Organic Computing and Swarm Intelligence

Daniel Merkle, Martin Middendorf, and Alexander Scheidler

Department of Computer Science
University of Leipzig, Leipzig, Germany
{merkle,middendorf,scheidler}@informatik.uni-leipzig.de

Summary. The relations between swarm intelligence and organic computing are discussed in this chapter. The aim of organic computing is to design and study computing systems that consist of many autonomous components and show forms of collective behavior. Such organic computing systems (OC systems) should possess self-x properties (e.g., self-healing, self-managing, self-optimizing), have a decentralized control, and be adaptive to changing requirements of their user. Examples of OC systems are described in this chapter and two case studies are presented that show in detail that OC systems share important properties with social insect colonies and how methods of swarm intelligence can be used to solve problems in organic computing.

1 Introduction

Organic computing is a new field of computer science with the aim to design and understand computing systems that consist of many components and possess so-called self-x properties where "x" stands, for example, for "healing", "managing", "organizing", "optimizing" (e.g., [19, 37, 44]). One idea of organic computing is to use principles of self-organization in order to obtain systems with self-x properties. Computing systems that possess self-x properties and follow such design principles are called *organic computing systems* (*OC systems*).

Social insects, like ants and bees, are a particularly interesting source of inspiration for the design of OC systems. The main reason is that social insect colonies show a complex behavior even though the members of the colony are relatively simple individuals. Most of these behaviors can be called self-organized because there exists neither a central control nor a global work plan. A behavior of the colony which can be seen on a large scale (e.g., nest building or the formation of aggregations of thousands of individuals) but where the individuals act only according to simple rules that use input from their senses only about their local environment is called emergent. Examples of such emergent behavior are nest building of termites ([21]), the formation

of bucket brigades during foraging of ants ([2]), the election of a new nest site by a swarm of bees ([26]), and the trail-laying behavior of ants that leads to short paths between their nest and food sources. The latter behavior has inspired the Ant Colony Optimization metaheuristic (see Chap. 2) that is used to solve combinatorial optimization problems ([15]). Another example is the behavior of ants to cluster larvae or corpses of dead ants, which has inspired the design of different clustering algorithms (e.g. [18, 23]).

Since self-organized systems can show emergent effects it is important to understand under which circumstances these effects might occur. Therefore, researchers have developed models that help explain the emergent behavior of social insects. Examples are threshold response models that have been proposed to explain the foraging behavior of ants ([4, 48, 50]); self-synchronization effects in the activity schedule of ants have been explained by models [12], and models have been used to describe the self-organized emergence of aggregations in social insects ([13, 14]). The emergent behavior of social insects and the related biological models have been used by researchers in swarm intelligence to build agent systems or swarm robots and to develop new optimization methods.

In this chapter we discuss connections between swarm intelligence and organic computing. Since organic computing is a relatively new research field it is too early to give an overview on the relations between organic computing and swarm intelligence. Therefore, we shortly present some example applications of organic computing methods from different areas (see Sec. 2). Then we present two case studies that show in some detail how methods of swarm intelligence are connected to problems in organic computing (see Sections 3 and 4).

The first study deals with the control of emergence effects in OC systems. In general, emergent behavior is considered to be an important aspect for OC systems (e.g., [42]). So far researchers have considered mostly the positive aspects of emergent behavior. They try to apply the principles of emergent behavior of natural systems to increase the capabilities of OC systems. Ideally, the autonomous components of an OC system should be able to create a complex emergent behavior without the knowledge of a global plan and without control information that they receive from a central controller. An example of such an emergent behavior that results from self-organization could be the task allocation between the components or the specialization of the components to different tasks by using reconfigurable hardware. Since a certain emergent behavior is typically seen as a desired property of an OC system there exist efforts to develop quantitative measures for emergence ([36]) in order to compare the strength of different OC systems. But, there exists also the potential danger that an OC system might show emergent properties that are unwanted and have not been foreseen when the system was designed. The question that is considered in the first part of this chapter is how an OC system can be controlled such that certain unwanted emergent behaviors can be prevented.

As an example model for this study the emergent clustering behavior of ants is used.

The second study in this chapter is related to the following observation. Typical for OC systems is that their components can adapt to environmental conditions. Hence, even if the components are all equal in principle they will show a slightly different behavior due to individual adaptations. Therefore, it is interesting to investigate what types of emergent effects might occur due to such slight differences in individual behavior. In this context, it is interesting that, as has been observed, ants with slightly different movement behavior can be found most often in different parts of the nest (see [45]). It is discussed in the second part of this chapter what patterns might occur in OC systems with moving components and what are their possible consequences on the behavior of OC systems.

The content of Section 3 and 4 is based on work that has been done within the project "Organization and Control of Self-Organizing Systems in Technical Compounds" within the German Research Foundation (DFG) priority programme on organic computing, and is based on publications [43] and [34].

2 Examples of Organic Computing Systems

In this section we shortly present some examples of the application of organic computing methods to different areas.

One application of organic computing in the field of hardware is the organic computing approach for very fast image processing that was proposed in [28, 16]. This approach is called Marching Pixels (MPs). The basic idea of MP is to use an embedded massively parallel array of pixel processor elements (PEs) to exploit emergent algorithms in order to solve difficult image-processing tasks. Marching pixels are seen as virtual organic units which are born, move, unite, are mutated, leave signatures on the ground, and die on the processor field. The task of the marching pixels is to carry out autonomously image preprocessing tasks, e.g., detection and tracking of moving objects. For the underlying technology there exist plans for future smart sensor chips which will integrate hundreds of millions of transistors. One idea for realizing the MPs approach is to use principles from the pheromone communication of ants to guide the pixels.

An Organic-Computing-inspired System-on-Chip (SoC) architecture which applies self-organization and self-calibration concepts to build reliable SoCs was proposed in [5]. This type of SoC architecture — called Autonomic SoC (ASoC) — provides lower overheads and a broader fault coverage than classical fault-tolerance techniques. The architecture essentially splits the SoC into two logical layers: the functional layer which contains the usual Intellectual Property components or Functional Elements (FEs), and the autonomous layer which consists of Autonomic Elements (AEs) and an interconnect structure among various AEs. FEs are either general-purpose CPUs, memories, on-chip

busses, special-purpose processing units, or system and network interfaces as in a conventional design. AEs contain any extensions necessary to improve the reliability of the FE and convert the FE-AE pairs into autonomous units. The feasibility of this approach has been shown for the processing pipeline of a public-domain RISC CPU core.

Traffic systems are another application area of organic computing. It has been proposed to use self-organized inter-vehicle communication to recognize traffic jams [17]. One aim of this communication is to detect the front and the back of a traffic jam. Since the set of cars that forms the front or the back of the traffic jam changes, data about the traffic jam have to be transferred between the cars. Hence, a so-called "Hovering Data Cloud" is formed that is independent of the participating vehicles and stays with the beginning or the end of the traffic jam. This data is used to extract information for other cars to optimize the traffic flow.

Principles of organic computing are also applied to the design of controllers for traffic lights. Traffic flows in urban road traffic networks are changing constantly and on different time scales. Unfortunately, many such changes of traffic flow cannot be foreseen since the change might be due to public events, road works, or sudden incidents. Therefore, traffic light controllers need the ability to adjust quickly to changes in traffic situations and to react reasonably in situations that have not been anticipated in their design process. The Organic Traffic Control (OTC) project (see [41]) develops an adaptive traffic light controller architecture with learning capabilities. The overall architecture is self-optimizing because it is traffic-responsive and can adapt to larger changes in traffic due to a "simulation-based learning" approach.

Collaborating Traffic Lights (CTLs) is a project that tries to exploit the increasing amount of available sensor data about traffic to address the problem of global optimization of traffic flows (see [9]). The idea is to allow the controller or agent of the traffic lights at a junction to decide autonomously on the appropriate phase for the junction. The controller or agent would monitor the level of congestion at the junction under its control based on available sensor data and use this information to decide which action to take. Over time, the agent learns the appropriate action to take given the current level of congestion. In order to achieve optimal system-wide performance, the set of controllers or agents at traffic light junctions in the system should communicate their current status to controllers or agents at neighbouring upstream and downstream junctions.

Mobile robotics is also an area where organic computing methods are clearly useful. For example, the aim of the ORCA project ([35]) is to develop an architecture for mobile autonomous robots that is based on organic computing principles. The aim is to make the robots more reliable and robust. Also the design should eventually be easier compared to classical (fault-tolerant) approaches. The concepts that are used in the project are inspired by the functioning of the human autonomous nervous system and the human immune system. A robot shall be able to continuously monitor its own "health

status" and ensure that it is stable and performing its task with optimum performance. In contrast to more classical approaches, error situations are not explicitly described in advance. If a new and unknown deviation from the healthy case is observed by a robot a counteraction is taken first. Based on success or failure, the robot will learn how to handle similar situations and to react faster and more appropriately (similarly to how the human immune system learns to fight against reoccurring infections).

The presented example applications of organic computing show that mobile components play a central role in many OC systems. This is one reason why methods that are inspired by the self-organized behavior of social insects have a great potential for future designs of OC systems. In each of the two following sections one such example of social-insect-inspired methods is discussed in detail.

3 Swarm-Controlled Emergence

The use of principles of self-organization from nature seems to have great potential for the design of OC systems. But recently there have been some concerns that self-organized computing systems might show an emergent behavior that is neither wanted nor intended or foreseen when they were designed. The term negative emergence has been used by some authors for such unwanted emergent behavior (see also [38] for a discussion of emergence). One important research question is how negative emergent behavior of OC systems can be prevented.

One possible approach to prevent negative emergence in OC systems that has been proposed by several researchers is to equip the systems with a so-called observer controller subsystem [40, 46] where a set of observers collects information about the system and based on this information the controllers send control information to the components to influence their behavior.

A potential disadvantage of this approach is that it relies fundamentally on (classical) controllers that send control messages to the components and thus directly restrict the autonomy of the single components of an OC system. Since this is against some central principles of organic computing like self-organization and self-autonomy it is interesting to search for alternative approaches.

In this section we present a new approach to prevent negative emergence in OC systems. This approach was proposed in [34] and is called swarm-controlled emergence. The general idea of swarm-controlled emergence is to add so-called anti-emergence components (anti-components) to an OC system which can prevent the occurrence of certain negative emergence effects. Ideally, the anti-components should not behave too differently from the normal components of the OC system. Then they can still do normal work in the OC system (eventually less efficient though because otherwise all components could become anti-components).

Characteristics of the swarm-controlled emergence approach that differ fundamentally from the observer controller approach are: i) the autonomy of the components (neither of the normal components nor of the anti-components) is not restricted, and ii) it is not necessary to have a corresponding communication structure for delivering control information.

In the following, we describe the principal ideas of the swarm-controlled emergence approach and give proof of concept for a test system and start investigations on special properties of the new approach. The chosen test case is one of the famous examples of emergent behavior of social insects which has several applications in computer science — the clustering behavior of ants (see [23, 24, 25, 27, 29, 30, 31, 32]). Ant clustering has been applied to solve combinatorial problems (e.g., clustering and sorting) and to study emergence in robotics (e.g., [10]).

3.1 Ant Clustering

Ant clustering refers to the behavior of ants to cluster their brood within the nest center (e.g., [18]) or to cluster dead corpses so that they form so-called ant cemeteries (e.g., [8, 10]). Both phenomena can be seen as emergent behavior and have been addressed by simple multi-agent models.

In the ant-clustering model that was proposed in [10] (see also Sect. 4.2 of Chap. 2), several items are distributed in a two-dimensional array of cells (at most one item per cell). Each agent walks randomly within the cell array, picks up an item that it finds with a certain probability, carries it around, and drops it with a certain probability. Formally, the probability p_p of an unladen agent picking up an item is $p_p = (k_1/(k_1 + f))^2$, where f is the fraction of cells in the neighborhood of the agent that are occupied with items and k_1 is a threshold value. Analogously, the probability that a laden agent drops an item if it is on a cell that is not occupied by an item is given by $p_d = (f/(k_2 + f))^2$, with k_2 being a threshold value. Several methods for calculating the value f have been proposed. One method is to count how many items have been encountered by the agent during the last few time steps and define f as the fraction of time steps where the agent moved across cells that are occupied by an item. Another way to calculate f is to calculate the fraction of cells in the von Neumann neighborhood of the agent that are occupied with an item.

It was shown that the ant-clustering model fits the clustering behavior of real ants during the organization of cemeteries very well. More complicated patterns, as they occur, for example, in the clustering of brood of the ant *Lepthotorax* ([18]), where different types of brood are clustered in concentric rings, need more elaborated models. This is discussed in more detail in the second half of this chapter.

To illustrate the ant-clustering model and the concept of swarm-controlled emergence some experiments are described for a two-dimensional cell array. The size of the cell array that was used for the experiments is 500×500. In the initial state 2,500 cells (i.e., 1/100 of all cells) are occupied by an item.

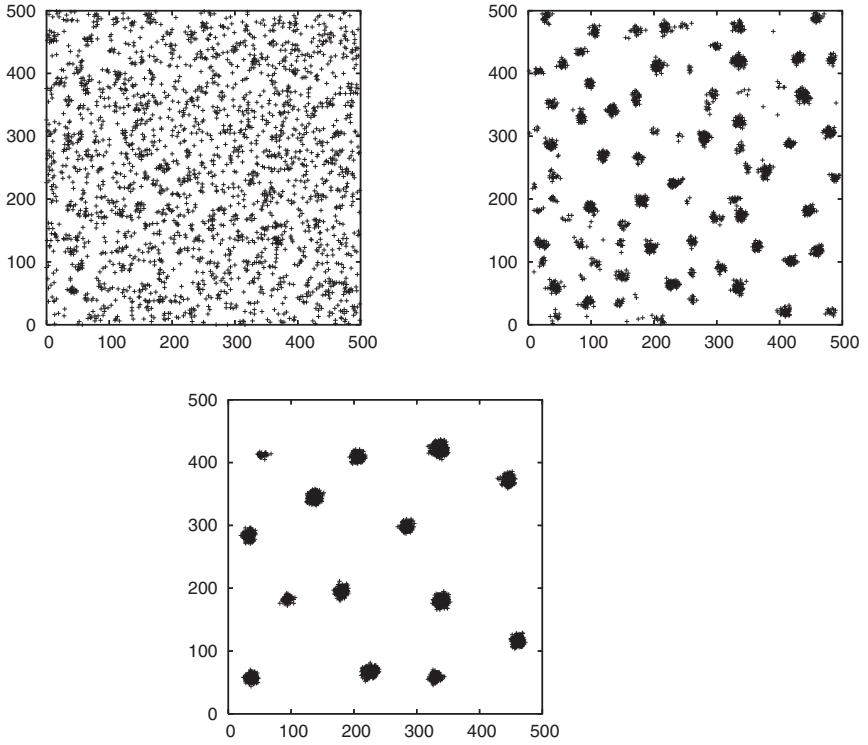

Fig. 1. Cell array with clustering agents: distribution of items after 100,000 (upper left), 1,000,000 (upper right), and 50,000,000 (lower) simulation steps

In all experiments 50 clustering agents were used that move on the cell array. The neighborhood of an agent is defined as the von Neumann neighborhood with radius 10, i.e., all cells for which $d_x + d_y \leq 10$ are said to be in the neighborhood of an agent, where d_x and d_y are the absolute distances of the considered cell to the cell of the agent in the two dimensions. The threshold parameters for the clustering agents were chosen as $k_1 = 0.05$ and $k_2 = 0.03$. The results of a typical test run are shown in Fig. 1. It can be seen that many small clusters have been formed after 100,000 simulation steps. With a growing number of simulation steps the number of clusters becomes smaller and the size of the clusters increases.

3.2 Cluster Validity and Clustering Measures

In order to study the effect of the swarm-controlled emergence approach to clustering it is necessary to measure the quality of a clustering. Since there exist several possibilities to define what a good clustering is several measures for the degree of clustering have been proposed in the literature. The

measures that are used in this section are *spatial entropy*, *summary function*, and *hierarchical social entropy*. These measures are described in the following.

The *spatial entropy* ([6, 20]) is a measure for classifying spatial distributions of items according to their cluster validity on different spatial scales. Therefore, the two-dimensional cell array is partitioned into so-called s-patches, i.e., subarrays of size $s \times s$. Let p_I be the fraction of cells in an s-patch I that are occupied by an item. Then the spatial entropy E_s at scale s is defined as $E_s = -\sum_{I \in \{s-patches\}} p_I \log p_I$.

Two functions are introduced in the following that are often used for data analysis because they provide a good statistic on the sizes of gaps between items of a set $\mathcal{R}$ in a cell array. The first function $\hat{F}(r)$ is the probability that a random empty cell has distance r from the nearest cell that is occupied by an item of $\mathcal{R}$. Function $\hat{F}(r)$ is called the Empty Space Function and characterizes the gaps between clusters. Similarly, let $\hat{G}(r)$ be the average distance from a random point of $\mathcal{R}$ to the nearest other point of $\mathcal{R}$. Function $\hat{G}$ is the Nearest-Neighbor Distance Distribution function and characterizes how close the items within the cluster are. The so-called *summary function* $\hat{J}(r)$ (see [33]) is a measure for the quality of a clustering that is based on the $\hat{F}$ and $\hat{G}$ and is defined as $\hat{J}(r) = (1 - \hat{G}(r))/(1 - \hat{F}(r))$. The value of $\hat{J}$ for a pattern of items can be compared to the corresponding value for a random pattern to find whether the pattern of items can be interpreted as a pattern that is more clustered (or more ordered) than a random pattern. Mathematically this is done by considering a complete random point process with intensity λ, for which $F(r) = G(r) = 1 - exp(-\lambda \cdot \pi \cdot r^2)$ and $J(r) = 1$ hold (F, G, and J are the random functions that correspond to $\hat{F}$, $\hat{G}$, and $\hat{J}$). Therefore, a value of $\hat{J}(r) < 1$ indicates a clustered pattern, whereas a value of $\hat{J}(r) > 1$ can be interpreted an an ordered pattern. For the computation of $\hat{J}(r)$ the corresponding function in the R package spatstat ([1]) was used.

The *hierarchical social entropy* measure was proposed in [3]. This measure is based on a hierarchy of clusters that is computed in a bottom-up manner as follows. The bottom of the hierarchy is formed by assigning each item its own cluster (i.e., each cluster at the bottom contains only one item). Then iteratively the two nearest clusters are merged, until there is only one single cluster left. The distance between two clusters can be computed in different ways. In this section the so-called complete linkage method measure is used where the dissimilarity between two clusters is the maximal distance between two arbitrary items of both clusters (the values are between 0 and 1 where 1 means two items have maximal distance). The hierarchy of clusters can be visualized as a dendrogram that shows the agglomeration process of forming the hierarchy as a tree. The leaves of the tree are identified with the items to be clustered. Two nodes of the tree are siblings if their corresponding clusters are agglomerated during the hierarchical clustering. Note that each inner node of the dendrogram corresponds to a taxonomic level h, i.e., the two corresponding

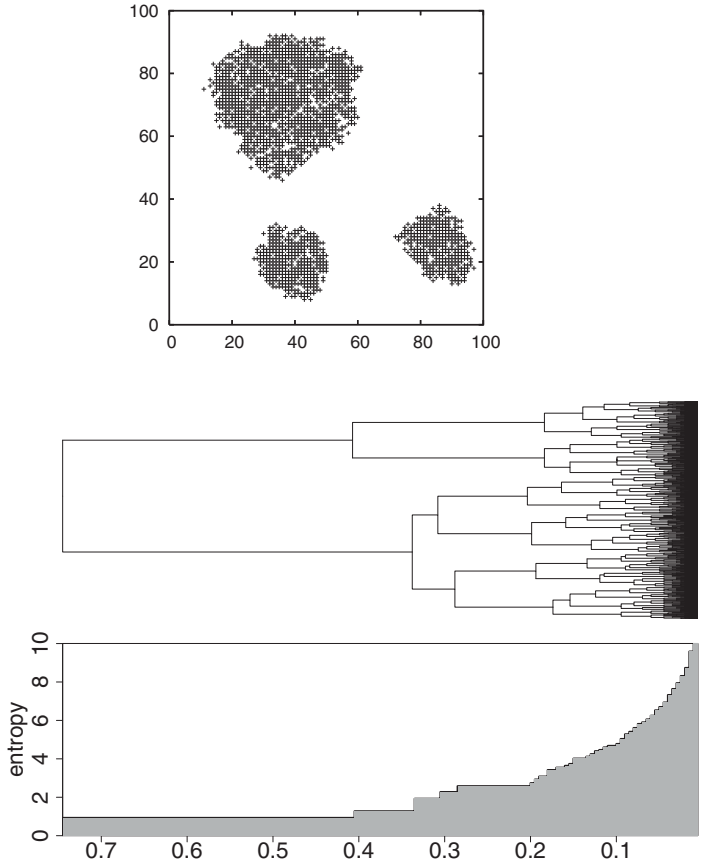

Fig. 2. Hierarchical social entropy; exemplary clustering situation on a 100×100 field (upper), resulting dendrogram (middle), and the value of the hierarchical social entropy $H(\mathcal{R}, h)$ at different taxonomic levels (lower)

clusters c_1 and c_2 have a dissimilarity of $d(c_1, c_2) = h$. For a given taxonomic level h the items are classified by the hierarchical clustering into clusters $\mathcal{C}(h) = \{c_1, \ldots, c_{M(h)}\}$. The hierarchical social entropy of a set of items $\mathcal{R}$ is defined as $\mathcal{S}(\mathcal{R}) = \int_0^\infty H(\mathcal{R}, h)dh$ where $H(\mathcal{R}, h) = -\sum_{i=1}^{M(h)} p_i \log_2(p_i)$ is the social entropy of $\mathcal{R}$ at level h (p_i is the fraction of items in cluster c_i). The hierarchical social entropy enables a total ordering according to the diversity of situations where items are distributed in a (two-dimensional) space. Note that the hierarchical social entropy is scale-invariant and allows us to address the extent of differences between clusters. In [3] the hierarchical social entropy was used to calculate the diversity of a group of robots in space. In this section it is used to distinguish between fine-grained and coarse-grained clustering

situations. In Fig. 2 a clustering situation, the resulting dendrogram, and the value of the social entropy at different taxonomic levels is depicted.

3.3 Anti-clustering

The emergent clustering effect in the clustering ant model that was described in Sect. 3.1 is considered in the rest of this section as an unwanted negative emergent effect. Clearly, the clustering effect has a biological relevance and it can be used positively for various applications in computer science. But it is also possible to consider it as unwanted to use it as an example application for the swarm-controlled emergence approach. The idea of swarm-controlled emergence is to add agents to the system that behave similarly to the standard agents but can prevent (or reduce) the clustering effect. These agents are called anti-clustering agents, or $\mathcal{AC}$-agents. In the following, three types of $\mathcal{AC}$-agents are described.

i. *Reverse $\mathcal{AC}$-agents* have a behavior which is opposite to the behavior of the standard agents in order to prevent clustering in the following sense. The values of two probabilities p_p and p_d that an agent picks up an item or drops an item in a certain situation are exchanged.
ii. *Random $\mathcal{AC}$-agents* pick up an item always when they enter a cell that is occupied by an item. If such an agent carries an item it drops it with a fixed probability (probability 0.1 is used in the experiments described in this section). The idea is that the introduction of sufficient randomness in the clustering process in the sense that items are placed on random cells should hinder a strong clustering.
iii. *Deterministic $\mathcal{AC}$-agents* use a deterministic strategy. An agent always picks up the item if it enters a cell that is occupied by an item and always drops the item if no item is in the neighborhood of the current cell.

Experimental results are described in the following, where the influence of the different types of anti-clustering agents is described when they are added to a system with clustering agents.

Experiments with Reverse $\mathcal{AC}$-Agents

The influence of reverse $\mathcal{AC}$-agents is shown in Fig. 3. In the figure the distribution of the items after 1,000,000 simulation steps is shown for different numbers of reverse $\mathcal{AC}$-agents together with 50 clustering agents. It can be seen that it is not possible for the reverse $\mathcal{AC}$-agents to hinder the standard agents from performing a clustering. Even if 100 times more reverse $\mathcal{AC}$-agents are used the item distribution is similar for only standard agents after 1,000,000 simulation steps (see upper right part of Fig. 1). The differences between using 100 and 5000 reverse $\mathcal{AC}$-agents are relatively small. For the latter the clusters are more diffuse but the number of clusters is similar and nearly the same.

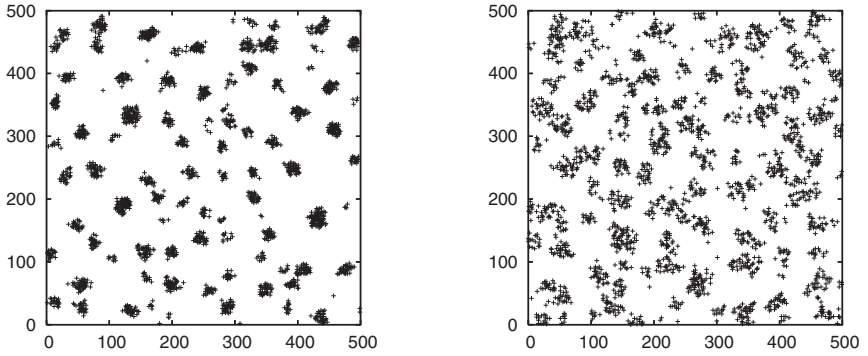

Fig. 3. Distribution of items with different numbers of reverse $\mathcal{AC}$-agents together with 50 standard agents after 1,000,000 steps; 100 reverse $\mathcal{AC}$-agents (left), 5000 reverse $\mathcal{AC}$-agents (right)

This results show that it is not a trivial task to find efficient anti-clustering agents.

The spatial entropy is $E_5 = 6.82$ and $E_5 = 7.39$ for 100 and 5000 reverse $\mathcal{AC}$-agents, together with 50 clustering agents. This is similar to the value of $E_5 = 6.53$ for the case with only clustering agents. The same holds for the hierarchical social entropy, which is $\mathcal{S} = 11.67$ and $\mathcal{S} = 12.60$ for 100 and 5000 reverse $\mathcal{AC}$-agents, together with 50 clustering agents. For the case with only clustering agents a similar value of $\mathcal{S} = 11.88$ is obtained for the hierarchical social entropy.

Experiments with Random $\mathcal{AC}$-Agents

The distribution of items for different numbers of random $\mathcal{AC}$-agents together with 50 clustering agents after 50,000,000 simulation steps are depicted in Fig. 4. The figure shows that no strong clustering occurs for 100 random $\mathcal{AC}$-agents. Hence, a reasonable number of random $\mathcal{AC}$-agents is able to hinder the clustering and they might therefore be attractive candidates for use as anti-clustering agents.

But it can also be seen that using a medium number of 50 random $\mathcal{AC}$-agents even enhances the degree of clustering compared to the cases with fewer (0 or 10) or more 100 random $\mathcal{AC}$-agents. It is an interesting observation that the quality of the clustering can even improve with respect to using only clustering agents when a certain number of random $\mathcal{AC}$-agents is used. Even if the clusters that occur are slightly more diffuse their number is clearly smaller than for the case when no anti-clustering agents are used. In the former case it can be observed that the fraction of clustering agents that carry items is higher in later phases of the simulation runs. The reason is that the slightly diffuse clusters make it more likely that the clustering agents pick up an item. The result is that smaller clusters dissolve faster. It should be noted that this

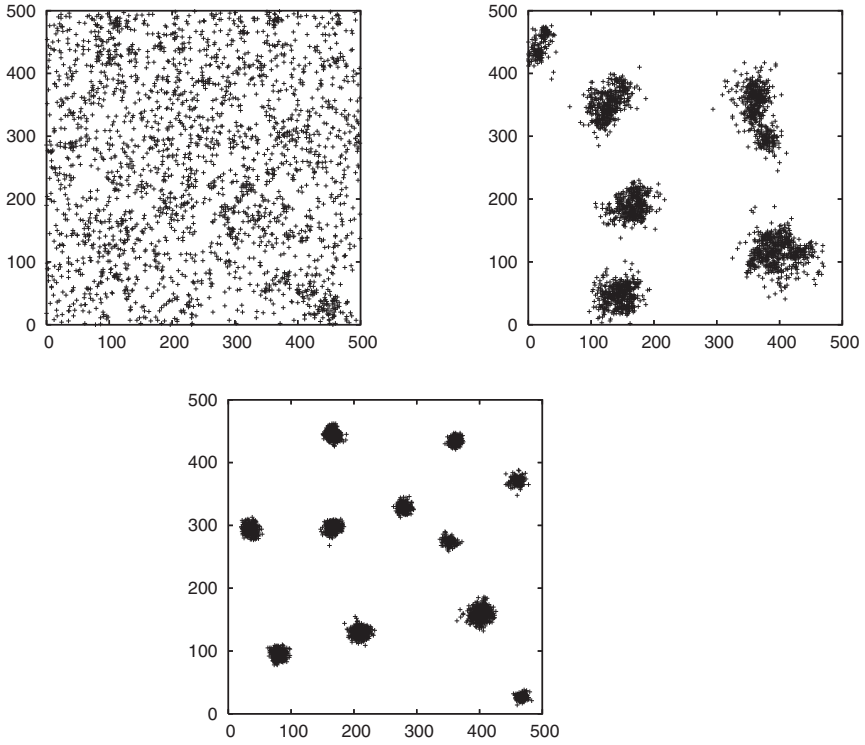

Fig. 4. Distribution of items with different numbers of random $\mathcal{AC}$-agents together with 50 clustering agents after 50,000,000 steps; 100 random $\mathcal{AC}$-agents (upper left), 50 random $\mathcal{AC}$-agents (upper right), 10 random $\mathcal{AC}$-agents (bottom)

finding is very interesting for ant-clustering algorithms in general because it shows the surprising fact that the addition of agents which have the effect of making clusters more diffuse can lead to improved clustering methods.

For the hierarchical social entropy a similar observation can be made. After 50,000,000 simulation steps the hierarchical social entropy is $\mathcal{S} = 13.64$, $\mathcal{S} = 8.07$, $\mathcal{S} = 6.91$, and $\mathcal{S} = 7.76$ for 100, 50, 10, and no random $\mathcal{AC}$-agents, together with 50 clustering agents. Hence, there is no good clustering for 100 and 50 random $\mathcal{AC}$-agents. But for a small number of 10 random $\mathcal{AC}$-agents it can be observed that the clustering quality is better than for the case with only clustering agents.

Clearly, what a good clustering method is depends on how clustering quality is defined and it can not be expected that for each quality measure there exist a suitable number of random $\mathcal{AC}$-agents that can improve the clustering quality when compared to the case with only clustering agents. This is illustrated for the Summary Function $J(r)$. It can be seen in Fig. 5 that the quality of the clustering always decreases with an increasing number of ran-

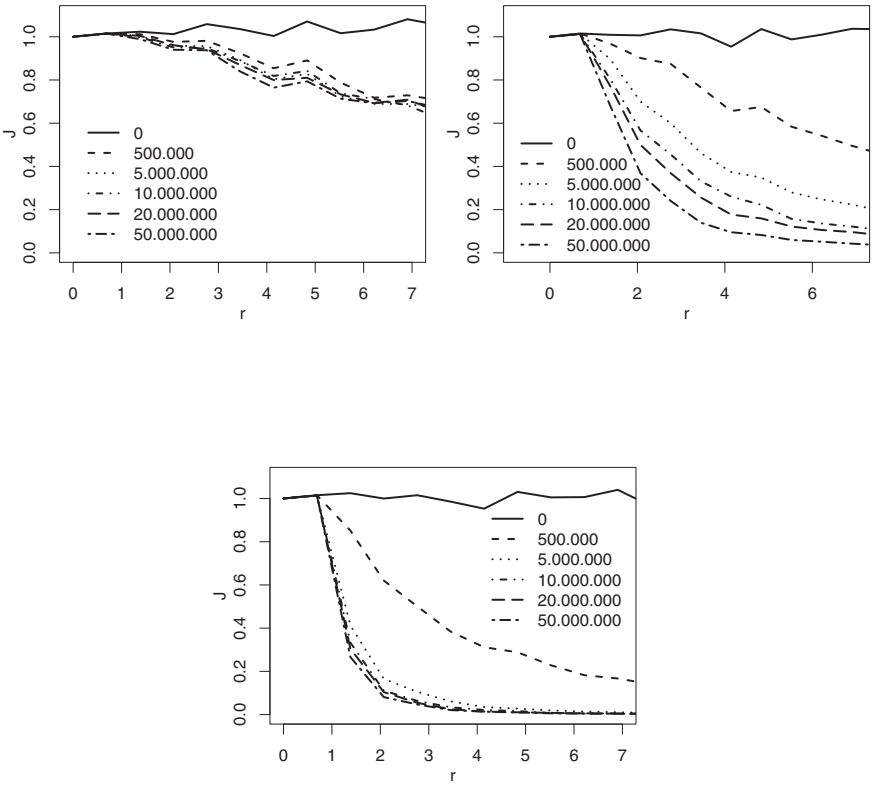

Fig. 5. Summary Function $J(r)$ for random $\mathcal{AC}$-agents together with 50 clustering agents at different time steps for the test runs for which the final item distribution is shown in Fig. 4; 100 random $\mathcal{AC}$-agents (upper left), 50 random $\mathcal{AC}$-agents (upper right), 10 random $\mathcal{AC}$-agents (bottom)

dom $\mathcal{AC}$-agents. The figure shows that no ordered item pattern occurs (the value $J(r)$ is always smaller than 1) and that, as expected, the more the random $\mathcal{AC}$-agents are used, the longer it takes until a certain degree of clustering occurs. This can be seen when comparing the curves for the same number of simulation steps in the three subfigures of Fig. 5. 100 random $\mathcal{AC}$-agents prevent a good clustering. The spatial entropy after 50,000,000 simulation steps is $E_5 = 7.56$, $E_5 = 6.73$, $E_5 = 5.78$, and $E_5 = 5.53$ for 100, 50, 10, and no random $\mathcal{AC}$-agents, together with 50 clustering agents. This shows that the clustering quality decreases with a growing number of random $\mathcal{AC}$-agents.

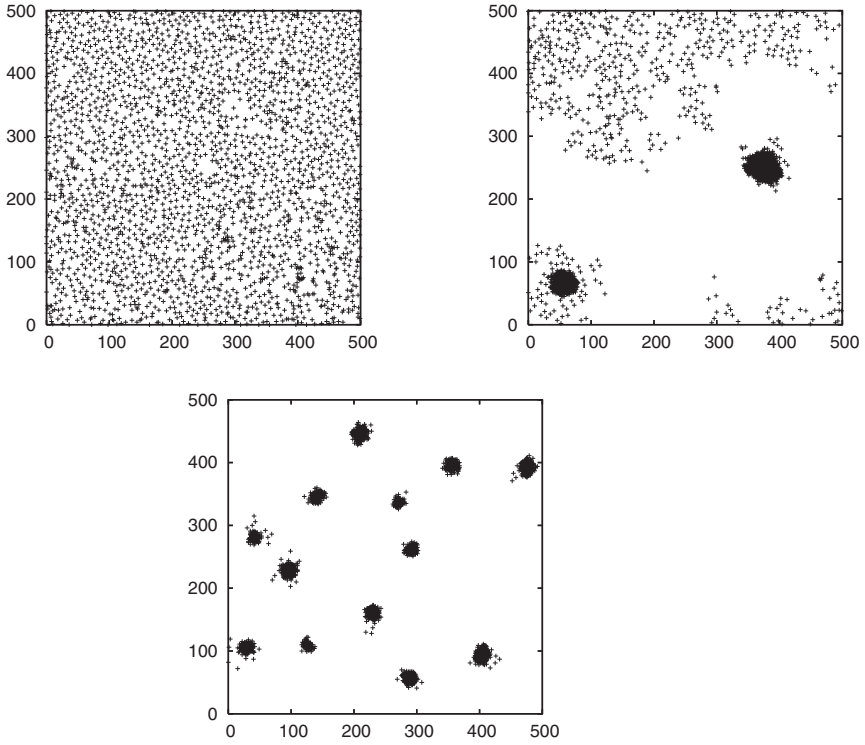

Fig. 6. Distribution of items with deterministic $\mathcal{AC}$-agents together with 50 standard agents after 50,000,000 steps; 50 deterministic $\mathcal{AC}$-agents (upper left), 35 deterministic $\mathcal{AC}$-agents (upper right), 10 deterministic $\mathcal{AC}$-agents (bottom)

Experiments with Deterministic $\mathcal{AC}$-Agents

The most interesting type of $\mathcal{AC}$-agents are the deterministic $\mathcal{AC}$-agents which have a deterministic picking and dropping behavior. Figure 6 shows the item distribution after 50,000,000 simulation steps for different numbers of anti-clustering agents. It can be seen that 50 deterministic $\mathcal{AC}$-agents clearly hinder the clustering agents from performing a successful clustering. This has been confirmed by tests where the initial item distribution was already clustered. In this case the deterministic $\mathcal{AC}$-agents destroyed the clustering successfully. The deterministic $\mathcal{AC}$-agents can be called efficient because they "win" against the same number of clustering agents.

A more detailed analysis is shown in Fig. 7, in which the Summary Function $J(r)$ is depicted. It can be seen in the upper-left subfigure that for 50 deterministic $\mathcal{AC}$-agents ordered patterns occur over time (i.e., no clustering occurs). Note that ordered patterns have been observed only for this type of

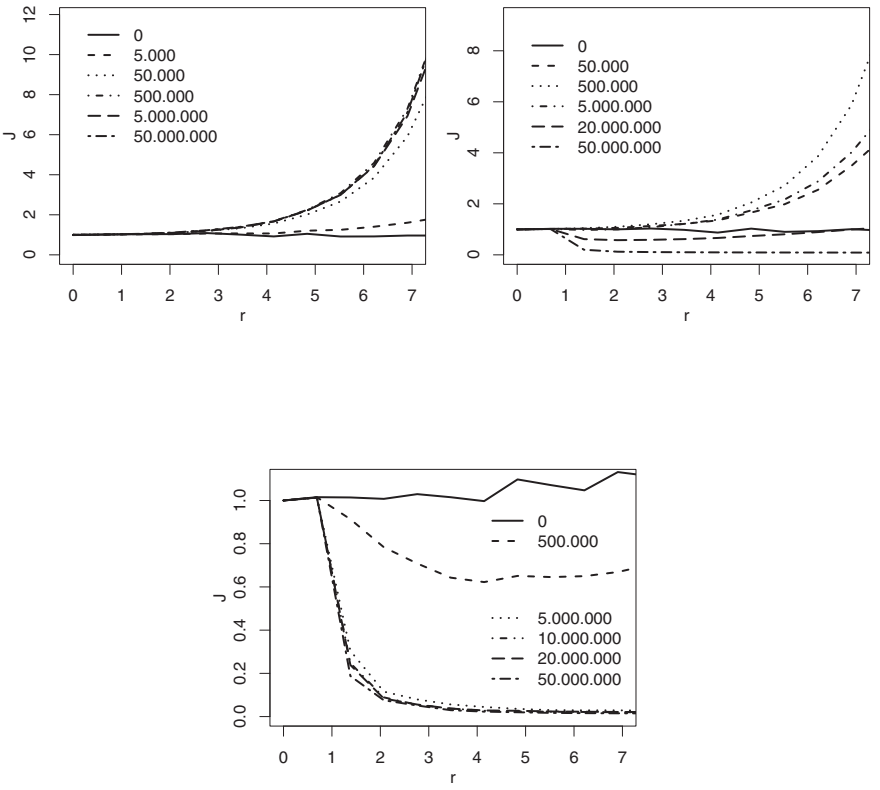

Fig. 7. Summary Function $J(r)$ after different number of time steps for different numbers of deterministic $\mathcal{AC}$-agents together with 50 standard agents for the test runs where the final clustering situation is shown in Fig. 6; 50 deterministic $\mathcal{AC}$-agents (upper left), 35 deterministic $\mathcal{AC}$-agents (upper right), 10 deterministic $\mathcal{AC}$-agents (bottom)

$\mathcal{AC}$-agent. Using only 10 $\mathcal{AC}$-agents cannot hinder the clustering agents from performing their task successfully (see Figs. 6 and 7).

An interesting behavior can be observed for a medium number of 35 deterministic $\mathcal{AC}$-agents. The upper-right subfigure of Figure 7 shows that after 500,000 steps ordered patterns occur, i.e., a clustering is prevented. But for an increasing number of steps, the ordered pattern disappears and a clustering occurs. After 20 million steps the value of the Summary Function at a radius of $r = 2$ is small ($J(2) \approx 0.5$) but for large radiuses the value becomes large (e.g., $J(7) \approx 7$). This is very interesting, because over time a situation

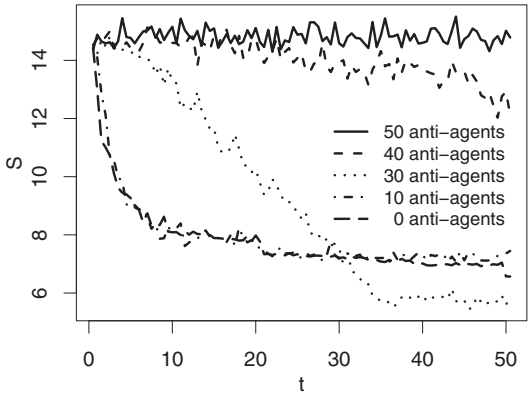

Fig. 8. Hierarchical social entropy S over time for different numbers $k \in 0, 10, 30, 40, 50$ of deterministic $\mathcal{AC}$-agents together with 50 standard agents

occurs where there is an ordering of items on a large scale but a clustering on a small scale. Moreover, it shows that a system where clustering agents are combined with anti-emergence agents can show a very complex behavior. The occurrence of ordered patterns that prevent the emergence behavior might exist only over certain time periods before the ordered patterns break down and the emergent behavior can appear.

Similarly as for the random $\mathcal{AC}$-agents, a medium number of deterministic $\mathcal{AC}$-agents even supports the clustering agents in their task. This can be seen in Fig. 8 where the values of the hierarchical social entropy $\mathcal{S}$ are shown over time for different numbers of deterministic $\mathcal{AC}$-agents. 50 agents can prevent a clustering ($\mathcal{S} > 14.0$); when using no $\mathcal{AC}$-agents a clustering with $\mathcal{S} \approx 7$ was achieved. But a medium number of 30 deterministic $\mathcal{AC}$-agents improves the final clustering with $\mathcal{S} \approx 6$.

Summarizing Remarks

Altogether the experiments with different types of anti-clustering agents that are added to a system with clustering agents have shown that: i) a not very high number of random $\mathcal{AC}$-agents or deterministic $\mathcal{AC}$-agents is enough to prevent the emergence of clustering, ii) only few $\mathcal{AC}$-agents may help the clustering agents to cluster the items faster, iii) a combination of $\mathcal{AC}$-agents and clustering agents may lead to situations which have an ordered pattern on a large scale and a clustered pattern on a small scale.

4 OC Systems with Moving Components

Emergent patterns that occur when groups of simple agents move is obviously an interesting topic for biology (e.g., [13, 11, 47, 49]) and robotics (e.g., [10]) but it is also interesting for the design of OC systems that consist of moving components or where parts of the system are embedded into moving objects.

In the following we discuss the emergent sorting behavior of simple ant-like moving agents (see [45]). Sorting here means that agents with different behavior can be found most often in different parts of the nest area (or movement area). The starting point of this investigation is a study of Sendova-Franks and Lent [45] where the authors simulate the movement behavior of real ants in their nest. Using different models of moving behavior it was shown that sorting occurs in all models. In each model the moving behavior of the ants differs. The strongest sorting effect occurred when the ants' behavior differs by the strength of an attraction force towards the nest center (centripetal ant model). For the other three models the ants' behavior differs by the maximal turning angle during movement. Ants with a small turning angle tend to keep to the wall once they have collided with it. Thus it was concluded in [45] that the colony center or the wall can play the role of a pivot (or beacon) which appears to be necessary for the sorting.

First, the occurrence of high concentrations of agents in the center of the nest area is considered here as it was observed before in the simulations of [45]. Secondly, some changes to the movement models are described in order to obtain a behavior that is more realistic for organic computing applications. It is discussed which behavioral differences can lead to an emergent sorting behavior for these movement models. Thirdly, a movement model with attraction force is investigated for the case where there is more than one center of attraction. This scenario is interesting, for example, when there exist several service stations for the components of the OC system.

4.1 Cellular Automata Model

The experiments with the moving agents' models that are described in this part have been done with a probabilistic cellular automaton model. The cellular automaton was designed so that it is suitable for approximation of the behavior of the continuous model that was used in [45]. The latter model tries to reflect the situation and the dimensions in a real ant colony (see [43] for more details). The cellular automaton has an array R of cells where the length of a cell corresponds to a length of 0.4 mm. For most experiments $R = \{1, \ldots, 75\} \times \{1, \ldots, 50\}$ was used. The neighborhood of a cell (x_0, y_0) are all cells $(x_0 + x, y_0 + y)$ with $x^2 + y^2 \leq 13$. The body of an agent in the cellular automaton consists of 21 cells arranged in a circle. Formally, an agent at position (x_0, y_0) occupies all cells $(x_0 + x, y_0 + y)$ with $x^2 + y^2 \leq 5$. Each agent i has an internal parameter $0 \leq \mu_i \leq 1$ that influences its moving behavior and an actual direction of moving $0 \leq \alpha_i < 2\pi$. For each agent the

probabilities of moving to one of the directly neighbored cells (i.e., the Moore neighborhood $n \in N_M = \{(-1,-1),\ldots,(1,1)\}$) are calculated. In order for an agent to be able to move, all cells of the new place must be free (and within the array R).

In the cellular automaton model the agents move at each time step in random order so that each agent moves at most one cell per time step. Since an agent can move only to discrete positions it might not be possible for an agent to move exactly in direction α. Therefore, an agent has a probabilistic movement behavior where α is its expected direction. Similarly, the expected velocity is 0.3 mm/s = 0.75 cells/time step on average when considering a free run of an agent with no obstacles.

4.2 Movement Models

In this section we describe several movement models for agents. The first two models were proposed in [45] to model the movement behavior of real ants.

Avoiding Ant Model. In this model the agents do a correlated random walk. If unobstructed, i.e., an agent does not collide with a nestmate or the nest wall, the movement is as follows. The turning angle Θ_i is chosen randomly according to a uniform distribution between $-\Theta_i$ and Θ_i. The maximal turning angle Θ_i is different for all agents and depends on their individual parameter μ_i: $\Theta_i = (1-\mu_i)\Theta^0 + \mu_i\Theta^1$. In [45] the standard values were $\Theta^0 = 60$ and $\Theta^1 = 15$. If the nest wall or a nestmate is in the sensing range, the agent will not move but only change its moving angle. In this case it avoids the obstacle explicitly by turning in one direction until it can move again. To define the turning direction assume that agent i collides with agent j. The sign of the scalar product between the vector that is perpendicular to the vector of the moving direction of agent i and the vector from the center of agent i to the center of agent j determines the direction of turning: $\Theta_i \leftarrow \text{sign}((-\sin\alpha_i, \cos\alpha_i)\cdot(x_j - x_i, y_j - y_i))\cdot r$ where r is chosen randomly from a uniform distribution in the interval $(0, \Theta_i)$. A collision with the nest wall is handled analogously.

Centripetal Ant Model. It is very likely that agents have the ability to detect gradients in gas (CO_2) or pheromone concentrations [39]. Since the concentration of the gas is maximal in the center region of the nest where the brood is located [7], this could give the agent a chance to estimate the direction to the center. In the Centripetal Ant Model this is used to establish an attraction force towards the center of the nest. This attraction is different for different agents and depends on their internal parameter μ_i. For the calculation of the moving behavior a modified model from the clinotaxis model from [22] is used: $\Theta_i \leftarrow p_u\chi_i + p_b\tau_i \cdot (1-\cos(\phi_i))/2$ where ϕ_i is the angle between the actual moving direction α_i and the vector towards the center of the nest. The values of p_u and p_b are randomly chosen from $\{-1, 1\}$ and they determine the direction of turning. The turning behavior depends on ϕ_i and the larger this angle

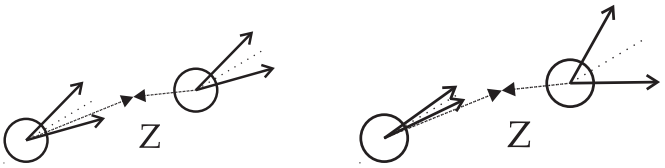

Fig. 9. Effect of different values of the internal parameter μ_i on the turning behavior in the Centripetal Ant Model; Z is the center of the nest; (left) for large μ_i there is only a slight difference between moving from or to the center; (right) for small μ_i the turning angle becomes significantly smaller the larger the angle between actual moving direction and the vector to the center

the more the agent will turn. The parameters χ_i and τ_i depend on the internal parameter μ_i of the agent: $\chi_i \leftarrow (1-\mu_i)\chi^0 + \mu_i\chi^1$ and $\tau_i \leftarrow (1-\mu_i)\tau^0 + \mu_i\tau^1$ with $\chi^0 = 0°$, $\chi^1 = 15°$, $\tau^0 = 30°$, and $\tau^1 = 0°$. Agents with larger μ_i will not be that much affected by their ϕ_i as agents with small μ_i (see Fig. 9). Therefore, for the agents with small μ_i the attraction to the colony center is larger than for agents with large μ_i.

It was shown in [43] that when the agents move according to the Centripetal Ant Model there occurs a cluster of non-moving agents in the center of the nest. Obviously, such a situation should not occur in OC systems. Therefore, the following variation of the movement models has been introduced. This model is also simple but the agents try to avoid a situation where they get stuck.

Model with Repulsive Behavior (Repulsive Model). The Centripetal Ant Model is slightly changed by modifying the agents' behavior in the case of a collision with a nestmate or the nest wall. In this case the agent turns according to the Avoiding Ant Model. Otherwise the moving behavior remains as in the Centripetal Ant Model. Hence, in the Repulsive Model the turning behavior of the agent is different for different situations.

For OC systems with moving components the question emerges whether small differences in speed or activity of the components can lead to a sorting effect. Differences in speed or activity can obviously occur when different types of components are used in such systems. But they might also occur when the components have power supplies of different quality or some components are loaded and carry items whereas other components are unloaded. An interesting question is whether differences in speed or activity between the agents can lead to a sorting effect even for very simple types of ant-like movement behavior. To investigate this two new movement models have been introduced that are described in the following.

Model with Speed Differences (Speed Model). In this model the agents have different velocities. The velocities are equidistantly distributed between ν_0,

$0 < \nu_0 < 1$, and 1, i.e., $\nu_i = \nu_0 + (i-1)(1-\nu_0)/(n-1)$ for $i \in \{1, \ldots, n\}$. Note that agent i moves ν_i times as fast as agent n. The position of the agents is now updated according to the formulas $x_i \leftarrow x_i + \nu_i \cdot \delta \cdot \cos\alpha_i$ and $y_i \leftarrow y_i + \nu_i \cdot \delta \cdot \sin\alpha_i$. The different velocities are realized in the stochastic cellular automaton such that agent i has expected velocity $\nu_i \cdot 0.3$ mm/s (recall that 0.3 mm/s is approximately the speed of a real ant). The movement and turning behavior are the same for all agents and do not depend on their internal parameter μ_i. All agents move and turn like an agent in the Centripetal Model with $\mu_i = 0$.

Model with Activity Differences (Activity Model). The Activity Model is similar to the Speed Model. The difference is that the agents have not only different velocities but also different turning behaviors. Similarly to the velocities, the turning angle is scaled by ν_i. Formally, if agent i can move, its turning angle is determined according to $\alpha_i \leftarrow \alpha_i + \nu_i \cdot \Theta_i$, with Θ_i calculated as in the Centripetal Ant Model. If the agent is obstructed, the turning behavior is defined as in the Avoiding Ant Model.

4.3 Experiments with Simple Environments

If not stated otherwise all experiments that are described in the following have been done over 100,000 time steps with 40 agents. The colony center is the point $Z = (35, 25)$. The distance of agent i from the colony center is computed as $r_i(t) = d((x_i \cdot 0.4 - 0.2, y_i \cdot 0.4 - 0.2), (15, 10))$ where (x_i, y_i) is the center of agent i in time step t. The distance r_i of agent i from the colony center is measured every 100 time steps. For a given time step t the mean distance $r_i^{\varnothing}(t)$ of agent i from the center is the average over all measured distances up to t. Let $r_i^{\varnothing} = r_i^{\varnothing}(100000)$ be the average distance measured over all time steps.

As in [45], Pearson's correlation coefficient k is used to measure the correlation of parameter μ_i and the mean distance of an agent from the nest center. A high value (≈ 1) for $k(t)$ indicates a strong correlation. As a second measure the slope of the linear relationship between the mean agent distance from the nest center and the internal parameter μ_i is measured. Given the mean distances $r_i^{\varnothing}(t)$ for all agents $i = 1, \ldots n$, the linear regression determines values $\alpha(t)$ and $\beta(t)$ such that the sum $\sum_{i=1\ldots n}(r_i^{\varnothing}(t) - (\alpha(t) + \beta(t)\mu_i))^2$ is minimized. The slope $\beta(t)$ can be seen as a measure for the degree of sortedness of the agents.

To depict the spatial distribution of the agents a similar strategy is used as proposed in [45]. The nest is divided into $15 \times 10 = 150$ squares and the agents are divided into five groups depending on their internal parameter $\mu_i : [0.0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0)$. For each of these intervals the number of agents in every square was counted and divided by the total number of agents in the group. To investigate how active the agents are in

different areas of the nest it was counted for every cell how often an agent enters that cell (measured over all time steps).

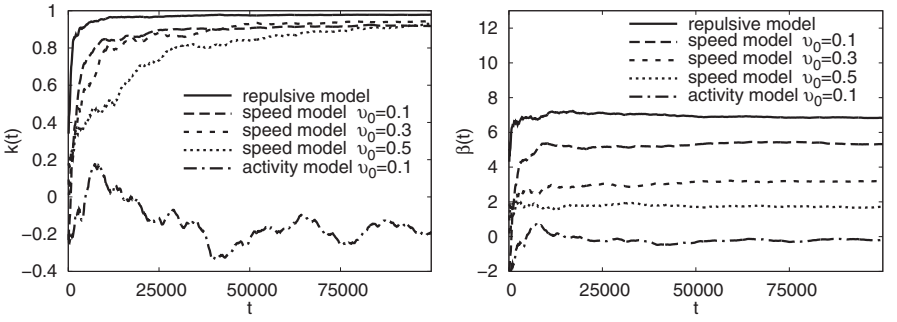

Fig. 10. Changes of correlation coefficient $k(t)$ (left) and slope $\beta(t)$ (right) for Repulsive Model, Speed Model, and Activity Model

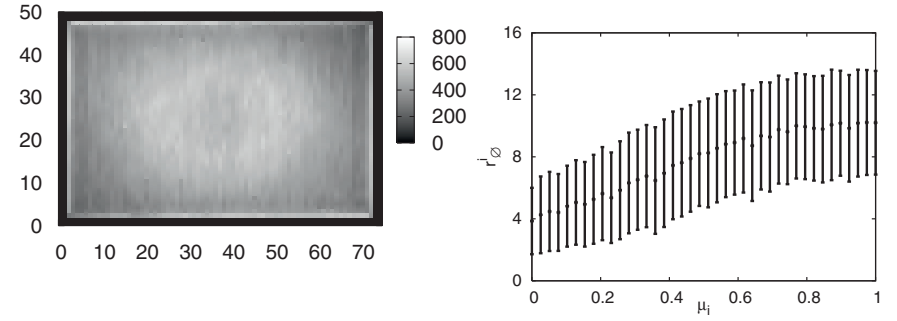

Fig. 11. Repulsive Model: number of time steps where an agent enters a cell (left) and average distance to nest center $r_i^{\varnothing}$ (right)

The results for different movement models with respect to the correlation coefficient $k(t)$ and changes of the slope $\beta(t)$ of the regression function are compared in Fig. 10. The figure shows that in the Repulsive Model and in the Speed Model the agents show a clear sorting behavior. In the Activity Model there is no clear indication for an agent sorting.

The motivation to introduce the Repulsive Model was to prevent the agents from getting stuck in the center of the nest. The left part of Fig. 11 shows that the agents in the center have a high movement activity and do not get stuck. The strength of the sorting behavior is shown in the right part of Fig. 11. Ants with $\mu \approx 0$ have an average distance of approximately 4 from the

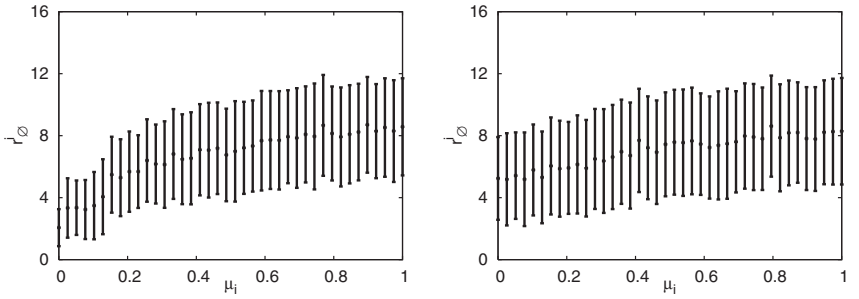

Fig. 12. Speed Model: average distance to nest center $r_i^{\emptyset}$; movement speed $\nu_0 = 0.1$ (left) and $\nu_0 = 0.3$ (right)

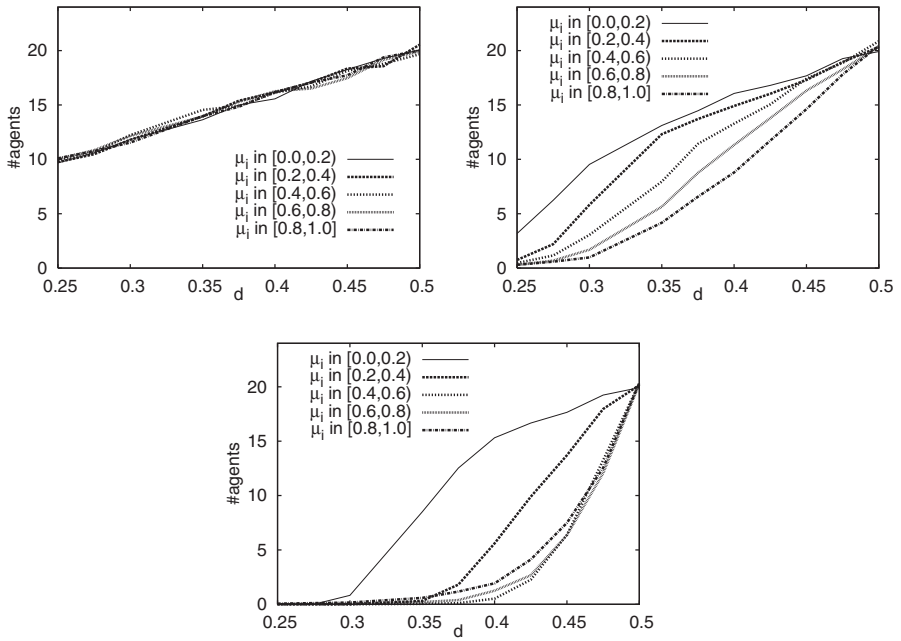

Fig. 13. Two nest centers; average number of agents for different classes $\mu_i \in [0.0, 0.2)$, $[0.2, 0.4)$, $[0.4, 0.6)$, $[0.6, 0.8)$, $[0.8, 1.0)$ in the smaller of the two areas at time step 0 (upper left), 2000 (upper right), and 50,000 (bottom); results are averaged over 100 test runs

center whereas agents with $\mu \approx 1$ have an average distance of approximately 10. The strength of the sorting behavior for the Speed Model is shown in Fig. 12. For large relative differences in movement speed ($\nu_0 = 0.1$) the sorting behavior is stronger than for smaller differences ($\nu_0 = 0.3$).

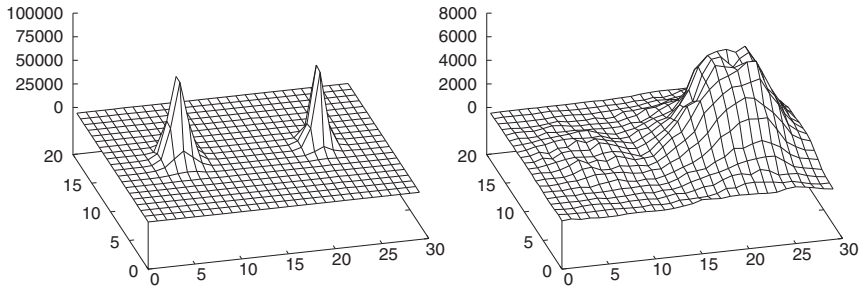

Fig. 14. Two nest centers; $d = 0.4$; $\mu_i \in [0, 0.2)$ (left) and $\mu_i \in [0.8, 1]$ (right); for both influence areas it is counted how often an agent enters a cell (measured over 50,000 time steps)

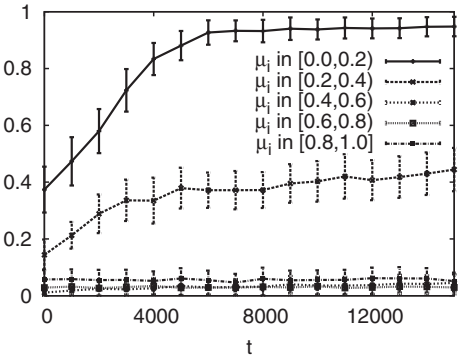

Fig. 15. Activation of the nest center in the larger area after $t \in \{0, 1000, \ldots, 15000\}$ simulation steps; $d = 0.4$; depicted is the fraction of agents for different classes $\mu_i \in [0.0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0)$ that are in the smaller area after 50,000 simulation steps

4.4 Experiments with Complex Environments

For applications in organic computing it is interesting to consider complex environments with more than one focal point for the movement behavior. An example where this occurs is moving components which have several service stations they can visit. Therefore, a much larger environment of 600 cells $\times$ 400 cells with two centers (located at $(150, 200)$ and $(450, 200)$) is used for the experiments. Simulations were done with the Speed Model and 200 agents. The area is divided vertically at position $d \times 600$, $d \in [0, 1]$, such that in the left (right) part of area the turning behavior of the agents is influenced

by the left (right) center. The effect is that the agents tend to turn toward the corresponding center. Note that for $d = 0.25$ the line dividing both areas passes exactly the left nest center. The agents are divided into five classes according to their μ-values. For all classes the number of agents in the left and right part of the area are counted for $d \in \{0.25, 0.30, \ldots, 0.50\}$.

The number of agents at different time steps is shown in Fig. 13 (results are averaged over 100 runs). A clear differentiation of the five agent classes is occurring over time. At the beginning the agents are equally distributed among the five classes for different values of d. Over time the faster agents (large μ_i values) occur more often than the slow agents in the larger part of the area.

The movement activity of agents with $\mu_i \in [0, 0.2)$ and agents with $\mu_i \in [0.8, 1)$ in different parts of the nest area for $d = 0.4$ is compared in Fig. 14. It can be seen that the slow agents occur next to both centers whereas the fast agents occur mainly in the part of the area with the center that has the larger influence region. These results show that moving agents with slightly different moving behavior can have very different spatial distributions in areas with several service stations.

A dynamic scenario is also considered where the service stations for the agents are not available starting from the same time. In the corresponding experiment it is assumed that the center in the larger part of the area becomes active several time steps later than the center in the smaller region. The results are given in Fig. 15 for the case where the center in the larger part of the area becomes active at time step $t \in \{0, 1000, \ldots, 15000\}$. Shown is the fraction of agents in each of the classes $\mu_i \in [0.0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0)$ that are in the smaller area after 50,000 simulation steps. It can be seen, that the slow agents with $\mu_i \in [0.0, 0.2)$ are much more concentrated within the small part of the area if the center in the other part of the area becomes active late (more than 90% of these agents appear in the smaller part of the area if the service station appeared after $t >$6000 simulation steps). The reason for this is that most of the slower agents are still fast enough to concentrate around the center in the small region during the first 6000 time steps. After that time they will leave the small area only with a very small probability. On the other hand it can be seen, that a large fraction of the faster agents ($\mu_i > 0.4$) can always be found in the larger area regardless of when the second center was added. This mechanism can possibly be used to implement a controlled separation process of agents with different properties in OC systems.

4.5 Summarizing Remarks

Emergent spatial sorting patterns for groups of randomly moving ant-like agents can be observed for simple movement models when there exist slight differences in the individual behavior of the agents. For different movement

models the emergent spatial sorting effects have been described based on the results of simulation studies.

Scenarios with more complex environments where the movement of the agents can be influenced by several "center points" have also been considered. Such scenarios are relevant for applications in organic computing where the center points can be seen as service points for moving components of the system. It is interesting that the relative size of the influence area of the service points leads to an emergent effect that the spatial distribution of agents might differ strongly for agents with only slightly different moving behavior. A dynamic scenario where different times span between the activation times of two service stations for the agents was studied. Simulations have shown that the length of the time span has a significant influence on the distribution pattern of the agents and the type of influence is different for different moving behaviors.

5 Final Remarks

The connections between swarm intelligence and the new field of organic computing have been discussed in this chapter. Organic computing systems (OC systems) consist of many autonomous components that interact and show forms of collective behavior. OC systems are designed to possess different self-x properties (e.g., self-healing, self-managing, self-optimizing). Typically, OC systems will have a decentralized control and are able to adapt to their environment or the requirements of the user. Thus, OC systems share some important properties with social insect colonies. Clearly, there exist also many differences between technical systems like OC systems and biological systems like social insect colonies. Nevertheless, it was argued in this chapter that the similarities make methods of swarm intelligence that are often inspired by principles of collective behavior of social insects attractive for organic computing.

We have described some examples of OC systems and presented two case studies that show in detail how methods of swarm intelligence are connected to problems in organic computing. The topic of the first case study is a new approach to control emergent behavior in OC systems. This method is called swarm-controlled emergence and it was applied to control emergent clustering effects that can occur when a group of ant-like agents take up items, carry them around, and drop them. The starting point of the second study is an observation that biologists made with ants. Ants with slightly different movement behavior can be found on average in different parts of the nest. It is discussed what consequences this emergent effect might have on OC systems with moving components. Both case studies have shown that organic computing is strongly linked to swarm intelligence. There is a large potential for applications of swarm intelligence methods in organic computing and design

problems for OC systems will provide new challenges for the development of new swarm intelligence methods.

References

1. A. Baddeley, R. Turner: `spatstat` website, URL: `www.spatstat.org`
2. C. Anderson, J. J. Boomsma, J. J. Bartholdi, III: Task partitioning in insect societies: bucket brigades. *Insectes Sociaux*, 49(2):171–180, 2002.
3. T. Balch: Hierarchical Social Entropy: An information theoretic measure of robot team diversity. *Autonomous Robots*, 8(3):209–237, 2000.
4. J.-C. de Biseau, J. M. Pasteels: Response thresholds to recruitment signals and the regulation of foraging intensity in the ant *Myrmica sabuleti* (Hymenoptera, Formicidae). *Behavioural Processes*, 48(3):137–148, 2000.
5. A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, A. Bernauer, O. Bringmann, W. Rosenstiel: Organic computing at the system on chip level. In: G. De Micheli, S. Mir, and R. Reis (editors), *Proc. IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC 2006)*. Springer, Berlin, Germany, 338–341, 2006.
6. E. Bonabeau, M. Dorigo, G. Theraulaz: Swarm Intelligence: ¿From Natural to Artificial Systems. Oxford University Press, New York, 1999.
7. M.D. Cox, G.B. Blanchard: Gaseous templates in ant nests. *Journal of Theoretic Biology*, 204:223–238, 2000.
8. L. Chrétien: Organisation spatiale du matériel provenant de l'excavation du nid chez Messor barbarus et des cadavres d'ouvrières chez Lasius niger (Hymenopterae: Formicidae). Université Libre de Bruxelles, Département de Biologie Animale, 1996.
9. R. Cunningham, J. Dowling, A. Harrington, V. Reynolds, R. Meier, V. Cahill: Self-Optimization in a Next-Generation Urban Traffic Control Environment. *ERCIM News*, 64:55–56, 2006.
10. J.-L. Deneubourg, S. Goss, N. Franks, A.B. Sendova-Franks, C. Detrain, L. Chretien: The dynamics of collective sorting: Robot-like ants and ant-like robots. In: J.-A. Meyer and S. Wilson, editors, *Proc. of the 1st Int. Conf., on Simulation of Adaptive Behavior: From Animals to Animats*, MIT Press/ Bradford Books, 356–363, 1991.
11. S. Depickère, D. Fresneau, J.-L. Deneubourg. A basis for spatial and social patterns in ant species: dynamics and mechanisms of aggregation. *Journal of Insect Behavior*, 17(1):81–97, 2004.
12. J. Delgado, R.V. Sole: Self-synchronization and task fulfilment in ant colonies, *Journal of Theoretical Biology*, 205(3):433–441, 2000.
13. J.L. Deneubourg, A. Lioni, C. Detrain: Dynamics of aggregation and emergence of cooperation. *The Biological Bulletin*, 202:262–267, 2002.
14. S. Depickère, D. Fresneau, J.-L. Deneubourg: Dynamics of aggregation in *Lasius niger* (Formicidae): influence of polyethism. *Insectes Sociaux*, 51(1):81–90, 2004.
15. M. Dorigo, T. Stützle: Ant Colony Optimization. MIT Press, 2004.
16. D. Fey, M. Komann, F. Schurz, A. Loos: An organic computing architecture for visual microprocessors based on marching pixels. *Proc. IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, IEEE Press, 2689–2692, 2007.

17. S. P. Fekete, C. Schmidt, A. Wegener, S. Fischer: Hovering data clouds for recognizing traffic jams. In: T. Margaria, A. Philippou, and B. Steffen (editors), *Proc. 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (IEEE-ISOLA 2006)*, IEEE Press, 213–218, 2006.
18. N.R. Franks, A. Sendova-Franks: Brood sorting by ants: distributing the workload over the work surface. *Behav. Ecol. Sociobiol.*, 30:109–123, 1992.
19. Gesellschaft für Informatik: Organic Computing / VDE, ITG, GI - Positionspapier. 2003.
20. H. Gutowitz: Complexity Seeking Ants. Unpublished report, 1993.
21. P. P. Grassé: La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La theorie de la stigmergie: Essai d'interpretation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959.
22. D. Grünbaum, Schooling as a strategy for taxis in a noisy environment. *Evolutionary Ecology*, 12:503–522, 1998.
23. J. Handl, B. Meyer: Improved ant-based clustering and sorting in a document retrieval interface. In: J.-J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Proc. Seventh Int. Conference on Parallel Problem Solving from Nature (PPSN VII)*. volume 2439 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 913–923, 2002.
24. J. Handl, J. Knowles, M. Dorigo: Strategies for the increased robustness of ant-based clustering. In: G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, editors, *Postproc. of the First International Workshop on Engineering Self-Organising Applications (ESOA 2003)*, volume 2977 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 90–104, 2003.
25. J. Handl, J. Knowles, M. Dorigo: On the performance of ant-based clustering. In: A. Abraham, M. Köppen, and K. Franke, editors, *Proc. 3rd Int. Conf. on Hybrid Intell. Systems (HIS 2003)*, IOS Press, 2003.
26. S. Janson, M. Middendorf, M. Beekman: Searching for a new home — scouting behavior of honeybee swarms. *Behavioral Ecology*, 18:384–392, 2007.
27. P. M. Kanade, L. O. Hall: Fuzzy Ants as a clustering concept. In: E. Walker, editor, *Proceedings 22nd International Conference of the North American Fuzzy Information Processing Society NAFIPS 2003*, IEEE Systems Man and Cybernetics Society, 227–232, 2003.
28. M. Komann, D. Fey: Marching pixels computing principles in embedded systems using organic parallel dardware. *Proc. International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, 369–373, IEEE CS Press, 2006.
29. P. Kuntz, D. Snyers: Emergent colonization and graph partitioning. In: D. Cliff et al., editors, *Proc. of the 1st Int. Conf, on Simulation of Adaptive Behavior: From Animals to Animats*, MIT Press, 494–500, 1994.
30. N. Labroche, N. Monmarche, G. Venturini: A new clustering algorithm based on the chemical recognition system of ants. In: F. van Harmelen, editor, *Proc. of the 15th European Conference on Artificial Intelligence, ECAI 2002*, IOS Press, 345–349, 2002.
31. N. Labroche, N. Monmarche, G. Venturini: AntClust: Ant clustering and web usage mining. In: E. Cantú-Paz et al., editors, *Proc. of the 2003 Genetic and Evolutionary Computation Conference*, volume 2723 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 25–36, 2003.

32. N. Labroche, N. Monmarche, G. Venturini: Visual clustering with artificial ants colonies. In: V. Palade, R.J. Howlett, L.C. Jain, editors, *Proc. of the 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES 2003)*, volume 2773 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 332–338, 2003.
33. M.N.M. van Lieshout, A.J. Baddeley: A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica*, 50:344–361, 1996.
34. D. Merkle, M. Middendorf, A. Scheidler: Swarm controlled emergence — Designing an anti-clustering ant system. *Proc. of the 2007 IEEE Swarm Intelligence Symposium*, IEEE Press, 10 pages, 2007.
35. F. Mösch, M. Litza, A. El Sayed Auf, E. Maehle, K.-E. Großpietsch, W. Brockmann: ORCA – Towards an organic robotic control. In: H. de Meer and J.P.G. Sterbenz, editors, *Proc. 1st International Workshop on Self-Organizing Systems (IWSOS 2006) and 3rd International Workshop on New Trends in Network Architectures and Services (EuroNGI 2006)*, volume 4124 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 251–253, 2006.
36. M. Mnif, C. Müller-Schloer: Quantitative emergence. In: *Proc. of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (IEEE SMCals 2006)*, 2006.
37. C. Müller-Schloer, C. von der Malsburg, R. P. Würtz: Organic computing. *Informatik Spektrum*, 27(4):332–336, 2004.
38. C. Müller-Schloer, B. Sick: Emergence in organic computing systems: Discussion of a controversial concept. In: L.T. Yang, H. Jin, J. Ma, and T. Ungerer, editors, *Proc. 3rd International Conference on Autonomic and Trusted Computing*, volume 4158 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 1–16, 2006.
39. G. Nicolas and D. Sillans: Immediate and latent effects of carbon dioxide on insects. *Annual Review of Entomology*, 34:97–116, 1989.
40. U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, H. Schmeck: Towards a generic observer/controller architecture for organic computing. Köllen Verlag, LNI P-93, 112–119, 2006.
41. F. Rochner, H. Prothmann, J. Branke, C. Müller-Schloer, H. Schmeck: An organic architecture for traffic light controllers. *Proc. GI Jahrestagung 2006*, Lecture Notes in Informatics 93:120–127, 2006.
42. F. Rochner, C. Müller-Schloer: Emergence in technical systems. *it — Special Issue on Organic Computing*, 47:188–200, 2005.
43. A. Scheidler, D. Merkle, M. Middendorf: Emergent sorting patterns and individual differences of randomly moving ant like agents. In: S. Artmann and P. Dittrich, editors, *Proc. 7th German Workshop on Artificial Life (GWAL-7)*, IOS Press, 11 pp., 2006.
44. H. Schmeck: Organic computing – A new vision for distributed embedded systems. *Proc. of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, IEEE CS Press, 201–203, 2005.
45. A.B. Sendova-Franks, J.V. Lent: Random walk models of worker sorting in ant colonies. *Journal of Theoretical Biology*, 217:255–274, 2002.
46. T. Schöler, C. Müller-Schloer: An observer/controller architecture for adaptive reconfigurable stacks. In: M. Beigl and P. Lukowicz, editors, *Proc. of the 18th International Conference on Architecture of Computing Systems (ARCS 05)*,

volume 3432 of Lecture Notes of Computer Science, Springer, Berlin, Germany, 139–153, 2006.
47. R.V. Sole, E. Bonabeau, J. Delgado, P. Fernandez, J. Marin: Pattern formation and optimization in army ant raids. *Artificial Life*, 6(3):219–226, 2000.
48. G. Theraulaz, E. Bonabeau, J.L. Deneubourg: Response threshold reinforcement and division of labour in insect colonies. *Proc. Roy. Soc. London B*, 265:327–335, 1998.
49. G. Theraulaz, E. Bonabeau, S.C. Nicolis, R.V. Sole, V. Fourcassie, S. Blanco, R. Fournier, J.L. Joly, P. Fernandez, A. Grimal, P. Dalle, J.L. Deneubourg: Spatial patterns in ant colonies. *Proc. Natl. Acad. Sci.*, 99(15):9645–9649, 2002.
50. G. Theraulaz, E. Bonabeau, R.V. Sole, B. Schatz, J.L. Deneubourg: Task Partitioning in a ponerine ant. *Journal of Theoretical Biology*, 215(4):481–489, 2002.