Enhancing the Performances of D-MASON
A Motivating Example

Michele Carillo[1], Gennaro Cordasco[2], Rosario De Chiara[1],
Francesco Raia[1], Vittorio Scarano[1] and Flavio Serrapica[1]
[1]*ISISLab, Dipartimento di Informatica, Università degli Studi di Salerno, Salerno, Italy*
[2]*Dipartimento di Psicologia, Seconda Università degli Studi di Napoli, Caserta, Italy*

Keywords: Agent-based Simulation, Load-balancing, Visualization of Distributed Models, Performance Evaluation.

Abstract: Agent-based simulation models are an increasingly popular tool for research and management in many, different and diverse fields. In executing such simulations the "speed" is one of the most general and important issues and the traditional answer to this issue is to invest resources in deploying a dedicated installation of dedicated computers, with highly specialized parallel applications, devoted to the purpose of achieving extreme computational performances.
In this paper we present our experience with a distributed framework, D-MASON, that is a distributed version of MASON, a well-known and popular library for writing and running Agent-based simulations. D-MASON introduces the parallelization at framework level so that scientists that use the framework (domain expert but with limited knowledge of distributed programming) can be only minimally aware of such distribution.
The framework allowed only a static decomposition of the work among workers, and was not able to cope with load unbalance among them, therefore incurring in serious performance degradation where, for example, many of the agents were concentrate on one specific part of the space. We elaborated two strategies for ameliorate the balancing and enhance the synchronization among workers. We present their design principles and the experimental tests that validate our approach.

1 INTRODUCTION

Agent-Based simulation Models (ABMs) are an increasingly popular tool for research and management in many, different and diverse fields such as biology, ecology, economics, political science, etc.. In some fields, such as social sciences, ABMs are seen as a key instrument (López-Paredes et al., 2012) to the generative approach (Epstein, 2007), essential for understanding complex social phenomena. But also in policy making and economics (eco, 2010; for Economic Co-operation and Forum, 2009), the relevance and effectiveness of ABMs is recently recognized.

Computer science community has responded to the need for tools and platforms, that can help the development and testing of new models in each specific field, by providing tools, libraries and frameworks that speed up and make easier the task of (massive) simulations. Several important issues in evaluating different platforms for ABM, well identified in the reviews (Berryman, 2008; Najlis et al., 2001; Railsback et al., 2006), are speed of execution, flexibility, reproducibility, documentation, open-source and facilities for recording and analyze data.

Our work is based on D-MASON, a parallel version of the MASON library for writing and running simulations of ABMs. D-MASON addresses, in particular, the speed of execution with no harm on other features that characterize MASON. The intent of D-MASON is to provide an effective and efficient way of parallelizing MASON ABMs: effective because with D-MASON *you can do more* (e.g. faster and/or larger simulations) than what you can do with MASON; efficient because, in order to obtain this additional computing power, the developer has to do some incremental modifications to the MASON ABMs he has already written without re-designing them.

While D-MASON is efficient and its scalability has been proved to be high (Cordasco et al., 2011; Cordasco et al., 2012), no load balancing mechanism is available, thereby impeding any kind of dynamic adaptation to the possible unbalance in the spatial decomposition of the world where the agents are located. This feature is extremely important in a class

Carillo M., Cordasco G., De Chiara R., Raia F., Scarano V. and Serrapica F..
Enhancing the Performances of D-MASON - A Motivating Example.
DOI: 10.5220/0004060501370143
In *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2012), pages 137-143
ISBN: 978-989-8565-20-4

of simulations where "spatially defined" goals have to be pursued (i.e. a specific position must be searched for and located by the agents) where all the agents are probable to gather, thereby loading one node of the distributed system more than the others.

In this paper we describe how, starting from a parallelization of a specific simulation, sprung two modifications of D-MASON that had measurable positive effects on both performances and scalability.

Distributed Simulations. The research in many fields that uses the simulation toolkits for ABMs is often conducted interactively, since the "generative" paradigm described in (Epstein, 2007) describes an iterative methodology where models are designed tested and refined to reach the generation of an outcome with a simple generative approach. In this context, given that scientists of the specific domain often are not computer scientists, usually they do not have access to systems for high performances computations for a long time, and usually they have to perform preliminary studies within their limited resources and, only later (if needed), allow extensive testing on large supercomputing centers. In social sciences, for example, the need for "*the capacity to model and make up in parallel, reactive and cognitive systems, and the means to observe their interactions and emerging effects*" (Conte and Castelfranchi, 1995) clearly outlined, since 1995, the needs of flexible, though powerful, tools.

In this scenario, D-MASON's goal is to offer to such scientists a setting where a traditional MASON program can be run on one desktop, first, but can immediately harness the power of other desktops in the same laboratory (available, maybe, during off-peak hours) by using D-MASON, thereby providing scaling up the size they can treat or reducing significantly the time needed for each iteration.

Of course, since the resulting distributed system, collecting hardware from research labs, administration offices, etc. is highly heterogeneous in nature, the challenge that is tackled by D-MASON is also how to use efficiently all the hardware without an impact on the "legitimate" user (i.e., the owner of the desktop) both on performances and on installation/customization of the machine. On the other hand, one of the objectives pursued by D-MASON is that the program in MASON should not be very different than the corresponding program in D-MASON so that the scientist can easily modify it to run over an increasing number of hosts.

The need for efficiency among the Agent-Based modeling tools is well recognized in literature: many reviews of state-of-the-art frameworks (Berryman, 2008; Najlis et al., 2001; Railsback et al., 2006) place "speed" upfront as one of the most general and important issues. While a consistent work has been done to allow the distribution of agents on several computing nodes (see (Collier and North, 2011; Mengistu et al., 2008; Pawlaszczyk and Strassburger, 2009)), D-MASON's claims to have a different approach in principle: distribution is introduced at the framework level, so that scientists who use the framework (domain experts but with limited knowledge of computer programming and systems) can be unaware of such distribution. Several works in this field (Collier and North, 2011; Mengistu et al., 2008; Pawlaszczyk and Strassburger, 2009) directly affects the implementation and the architecture of a distributed agent model (dealing with lazy synchronization etc.), D-MASON's approach is concentrated on the upper layer of the simulation framework, thereby hiding, as much as possible, the details of the architecture. In this way, D-MASON provides a certain degree of backward-compatibility with pre-existing MASON applications, ensuring cost-effectiveness of porting a sequential implementation into a distributed setting.

Outline of the Paper. The rest of the paper is organized as follows: Section 2 introduces D-MASON. In Section 3, we briefly discuss our motivating example Ants Foraging and introduce the two improvements on D-MASON. In Section 4 we report on and discuss the tests we performed on the enhanced version of D-MASON. In Section 5, we conclude and discuss some possible extensions of this work.

2 MASON AND D-MASON

Before presenting the features of D-MASON, we will, briefly, introduce MASON.

MASON. MASON toolkit is a discrete-event simulation core and visualization library written in Java, designed to be used for a wide range of ABMs. The toolkit is composed of two independent layers: the *simulation* layer and the *visualization* layer. The simulation layer is the core of MASON and is mainly represented by an event scheduler and a variety of fields which hold agents into a given simulation space. MASON is mainly based on step-able agent: a computational entity which may be scheduled to perform some action (step), and which can interact (communicate) with other agents. The visualization layer permits both visualization and manipulation of the model.

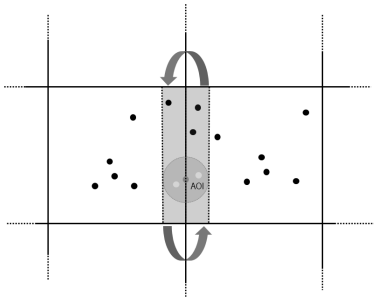

Figure 1: Field partitioning.

D-MASON. D-MASON adds a new layer named *D-simulation* which extends the MASON simulation layer. The new layer adds some features that allows the distribution of the simulation work on multiple, even heterogeneous, machines (workers). Notice that the new layer does not alter in any way the existing layers. Moreover, it has been designed so as to enable the porting of existing applications on distributed platforms in a transparent and easy way.

From a functional point of view D-MASON architecture is divided into three functional blocks: *Management*, *Workers* and *Communication*. The Management layer provides a *master application* which will be used for coordinating the workers, handle the bootstrap and running the simulation. D-MASON is based on a master/workers paradigm which exploits a space partitioning approach: the master partitions the space to be simulated (the field) into regions (see Figure 1). Each region, together with the agents contained in it, is assigned to a worker; then each worker is in charge of: simulating the agents that belong to the assigned region; handling the migration of agents; managing the synchronization between neighboring regions (this information exchange is required in order to let the simulation run consistently). Workers communicate by using a publish–subscribe mechanism.

D-MASON uses a standard approach to achieve a consistent local synchronization of the distributed simulations: each step is associated with a fixed state of the simulation. Regions are simulated step by step. Since the step i of region r is computed by using the states $i-1$ of r's neighborhood, the step i of a region cannot be executed until the states $i-1$ of its neighborhood have been computed and delivered. In other words, each region is synchronized with its neighborhood before each simulation step.

3 D-MASON IMPROVEMENTS

A Motivating Example: Ants Forage. In (Panait and Luke, 2004b; Panait and Luke, 2004a; Panait and Luke, 2004c) is described the *Ants Foraging Model*. *Ants Forage* is an agent based simulation of the Ants Foraging Model and can be found in the standard distribution of MASON (from http://cs.gmu.edu/ eclab/projects/mason/). In Ants Forage the space is represented by a grid of square cells where there are some food sources, a nest and a number of optional obstacles. When the simulation starts, the ants leave the nest in search for the food. Each ant occupying a cell may move to any of the eight cells in the neighbor which is not occupied by either other ants or an obstacle. When an ant reaches a food source it becomes laden with food and begins its search for the nest. When it has reached the nest again, the ant leaves the food and begins searching for food again. The goal is to maximize the rate of food brought to the nest from the food sources. A key factor of ants foraging simulation is that there is no direct communication among agents: once an ant finds the food source it does not communicate where it is located to other component of the colony. On the other hand, an indirect communication is provided by pheromone trails: ants release two of different kinds of pheromones on cells that have crossed depending on where they are heading to, the nest or the food. Pheromones evaporate over time therefore each cell is associated with a level of pheromone that depends on the time. Ants take local decisions that depend on their state (going to the nest or going to the food), the kind and level of pheromone that is located in adjacent cells.

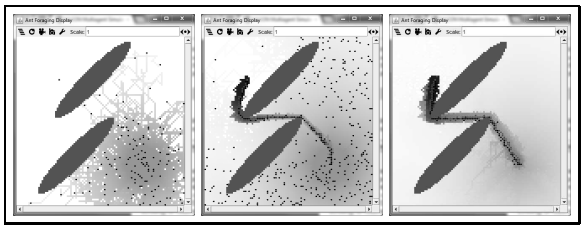

Figure 2: Ants foraging (left to right): ants leave the nest, part of the ants have found the food and head toward the nest and all the ants walk along shortest path between the food and the nest. The grey gradient represents the different levels of pheromone.

In Figure 2 are depicted the three typical phases of an ants foraging simulation, from left to right: in the beginning they leave the nest wandering around so the area around it is overcrowded; after a phase of equilibrium during which the ants almost equally spread in the field looking for food, some of the ants find it and get back to the nest; the last phase is characterized, once again, by an uneven distribution of the ants on the field along the shortest path between the nest and the food source.

This kind of simulations, characterized by dy-

namic unbalancing distribution of the agents throughout the steps, motivates the need for a specific load balancing policy that measures the amount of computational work needed by each worker to carry out the simulation and, consequently, takes decisions about how to re-distribute the work among workers. Ants Forage is based on 5 different specialized fields: one for the ants, two for the pheromones level, one for the obstacles and one for food sources. The first three fields are dynamic and updated at every step. Ants Forage has pushed us to meditate about two needs:

3.1 Enhancing Communication

The performances of distributed systems, like D-MASON, are strongly bound by the performances of the slowest component in the system when the various components (i.e. workers) needs to be synchronized. In D-MASON the synchronization of the fields immediately follows the simulation phase and is carried out sequentially by running along the list of the dynamic fields (static fields are not synchronized). Together with each of field synchronization there is some overhead due to both the communication channel and the *barrier*, the mechanism which allows to the different workers to "wait" for the slowest one to complete its work and to begin the successive step on the same time. Hence, synchronization phases adds dependencies between the operations carried out by workers that harm the parallelization process (Amdahl, 1967).

A reasonable solution to this waste of computing power has been adopted by moving to a multi thread communication phase during which all the updates on the fields are transmitted in parallel reducing the communication overhead. A single synchronization phase is done at the end of all the communications.

3.2 Load Balancing

The Need for Load Balancing. As described before D-MASON uses a space partitioning approach where the fields are subdivided in regions assigned to workers; this approach allows to limit the communication among the workers. Indeed, since each agent interacts only within a small area around it, the communication is limited to local messages (messages between workers, managing neighboring spaces, etc.). The problem with this approach is that agents can migrate between regions and consequently the association between workers and agents changes during the simulation. Moreover, load balancing is not guaranteed and needs to be addressed by the application. To better exploit the computing power provided by the workers of the system, it is necessary to design the system so that the simulation always evolves in parallel, avoiding bottlenecks. Since the simulation is synchronized after each step, the system advances with the same speed provided by the slower peer in the system. For this reason it is necessary to design the system in order to balance the load between the workers.

Our Approach. The choice of the partitioning strategy is important for the efficiency of the whole system. Two key factors need to be considered: (i) Static vs Dynamic Partitioning; (ii) The granularity of the world decomposition. Dynamic partitioning can be useful, for instance, when the workload of the simulation changes along the time, as in the case of Ants foraging. In this cases, in order to balance the workload across the workers the system can adapt the partitioning step by step. Unfortunately, the management of dynamic regions requires a large amount of communication between workers that consumes bandwidth and introduces latency. Similarly the granularity of the world decomposition (that is, the region size and, consequently, the number of regions, which a given space is partitioned into) determines a trade off between load balancing and communication overhead. The finer is the granularity adopted, the higher is the balancing that, ideally, can be reached by the system. However, due to regions' interdependency and system synchronizations, fine granularity usually determines a huge amount of communication which may harm the overall performances.

Based on the above considerations, we decided to opt for a system that allows a dynamic partitioning with two levels of granularity. At each step every worker compares the amount of agents it has to simulate with the ideal number of agents per region, that is the total number of ants divided by the number of regions the field is split into. When the ratio between this two values is above a given threshold the worker decides to move on a finer granularity by splitting its region.

The balancing phase is depicted in Figure 3: on the left there is the field partitioned in 9 (3×3) regions, this is the coarse grained subdivision of the work, while in the middle image is depicted the fine grained subdivision of the work. The last image shows what happens when a worker decides to split its region, in this particular case the worker that is in charge of the central region decompose the region in 9 sub-regions then assigns 8 of this sub-regions to its 8 neighboring workers. Please note that each sub-region is assigned in way that allows to minimize the communication between the neighbors.

Symmetrically when a worker notices that the

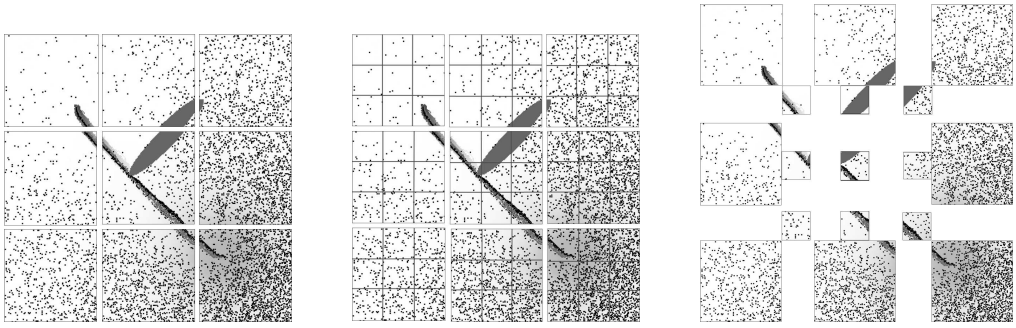

Figure 3: (Left to right): coarse grained partitioning, finer grained partitioning and balancing phase.

workload among the split subregions is below a given threshold the worker may decide to merge back the subregions returning to the initial (coarser) subdivision (Figure 3 (left)).

4 TESTING ENHANCED D-MASON

We performed a number of tests of the enhanced version of D-MASON in order to assess the effectiveness of the two improvements described above.

Setting of the Experiments. Simulations were conducted on a scenario consisting of two different configuration of hosts/workers: for the enhanced communication schema we conducted a series of tests on a single host (CPU *i*7, 8*GB* RAM) while for the load balancing experiments we also performed the tests on a network of 6 machines: a master machine, one communication server and 4 hosts each running an evenly distributed amount of workers. In the load balancing tests, each region is simulated by using a dedicated Java Virtual Machine (JVM). The communication is managed by a dedicated host running Apache ActiveMQ Server. Master, workers and the communication server are connected using a standard 100Mbit LAN network.

The *DAnts* Testbed. We have performed our tests on *DAnts*, the distributed version of *Ants Forage*, by considering more than 32 different test settings. Each setting is characterized by the choice of the following parameter: number of ants (the size of the field is determined by the number of agents in order to maintain a fixed density), number of regions, the kind of synchronization/load balancing policy.

4.1 Discussion of Results

In the following we will briefly discuss the results. We have decided to test the two improvements by using an incremental approach, in the first set of tests we tested the communication enhancement while in the second batch we added the load balancing.

Enhanced Communication. The rationale behind this test is to check the new communication mechanism against the previous one. Figure 4 depicts two square partitioning, 4×4 and 6×6. In each plot the X-axis represents the increasing number of ants while on the Y-axis are reported the performances of the system in terms of simulation steps per second.

In both test settings the improved communication strategy works more efficiently than the older one. The reason why, as along as the number of agents increases the delta between the curves decreases, is that the impact of simulation time augments proportionally to the number of agents while our improvement affects only the communication phase.

Load balancing. The batch of tests we performed simulates $100,000$ ants running in two settings: 1 host and 6 host. Each test lasted $40,000$ simulation steps. In both settings we used 3.0 for the split threshold (i.e., a worker decides to split its region when the number of agents in the region are 3.0 times the ideal number of agents per region) and 1.5 as the merge threshold.

Figure 5 shows the results. In each plot the X-axis represents the partitioning while on the Y-axis are reported the performances of the system in terms of simulation steps per second. The results are encouraging and show that the load balancing policy is effective in mitigating the unbalancing. A first consideration must be done on the fact that by using load balancing is possible “to do more” with the D-MASON even on a single machine. In the multiple machines

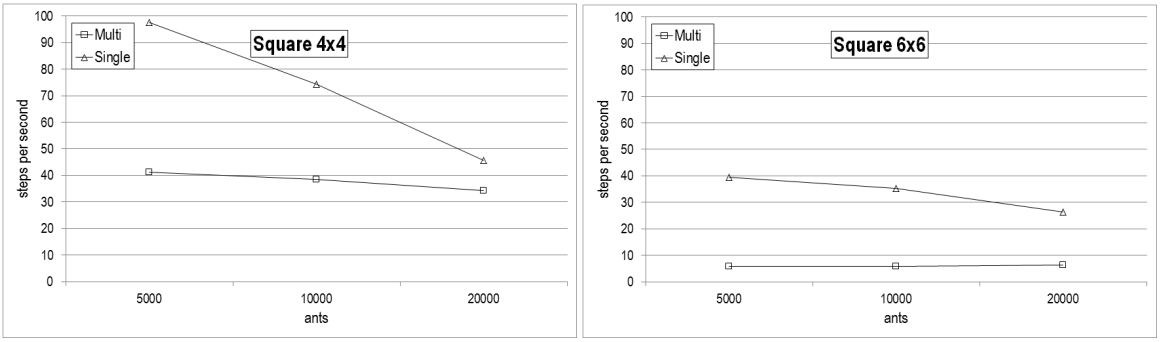

Figure 4: Multi message vs single message.

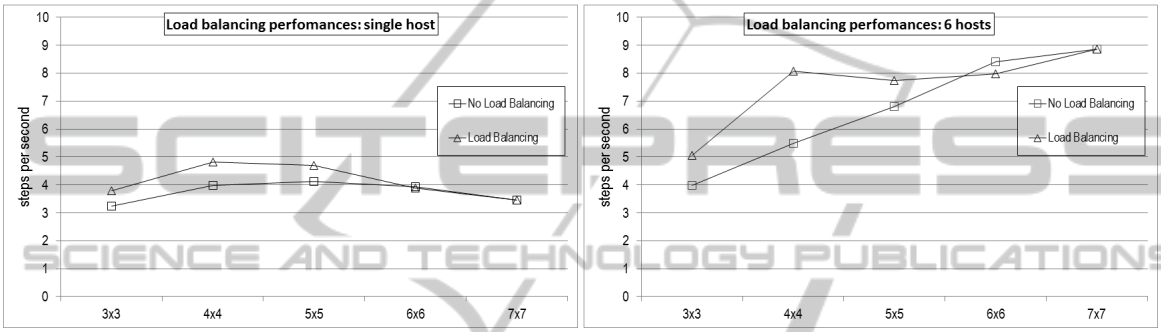

Figure 5: D-MASON load balancing effectiveness.

setting we can obtain even better improvements because in the previous setting the load balancing was performed on workers running on a single host, while in this setting each host is serving a smaller number of workers. Clearly as the number of regions increases the finer granularity of the subdivision naturally balance better the load but, on the other hand, the communication overhead increases, coherently, limiting the overall improvement.

5 CONCLUSIONS AND FUTURE WORK

This paper reports our experience with D-MASON a distributed version of MASON. D-MASON has been developed with the purpose of speeding up the performances of MASON by letting the computational work to be distributed among several machines. Hence by harvesting the unused CPU power usually largely available in installations like laboratories.

This work has been motivated by the development of the distributed version of Ants Forage. We observed that this kind of simulations have common characteristics that needed to be better addressed by D-MASON: (i) the simulation is based on more than a field dynamically updated during the simulation; (ii) agents are not balanced among the space often accumulating in some zones. We have showed two strategies that deal with the issues presented above and we have validated the effectiveness of the strategies by several experimental tests.

Some work still need to be done, for instance the load balancing policy has to be tuned in order to better exploit the work subdivision, the control of the thresholds is the key to leverage this mechanism. D-MASON is available at http://www.isislab.it/projects/dmason/. The project will be soon released under a Free and Open Software license.

REFERENCES

The Economist (2010). Agents of change.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of AFIPS '67*, pages 483–485.

Berryman, M. (2008). Review of Software Platforms for Agent Based Models. Tech. Rep. DSTO-GD-0532, Australian Government, Department of Defence.

Collier, N. and North, M. (2011). A platform for large-scale agent-based modeling. In *W. Dubitzky, K. Kurowski, and B. Schott, eds., Large-Scale Computing Techniques for Complex System Simulations, Wiley.*

Conte, R. and Castelfranchi, C. (1995). *Cognitive and Social Action*. UCL Press.

Cordasco, G., De Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V., and Spagnuolo, C. (2011). A Framework for distributing Agent-based simulations. In *Ninth Inter. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2011)*.

Cordasco, G., De Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V., and Spagnuolo, C. (2012). D-MASON: A Distributed Framework for Agent-based simulations. *Simulation SI: Advancing Simulation Theory and Practice with Distributed Computing*, (Submitted).

Epstein, J. M. (2007). *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press.

For Economic Co-operation, O. and Forum, D. O. G. S. (2009). Applications of complexity science for public policy: new tools for finding unanticipated consequences and unrealized opportunities.

López-Paredes, A., Edmonds, B., and Klugl, F. (2012). Editorial of the special issue: Agent based simulation of complex social systems. *Simulation*, 88(1):4–6.

Mengistu, D., Troger, P., Lundberg, L., and Davidsson, P. (2008). Scalability in Distributed Multi-Agent Based Simulations: The JADE Case. In *Proc. of FGCNS '08*, volume 5, pages 93–99.

Najlis, R., Janssen, M. A., and Parkerx, D. C. (2001). Software tools and communication issues. In Parker, D. C., Berger, T., and Manson, S. M., editors, *Proc. Agent-Based Models of Land-Use and Land-Cover Change Workshop*, pages 17–30.

Panait, L. and Luke, S. (2004a). Ant foraging revisited. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE-IX)*.

Panait, L. and Luke, S. (2004b). Learning ant foraging behaviors. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE-IX)*.

Panait, L. and Luke, S. (2004c). A pheromone-based utility model for collaborative foraging. In *Proceedings of AAMAS 2004*.

Pawlaszczyk, D. and Strassburger, S. (2009). Scalability in distributed simulations of agent-based models. In *Proc. of WSC 2009*, pages 1189–1200.

Railsback, S. F., Lytinen, S. L., and Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82:609–623.