Signal-Driven Swarming: A Parallel Implementation of Evolved Autonomous Agents to Perform A Foraging Task

Aleksandr Drozd[1†], Olaf Witkowski[2], Satoshi Matsuoka[1] and Takashi Ikegami[2]

[1]Global Scientific Information and Computing Center,
Tokyo Institute of Technology, Tokyo, Japan
(Tel: +81-(0)3-5734-3876; E-mail: {alex@smg.|matsu}@is.titech.ac.jp)
[2]Department of Multi-Disciplinary Sciences, University of Tokyo, Tokyo, Japan
(Tel: +81-3-5454-6535; E-mail: {olaf|ikeg}@sacral.c.u-tokyo.ac.jp)

Abstract: From colonies of bacteria to swarms of bees and flocks of birds, countless organisms exhibit a swarming behavior based on local, individual decision making. In such species, the information is used efficiently at the group level to reach optimal behaviors in tasks such as food foraging, which allow to overcome noisy sensory inputs and local minima. In this paper, we extend an abstract agent-based swarming model based on the evolution of neural network controllers, in order to explore further the emergence of swarming. However, we ground our model in a more realistic setting where information about the resource location made partly accessible to the agents, but only through a highly noisy channel. The swarming is shown to critically improve the efficiency of group foraging, by allowing agents to reach resource areas much more easily by correct 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 in simulation sufficient computing power to allow for the evolution of the whole set of dynamics. Because this type of simulations based on neural controllers and information exchanges between agents is computationally intensive, it is critical to optimize the implementation in order to be able to analyze critical masses of individuals. In this work, we address implementation challenges, by showing how to apply techniques from astrophysics known as treecodes to compute the signal propagation, and efficiently parallelize for multi-core architectures. The results confirm that signal-driven swarming improves foraging performance. The agents overcome their noisy individual channels by forming dynamic swarms. The measured fitness is found to depend on the population size, which suggests that large scale swarms may behave qualitatively differently. The minimalist study presented in this paper together with crucial computational optimizations opens the way to future research on the emergence of signal-based swarming as an efficient collective strategy for uninformed search. Future work will focus on further information analysis of the swarming phenomenon and how swarm sizes can affect foraging efficiency.

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

1. INTRODUCTION

The ability of swarms of organisms to coordinate their motion in space has been studied extensively because of their implications for the evolution of social cognition, collective animal behavior and artificial life [1]. Swarming is defined as the very organization of a large number of individuals into a coordinated formation. Using only the information at their disposition in the environment, they are able to aggregate together and move in groups, using various types of dynamics [2-4].

A great interest in swarming consists in the capacity that individuals possess to overcome noise and local minima, in a search space, based on an explicit or implicit exchange of information between agents.

As experiments involving real animals, either inside an experimental setup [5, 6] or in their original ecological environment [7] is costly to reproduce and does not offer the possibility to test for evolutionary paths, researchers have soon turned to computational modeling. Simulating individuals on machines offers easy modification of setup conditions and parameters, tremendous data generation, full reproducibility of every experiment, and easier identification of the underlying dynamics of complex phenomena.

Reynolds [8], with his *boids* (short for *birdoids*) model, was the first to simulate agents swarming in 3D, with only three simple rules. Many researchers have since then tried to improve on this type of approach [9-13].

The evolutionary robotics approach dramatically changed the way experimenters could simulate swarming. Instead of a fixed set of rules, each agent is given an artificial neural network brain that controls its movements. The swarming behavior is evolved by copy with mutations of chromosomes encoding the neural network parameters. By comparing the impact of different selective pressures, this type of methodology, first used in [14] to solve optimization problems, eventually allowed to study the evolutionary emergence of swarming.

In this paper, we describe a simplistic model of agents moving in a 3D map to find a vital resource. The population of agents 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 based on their

† Aleksandr Drozd is the presenter of this paper.

energy, as they are removed from the simulation if their energy level is too low, implementing a darwinian-like process.

We specifically focus on the addition of noise to the food detection sense that the agents possess, 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 (Schelling 1978). 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 of application [15, 16]. Similarly, the size of the formed groups of agents may here be of a crucial importance, in order to reach a critical mass in swarms, enough to overcome very noisy environments. Part of the focus will therefore be on 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 [17], 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 $\vec{c}_t$ is obtained by rotating its velocity vector at the previous time step $\vec{c}_{t-1}$ 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 $o2$, 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 $\vec{x}_t$ is then updated according to its current velocity with $\vec{x}_t = \vec{x}_{t-1} + \vec{c}_t$.

In the original model [17], the individuals were blind, in the sense that they don't 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 10000% 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 is also 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.

Each agent detects both the average value and the total intensity of signals produced in the cone of reception. Those are distinct values, since the intensity of a signal decreases with the distance from the emitter, whereas the value of the signal remains the same. First, the intensity sensor receives a float for the intensity, equal to the sum of every signal emitted within range, divided by the distance, and normalized between 0 and 1. Second, the value sensor receives a float for the average value of the signals from the agents in range.

The agent's neural controller is implemented by a modified Elman artificial neural network (Figure 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, 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, comprised between 0 and 1, are 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 map, and the parent's energy is decreased by a certain cost (arbitrary 10 units). The genotype of an agent defines the range of signals it is able to produce. The values of produced signals are modulated by each agent's decision, modeled by an artificial neural network with the weights encoded in each agent's genotype.

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.

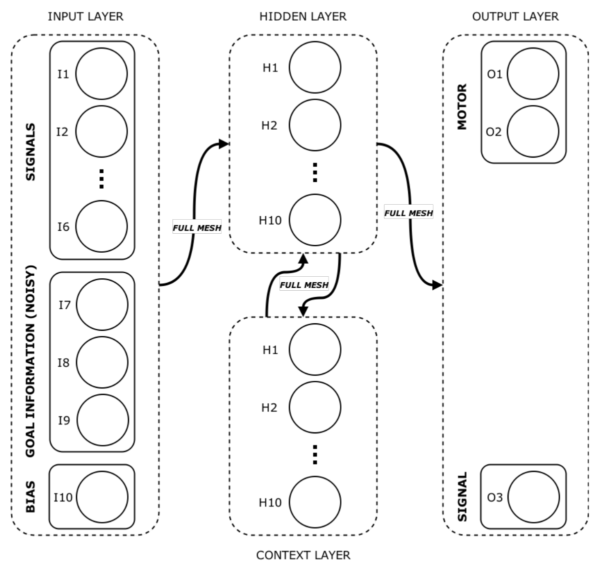


Fig. 1: Architecture of the agent's controller, a recursive neural network composed of 10 input neurons (I_1 to I_10) , 10 hidden neurons (H_1 to H_{10}) , 10 context neurons (C_1 to C_10) and 3 output neurons (O_1 to O_3). Every layer is fully connected to the next one as shown on the diagram. The input neurons receive the average signal value and the total intensity from the other agents. The output neurons O_1 and O_2 control the agent's motion, and O_3 controls the signal it emits.

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. The simulation uses a genetic algorithm with overlapping generations to evolve the weights of the neural networks. Whenever an agent accumulates 4.0 in energy, it becomes able to mate. When a potential mate is within 100 units of distance, they have a probability to mate proportional to the similarity of their genotypes, calculated by euclidian distance. In case of success, an infant individual is created, using a 2-point crossover, with a 5% mutation in the genotype, and added in the neighborhood of the parents, at a distance lower than 100 units to at least one of them. The parents' energy is decreased by 2.0 and the new replica's energy is set to 2.0.

3. IMPLEMENTATION

3.1. Computational Challenges

One iteration of the algorithm performs roughly following steps:

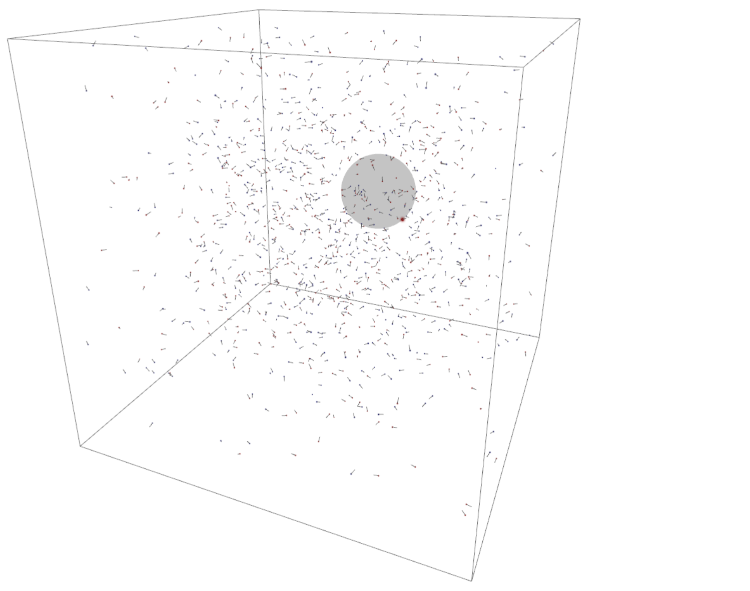


Fig. 2: Visualization of the simulation

- simulate signal propagation
  - build the octree
  - update signal intensities in the intermediate nodes.
  - compute signals perceived by each agent
- perform the feed-forward step in the ANN s
- spawn new agents/remove exhausted agents from the simulation depending on their energy levels.
- (optionally) collect and analyze statistical information

Performance profiling done with Linux Perf tool revealed the computing signal propagation consumes up to 87% of the execution time.

3.2. Signal Propagation with Tree Code

Main performance bottlenecks of the model is the computation of aggregated signal that boid perceives from all other boids. Straight-forward implementation has daunting $O(n^2)$ complexity and becomes prohibitive already at thousands-of-agent scale.

This task, however, resembles 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. As there is no analytical solution available for the N-body problem - N-body simulation is used in practice. For every particle gravitational force $\vec{F} = -\sum_{i\neq j} G\frac{m_i m_j(\vec{r_i} -\ vecr_j)}{|\vec{r_i}-\vec{r_j}|^3+\epsilon}$ is computed at each time step, then positions and velocities of every particle are updated and the computation is repeated. Here $\vec{r}$ is the position of the particle, m is the mass G is gravitational constant and ϵ is small constant to avoid infinite values for the gravitational force when distance between particles goes to zero.

Direct (particle-to-particle) method of computations has quadratic computational complexity and several optimized methods were proposed. In hierarchical methods, such as the BarnesHut simulation [18], an octree is used to divide the volume into cubic cells, so that only particles from nearby cells need to be treated individually, and particles in distant cells can be treated as a single large particle centered at the cell's center of mass. This can dramatically reduce the number of particle pair interactions that must be computed. Octree depth is larger (i.e. cells are refined to smaller cells) for denser parts of the domain.

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 aggregated signal from other agents independently by each senor in directed fashion. These signals thus can not be aggregated simply by summing corresponding vectors. For each cell of the tree which contributes to the signal we compute the projection of the signal from this cell to the vectors representing orientation of sensors.

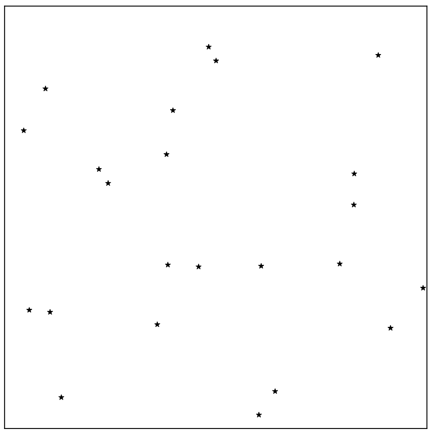
Tree construction is done by recursive binary splitting of the domain so that at most one agent is on one cell. Fig. 3) illustrates space decomposition and corresponding quad-tree (fig. 3c) for 2-dimensional case. Then the octree is fully traversed once, such that each node stores the summed aggregated 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.

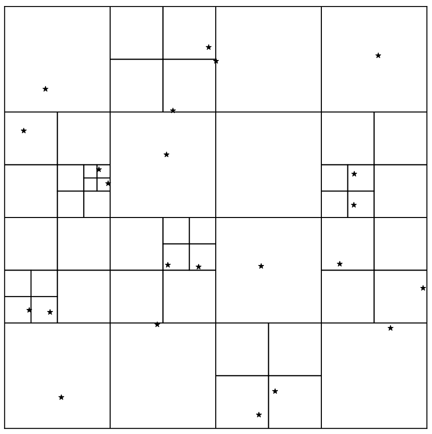
3.3. Parallelization

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, and it has been implemented on many platforms, such as clusters, GPU accelerators, and even FPGA devices [19-21]. We limit our interest to 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 3.60GHz CPU with 4 hyperthreaded cores (8 log-


(a) The original domain


(b) 2D space decomposition

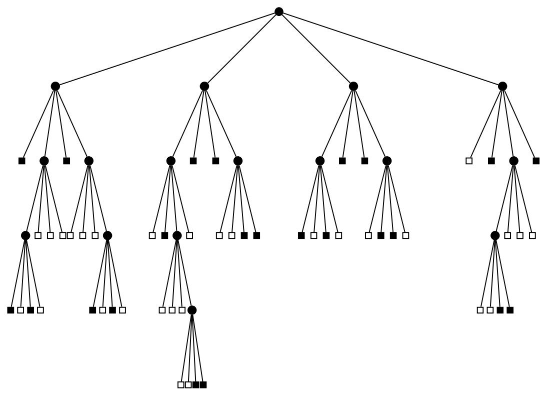

(c) Corresponding quad-tree

Fig. 3: 2-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"

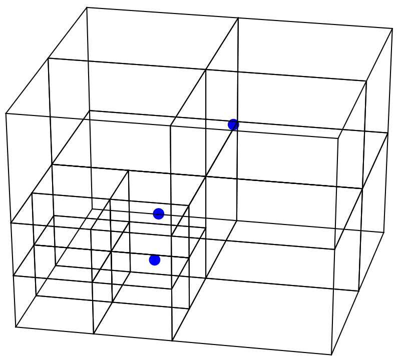


Fig. 4: 3-dimensional space partitioning for 3 agents

ical 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 [22] used for multi-threading support and compiled with gcc 4.9.2 compiler. Figure 5 shows performance scaling with the increase of the number of agents (spawning of new agents was disabled for this benchmark).

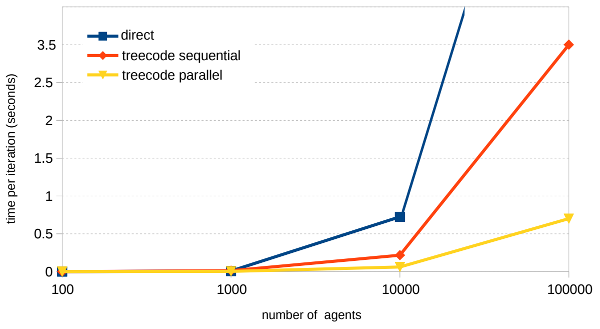


Fig. 5: Performance scaling

3.4. Prototype GPU implementation

Graphics Processing Units (GPUs) provide high computational capacity and memory bandwidth and have been used to accelerate applications in multiple domains. However, GPU programming and execution models impose certain restrictions to the implementation of algorithms to this platform. While classic CPU architecture is optimized for low-latency access to cached data sets, the GPU architecture is optimized for high data parallel throughput computation. The GPU cores are all managed by a thread manager, that can spawn and manage tens of thousands of threads simultaneously. Unlike CPU threads which can run independently, threads on GPU are organized in warps of (currently) 32 threads. Threads in a warp are executed in lock-step, i.e. executing the same instruction on each step. This implies that if one of the threads in the warp will follow one branch in *if-then-else* statement - all threads will execute the same branch, discarding the results later. This is called branch divergence and is one of the main reasons for GPU codes to be inefficient.

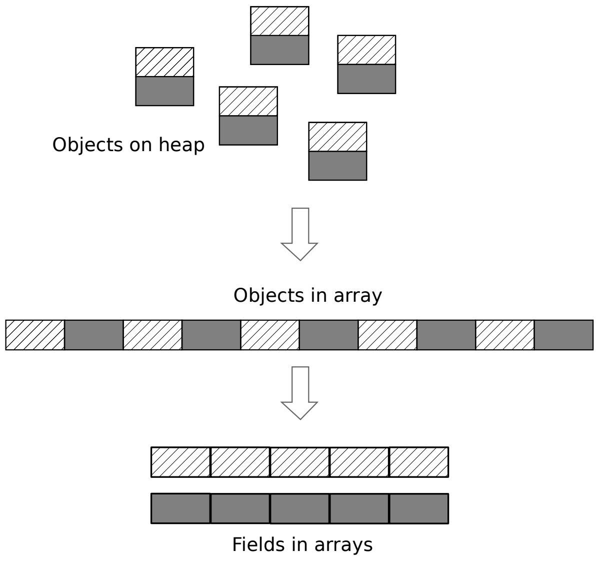


Fig. 6: Memory layout

The second main difference of the GPU execution model is the memory access mechanism. If all threads of the warp simultaneously access adjacent memory elements - such access can be executed in one step. Such coalesced memory access require reorganization of application data layout. In object-oriented agent-based simulation it is natural to allocate the memory for agents on *heap*, including motion parameters, neuron values and weights of the artificial neural net in our case, etc. Dynamic memory allocation became only recently available on GPUs, and still is very expensive. It is possible and fairly easy to allocate contiguous array for N objecte beforehand, but such memory layout would not allow for the coalesced memory access. Bad memory access pattern can cause easily 10 times slow-down of a GPU application [23].

Finally, array of fields layout allows for optimal memory access. Thus, all the i_{th} weight coefficients of each layer of the ANN of each agent should be places on the adjacent memory words. This requires complete memory layout transformation as showed on the fig. 6.

The algorithm for the signal propagation as we implemented it in the parallel CPU version of the code has highly irregular structure and thus is extremely challenging to implement efficiently on GPU. For the original Barnes-Hut algorithm successful GPU implementations have been reported, as well as the significant effort required to achieve sufficient efficiency [24].

However, up to certain number of agents, direct compute on GPU can outperform hierarchical method on parallel CPU/GPU. In the scope of the current work we limit our implementation to N-to-N method for signal propagation and leave the hierarchical algorithm to the future work. Also, although asymptotically the hierarchical method offers clearly superior computational complexity, direct method blends well with GPU computational model and up to certain number of agents can outperform the hierarchical method. Although in general the ability to simulate larger number of agents in desirable, faster runs of smaller number of agents can be used to tune model parameters, like fitness constraints, etc.

4. RELATED WORK

In Particle Swarm Optimization (PSO) problem, a large number of particles is moving though the domain, possibly updating behavioral parameters every iteration. Highly efficient parallel methods for the PSO have been proposed and implemented [25]. However, as there is no interaction between particles - the parallelization strategy is fairly straightforward.

In Reynolds' boids 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 also because boids tend to be fairly separated - grid based methods or methods based on Smoothed Particle Hydrodynamics technique (and corresponding parallelization approaches) works quite well for boids and similar models [26, 27].

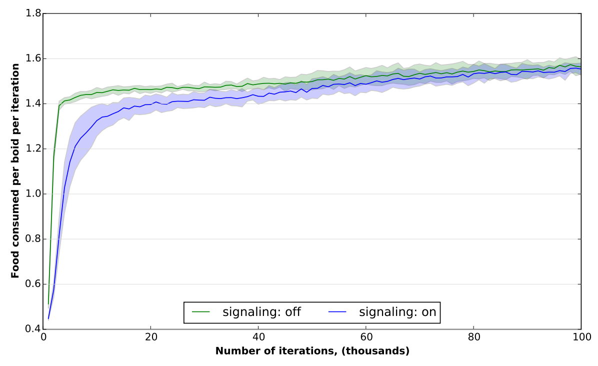
Our model is different in three key aspects: every agent has to receive signals from all other agents; every agent contains an artificial neural network that has to be evaluated at each iteration; the number of agents changes over the simulation. The first aspect makes the above-mentioned optimization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

5. RESULTS

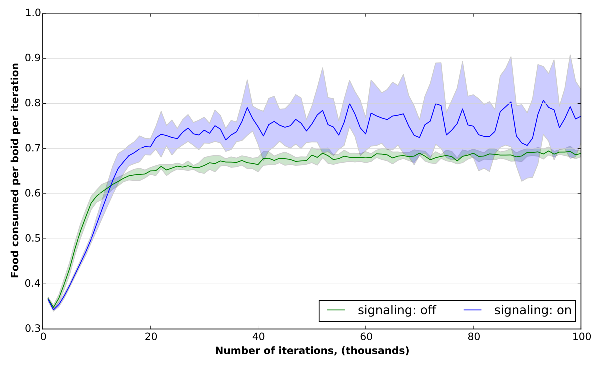
In Figure 7, we can observe that signaling improves the foraging of agents. We use the average amount of food resource obtained per agent per iteration, as a measure of the population's fitness. Without noise, the agents using signaling are less efficient than their silent counterpart, which we found 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 from a certain noise level, the cost to signal is fully compensated by the benefits of signaling, as it helps the foraging of agents. 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.

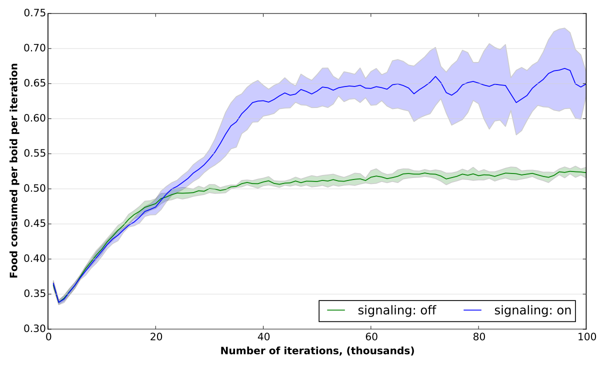
We also observe scale effects in the influence of the signal propagation on the average fitness of the population. Figure 8 shows the results of different population sizes and propagation parameters on foraging efficiency. For a smaller population, only middle values of signal propagation seem to bring about fitter behaviors, whereas this is not the case for larger sizes of population. On the contrary, larger populations are most efficient for lower



(a) Without noise



(b) With noise level 20



(c) With noise level 40

Fig. 7: Efficiency with and without signal with constant noise , mean (central line) and standard deviation range (area plot) over 10 runs

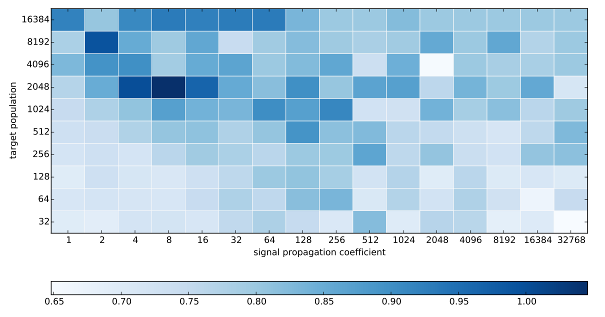


Fig. 8: Effects of population size and signal propagation coefficient

levels of signal propagation. This may suggest a phase transition in the agents' behavior for large populations, eventually in the way the swarming itself helps foraging.

6. CONCLUSION

In this paper, we used an agent-based simulation to show how signal-driven swarming, emerging in an evolutionary simulation such as in [17], allow agents to overcome noisy information channels an improve their performance in a resource finding task. Our first contribution is the very introduction of noise, demonstrating that the algorithm performs well against noises filling up channels of information almost up to their full capacity, in the inputs of agents. The individuals, by means of a swarming behavior helped by basic signaling, manage to globally filter out the noise present in the information from their sensory inputs, to reach the food sites.

We proposed a hierarchical method based on the Barnes-Hut simulation in computational physics and its parallel implementation. We achieved a performance improvement of a few orders of magnitude over the previous implementation [17]. We leave GPU implementation of the hierarchical signal computation method to future work. This implementation is crucial to achieve the simulation of a sufficient number of agents to test for large-scale swarms (i.e. involving a very large number of individuals), which have been suggested to generate qualitatively different dynamics.

The optimization of the fitness 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 the 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.

In future work, it would be interesting to explore further the information flow between agents in this experiment with noise.

7. ACKNOWLEDGMENTS

This paper was partially supported by 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] I. D. Couzin, "Collective cognition in animal groups," *Trends in cognitive sciences*, vol. 13, no. 1, pp. 36–43, 2009.

[2] E. O. Budrene, H. C. Berg, *et al.*, "Complex patterns formed by motile cells of escherichia coli," *Nature*, vol. 349, no. 6310, pp. 630–633, 1991.

[3] N. Shimoyama, K. Sugawara, T. Mizuguchi, Y. Hayakawa, and M. Sano, "Collective motion in a system of motile elements," *Physical Review Letters*, vol. 76, no. 20, p. 3870, 1996.

[4] A. Czirók, A.-L. Barabási, and T. Vicsek, "Collective motion of self-propelled particles: Kinetic phase transition in one dimension," *arXiv preprint cond-mat/9712154*, 1997.

[5] B. L. Partridge, "The structure and function of fish schools," *Scientific american*, vol. 246, no. 6, pp. 114–123, 1982.

[6] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic, "Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1232–1237, 2008.

[7] J. K. Parrish and L. Edelstein-Keshet, "Complexity, pattern, and evolutionary trade-offs in animal aggregation," *Science*, vol. 284, no. 5411, pp. 99–101, 1999.

[8] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, ACM, 1987.

[9] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 3, pp. 304–312, 1992.

[10] C. Hartman and B. Benes, "Autonomous boids," *Computer Animation and Virtual Worlds*, vol. 17, no. 3-4, pp. 199–206, 2006.

[11] F. Cucker and C. Huepe, "Flocking with informed agents," *Mathematics in Action*, vol. 1, no. 1, pp. 1–25, 2008.

[12] H. Su, X. Wang, and Z. Lin, "Flocking of multi-agents with a virtual leader," *Automatic Control, IEEE Transactions on*, vol. 54, no. 2, pp. 293–307, 2009.

[13] W. Yu, G. Chen, and M. Cao, "Distributed leader–follower flocking control for multi-agent dynamical systems with time-varying velocities," *Systems & Control Letters*, vol. 59, no. 9, pp. 543–552, 2010.

[14] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1, pp. 39–43, New York, NY, 1995.

[15] G. Marwell and P. Oliver, *The critical mass in collective action.* Cambridge University Press, 1993.

[16] P. E. Oliver and G. Marwell, "Whatever happened to critical mass theory? a retrospective and assessment," *Sociological Theory*, vol. 19, no. 3, pp. 292–311, 2001.

[17] O. Witkowski and T. Ikegami, "Asynchronous evolution: Emergence of signal-based swarming," *Proceedings of the Fourteenth International Conference on the Simulation and Synthesis of Living Sys-*

tems (Artificial Life 14), vol. 14, pp. 302–309, July 2014.

[18] J. Barnes and P. Hut, "A hierarchical o(n log n) force-calculation algorithm," *Nature*, vol. 324, pp. 446–449, Dec 1986.

[19] D. Blackston and T. Suel, "Highly portable and efficient implementations of parallel adaptive n-body methods," in *In SC'97*, 1997.

[20] T. Hamada, R. Yokota, K. Nitadori, T. Narumi, K. Yasuoka, and M. Taiji, "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, Nov 2009.

[21] J. Coole, J. Wernsing, and G. Stitt, "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, Dec 2009.

[22] "Openmp application program interface, version 4.0." openmp.org, 2013. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

[23] I.-J. Sung, G. Liu, and W.-M. Hwu, "Dl: A data layout transformation system for heterogeneous computing," in *Innovative Parallel Computing (InPar), 2012*, pp. 1–11, May 2012.

[24] M. Burtscher and K. Pingali., *GPU Computing Gems Emerald Edition*, ch. 6, An Efficient CUDA Implementation of the Tree-based Barnes Hut n-Body Algorithm, pp. 75–92. January 2011.

[25] L. Mussi, F. Daolio, and S. Cagnoni, "Evaluation of parallel particle swarm optimization algorithms within the cuda architecture," *Information Sciences*, vol. 181, no. 20, pp. 4642 – 4657, 2011. Special Issue on Interpretable Fuzzy Systems.

[26] A. R. D. Silva, W. S. Lages, and L. Chaimowicz, "Boids that see: Using self-occlusion for simulating large groups on gpus," *Comput. Entertain.*, vol. 7, pp. 51:1–51:20, Jan. 2010.

[27] L. Pimenta, G. Pereira, N. Michael, R. Mesquita, M. Bosque, L. Chaimowicz, and V. Kumar, "Swarm coordination based on smoothed particle hydrodynamics technique," *Robotics, IEEE Transactions on*, vol. 29, pp. 383–399, April 2013.