Artif Life Robotics (2016) 21:317–323
DOI 10.1007/s10015-016-0303-8

SPECIAL FEATURE: ORIGINAL ARTICLE

Critical mass in the emergence of collective intelligence: a parallelized simulation of swarms in noisy environments

Aleksandr Drozd[1] · **Olaf Witkowski**[2] · **Satoshi Matsuoka**[1] · **Takashi Ikegami**[3]

Received: 1 May 2016 / Accepted: 26 June 2016 / Published online: 8 August 2016

Abstract We extend an abstract agent-based swarming model based on the evolution of neural network controllers, to explore further the emergence of swarming. Our model is grounded in the ecological situation, in which agents can access some information from the environment about the resource location, but through a noisy channel. Swarming critically improves the efficiency of group foraging, by allowing agents to reach resource areas much more easily by correcting individual mistakes in group dynamics. As high levels of noise may make the emergence of collective behavior depend on a critical mass of agents, it is crucial to reach sufficient computing power to allow for the evolution of the whole set of dynamics in simulation. Since simulating neural controllers and information exchanges between agents are computationally intensive, to scale up simulations to model critical masses of individuals, the implementation requires careful optimization. We apply techniques from astrophysics known as treecodes to compute the signal propagation, and efficiently parallelize for multi-core architectures. Our results open up future research on signal-based emergent collective behavior as a valid collective strategy for uninformed search over a domain space.

Keywords Artificial life · Artificial neural networks · Bio-inspired computation · Evolutionary robotics · Foraging · Swarming · Treecode

This work was presented in part at the 1st International Symposium on Swarm Behavior and Bio-Inspired Robotics, Kyoto, Japan, October 28–30, 2015.

✉ Aleksandr Drozd
alex@smg.is.titech.ac.jp

Olaf Witkowski
okw@elsi.jp

Satoshi Matsuoka
matsu@acm.org

Takashi Ikegami
ikeg@sacral.c.u-tokyo.ac.jp

1 Global Scientific Information and Computing Center, Tokyo Institute of Technology, Tokyo, Japan

2 Tokyo Institute of Technology, Earth-Life Science Institute, Tokyo, Japan

3 Department of Multi-Disciplinary Sciences, The University of Tokyo, Tokyo, Japan

1 Introduction

The ability of swarms of organisms to coordinate their motion in space has important implications for the evolution of social cognition, collective animal behavior, and artificial life [7]. Swarming is defined as organization of a large number of individuals into a coordinated formation. Using only the information from their environment, they can gather together and move in groups, using various types of dynamics [5, 9, 21]. One of the most interesting aspects of swarming is that individuals are able to overcome noise and local minima in a search space, based on an explicit or implicit exchange of information between agents.

As experiments on real animals, either in an experimental setup [1, 18] or in their ecological environment [17], are costly to reproduce and do not offer the possibility to easily explore different evolutionary paths, researchers have turned to computational modeling. It allows for easy modification of setup conditions and parameters, tremendous data generation, reproducibility of every experiment, and easier identification of the underlying dynamics of complex phenomena.

The first model to simulate agents swarming in 3D is the boids [20] (short for *birdoids*) model, with only three simple rules. This approach has been explored in many subsequent studies [8, 12, 14, 23, 27]. The next step in swarming simulations came with evolutionary robotics where, instead of a fixed set of rules, each agent is given an artificial neural network brain that controls its movements. The swarming behavior evolves by copying (with mutations) of chromosomes which encode the neural network parameters. By comparing the impact of different selective pressures, this methodology, first used in [10], eventually allowed for studying the evolutionary emergence of swarming.

In this paper, we describe a simplistic model of agents moving in a 3D space to find a vital resource. The population is simulated with an asynchronous evolutionary algorithm, meaning that new agents are created during the experiments, eventually replacing the previous population. Agents are indirectly selected in a Darwinian-like process based on their energy: they are removed from the simulation if their energy level is too low. We specifically explore addition of noise to the food detection sense of the agents, and hypothesize that it can be overcome by the emergence of a collective behavior involving sufficiently large groups of agents.

An atomic pile is said to "go critical" when a chain reaction of nuclear fission becomes self-sustaining. A minimal amount of fissionable material has to be compacted together to keep the dynamics from fading away. The notion of critical mass as a crucial factor in collective behavior has been studied in various areas [13, 16]. Similarly, it may be important for the swarms to reach a critical mass, enough to overcome very noisy environments. We will, therefore, explore the optimization of the computer simulation itself, as large-scale swarms may qualitatively differ in behavior from regular-sized ones.

2 Model

The model extends [24], which proposed an asynchronous simulation evolving a swarming behavior based on signaling between individuals. A population of agents is simulated in a three-dimensional space of $600.0 \times 600.0 \times 600.0$, gaining a vital amount of energy from a resource gathering task. Food spots are randomly placed in the environment and moved around every 1000 iterations. By getting close to one of those food spots, the agents gain more energy, allowing them to compensate for the energy losses due to their movement and their signaling. If an agent's energy level drops to zero, it is removed from the simulation. Only agents with energy higher than 4.0 are allowed to reproduce. In this regard, the energy also represents the agent's fitness, and both terms will here be used interchangeably.

The agent's position is determined by three floating point coordinates between 0.0 and 600.0. Each agent is positioned randomly at the start of the simulation, and then moves at a fixed speed of 1 unit per iteration. Every iteration, the agent's new velocity $\mathbf{c}_t$ is obtained by rotating its velocity vector at the previous time step $\mathbf{c}_{t-1}$ by two Euler angles: ψ for the agent's pitch (i.e., elevation) and θ for the agent's yaw (i.e., heading). The rotation is determined by the two motor output values of the neural controller o_1 and o_2, determining, respectively, the acceleration in y and z in the agent's inertial frame of reference, while the norm of the velocity is kept constant. The agent's position $\mathbf{x}_t$ is then updated according to its current velocity with $\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{c}_t$.

In the original model [24], the individuals were blind, in the sense that they do not see either the food patches or the other agents around them. In our model, we add a sense of vision to every agent, allowing them to detect nearby resources. However, we add a high level of noise (a randomly generated term at comparatively 10,000 % of the original value range) to make this information highly imperfect.

The agents' interaction is limited to the exchange of signals between each others. Every agent is capable of sending signals of variable intensity, encoded as floating point values ranging from 0.0 to 1.0. Each agent also has a directional communication sensor allowing it to detect signals produced by other agents in a 60-degree frontal cone. The distance to the source proportionally affects the intensity of a received signal, and signals from agents above a 100 distance are ignored.

The agent's neural controller is implemented by a modified Elman artificial neural network (Fig. 1) with 10 input neurons, 10 hidden neurons, and 3 output neurons. The outputs control the two motor angles and the communication signal emitted by the agent. The hidden layer is given a form of memory feedback from a 10-neuron context layer, containing the values of the hidden layer from the previous time step. The input neurons correspond to 6 directional signal sensors, 3 angle-to-goal sensors, and 1 fixed bias input. All nodes in the neural network take floating point input values between 0 and 1. All output values are also floating values between 0 and 1, and the motor outputs are then converted to angles between $-\pi$ to π. The activation state of each internal neuron is updated according to a sigmoid function. The weights of each connection in the neural network are directly stored in an array. That array, constituting the agent's genotype, is then evolved using a specific genetic algorithm described below.

The agents reproduce by replicating whenever they reach a minimal level of fitness, that is, whenever their level of energy reaches a certain threshold value (arbitrary 12 units), a child agent is added to a random position on the

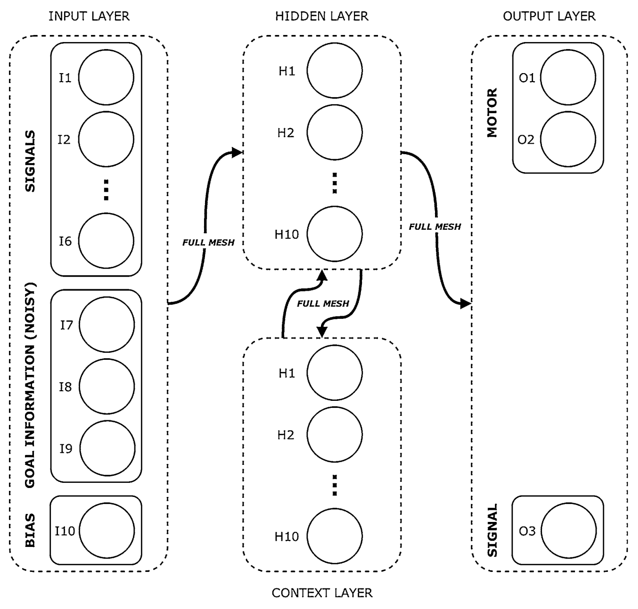


Fig. 1 Architecture of the agent's controller, a recursive neural network composed of 10 input neurons (I_1 to I_10), 10 hidden neurons (H_1 to H_{10}), 10 context neurons (C_1 to C_10), and 3 output neurons (O_1 to O_3). Every layer is fully connected to the next one

map, and the parent's energy is decreased by a certain cost (arbitrary 10 units).

Every new agent is born with an energy equal to 2.0. In the course of the simulation, each agent can gain or lose a variable amount of energy. At iteration t, the fitness function f_i for agent i is defined by $f_i(t) = \frac{r}{d_i(t)}$, where r is the reward value and d_i is the agent's distance to any food spot. The reward value is controlled by the simulation, such that the population remains between 100 and 1000 agents, and as close as possible to 500 agents. All the way through the simulation, the agents also spend a fixed amount of energy for movement (0.01 per iteration) and a variable amount of energy for signaling costs ($0.001 \times$ signal intensity per iteration).

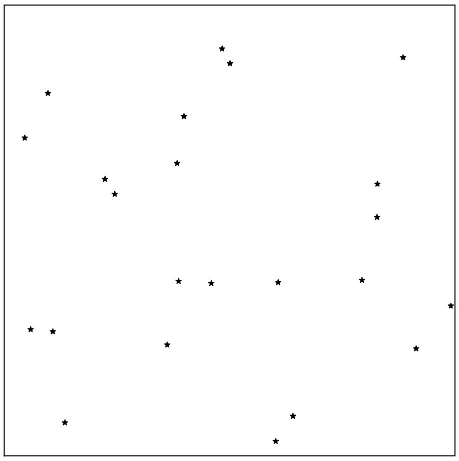
The weights of every connection in the neural network (apart from the links from hidden to context nodes, which have fixed weights) are encoded in genotypes and evolved through successive generations of agents. Each weight is represented by a unique floating point value in the genotype vector, such that the size of the vector corresponds to the total number of connections in a neural network. Agents inherit parental genotype with 10 % chance of mutation of each element. Mutation increases or decreases the weight by the random value within 10 % range of original.

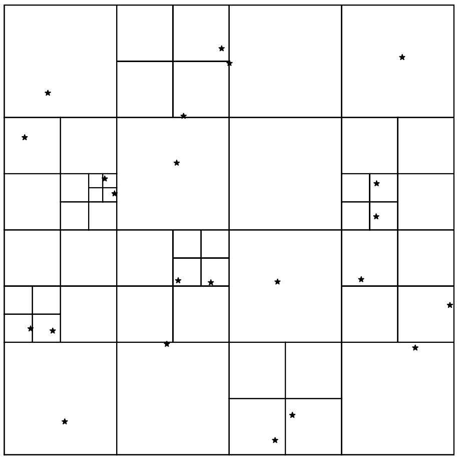
3 Implementation

The main performance bottleneck of the model is the computation of aggregated signal that a boid perceives from all other boids. Straightforward implementation has a daunting $O(n^2)$ complexity and becomes prohibitive already at a thousands-of-agent scale. This task, however, resembles the classical N-Body problem from computational physics—the problem of predicting the individual motions of a group of celestial objects (represented as particles) interacting with each other gravitationally. This problem has an efficient approximate implementation known as the Barnes–Hut algorithm [2]. The key insight is to approximate the gravitational pull from remote particle clusters with the force coming from one particle positioned at the center of mass of the cluster, and having the same aggregated mass. Please note that this aggregation refers to the computational approximation of signals coming from clusters of agents as single vectors, which should not be confused with the swarming behavior of the boids.

Our signal-propagation method is based largely on the Barnes–Hut algorithm, but accounts for the specifics of the signal perception model of the agents. Instead of being pulled by gravitational force, every particle (agent in our case) perceives an aggregated signal from the other agents independently by each sensor in a directed fashion. These signals thus cannot be simply aggregated by summing the corresponding vectors. For example, if an agent is receiving two signals of the same intensity coming, respectively, from the left and from the right, they should not cancel each other as in case of gravitation pull. Instead, in that case, they will be perceived independently, respectively, by the left and right sensors. To achieve this, independently for each of the 6 sensors, the recursive tree traversal procedure is informed by its direction. We then compute the projection of these signals coming from each cluster of signaling agents, onto the vector corresponding to that sensor, discarding negative values.

The algorithm works by constructing an octree corresponding to the hierarchical decomposition of the simulation space into cubic cells with the root of the tree representing the whole space. Tree construction is done by recursive splitting of the domain, so that at most one agent is on one cell. Figure 2 illustrates space decomposition and the corresponding quad-tree (Fig. 2c) for the two-dimensional case. Fig. 3 Illustrates a 3D case. Then, the octree is fully traversed once, such that each node stores the summed aggregated signal for each of the nodes in its subtree. To compute the signal, the algorithm starts from the root and checks if the current node is a leaf node or if the node is sufficiently remote from the target agent, i.e., the ratio s/d is smaller than the threshold parameter θ, where s is the width of the region represented by the internal node, and d is the distance between the agent and the node's center of signal intensity. If the node is not a leaf and is not sufficiently remote, the algorithm recursively aggregates signals from all of the child nodes.


(a) The original domain


(b) 2D space decomposition

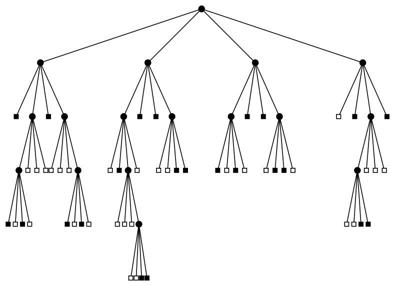

(c) Corresponding quad-tree

Fig. 2 Two-dimensional example of tree construction. *Black squares* mark the leaf nodes containing an agent, *white square* nodes are empty leaf nodes, and the nodes marked with *circles* are "inner nodes"

Further performance improvement can be achieved by utilizing modern parallel hardware architectures. The signal-propagation logic in our code is roughly similar to the logic of the original Barnes–Hut algorithms, and similar parallelization schemes can be applied to it. The Barnes–Hut algorithm has been extensively studied and optimized,

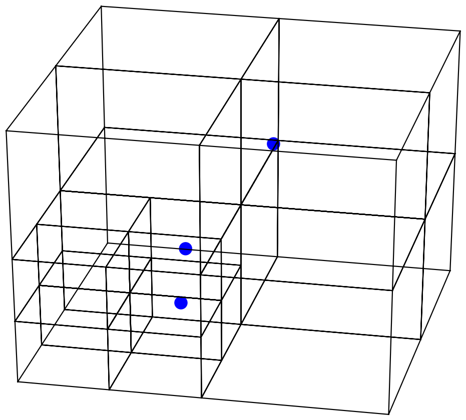


Fig. 3 Three-dimensional space partitioning for three agents

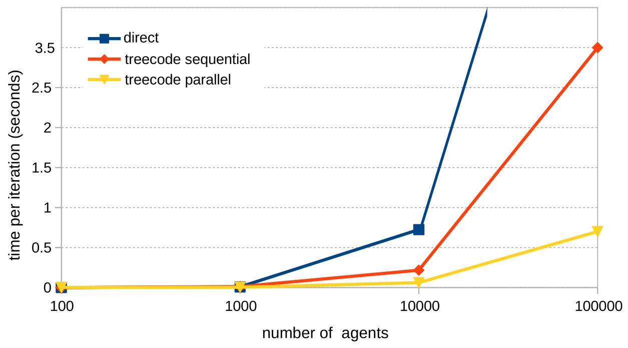


Fig. 4 Performance scaling

and it has been implemented on many platforms, such as clusters, GPU accelerators, and even FPGA devices [3, 6, 11]. We focus on the traditional multi-core CPUs, as this can be done with reasonable effort, but still lead to significant performance improvement.

For the tree construction, we use a top–bottom approach: each thread independently inserts particles into the tree, starting traversing from the root to the desired last-level cell, and then attempts to lock the appropriate child pointer, as only the leaf nodes can be changed during the insertion.

Computing the signals does not modify the tree and does not require any synchronization. The same is true for the feed-forward computation of the ANN. Performance evaluation was performed on a PC with Intel Core i7-3820 3.60 GHz CPU with 4 hyperthreaded cores (8 logical cores) and 16 Gb RAM under Ubuntu 15.04 (Linux kernel 3.19.0) operating system. Source codes written in C++ programming language with OpenMP programming interface [4] used for multi-threading support and compiled with gcc 4.9.2 compiler. Figure 4 shows performance scaling with the increase of the number of agents (spawning of new agents was disabled for this benchmark).

4 Related work

In the particle swarm optimization (PSO) problem, a large number of particles are moving though a domain, with the possibility of updating behavioral parameters every iteration. Highly efficient parallel methods for the PSO have been proposed and implemented [15]. However, as there is no interaction between particles, the parallelization strategy is fairly straightforward.

In the case of Reynolds' boids, the agents have to be aware of each other to follow separation, alignment, and cohesion rules. Naive implementation can also lead to quadratic performance complexity. However, since no long-range interaction is required, and boids tend to be fairly separated—the grid-based methods or methods based on Smoothed Particle Hydrodynamics technique (and corresponding parallelization approaches) work quite well for boids and similar models [19, 22].

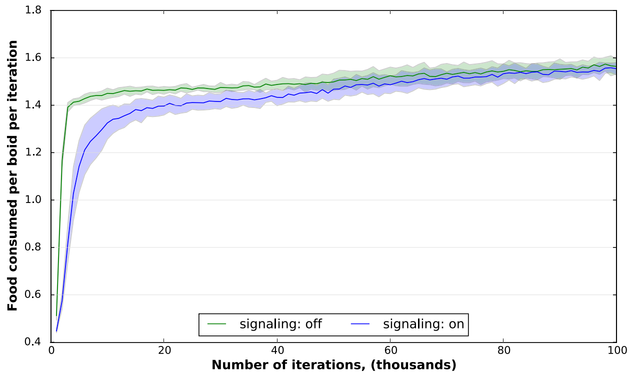
Our model is different in three key aspects: (1) every agent has to receive signals from all other agents; (2) every agent contains an artificial neural network that has to be evaluated at each iteration; and (3) the number of agents changes over the simulation. The first aspect makes the above-mentioned optimization and parallelization techniques inapplicable to our model.

Yokoi et al. [25, 26] used a vibrating potential field method to coordinate the motion of their morpho-functional machines (amoeba-like autonomous agents). The equation for the potential field at a given point in space contains the sum of individual fields of the agents. As the values of the potential have to be computed for every agent in a set, this model also results in $O(N^2)$ computational complexity and cannot scale to a large number of agents.

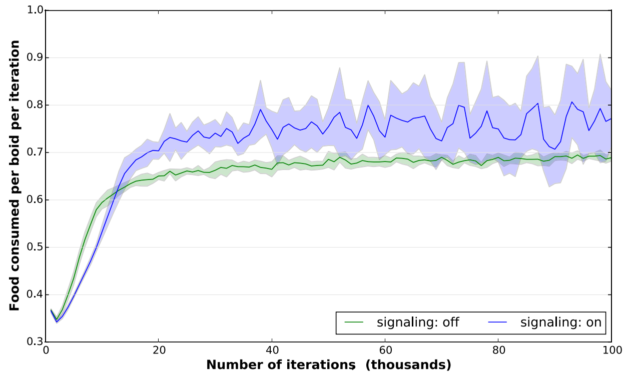
5 Results

In Fig. 5, we can observe that signaling improves the foraging of agents. We use the average amount of food resource obtained per agent per iteration as a measure of the population's fitness. Without noise, the agents using signaling are less efficient than their silent counterparts. We found that this is not due to the cost of signaling (we factored out this cost from the graph), but rather because of the excess of noise brought by the signal inputs. The difference remains very small between signaling and non-signaling agents.

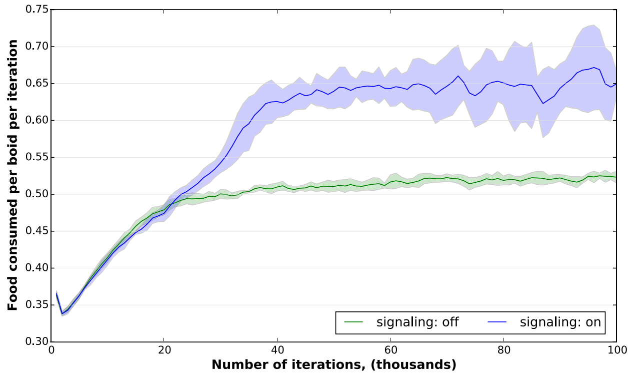
We find, however, that above a certain noise level, the cost of signaling is fully compensated by its benefits, as it helps foraging. The average fitness becomes even higher, as we increase the noise level, which suggest that the signaling behavior increases in efficiency for high levels of noise, allowing the agents to overcome imperfect information by forming swarms.



(a) Without noise



(b) With noise level 20



(c) With noise level 40

Fig. 5 Efficiency with and without signals with constant noise, mean (*central line*), and standard deviation range (area plot) over 10 runs

We also observe scale effects in the influence of the signal propagation on the average fitness of the population. Figures 6 and 7 show the effect of different population sizes and propagation parameters on foraging efficiency. For a smaller population, only middle values of signal propagation seem to bring about fitter behaviors, whereas this is not the case for larger populations. On the contrary, larger populations are the most efficient for lower levels of signal propagation. This may suggest a phase transition in

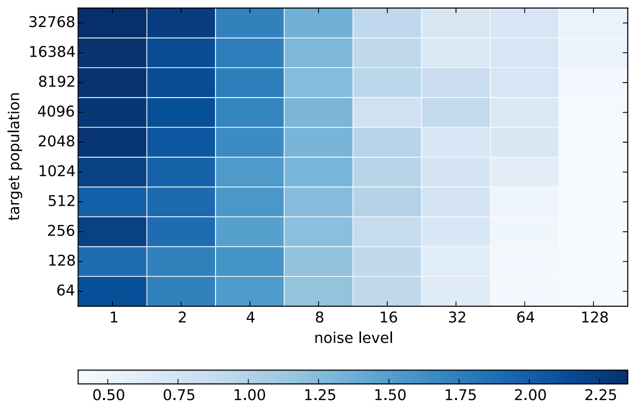


Fig. 6 Effects of population size and noise level

the agents' behavior for large populations, eventually in the way, the swarming itself helps foraging.

The analysis of the phylogeny (Fig. 8) though the whole simulation showed that at the initial step, one or a few of the "fittest" individuals are selected. The following generations branch more uniformly, slowly approaching the optimal genotypes within several sub-groups 5. This corresponds to the dynamics that we observe with the foraging efficiency over iterations: a noticeable jump in the beginning, followed by slow improvement towards saturation.

6 Conclusion

We used an hierarchical method based on the Barnes–Hut simulation in computational physics and its parallel implementation to speed up signal-propagation simulation between autonomous agents. This achieved performance improvement of a few orders of magnitude over the previous implementation [24], and allowed us to explore the

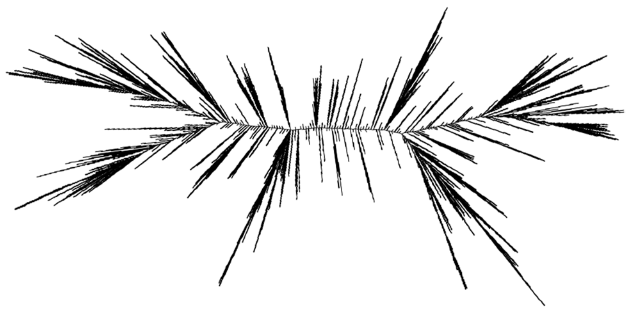


Fig. 8 Phylogenetic tree

behavior of large-scale swarms which have been suggested to generate qualitatively different dynamics.

We showed how signal-driven swarming, emerging in an evolutionary simulation, such as in [24], allows agents to overcome noisy information channels and improves their performance in a resource finding task. Our first contribution is the introduction of noise, demonstrating that the algorithm performs well against noise filling up channels of information almost up to their full capacity, in the inputs of agents. The swarming behavior helped by basic signaling enables the agents to globally filter out the noise present in the information from their sensory inputs, and to reach food sites.

The optimization of fitness is acquired by phenotypes (agents) using efficient patterns of behavior (motion and signaling), which themselves are encoded in the weights of agents' neural networks. The real optimization, therefore, occurs at a higher level of the Darwinian-like process in the genotypic search space. Efficient genotypes are selected by the asynchronous genetic algorithm throughout a simulation run.

Our results indicate non-linear dependencies of the signal propagation with respect to the population size,

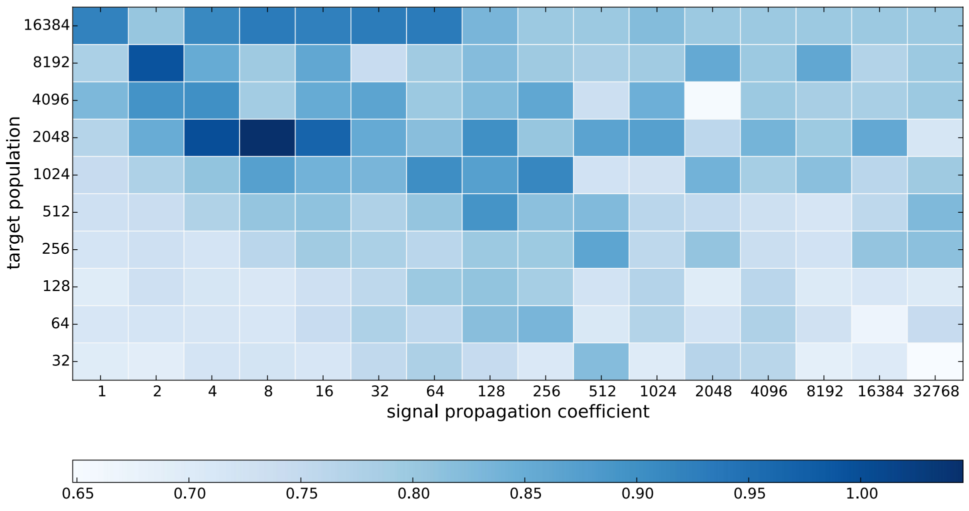


Fig. 7 Effects of population size and signal-propagation coefficient

suggesting the existence of a critical mass in swarms which enables them to overcome noisy environments. This effect could only be shown, thanks to the efficient simulation of a large-scale swarm, with a behavior qualitatively different from that of relatively small swarms.

Acknowledgments This paper was partially supported by a Grant-in-Aid for Scientific Research on Innovative Areas (Research Project Number: 15H01612). This paper was partially supported by JST, CREST (Research Area: Advanced Core Technologies for Big Data Integration).

References

1. Ballerini M, Cabibbo N, Candelier R, Cavagna A, Cisbani E, Giardina I, Lecomte V, Orlandi A, Parisi G, Procaccini A, Viale M, Zdravkovic V (2008) Interaction ruling animal collective behavior depends on topological rather than metric distance: evidence from a field study. Proc Natl Acad Sci 105(4):1232–1237
2. Barnes J, Hut P (1986) A hierarchical o(n log n) force-calculation algorithm. Nature 324:446–449
3. Blackston D, Suel T (1997) Highly portable and efficient implementations of parallel adaptive n-body methods. In: In SC'97: 1–20
4. Board OAR (2013) OpenMP Application Program Interface, Version 4.0. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
5. Budrene EO, Berg HC et al (1991) Complex patterns formed by motile cells of *Escherichia coli*. Nature 349(6310):630–633
6. Coole J, Wernsing J, Stitt G (2009) A traversal cache framework for fpga acceleration of pointer data structures: A case study on barnes-hut n-body simulation. In: Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on, pp 143–148
7. Couzin ID (2009) Collective cognition in animal groups. Trends Cogn Sci 13(1):36–43
8. Cucker F, Huepe C (2008) Flocking with informed agents. Math Action 1(1):1–25
9. Czirók A, Barabási AL, Vicsek T (1997) Collective motion of self-propelled particles: kinetic phase transition in one dimension. arXiv preprint. arXiv:9712154
10. Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, vol 1. New York, NY, pp 39–43
11. Hamada T, Yokota R, Nitadori K, Narumi T, Yasuoka K, Taiji M (2009) 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In: High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on, pp. 1–12. doi:10.1145/1654059.1654123
12. Hartman C, Benes B (2006) Autonomous boids. Comput Animat Virtual Worlds 17(3–4):199–206
13. Marwell G, Oliver P (1993) The critical mass in collective action. Cambridge University Press, New York
14. Mataric MJ (1992) Integration of representation into goal-driven behavior-based robots. IEEE Trans Rob Autom 8(3):304–312
15. Mussi, L., Daolio, F., Cagnoni, S.: Evaluation of parallel particle swarm optimization algorithms within the cuda architecture. Inf Sci 181(20):4642–4657 (2011) (Special Issue on Interpretable Fuzzy Systems)
16. Oliver PE, Marwell G (2001) Whatever happened to critical mass theory? A retrospective and assessment. Sociol Theory 19(3):292–311
17. Parrish JK, Edelstein-Keshet L (1999) Complexity, pattern, and evolutionary trade-offs in animal aggregation. Science 284(5411):99–101
18. Partridge BL (1982) The structure and function of fish schools. Sci Am 246(6):114–123
19. Pimenta L, Pereira G, Michael N, Mesquita R, Bosque M, Chaimowicz L, Kumar V (2013) Swarm coordination based on smoothed particle hydrodynamics technique. IEEE Trans Robot 29(2):383–399
20. Reynolds CW (1987) Flocks, herds and schools: a distributed behavioral model. In: ACM SIGGRAPH Computer Graphics, vol 1. ACM, pp 25–34
21. Shimoyama N, Sugawara K, Mizuguchi T, Hayakawa Y, Sano M (1996) Collective motion in a system of motile elements. Phys Rev Lett 76(20):3870
22. Silva ARD, Lages WS, Chaimowicz L (2010) Boids that see: using self-occlusion for simulating large groups on gpus. Comput Entertain 7(4):51:1–51:20
23. Su H, Wang X, Lin Z (2009) Flocking of multi-agents with a virtual leader. IEEE Trans Autom Control 54(2):293–307
24. Witkowski O, Ikegami T (2014) Asynchronous evolution: emergence of signal-based swarming. In: Proceedings of the Fourteenth International Conference on the Simulation and Synthesis of Living Systems (Artificial Life 14) vol 14, pp 302–309
25. Yokoi H, Yu W, Hakura J (1999) Morpho-functional machine: design of an amoebae model based on the vibrating potential method. Rob Auton Syst 28(2–3):217–236
26. Yokoi H, Yu W, Pfeifer R (2003) Morpho-rate: a macroscopic evaluation and analysis of the morpho-functional machine. In: Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation: Computational Intelligence in Robotics and Automation for the New Millennium, CIRA 2003, Kobe, Japan, July 16–20, 2003, pp 788–793
27. Yu W, Chen G, Cao M (2010) Distributed leader-follower flocking control for multi-agent dynamical systems with time-varying velocities. Syst Control Lett 59(9):543–552