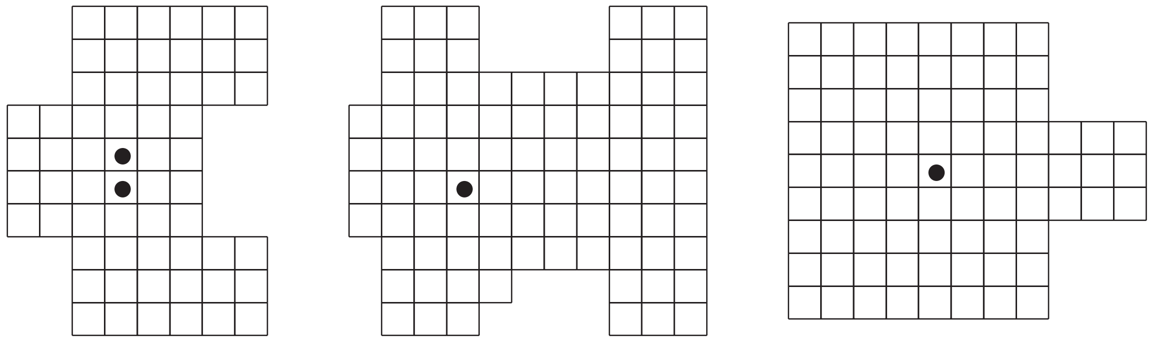

Fig. 7. The black circles denote the deepest tiles within the shapes. Note that while in the left shape there are two tiles of maximum depth, there is only one tile of maximum depth in the other two shapes. The widths of the shapes equal the depth of the deepest tiles, and are (from left to right) 2, 3 and 4.

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 CLEAN 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 from each one of its four neighbors, at most once), the maximum length of this traversal is $4s_t$. As agents may indeed delay one another, it is shown in Lemma 6 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 CLEAN algorithm.

- 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 when no other dirty points are left in F_t. □

5.3. Detailed Analysis

In this section, we shall discuss an upper bound for the cleaning time of k agents employing the CLEAN cleaning protocol for some dirty shape 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 shape in a given number of time steps can be derived.

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

Lemma 4. *The cardinality of the region's circumference always decreases after being traversed by an agent applying the CLEAN protocol, namely:*

$$|\partial F'| \leq |\partial F| - 8.$$

Proof. 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 travels in a straight line, makes an internal turn (*left*, if we assume a clockwise movement) or an external turn (*right*). Suppose w.l.o.g 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 twice as long as it contains an additional movement up and another towards the left. Similarly, an external turn creates a new path which is shorter by two tiles since a single horizontal movement and a single vertical movement are no longer necessary.

Since ∂F is a simple rectilinear polygon, it always has four more *right* turns than *left*. (This is a simple consequence of the 'rotation index' theorem (see e.g. Do-Carmo 1976 pp. 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.) 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 example in Figure 8). □

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

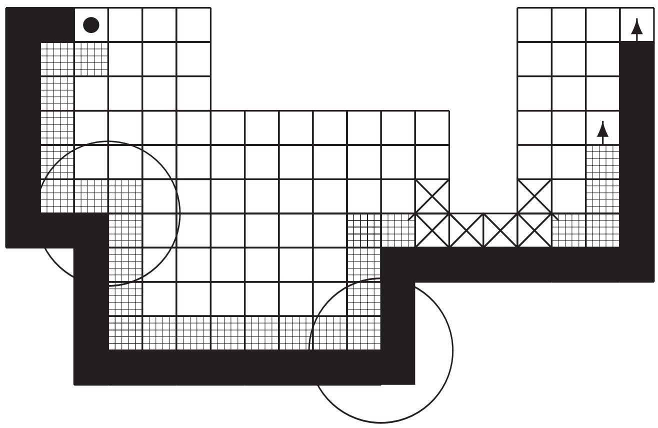

Fig. 8. An example of Lemma 4 and of the CLEAN protocol. The black dot denotes the starting point of the agents cleaning F (which is the entire shape presented). The X's mark the *critical points* which are not cleaned. The black tiles are some of the tiles of ∂F, cleaned by the first agent. The second 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 5. *Every time a region is traversed by an agent using the CLEAN protocol, its width is decreased by at least one, i.e.*

$$W(F_t') \leq W(F_t) - 1.$$

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

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 CLEAN 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. The *resting* mechanism (Section 4.7) prevents situations in which a certain agent is never leaving the tile it is located in. Therefore, the only time in which a tile might constantly contain agents is when the number of agents is greater than the number of tiles. The termination mechanism (Section 4.8) is in charge of detecting such scenarios and stopping the operation of agents until the number of active agents decreases.

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 shape was decreased by (at least) one. As a result, since 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 CLEAN 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 maximum time it takes an agent using the CLEAN protocol to move along it.

Lemma 6. *The time it takes an agent using the CLEAN protocol to move along a path of length c_t (including delays caused by other agents located in the same tiles) is at most $4 \times c_t$.*

Proof. According to the specifications of the problem, 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 CLEAN 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 reduction in the performance of the swarm.

Let v be an empty tile. Let us assume w.l.o.g. that at some time step t, 4 agents enter v (this is of course the maximum 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 4 agents which entered v intends to move in 4 different directions (since according to the CLEAN protocol, each of them has a 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 CLEAN 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 step $T > t + 2$. 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 $k/4$ 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 $4 \times k \times c_t$ (and the average traver-

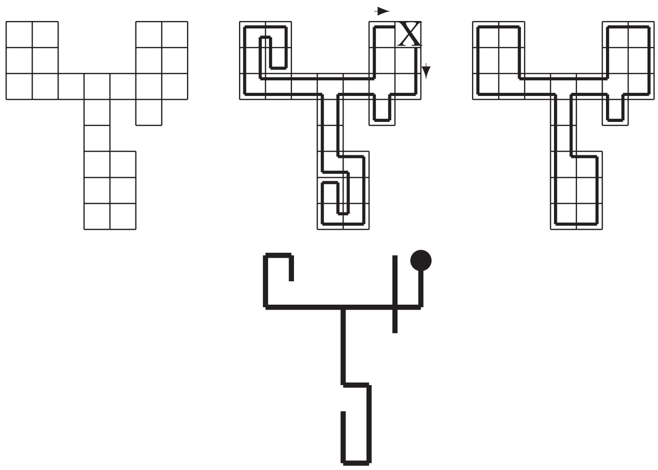

Fig. 9. The left chart represents the dirty region F_t. The middle chart shows the DFS scan of ∂F_t according to the SPAN-DFS protocol, where X marks p_s, the *non-critical* starting point. The right chart shows the traversal path which is contained in this DFS scan. The drawing on the bottom shows the spanning tree that is created by the protocol and is searched by the DFS algorithm.

sal time of an agent, amortized for the entire group of agents, is at most 4 times its minimal traversal time). 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 CLEAN protocol is traversing F_t it does so by moving along a path which contains all the tiles of ∂F_t. Since by doing this, the agent may pass in 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 7. *c_t, the length of the circumference of F_t, never exceeds twice its cardinality, i.e.*

$$c_t \leq 2 \times |\partial F_t| - 2.$$

Proof. ∂F_t is a connected graph and thus it has spanning trees. A protocol for constructing a spanning tree for ∂F_t and a *Depth First Search* (DFS) scan was constructed, such that the path generated using the DFS algorithm contains a traversal around F_t. The DFS scan generates a path which, after having several of its tiles removed, equals a path which traverses F_t. Figure 9 depicts an example of the spanning tree protocol.

The protocol (listed in Figure 10) receives p_s, a *non-critical* tile as its starting point and an empty list of tiles, L. (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.) The protocol constructs a spanning tree of ∂F_t as well as a DFS scan for this tree, by adding tiles to the list. The protocol also marks the tiles in L which should 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 Figure 9.

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 $2 \times (|\partial F_t| - 1)$ transitions of edges. Since there exists such a spanning tree that contains a 'tour' around F_t then $c_t \leq 2 \times |\partial F_t| - 2$. □

Let $W_{\text{REMOVED}}(t)$ denote the decrease in the width of F due to the agents' activities until time t.

Lemma 8.

$$W_{\text{REMOVED}}(t+1) \geq \left\lfloor \sum_{i=1}^{t} \frac{k}{4 \times c_i} \right\rfloor$$

Proof. $W_{\text{REMOVED}}(t)$ 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 $1/c_t$ 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, then the decrease in the original width of the shapes caused by the cleaning process of the agents is bounded as described. □

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

Theorem 9. *Assume that k agents start cleaning a simple connected region F_0 at some boundary point p_0 and work according to the CLEAN algorithm. Denote by $t_{\text{success}}(k)$ the time needed for this group to clean F_0. Then it holds that*

$$\max\left\{2k, \left\lceil \frac{s_0}{k} \right\rceil, 2l(F_0)\right\}$$
$$\leq \quad t_{\text{success}}(k) \leq \frac{8(|\partial F_0| - 1) \times (W(F_0) + k)}{k} + 2k.$$

Proof. The lower bound is quite obvious: the left term $2k$ is the time needed to release the k agents from the pivot point and 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 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 shape

Protocol **SPAN-DFS**(x, y):
Add (x, y) to L and mark it as **OLD**;
If the *rightmost* neighbor of (x, y) is p_s then
  Delete all the marked tiles from L and **STOP**;
If (the *rightmost* neighbor of (x, y) is marked as **OLD**) and (no circle was formed) then
  / The traversal repeats this tile as well */*
  Call **SPAN-DFS** for the *rightmost* neighbor of
  (x, y) and **STOP** upon return;
If (the *rightmost* neighbor of (x, y) is marked as **OLD**) and (a circle was formed) then
  / Continue, and clean the redundant tiles */*
  Add to L the sequence of tiles from L, starting
  with (x, y) and backwards to (x, y)'s *rightmost*
  neighbor;
  Mark these tiles (expect (x, y)) to be deleted;
  Call **SPAN-DFS** for the *rightmost* neighbor of
  (x, y) and **STOP** upon return;
/ The rightmost neighbor was not OLD */*
Call **SPAN-DFS** for the *rightmost* neighbor of (x, y) and **STOP** upon return;
End **SPAN-DFS**;

Fig. 10. The recursive spanning tree construction protocol. The term *rightmost* has the same meaning as in the CLEAN 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).

will be 0. Note that once the dirty region was reduced to a 'skeleton' comprising only tiles of depth zero, an additional traversal must be done in order to clean it entirely.

By combining Lemmas 4 and 7, we know that for $c_j(i)$ and $\partial F_j(i)$, the length and cardinality, respectively, of the ith traversal of agent j:

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

From Lemma 8, we see that

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

We are therefore interested in

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

where $+k$ represents the additional traversal required to clean the dirty 'skeleton'.

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

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

□

Using the above theorem we can bound the *efficiency ratio*, defined as $t_{\text{success}}(k)/s_0$ which expresses the benefit of using k agents for a cleaning mission.

Corollary 10.

$$\max\left\{\frac{2k}{s_0}, \frac{1}{k}, \frac{2l(F_0)}{s_0}\right\} \leq \frac{t_{\text{success}}(k)}{s_0} \leq \frac{8\,(|\partial F_0| - 1) \times (W(F_0) + k)}{k \times s_0} + \frac{2k}{s_0} \quad (3)$$

An interesting result of Corollary 10 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 $8|\partial F_0|/s_0$. Note here the similarity to the ratio $c_0/\sqrt{s_0}$, known as the *shape factor*.

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

Corollary 11.

$$\text{As } n \to \infty, \qquad L_\infty \le \frac{t_{\text{success}}(k)}{S} \le U_\infty$$

where

$$L_\infty = \frac{1}{k}, \quad U_\infty = 8\frac{\partial F \times W}{k \times S} = 8\frac{|\partial F_0| \times W(F_0)}{k \times s_0}$$

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 the robustness and fault-tolerance of an algorithm is a major aspect of robotics systems, as the environments in which they operate may often include unpredicted changes from the original specification of the system such as various kinds of noises and robot malfunctions.

As discussed in Section 4.1, one of the requirements of the CLEAN protocol is a complete fault-tolerance to malfunctions in the agents. This means that even if one or several agents stop working (die) the rest of the agents will continue the cleaning process as efficiently as their number allows them, and as long as there exist at least one agent which functions properly, the cleaning completion is guaranteed. The fulfillment of this requirement is immediately derived from the completeness of the algorithm for a group of agents of any size (and specifically, for a single agent) shown in Theorem 3. The basic idea enabling this robustness is the complete independence between the agents, due to the fully decentralized nature of the system. 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. Noise can generally occur in various aspects of the system, e.g. while sensing whether a tile should be cleaned, for the presence of other agents in an agent's vicinity or noise in the agent's movement system. As in almost any system, it is easy to see that an exaggerated noise level may significantly decrease the algorithm's efficiency and may even prevent the agents from completing 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 the event that one of these components does not have any agent located on its tiles, it will remain dirty even after the agents stop and report a successful termination of their mission. In addition, noises in the agents' movement system may cause the agents to detach from the dirty region, thus delaying or even preventing the completion of the cleaning process.

To keep the noise level at a reasonable rate such that the completion of the algorithm can still be guaranteed, cleaning time may be longer. This may happen when the agent suffers difficulties in the sensors in charge of detecting other agents in its vicinity, and interpreting their signaling (see relevant sections for more details about signaling and synchronization). Several problematic scenarios, which these mechanisms were designed to avoid, may then occur. However, simulations show that in such cases only the cleaning time is affected and not the completion of the cleaning itself. Nevertheless, as this issue has not yet been fully investigated, a zero noise level is still one of the algorithm's requirements.

One example for the above is described in Figure 4, in which a failure in the sensing mechanism may damage the simple connectivity of the dirty region. Note that in such scenarios, there is at least a single agent in each of the new dirty components. Therefore, each component is an instance of the cooperative cleaners problem where complete cleaning is guaranteed. However, the completion of the cleaning for all of those components may require longer time than would have been required for the original problem.

Another example is demonstrated in Figure 5, where a bug in the *waiting* mechanism (see Section 4.6 for more details) may prevent some of the agents from cleaning dirty tiles. In this example, the existence of at least a single fully functioning agent is guaranteed and, as a result, also the successful completion of the cleaning mission. The same holds for possible bugs in the *resting* mechanism (Section 4.7) 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 repositioned at some arbitrary location onboard the dirty region. As a result of the CLEAN protocol, the robots will navigate to the boundary of the region and resume their cleaning. The overall cleaning time might be affected; however, the completion of the cleaning is still guaranteed.

5.5. Comparison to Existing Work

Examining other work that has been done in the area of multi-robotic systems designed for cooperative cleaning/coverage, several analytic results concerning the completion time of the mission can be found.

An interesting result is presented in Svennebring and Koenig (2004). 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 Koenig and Szymanski (1999), the bound: "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" is given.

The covering time of k robots by $f(k)$ is given by

$$f(k) = O(2^{\sqrt{s_0}}). \qquad (4)$$

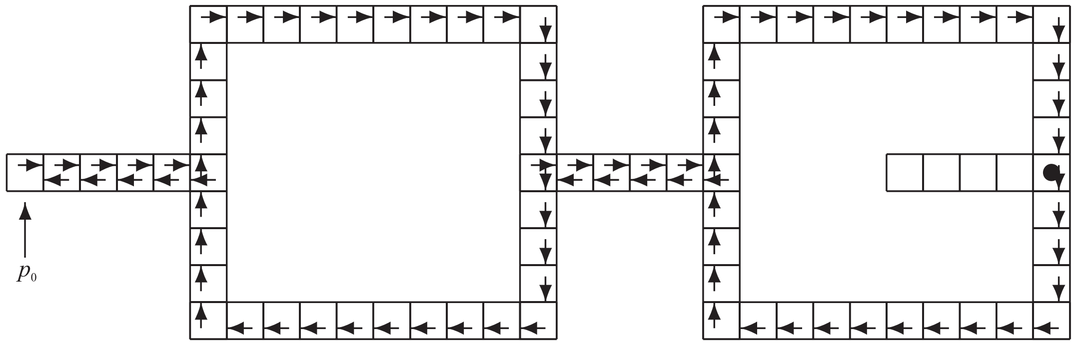

Fig. 11. An example of agents' odoring used in Section 6 for cleaning regions with obstacles. The figure demonstrates the labels on the region's tiles after an agent reached the *useless* state, in which it did not clean any tile in its last tour. Note that any of the tiles labeled with one direction 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 cannot be cleaned, since it is indeed a critical point. Such a point is marked in the figure with a black circle.

Recalling Theorem 9 in which

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

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

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

For practical reasons, we assume that $k < S_0$ so can see

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

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

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

Comparing this to the bound of equation (4), we see that the time complexity of the CLEAN protocol is much smaller than of the nodes counting technique.

It should be mentioned, however, that in Svennebring and Koenig (2004) 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 Svennebring and Koenig (2004) 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 CLEAN 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 CLEANing algorithm 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 adding an 'odoring' feature to our cleaners, i.e. 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 in which agents traveled through the labeled tiles.

Upon arriving at 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. It will now look for a point which has a single label, i.e. 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 Figure 11 for an example). This will open one cycle; hence, if there are m obstacles, we will need m/k such tours before the region is completely clean. On the other hand, Lemma 4 no longer holds (see Figure 12) 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 12. Assume that k agents start at some boundary point of a non-simply connected region F_0 with s obstacles, and work according to the CLEAN-WITH-OBSTACLES algorithm. Denote by $t_{\text{success}}(k)$ the time needed for this group to clean F_0. Then

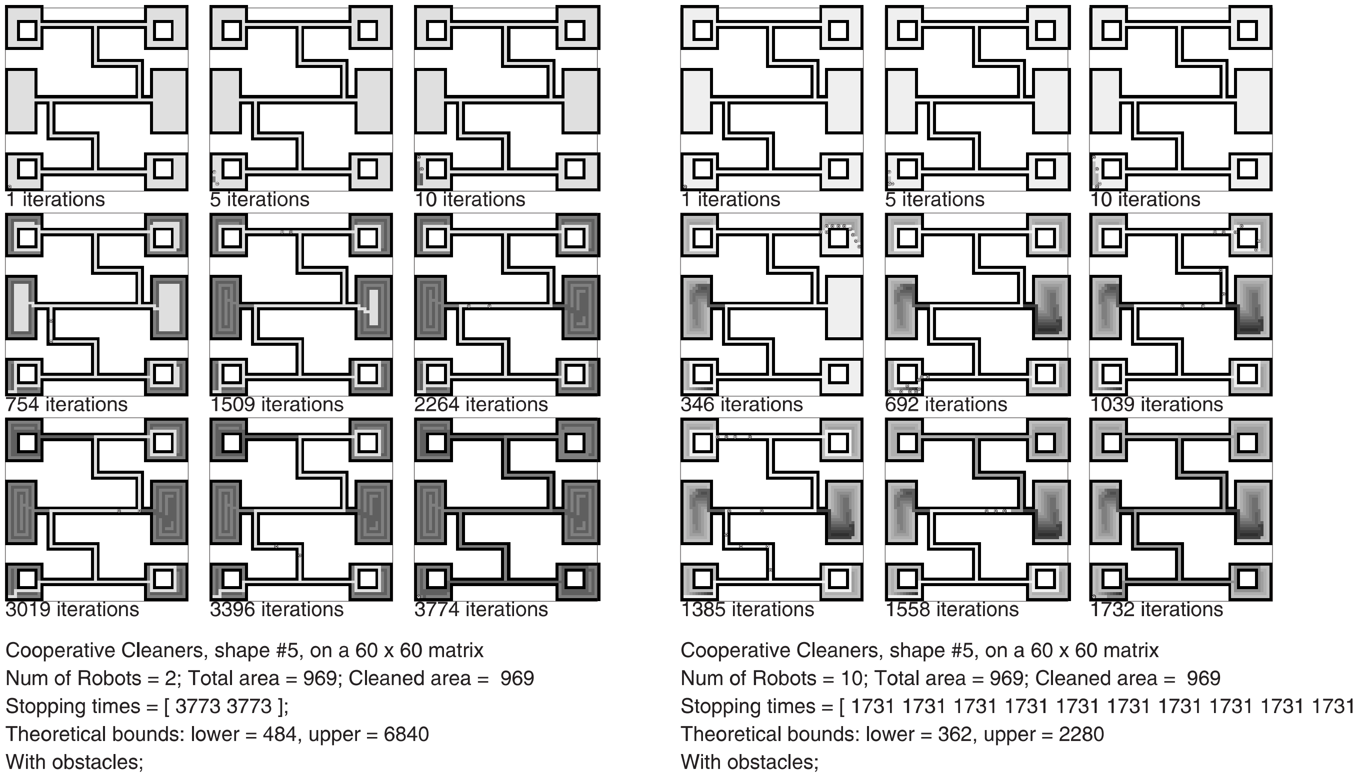

Fig. 12. Cooperative cleaners: two agents, six rooms with obstacles (left), ten agents, six rooms with obstacles (right).

$$\max\left\{2k, \left\lceil \frac{s_0}{k} \right\rceil, 2l(F_0)\right\}$$

$$\leq \quad t_{\text{success}}(k) \leq 8 \times s_0 \left(\frac{W(F_0)}{k} + \left\lceil \frac{s}{k} \right\rceil \right) + 2k \qquad (5)$$

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.

Note, however, that the completion of the cleaning process is still guaranteed.

7. Experimental Results

A computer simulation was constructed, implementing the CLEAN protocol. The algorithm was ran on several shapes of regions and for numbers of agents varying from 1 to 20. See Figures 12 and 13 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 Figure 13 shows the same region and number of agents as in the left side, 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 noted that all the theory we developed in the previous sections (up to 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 12 shows the evolution of the CLEAN-WITH-OBSTACLES algorithm for the same shapes with four additional obstacles in each, with 2 agents (left) and 10 agents (right).

Figure 14 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 Figure 15 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 results are presented in Figure 16, comparing the various efficiency ratios obtained for initial dirty regions of various sizes. This is examined even further by Figure 17, which shows the behavior of $t_{\text{success}}(50)/t_{\text{success}}(1)$, i.e. 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, implying that as the initial region is larger the addition of more agents can more efficiently decrease the cleaning time of the agents.

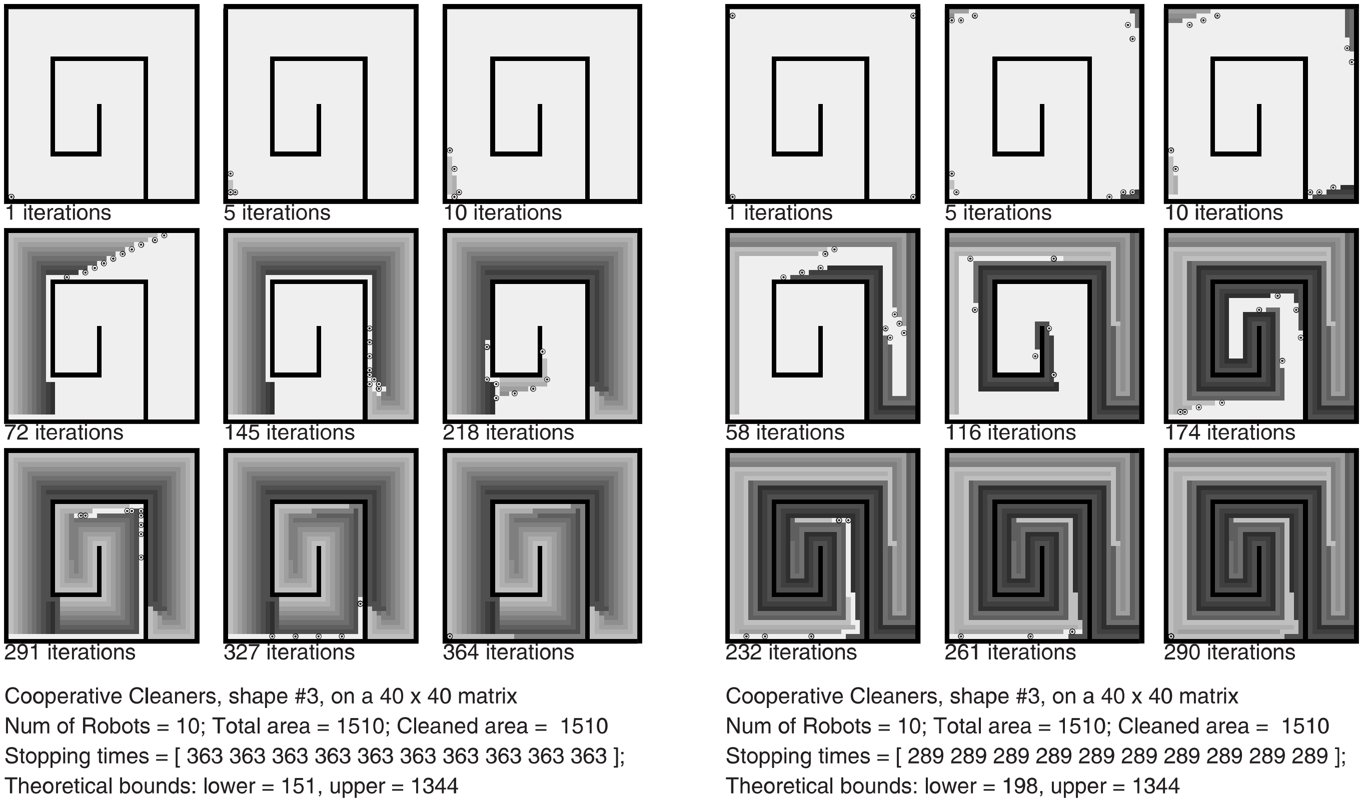

Fig. 13. Cooperative cleaners: maze, ten agents (left), and ten agents with random initial locations on the boundary (right).

8. Discussion and Conclusion

Our cooperative CLEANing algorithm can be considered as a case of social behavior in the sense of Shoham and Tennenholtz (1995), where one induces multi-agent cooperation by forcing the agents to obey some simple social guidelines. This raises the question of 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. In the CLEAN protocol we resolve such a problem by giving each agent a priority measure depending on its previous location. However, it is an interesting open question that simply flipping a coin might be better.

The cleaning problem discussed is related to the geometric problem of pocketmachining; see Held (1991) for details. An interesting problem of cleaning and maintenance of a system of pipes by an autonomous agent is discussed in Neubauer (1993). The importance of cleaning hazardous waste by agents is described in Hedberg (1995).

Our approach is that cleaning is always done at the boundary. It is possible that a better efficiency will be achieved using other approaches.

1. Given that several neighbors are dirty, visit the non-critical ones first (even if not on the boundary). This approach is quite efficient for one agent, but can be a mess for several agents.

2. Once entering a large ‘room’ (that is, upon passing from a critical area to a non-critical area), designate the entrance by a special type of token so that other agents will enter only in the case there is no other work to do. This approach guarantees that the agents will be distributed between the large rooms of the F-configuration. This is attractive if the region has rooms of similar areas.

3. Quite a different idea is to divide the work into two phases. The first phase is of ‘distribution’, where the agents locate themselves uniformly around the area. In the second phase, each agent cleans around his ‘center’. If the distribution is appropriate, there will be a minimum of interactions between agents in the second phase.

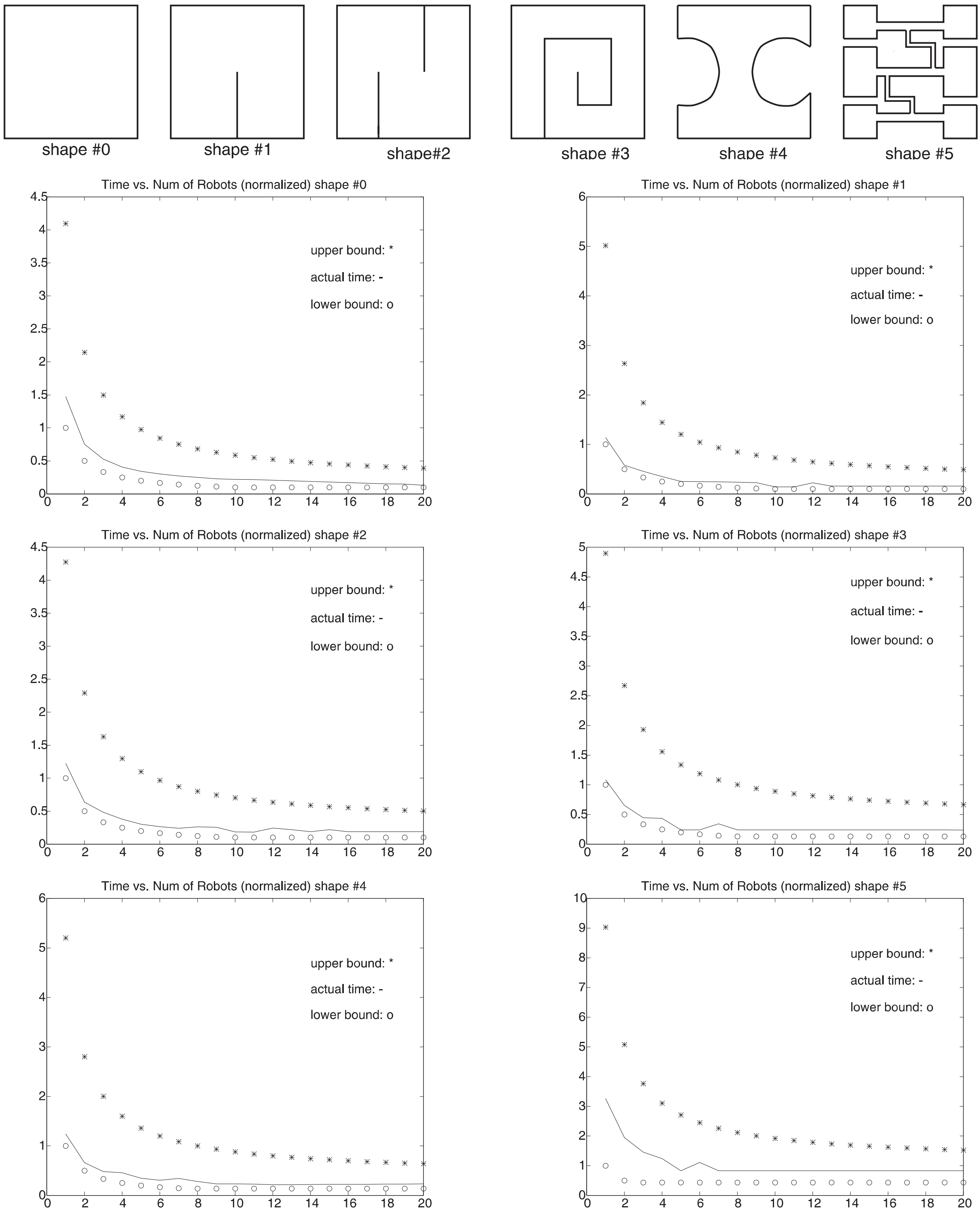

Fig. 14. Various shapes tested in CLEAN simulations and their normalized cleaning time $t(k) = t_{\text{success}}(k)/s_0$ as a function of the number of agents k for each region, together with the lower and upper bounds according to Theorem 9. For a confidence level of 95%, based on at least 10 simulation instances per dirty region, the appropriate confidence interval is less than $\pm 6\%$.

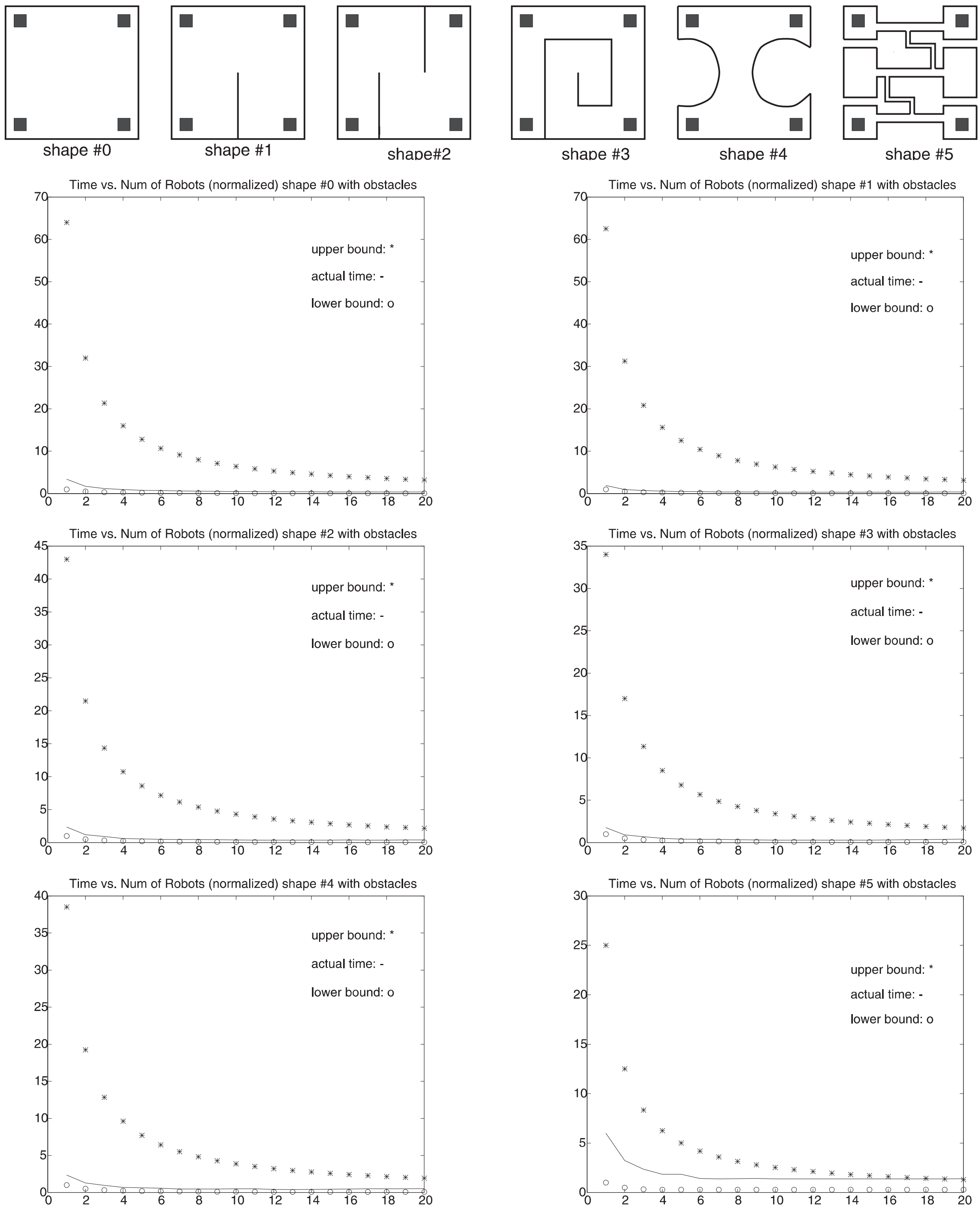

Fig. 15. Various shapes tested in CLEAN-WITH-OBSTACLES simulations and their normalized cleaning time $t(k) = t_{\text{success}}(k)/s_0$ as a function of the number of agents k for each region, together with the lower and upper bounds according to Theorem 9. For a confidence level of 95%, based on at least 10 simulation instances per dirty region, the appropriate confidence interval is less than ±10%.

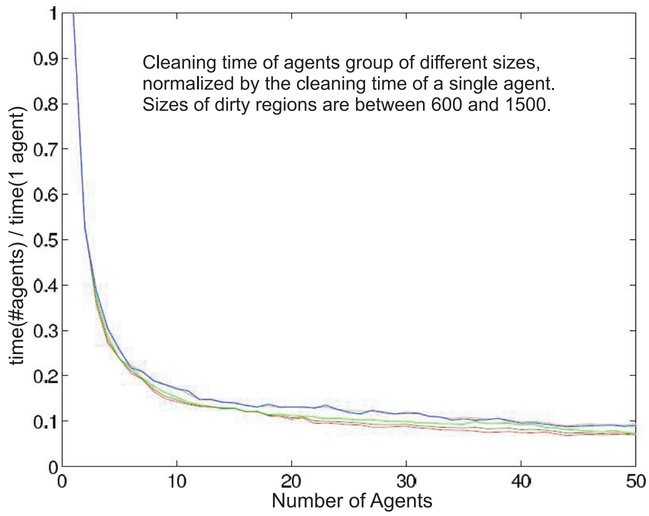

Fig. 16. Cooperative cleaners: from $1 - 50$ agents, dirty regions of sizes $600 - 1500$ tiles, 10 simulations per region's size, cleaning time normalized by the cleaning time of a single agent. Note that as the initial dirty area is larger, the ratio between $t_{\text{success}}(k)$ and $t_{\text{success}}(1)$ 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 $\pm 15\%$.

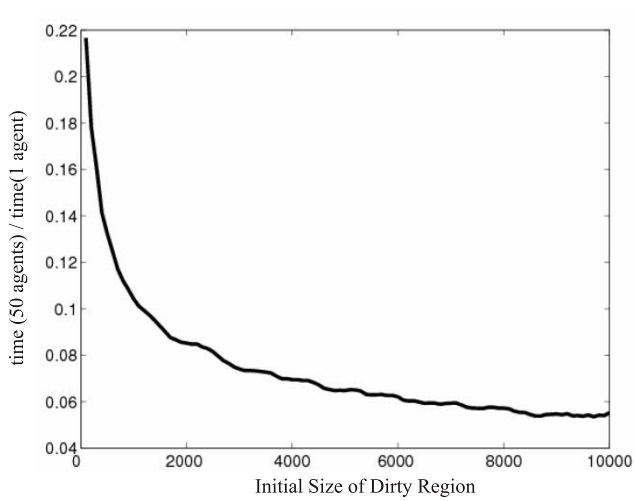

Fig. 17. Cooperative cleaners: the behavior of $t_{\text{success}}(50)/t_{\text{success}}(1)$ for dirty regions of sizes $100 - 10\,000$ tiles. Ten simulations were carried out per region's size. Note how as the initial dirty area gets larger, $t_{\text{success}}(50)/t_{\text{success}}(1)$ decreases. The minimal value possible is of course $1/50 = 0.02$. For a confidence level of 90%, the confidence interval is less than $\pm 10\%$.

We use the dirt on the floor as a means of inter-agent communication, but other ways for communication between agents have been suggested. One is to use heat trails for this end, as was reported in Russell (1993). In Steels (1990), self-organization is achieved among a group of lunar robots that have to explore an unknown region and bring particular rock samples back to the mother-spaceship. Each robot is programmed to drop a crumb at each point visited, and walk around at random with a bias toward the negative gradient of crumb concentration.

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

It is of interest to notice here that an off-line version of the problem, i.e. 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 a Hamilton path in a non-simple grid-graph is *NP-complete* (Itai et al. 1982).

In summary, we would like to cite a statement (Simon 1981) made after watching an ant laboring across a wind-and-wave-molded beach.

> An ant, viewed as a behaving system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the environment in which it finds itself.

Such a point of view, as well as the results of our simulations and analysis, make us believe that even simple, ant-like creatures, yield very interesting, adaptive and quite efficient goal-oriented behavior.

Acknowledgements

This research was supported in part by the Israeli Ministry of Science Infrastructural Grant No. 3-942 and the Devorah fund.

References

Acar, E. U., Zhang, Y., Choset, H., Schervish, M., Costa, A. G., Melamud, R., Lean, D. C. and Gravelin, A. (2001). Path Planning for Robotic Demining and Development of a Test Platform. *Proceedings of International Conference on Field and Service Robotics*, Helsinki, Finland. 161–168.

Adler, F.R. and Gordon, D.M. (1992). Information collection and spread by networks of partolling agents. *The American Naturalist*, **140**(3): 373–400.

Alami, R., Fleury, S., Herrb, M., Ingrand, F. and Robert, F. (1997). Multi-Robot Cooperation in the Martha Project. *IEEE Robotics and Automation Magazine*, **5**(1): 36–47.

Arai, T., Ogata, H. and Suzuki, T. (1989). Collision Avoidance Among Multiple Robots Using Virtual Impedance. In

Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 479–485.

Arkin, R.C. (1990). Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. *Robotics and Autonomous Systems*, **6**: 105–122.

Arkin, R.C. and Balch, T. (1997). AuRA: Principles and Practice in Review. *Journal of Experimental and Theoretical Artificial Intelligence*, **9**(2–3): 175–188.

Arkin, R.C. and Balch, T. (1998). *Cooperative multi-agent Robotic Systems. Artificial Intelligence and Mobile Robots.* MIT/AAAI Press: Cambridge, MA.

Baeza-Yates, R. and Schott, R. (1995). Parallel Searching in the Plane. *Computational Geometry*, **5** :143–154.

Balch, T and Arkin, R. (1998). Behavior-Based Formation Control for Multi-Robot Teams. *IEEE Transactions on Robotics and Automation*, **14**(6):926–939.

Batalin, M. A. and Sukhatme, G. S. (2002). Spreading Out: A Local Approach to Multi-Robot Coverage. In *Proceedings of 6th International Symposium on Distributed Autonomous Robotics Systems*, Fukuoka, Japan.

Benda, M., Jagannathan, V. and Dodhiawalla, R. (1985). On Optimal Cooperation of Knowledge Sources. Technical Report BCS-G2010-28, Boeing AI Center.

Beni, G. and Wang, J. (1991). Theoretical Problems for the Realization of Distributed Robotic Systems. In *Proceedings of 1991 IEEE Internal Conference on Robotics and Automation*, Sacramento, CA, 1914–1920.

Braitenberg, V. (1984). *Vehicles*. MIT Press, Cambridge, MA.

Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, **RA-2**(1): 14–23.

Brooks, R. A. (1990). Elephants Don't Play Chess. In *Designing Autonomous Agents*, P. Maes (ed.) MIT press/Elsevier, Cambridge, MA; 3–15.

Bruckstein, A. M. (1993). Why the Ant Trails Look So Straight and Nice. *The Mathematical Intelligencer*, **15**(2): 59–62.

Bruckstein, A. M., Mallows, C. L. and Wagner, I. A. (1994). Probabilistic Pursuits on the Integer Grid. Technical report CIS-9411, Center for Intelligent Systems, Technion, Haifa.

Butler, Z., Rizzi, A. and Hollis, R. (2001). Distributed Coverage of Rectilinear Environments. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

Candea, C., Hu, H., Iocchi, L., Nardi, D. and Piaggio, M. (2001). Coordinating in Multi-Agent RoboCup Teams. *Robotics and Autonomous Systems*, **36**(2–3): 67–86.

Chevallier, D. and Payandeh, S. (2000). On Kinematic Geometry of Multi-Agent Manipulating System Based on the Contact Force Information. In *Proceedings of 6th International Conference on Intelligent Autonomous Systems (IAS-6)*, 188–195.

Conner, D. C., Greenfield, A., Atkar, P. N., Rizzi, A. and Choset, H. (2005). Paint Deposition Modeling for Trajectory Planning on Automotive Surfaces. *IEEE Transactions on Automation Science and Engineering*, **2**(4): 381–392.

Deneubourg, J., Goss, S., Sandini, G., Ferrari, F. and Dario, P. (1990). Self-Organizing Collection and Transport of Objects in Unpredictable Environments. In *Proceedings of Japan-U.S.A. Symposium on Flexible Automation*, Kyoto, Japan, 1093–1098.

Dias, M. B. and Stentz, A. (2001). A Market Approach to Multirobot Coordination. Technical Report, CMU-RI-TR-01-26, Robotics Institute, Carnegie Mellon University.

Do-Carmo, M. P. (1976). *Differential Geometry of Curves and Surfaces*. Prentice-Hall: New Jersey.

Drogoul, A. and Ferber, J. (1992). From Tom Thumb to the Dockers: Some Experiments With Foraging Robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Honolulu, Hawaii, 451–459.

Dudek, G., Jenkin, M. R. M., Milios, E. and Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots Journal*, **3**(4): 375–397.

Ferrari, C., Pagello, E., Ota, J., and Arai, T. (1998). Multirobot Motion Coordination in Space and Time. *Robotics and Autonomous Systems*, **25**: 219–229.

Fredslund, J. and Mataric, M. J. (2001). Robot Formations Using Only Local Sensing and Control. In *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation (IEEE CIRA 2001)*, Banff, Alberta, Canada, 308–313.

Gerkey, B. P. and Mataric, M. J. (2002). Sold! Market Methods for Multi-Robot Control. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, **18**(5): 758–768.

Golfarelli, M., Maio, D. and Rizzi, S. (1997). A Task-Swap Negotiation Protocol Based on the Contract Net Paradigm. Technical Report, 005-97, CSITE (Research Center For Informatics And Telecommunication Systems, associated with the University of Bologna, Italy).

Gordon, D. M. (1995). The expandable network of ant exploration. *Animal Behaviour*, **50**: 372–378.

Gordon, N., Wagner, I. A. and Bruckstein, A. M. (2003). Discrete Bee Dance Algorithms for Pattern Formation on a Grid. In *Proceedings of IEEE International Conference on Intelligent Agent Technology (IAT03)*, 545–549.

Haynes, T. and Sen, S. (1986). Evolving Behavioral Strategies in Predators and Prey. In *Adaptation and Learning in Multi-Agent Systems*, Weiss G. and Sen S. eds. 113–126.

Hedberg, S. (1995). Robots Cleaning Up Hazardous Waste. *AI Expert*, **10**(5): 20–24.

Held, M. (1991). *On the Computational Geometry of Pocket Machining*. Lecture Notes in Computer Science. Springer-Verlag, New York, NY.

Henrich, D. (1994). Space Efficient Region Filling in Raster Graphics. *The Visual Computer*, **10**: 205–215.

Itai, A., Papadimitriou, C. H., and Szwarefiter, J. L. (1982). Hamilton Paths in Grid Graphs. *SIAM Journal of Computing*, **11**: 676–686.

Koenig, S. and Szymanski, B. (1999). Value-update rules for real-time search. In *Proceedings of the National Conference on Artificial Intelligence*, 718–724.

Koenig, S. and Liu, Y. (2001). Terrain Coverage with Ant Robots: A Simulation Study. In *Proceedings of 5th International Conference on Autonomous Agents 2001*, Montreal, Quebec, Canada, 600–607.

Koenig, S., Szymanski, B. and Liu, Y. (2001). Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, **31**: 41–76.

Latimer, D., Srinivasa, S., Lee-Shue, V., Sonne, S., Choset, H. and Hurst, A. (2002). Toward Sensor Based Coverage with Robot Teams. In *Proceedings of The International Society for Optical Engineering (SPIE)*, **4573**: 1–9, Mobile Robots XVI.

LaValle, S. M., Lin, D., Guibas, L. J., Latombe, J. C. and Motwani, R. (1997). Finding an Unpredictable Target in a Workspace with Obstacles. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation (ICRA-97)*, 737–742.

Levi, S. (1992). *Artificial Life – the Quest for a New Creation*. Penguin, New York, NY.

Lumelsky, V. J. and Harinarayan, K. R. (1997). Decentralized Motion Planning for Multiple Mobile Robots: The Cocktail Party Model. *Autonomous Robots*, **4**(1): 121–136.

MacKenzie, D., Arkin, R., and Cameron, J. (1997). Multiagent Mission Specification and Execution. *Autonomous Robots*, **4**(1): 29–52.

Mataric, M. J. (1992). Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. Meyer, J., Roitblat, H. and Wilson, S. (eds). MIT Press: Honolulu, Hawaii; 432–441.

Mataric, M. J. (1994). *Interaction and Intelligent Behavior*. PhD Thesis, Massachusetts Institute of Technology.

Min, T. W. and Yin, H. K. (1998). A Decentralized Approach for Cooperative Sweeping by Multiple Mobile Robots. In *Proceedings of 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, **1**: 380–385.

Neubauer, W. (1993). Locomotion with Articulated Legs in Pipes or Ducts. *Robotics and Autonomous Systems*, **11**: 163–169.

Pagello, E., D'Angelo, A., Montesello, F., Garelli, F. and Ferrari, C. (1999). Cooperative Behaviors in Multi-Robot Systems Through Implicit Communication. *Robotics and Autonomous Systems*, **29**(1): 65–77.

Pagello, E., D'Angelo, A., Ferrari, C., Polesel, R., Rosati, R. and Speranzon, A. (2003). Emergent Behaviors of a Robot Team Performing Cooperative Tasks. *Advanced Robotics*, **17**(1): 3–19.

Parker, L. E. (1998). ALLIANCE: An Architecture for Fault-Tolerant Multi-Robot Cooperation. *IEEE Transactions on Robotics and Automation*, **14**(2): 220–240.

Parker, L. E. and Touzet, C. (2000). Multi-Robot Learning in a Cooperative Observation Task. *Distributed Autonomous Robotic Systems*, **4**: 391–401.

Polycarpou, M., Yang, Y. and Passino, K. (2001). A Cooperative Search Framework for Distributed Agents. In *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*, Mexico City, Mexico. IEEE: New Jersey, 1–6.

Premvuti, S. and Yuta, S. (1990). Consideration on the Cooperation of Multiple Autonomous Mobile Robots. In *Proceedings of the IEEE International Workshop of Intelligent Robots and Systems*, Tsuchiura, Japan, 59–63.

Rabideau, G., Estlin, T., Chien, T. and Barrett, A. (1999). A Comparison of Coordinated Planning Methods for Cooperating Rovers. In *Proceedings of the American Institute of Aeronautics and Astronautics (AIAA) Space Technology Conference*.

Rekleitis, I. M., Dudek, G. and Milios, E. (2003). Experiments in Free-Space Triangulation Using Cooperative Localization. In *Proceedings of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, **2**: 1777–1782.

Rekleitisy, I., Lee-Shuey, V., Peng Newz, A. and Choset, H. (2004). Limited Communication, Multi-Robot Team Based Coverage. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, 3462–3468.

Russell, R. A. (1993). Mobile Robot Guidance Using a Short-Lived Heat Trail. *Robotica*, **11**: 427–431.

Sen, S., Sekaran, M., Hale, J. (1994). Learning to Coordinate Without Sharing Information. In *Proceedings of American Association for Artifical Intelligence 1994*, 426–431.

Shoham, Y. and Tennenholtz, M. (1995). On Social Laws for Artificial Agent Societies: Off Line Design. *Artificial Intelligence*, **73**(1–2): 231–252.

Simon. H. A. (1981). *The Sciences of the Artificial*. Second Edition. MIT Press, Cambridge, MA.

Smith, R. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, **C-29**(12): 1104–1113.

Steels, L. (1990). Cooperation Between Distributed Agents Through Self-Organization. Decentralized A.I. In *Proceedings of the first European Workshop on Modeling Autonomous Agents in Multi-Agents world*, DeMazeau, Y., Muller J. P. (eds). 175–196.

Stone, P. and Veloso, M. (1999). Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork. *Artificial Intelligence*, **110**(2): 241–273.

Svennebring, J. and Koenig, S. (2004). Building Terrain-Covering Ant Robots: A Feasibility Study. *Autonomous Robots*, **16**(3): 313–332.

Svestka, P. and Overmars, M. H. (1998). Coordinated Path Planning for Multiple Robots. *Robotics and Autonomous Systems*, **23**(3): 125–152.

Thayer, S. M., Dias, M. B., Digney, B. L., Stentz, A., Nabbe, B. and Hebert, M. (2000). Distributed Robotic Mapping of Extreme Environments. In *Proceedings of SPIE, 4195, Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*.

Wagner, I. A. and Bruckstein, A. M. (1994). Row Straightening via Local Interactions. Technical report CIS-9406, Center for Intelligent Systems, Technion, Haifa.

Wagner, I. A. and Bruckstein, A. M. (1997). Cooperative Cleaners: A Case of Distributed Ant-Robotics. In *Communications, Computation, Control, and Signal Processing: A Tribute to Thomas Kailath*. Kluwer Academic Publishers: The Netherlands; 289–308.

Wagner, I. A. and Bruckstein, A. M. (2000). ANTS: agents, networks, trees and subgraphs. *Future Generation Computer Systems Journal*, **16**(8): 915–926.

Wagner, I. A. and Bruckstein, A. M. (2001). From Ants to A(ge)nts: A Special Issue on Ant–Robotics. *Annals of Mathematics and Artificial Intelligence*, Special Issue on Ant Robotics, **31**(1–4): 1–6.

Wagner, I. A., Lindenbaum, M. and Bruckstein, A. M. (1998). Efficiently Searching a Graph by a Smell-Oriented Vertex Process. *Annals of Mathematics and Artificial Intelligence*, **24**: 211–223.

Wang, P. K. C. (1989). Navigation Strategies for Multiple Autonomous Mobile Robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 486–493.

Wellman, M. P. and Wurman, P. R. (1998). Market-Aware Agents for a Multiagent World. *Robotics and Autonomous Systems*, **24**: 115–125.

Yamashita, A., Fukuchi, M., Ota, J., Arai, T. and Asama, H. (2000). Motion Planning for Cooperative Transportation of a Large Object by Multiple Mobile Robots in a 3D Environment. In *Proceedings of IEEE International Conference on Robotics and Automation*, 3144–3151.

Yanovski, V., Wagner, I. A. and Bruckstein, A. M. (2001). Vertex-ants-walk: a robust method for efficient exploration of faulty graphs. *Annals of Mathematics and Artificial Intelligence*, **31**(1–4): 99–112.

Yanovski, V., Wagner, I. A. and Bruckstein, A. M. (2003). A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, **37**: 165–186.

Zlot, R., Stentz, A., Dias, M. B. and Thayer, S. (2002). Multi-Robot Exploration Controlled By A Market Economy. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3016–3023.