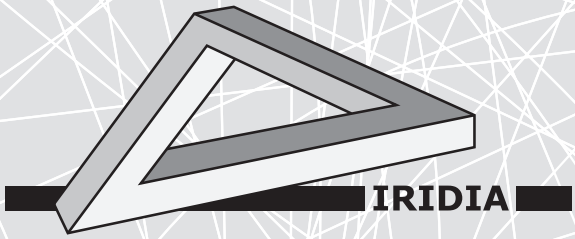


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Learning from House-Hunting Ants: Collective Decision-Making in Organic Computing Systems

Arne Brutschy, Alexander Scheidler, Daniel Merkle, and Martin Middendorf

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2008-016

June 2008

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2008-016

Revision history:
TR/IRIDIA/2008-016.001 June 2008

Learning from House-Hunting Ants: Collective Decision-Making in Organic Computing Systems

Arne Brutschy[1], Alexander Scheidler[2],
Daniel Merkle[3], and Martin Middendorf[2]

[1] IRIDIA, CoDE, Université Libre de Bruxelles,
Brussels, Belgium arne.brutschy@ulb.ac.be
[2] Parallel Computing and Complex Systems Group, Computer Science Department,
University of Leipzig, Leipzig, Germany {scheidler,middendorf}@uni-leipzig.de
[3] Department of Mathematics and Computer Science, University of Southern
Denmark, Odense M, Denmark daniel@imada.sdu.dk

Abstract. This paper proposes ant-inspired strategies for self-organized and decentralized collective decision-making in computing systems which employ reconfigurable units. The particular principles used for the design of these strategies are inspired by the house-hunting of the ant *Temnothorax albipennis*. The considered computing system consists of two types of units: so-called worker units that are able to execute jobs that come into the system, and scout units that are additionally responsible for the reconfiguration process of all units. The ant-inspired strategies are analyzed experimentally and are compared to a non-adaptive reference strategy. It is shown that the ant-inspired strategies lead to a collective decentralized decision process through which the units are able to find good configurations that lead to a high system throughput even in complex configuration spaces.

Key words: organic computing, reconfigurable units, swarm intelligence, decision-making, house-hunting, ant-inspired

1 Introduction

With the increasing complexity of modern computing systems, new design paradigms become important with regards to solving the resulting management and reliability issues. Computing systems that follow the paradigms of Autonomic Computing (e.g., [1]) or Organic Computing (e.g., [2, 3]) should ideally be able to address certain critical tasks autonomously, follow the principle of self-organization, and possess so-called self-x properties. Examples of self-x properties are self-management, self-reconfiguration, self-optimization, self-servicing and of course self-organization itself.

Supplementary online material: http://iridia.ulb.ac.be/supp/IridiaSupp2008-006/

Closely related to the property of self-reconfiguration is the field of dynamically reconfigurable hardware (e.g., [4]). A central problem in systems that use dynamically reconfigurable hardware is to find a good reconfiguration strategy for deciding when to carry out reconfiguration operations and which configurations to use. In these systems, a trade-off exists between increasing reconfiguration costs (i.e. when reconfiguration operations are carried out more frequently) and the possible speed gain resulting from adapted configurations.

Because designing computing systems that work autonomously and are able to make collective decisions is complicated, principles of self-organized decision-making in social insects (see, e.g., [5, 6]) have become an important source of inspiration for system designers. This approach has been successfully applied to systems in different technical domains, e.g., scheduling [7], task-allocation in networks [8] and robotics [9].

In this paper, we propose a nature-inspired reconfiguration strategy for systems that consist of reconfigurable units. The proposed strategy is inspired by principles that are used by the ant *Temnothorax albipennis* for house-hunting. To our best knowledge, these principles have been exploited so far only once for the design of a technical system [10]. It is shown that a cluster of decentralized and self-organized reconfigurable units which uses the proposed strategy can adapt itself to a dynamic environment by making an efficient compromise between accuracy and speed of the decision-making process.

The paper is organized as follows. A brief overview on the biological system that inspired the proposed strategy is given in Section 2. Section 3 describes the model of the computing system with reconfigurable units. Some details about the reconfiguration strategies are explained in Section 4. Experiments and results are presented in Section 5. Conclusions are given in Section 6.

2 House-Hunting in *Temnothorax Albipennis*

Temnothorax albipennis is an ant species that has small workers and lives in small colonies. As it tends to build the nests in structurally unstable places such as in crevices and under rocks, the ant has to emigrate frequently [11]. Due to its peculiar properties, and because having to deal with only a small number of individuals is an advantage, the emigration behaviour of *T. albipennis* has been studied thoroughly [11–13]. The details of the emigration process are described in the following.

When emigration is necessary, the scout ants (which form approximately 20-30% of the colony [14]) start to search the surrounding environment for a new nest site. Upon finding a candidate site, a scout starts to assess the site according to several criteria (e.g., size and darkness). If the scout considers the site to be superior to the current nest, it tries to get a "second opinion" by guiding another scout to the candidate nest site. The guidance is accomplished by *tandem-running*, which means that one ant teaches another ant the route by leading it while keeping close physical contact to allow bidirectional feedback. This makes this technique slow and therefore costly [15]. As soon as the other

scout reaches the candidate nest site, it assesses it and, if it considers it to be good, starts recruiting another scout as well. Scouts delay recruitment to a candidate nest site by a time that is inversely proportional to the perceived quality of the site. This behaviour ensures that better nests will attract scouts faster, thus making the emigration an autocatalytic process.

As soon as a certain number of ants prefer a certain candidate nest site, the scouts will switch from tandem-running to a transportation behaviour known as *social-carrying*. In social-carrying, a scout picks up a passive ant or brood and carries it to the new nest site. Social carrying is three times faster than tandem-running, but has the disadvantage that a carried ant does not learn the route between the old and the new nest site. The switch from tandem-running to social-carrying can be seen as the start of the actual emigration process towards the new nest.

The number of scouts that are required for the behaviour to switch to social-carrying (and thus for making the decision for the nest site) is called *quorum threshold*. By adapting the quorum threshold to the colony's needs, *T. albipennis* is able to make a compromise between the accuracy and speed of its decisions. A low quorum threshold leads to fast but error-prone decisions. This might be acceptable when a fast emigration is required and the quality of the new nest is of minor importance, for example, when the current nest has been destroyed by a predator. On the other hand, if the ants have more time for emigration (e.g. when the current nest is to small to support the colony), the ants can take enough time to select a superior site by requiring a high quorum threshold. As the behaviour of the ants makes them capable of decentralized and self-organized decisions, we think this strategy has potential to be applied successfully in the technical domain.

3 Model of the Organic Computing System

The model for an organic computing system that is used for our study is described in this section. The system consists of a set of computing units. These units are connected by a simple interconnection network that only allows one-to-one communication; broadcast operations are not possible. Units can communicate with any other single unit at any time. As no complicated communication is employed, we do not associate any costs with communication. The purpose of the system is to maximize the total throughput of jobs. Throughput is measured as the number of jobs processed by the system per time step. In the current model it is assumed that there are always j different job types in the system. Additionally, it is assumed that the system is saturated with jobs, meaning that jobs have to be executed at every time step and the units are never idle. All units work in parallel and each unit can only work on a single job at one time step.

Each computing unit consists of s slices that can be reconfigured independently (see also [16]). Jobs are executed by the unit using computational logic configured in these slices. The more slices are configured for a certain type of

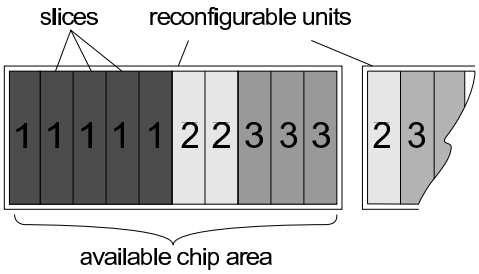


Fig. 1. Example partition of a unit's configuration resources in 10 slices. Each slice is configured to one of three possible job types. Numbers denote the job type the respective slice is configured to.

job, the bigger the computational logic can be. The degree of specialization of a unit for a certain type of job depends on the number of slices that are configured for this job type. Hence, the set of slices of a unit is partitioned with respect to the different types of jobs. This partition is called the configuration of the unit (see Figure 1 for an example). A configuration can be described as a vector $c(i)$ with $\sum c(i) = s$ and $1 \leq i \leq j$. Each unit has to be able to service every type of job, i.e. at least one slice has to be configured for every type of job. Hence, $1 \leq c(i) \leq m$ with $m = s - j + 1$ is the maximum number of slices available for a single job. As the purpose of the system is to maximize the throughput of jobs, configurations which result in higher throughput are considered to be superior to others.

It is assumed that the computing system consists of two types of units: units that are only able to execute jobs, and units that can execute jobs and are additionally able to reconfigure themselves or other units. The former units are referred to as *worker units*, whereas the latter units are called *scout units*. The computing system consists of n_w worker units and n_s scout units. Let $n = n_w + n_s$. The ratio between scouts and workers is assumed to be fixed. A worker unit can only change configuration with the help of a scout unit. One motivation for this is that a reconfigurable system needs to know, or must be able to compute reconfiguration data that is required to define the new configuration. In our case, only the scouts are able to compute this data. It is the task of the scout units to find and evaluate new configurations. In order to reconfigure another unit, the corresponding scout unit requires knowledge of the reconfiguration data for the new configuration. When this knowledge is additionally transmitted during a scout unit's reconfiguration, the newly reconfigured scout unit is able to reconfigure other units to the corresponding configuration as well. In this case, reconfiguration takes r_t time steps and is called a full reconfiguration. When a reconfigured scout did not receive the knowledge, it is not able to reconfigure other units to its new configuration. The cost r_c for such a reconfiguration (i.e., without transfer of the special knowledge) is lower than for a full reconfiguration ($r_c < r_t$). Worker units can only be reconfigured using the second reconfiguration type.

Each job has a certain run time that depends on the number of slices a unit has configured for it. Typically, one would assume that more slices result in shorter execution time. In reality, processing speed does not increase linearly

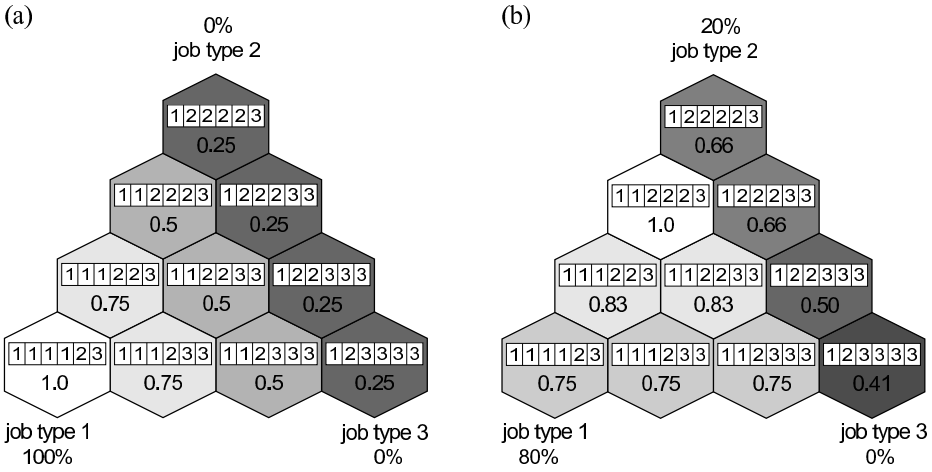


Fig. 2. Example configuration spaces with $s = 6$ and $j = 3$. Each cell denotes a configuration with configured slices per job type (in boxes) and its relative quality (below boxes). Colors illustrate the relative quality value. (a) The underlying run time function is linear, thus the configurations are ordered by their quality. In the displayed case the distribution of job type 1 is 100%, which makes the configuration in the lower left corner the optimum. (b) Non-linear (exponential) run time function. Job type 1 cannot utilize more than two slices, whereas job type 2 can. Thus, the optimal configuration gives more slices to job type 2 than the job distribution would indicate.

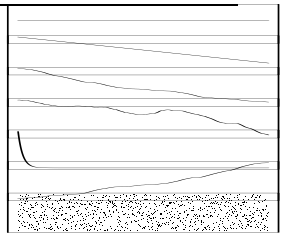
with processing resources (e.g. because of caches or verification mechanisms). Hence, we cannot necessarily assume that more slices result in shorter execution time. The function which defines the run time for each job type and each possible slice partition is referred to as the run time function. It is assumed that neither the global job distribution nor the underlying runtime function is known to the units. Hence, the units have to sample from the population of jobs.

The *quality* of a configuration o_c is defined as the average number of jobs which can be serviced per time step. Hence, $o_{\max} = 1$. The quality depends on the distribution of the different types of jobs that are in the system. Because the job types and their distribution can change in a dynamic environment, it is possible that the quality of a configuration changes over time. The ratio of the actual quality of a configuration to the quality of the optimal reconfiguration $o'_c = \frac{o_c}{o_{\max}}$ is referred to as *relative quality*.

The set of all possible configurations is called configuration space R_c. The size of the configuration space is $\binom{s-1}{j-1}$ and depends on the number of slices and number of job types. In the worst case, the size of the configuration space increases factorially with linear increasing parameters (see Figure 2 for two example configuration spaces). The configuration space is strongly characterized by the run times of the underlying job types. Job types which have a linear run time function result in spaces where configurations are ordered linearly by their quality. Hence, this type of configuration space is referred to as a linear configuration space. In order to study the impact of different underlying run time

functions on the system, configuration spaces were classified by the behaviour of their run time functions.[1] All classes of run time functions that have been used in this paper are given in Table 1.

Table 1. Classification of underlying run time functions

Name	Behaviour with increasing number of slices	Example
constant	constant	
linear	linearly decreasing	
monotone	monotonically decreasing	
polygonal	decreasing with local maxima	
exponential	exponentially decreasing	
increasing	increasing	
random	all functions not classified otherwise	

4 Search and Reconfiguration Strategies

The units of the computing system need to adapt constantly their configuration in order to deliver good performance in a dynamic environment. In order to accomplish this, the scout units need to search the configuration space and decide which configuration should actually be used. Several search and reconfiguration strategies have been developed which are described in the following.

4.1 Reference Model

As a point of reference, we developed a model that tries to solve the reconfiguration problem employing classical methods. It is therefore called the *reference model*. In this model, each scout unit has a fixed set of worker units assigned to it. A scout unit starts to search for a new configuration when it detects a change in the current job distribution; when no change is detected scout units act as workers and process jobs. New configurations are found by employing a simple heuristic. First, the scout unit generates a partition which reflects the actual job type distribution in the system. For example, if 80% of the jobs in the system are of type 1 and 20% of type 2, the heuristic will assign 80% of the computation resources to job type 1 and the rest to job type 2. After assessing the generated configuration by trial, the scout unit tries other configurations by making a random walk in the neighbourhood of this configuration (two configuration are neighboured when the assignment of the slices differs only for one slice), assessing

[1] Formal details on the generation of a run time model by using these classes are given in the supplementary online material.

a total of 4 other configurations. Thereafter, the scout unit starts to reconfigure the worker units that are assigned to it to the best configuration it has found. In order to make the model comparative it uses the same scout-worker ratio as the other models.

4.2 Ant-Inspired Models

Two models have been developed which utilize the emigration strategy of *T. albipennis* for solving the reconfiguration problem. In one of the models the system adapts to the environment by making a compromise between accuracy and speed of the decision-making process, whereas the other remains static. As both models share the basic principle, we first describe the common characteristics followed by the model-specific behaviour.[2]

Analogously to the case of the real ants, the decision-making process and the following reconfiguration of the units are referred to as *emigration*. Scout units start to search for a better configuration with the probability p_s per time step. In this study, new configurations are tried by generating a random partition c_n (if specific characteristics of the run time functions are known, specialized strategies might be of advantage). The scout unit then assesses the new configuration for 100 time steps by executing tasks. The perceived quality o_n is then compared with the quality of the currently preferred configuration, o_p. The preferred configuration c_p might be the old configuration if the scout unit just started its search, or another configuration which was assessed by the scout unit before. If the new configuration is considered to be better than the current preferred configuration, the new configuration becomes the preferred configuration. If not, the scout unit returns to its previous state. Upon finding a preferable configuration, the scout unit starts recruiting other scout units with a probability $p_r = \frac{o_p}{o_{\max}}$ per time step, which is proportional to the perceived quality of the configuration. If the quorum threshold has not been reached for this configuration (i.e., not enough scout units prefer this configuration), the scout unit tries to gather more opinions on its preferred configuration by reconfiguring other scout units to it. Analogously to the case of the real ants, this reconfiguration is referred to as *tandem-running*. As soon as the required quorum has been reached, the scout units start to reconfigure the remaining units to the new configuration. This is called *social-carrying*, because, as with the real ants, the scout units do not transmit the knowledge required for reconfiguring other units to the new configuration. In general, units are contacted randomly, regardless of their state.

When scout units are not participating actively in the emigration, they act as worker units and service jobs in order to increase the cluster's performance. When a scout unit tries to recruit another scout unit via tandem-running, it contacts a random scout unit of its colony. If the contacted scout unit is in working state, the scout stops working with the probability p_a per time step and joins the tandem-run. If the contacted scout unit is already searching for another

[2] A finite state diagram of the scout units' behaviour is available in the supplementary online material.

configuration, it switches preference with the probability p_p per time step. Scout units switch back to working state as soon as an emigration has been completed. As scout units are only asked to leave working state when there is an active emigration and possible better configuration, the number of scout units taking an active part in the emigration varies with the cluster's needs. In order to detect changes in the current configuration's quality, scout units reevaluate it periodically.

The following subsections describe variants in the handling of the quorum threshold.

Fixed Quorum Threshold Model The first model is the so-called *fixed quorum threshold model* (fixed model). In this model, the computing system is not able to adapt its quorum threshold to the environment. This means that there is always the same number of scout units required to make a decision regardless of the situation. Therefore, the decision speed and accuracy is fixed and does not change. This model is a simplification of the behaviour of the real ants.

Adaptive Quorum Threshold Model In the second model the computing system is able to adapt its quorum threshold to the environment. As for the real ants, the units are able to sense the current environment and adjust the compromise between speed and accuracy of the decision-making process accordingly. Thus, the model is referred to as the *adaptive quorum threshold model* (adaptive model). The adaptation is accomplished by scaling the quorum threshold with the relative throughput of the system. It is assumed that a system yielding a low performance should reconfigure as fast as possible. Hence, a low system throughput results in a low quorum threshold. On the other hand, when the system is delivering a high throughput, it uses a high quorum threshold. Two limits define the throughput values between which the quorum threshold is scaled linearly. Below the lower limit l_l the minimal threshold $T_{\min} = 1$ is used, whereas above the upper limit l_u the maximal quorum threshold $T_{\max} = n_s$ applies. The knowledge of the overall and optimal throughput of the cluster is required for the scaling mechanism. In this study it was assumed that the units possess this knowledge. Experiments have also been made concerning the distribution of this knowledge via gossiping protocols, but this cannot be described within the limited space of this paper.

5 Experiments

Experiments have been conducted in a standard environment with the following characteristics. In order to simulate a dynamic environment, one of the job types was replaced every 50000 time steps. Thus, always j type of jobs where present in the system. Jobs were generated randomly with a uniform distribution. The distribution of job types in the system was changed every 5000 time steps. The configuration space was generated using a polygonal run time function (see listing

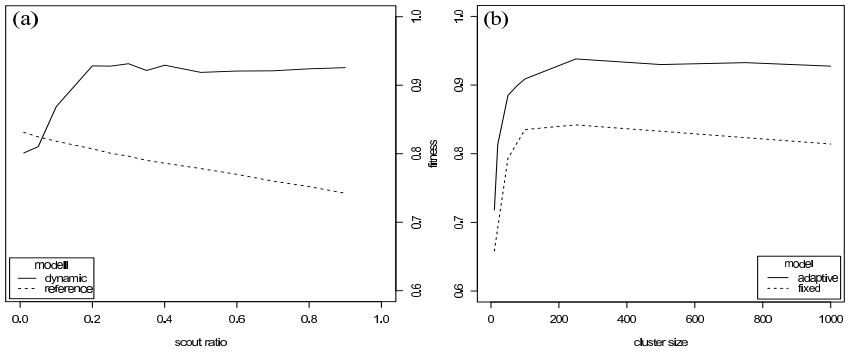


Fig. 3. (a) Fitness of the adaptive model and the reference model for scout ratios from 0.1 to 0.9, increased in steps of 0.05. (b) Fitness of the adaptive and the fixed model for increasing cluster sizes ($n \in \{10, 50, 100, 250, 1000\}$).

of run time functions in Table 1 for reference). The standard metric used in the experiments is a *fitness* metric which is defined as the ratio of average system throughput to optimal system throughput $f = \frac{d_a}{d_o}$. An optimal system exhibits a fitness of $f = 1$. The model was simulated with the Repast Agent Simulation Toolkit[3].

Each parameter set was evaluated by 20 simulation runs. If not stated otherwise, the following parameter values were used. The total number of units in the system was $n = 250$, with a scout ratio of $\frac{n_s}{n} = 0.3$, $s = 10$ slices and $j = 3$ job types. Default probabilities were $p_s = 0.005$, $p_a = 0.001$ and $p_p = 0.7$. The quorum threshold for the fixed threshold model $T_{\text{fix}} = 0.3 \cdot n_s$, which proved to be optimal when used in conjunction with the default parameter settings.[4] The quorum threshold scaling limits of the adaptive models l_l and l_u were set to 0.5 and 1.0 respectively. For the reconfiguration times, it was assumed that a scout reconfiguration takes three times as long as the reconfiguration of a worker unit ($r_t = 150$, $r_c = 50$). This is similar to the time difference found between tandem-running and social-carrying in colonies of *T. albipennis*.

5.1 Results and Discussion

One of the most characteristic features of the model is the distinction between scout and worker units. Therefore, we first studied the impact of the scout-worker ratio on the system fitness. Figure 3(a) shows the fitness of the system for different ratios. As it can be seen clearly, the adaptive model performs badly when used with a low number of scouts. If less than 20% of the units act as scouts, the fitness drops rapidly. On the other hand, with more than 20% of scouts the fitness remains nearly constant. This can be explained with the working-state mechanism: only those scouts that are actually required take part in an emigration, the other units continue to act as workers. Thus, increasing the

[3] http://repast.sourceforge.net/

[4] See the supplementary online material for an experimental validation of this choice.

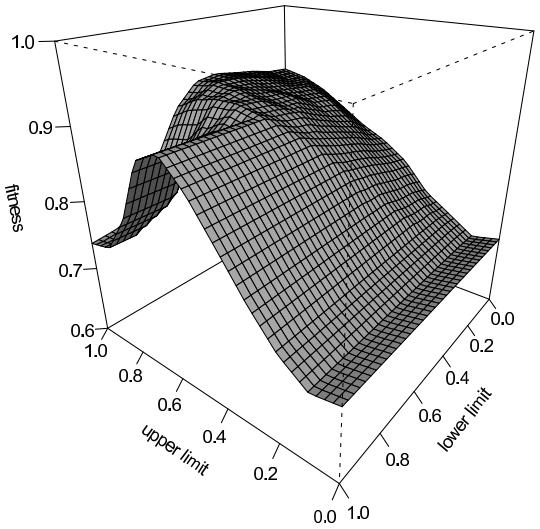


Fig. 4. Fitness of the adaptive model for different quorum threshold scaling limits l_l and l_u. Limits from 0.0 to 1.0 increased by 0.1.

percentage of scout units beyond the cost-optimal value of 20% does not increase the fitness of the system. This behaviour is analogous to the real ants, where the percentage of scouts has been found to be 20-30% [14]. The lower fitness in the reference model can be explained by the increased reconfiguration overhead when using more scouts.

The influence of the cluster size has been studied as well. The results are shown in Figure 3(b). In this experiment, the fixed model shows the same general behaviour as the adaptive model but exhibits a much lower fitness. This loss in fitness is attributed to the fixed quorum threshold, which enforces the same percentage of units on each decision, regardless of the situation. The decrease of fitness when using less than 100 units can be explained with the aforementioned minimal number of scout units required for the emigration to work. Apart from this minor restriction, the models scale well with an increasing number of units. As the fixed model is less effective than the adaptive model, we omitted it in the description of the following experiments.

In order to study the impact of the quorum threshold scaling on the adaptive model, we varied the scaling limits l_l and l_u as shown in Figure 4. As the results show, threshold scaling strongly affects the fitness of the system. An upper limit of less than 60% of the total number of units yields too many bad decisions even in situations where this is not acceptable, thus reducing the fitness substantially. The system does not react as sensitively to changes of the lower limit as it does to changes of to the upper limit, although the results show that a lower limit over 0.5 results in a quorum threshold that requires too many units to decide unanimously. Having a lower limit higher than the upper limit results in a dysfunctional quorum scaling. Hence, a strong decrease of fitness can be observed in left corner of Figure 4.

The final experiment studies the impact of the different configuration spaces on the fitness of the various models. Figure 5 shows a comparison of the fit-

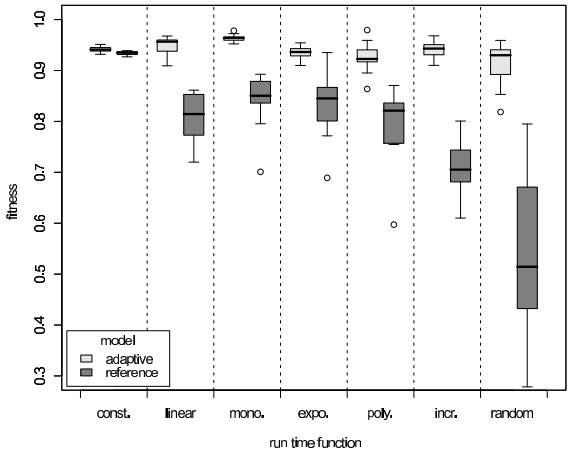


Fig. 5. Boxplot of the fitness on different configuration spaces. Whiskers indicate standard deviation and circles outliers (observations above/below $1.5 \cdot \mathrm{IQR}$).

ness on configuration spaces that are generated by the aforementioned different classes of run time functions (see Table 1). The ant-inspired models perform nearly constantly on all run time function classes, with the previously observed advantage of the adaptive model. The reference model's fitness drops with an increasing complexity of the configuration space, as the scout units are not able to find good configurations. This shows an important advantage of the ant-inspired model: The robustness of the self-organized reconfiguration enables it to deliver nearly constant performance even in dynamic and complex configuration spaces.

6 Conclusion

In this paper, we proposed ant-inspired strategies for self-organized reconfiguration in a computing system with reconfigurable units. In particular, we used principles that are also used by the ant *Temnothorax albipennis* for house-hunting to design the proposed strategies. In our model, the system consists of scout units and worker units which execute different jobs coming into the system. The units are able to specialize on different types of jobs by adjusting their reconfigurable hardware, thereby optimizing the system's throughput. The ant-inspired strategies for making a decision on which configuration should be employed have been analyzed experimentally and compared to a non-adaptive reference strategy. The ant-inspired adaptive strategy proved to be versatile and very robust on all tested environments. In contrast to the reference strategy, the ant-inspired strategies allowed the scout units to find good configurations even in complex configuration spaces. They were able to reach a collective, decentralized decision on which configuration was acceptable in the given situation, and to reconfigure all of the cluster's units to it. A interesting similarity to the natural system has been identified, as the optimal percentage of scouts in the system is about the same as observed for the real ants.

Acknowledgements This work was supported by the German Research Foundation (DFG) through the project "Organisation and Control of Self-Organising Systems in Technical Compounds" within SPP 1183.

References

1. IBM Research: Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM Research (2001)
2. Branke, J., Mnif, M., Müller-Schloer, C., Prothmann, H., Richter, U., Rochner, F., Schmeck, H.: Organic computing – addressing complexity by controlled self-organization. In: ISoLA 2006. (2006)
3. Würtz, R.P., ed.: Organic Computing. Springer (2008)
4. Compton, K., Hauck, S.: Configurable computing: A survey of systems and software. Technical report, Northwestern University, Department of ECE (1999)
5. Couzin, I.D., Krause, J., Franks, N.R., Levin, S.A.: Effective leadership and decisionmaking in animal groups on the move. Nature **433** (2005) 513–516
6. Conradt, L., Roper, T.J.: Group decision-making in animals. Nature **421** (2005) 155–158
7. Cicirello, V.A., Smith, S.F.: Wasp-like agents for distributed factory coordination. Autonom. Agents and Multi-Agent Sys. **8**(3) (2004) 237–266
8. Merkle, D., Middendorf, M., Scheidler, A.: Using decentralized clustering for task allocation in networks with reconfigurable helper units. In: IWSOS 2006, LNCS 4124. (2006) 137–147
9. Krieger, M., Billeter, J.B.: The call of duty: Selforganised task allocation in a population of up to twelve mobile robots. Robot. Autonom. Sys. **30** (2000) 65–84
10. Parker, C.A.C., Zhang, H.: Collective decision making: A biologically inspired approach to making up all of your minds. In: IEEE Int. Conf. on Robotics and Biomimetics. (2004) 250–255
11. Marshall, J.A.R., Dornhaus, A., Franks, N.R., Kovacs, T.: Noise, cost and speed-accuracy trade-offs: Decision-making in a decentralized system. J. Roy. Soc. Interface **3**(7) (2006) 243–254
12. Pratt, S.C., Sumpter, D.J.T., Mallon, E.B., Franks, N.R.: An agent-based model of collective nest choice by the ant temnothorax albipennis. Anim. Behav. **70**(5) (2005) 1023–1036
13. Pratt, S.: Quorum sensing by encounter rates in the ant temnothorax albipennis. Behav. Ecol. **16** (2005) 488–496
14. Pratt, S.C., Mallon, E.B., Sumpter, J., Franks, N.R.: Quorum sensing, recruitment, and collective decision-making during colony emigration by the ant leptothorax albipennis. Behav. Ecol. Sociobiol. **52**(2) (2002) 117–127
15. Franks, N.R., Richardson, T.: Teaching in tandem-running ants. Nature **439**(7073) (2006) 153–153
16. Merkle, D., Middendorf, M., Scheidler, A.: Self-organized task allocation for computing systems with reconfigurable components. In: IPDPS 2006. (2006) 25–29