Designing Emergent Swarm Behaviors using Behavior Trees and Grammatical Evolution

Extended Abstract

Aadesh Neupane
Brigham Young University
aadeshnpn@byu.edu

Michael A. Goodrich
Brigham Young University
mike@cs.byu.edu

KEYWORDS

Behavior Trees; Swarms; Grammatical Evolution

ACM Reference Format:
Aadesh Neupane and Michael A. Goodrich. 2019. Designing Emergent Swarm Behaviors using Behavior Trees and Grammatical Evolution. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 MOTIVATION AND PROBLEM STATEMENT

Bio-inspired collectives like honeybee, ant, and termite colonies provide elegant distributed solutions to complex collective problems like finding food sources, selecting a new site, and allocating tasks. Effective collective behaviors emerge from biological swarms through local interactions (see, for example, [7, 17, 19]).

Despite the potential benefits of bio-inspired algorithms, only a few organisms have been explored for their collective behavior; for example, very little is understood about the construction methods of termites [5]. One reason for slow research is the effort involved in understanding individual agent behavior and creating mathematical models to describe both individual and collective behaviors [2]. Mimicking an evolutionary process with artificial agents may yield useful collective behaviors in a reasonable time.

Conventional approaches for evolving swarms behaviors used Finite State Machines (FSM) with or without neuro-evolutionary algorithms [6, 10, 12, 14, 15]. When the system is complex and the number of states is huge, a hierarchical finite state machine (HFSM) offers benefits [1, 20]. Unfortunately, HFSMs must trade-off between reactivity and modularity [3]. Also, behaviors encoded in HFSMs can be hard to debug and extend [11]. Behaviour Trees (BTs), which are useful in game design, overcome some HFSM limitations [8].

BTs have recently been used to evolve behaviors for robot swarms. Jones et al. [9] used genetic evolution algorithm to evolve a BT for a Kilobot foraging task. For detailed information on BTs, see [4]. Distributed Grammatical Evolution (GE) coupled with BTs might be adequate for generating swarm behaviors.

This abstract summarizes a framework combining a distributed GE called GEESE [13] with BTs to generate swarm behaviors. We identified twenty-eight primitive individual behaviors designed to mimic behaviors frequently seen in the swarm literature. We then designed a BNF grammar that embeds the primitive behaviors as BT nodes and is general enough to solve many collective spatial allocation tasks. Results show that the performance of evolved swarm behaviors was better than a hand-coded solution for a Single-source foraging task.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

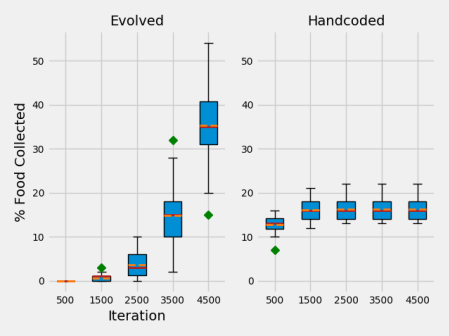

Figure 1: Single-source foraging

2 APPROACH

GE is a context-free grammar-based genetic program paradigm that is capable of evolving programs or rules in many languages [16]. GE adopts a population of genotypes represented as binary strings, which are transformed into functional phenotype programs through a genotype-to-phenotype transformation. The transformation uses a BNF grammar, which specifies the language of the solutions.

We extend a specific multi-agent GE algorithm called *GEESE* in two ways: First, the BNF grammar was redesigned to allow BT programs to be the evolutionary phenotype. Second, three levels of fitness function were designed to promote not only task-specific success but also exploration.

Swarm Grammar. This swarm grammar used with GEESE incorporates individual agent rules that can produce a valid BT which induce desirable swarm behaviors. The BNF grammar guides the genotype-to-phenotype mapping process.

⟨*s*⟩ ::= ⟨*sequence*⟩ | ⟨*selector*⟩ (1)
⟨*sequence*⟩ ::= ⟨*execution*⟩ | ⟨*s*⟩⟨*s*⟩ | ⟨*sequence*⟩ ⟨*s*⟩ (2)
⟨*selector*⟩ ::= ⟨*execution*⟩ | ⟨*s*⟩⟨*s*⟩ | ⟨*selector*⟩ ⟨*s*⟩ (3)
⟨*execution*⟩ ::= ⟨*conditions*⟩⟨*action*⟩ (4)
⟨*conditions*⟩ ::= ⟨*condition*⟩ ⟨*conditions*⟩ | ⟨*condition*⟩ (5)
⟨*condition*⟩ ::= NeighbourObjects | IsDropable_⟨*sobjects*⟩ | … (6)
⟨*action*⟩ ::= MoveTowards_⟨*sobjects*⟩ | Explore | … (7)
⟨*sobjects*⟩ ::= Hub | Sites | Obstacles (8)
⟨*dobjects*⟩ ::= Food | Debris (9)
⟨*cobjects*⟩ ::= Signal | Cue (10)
⟨*objects*⟩ ::= ⟨*sobjects*⟩ | ⟨*dobjects*⟩ (11)

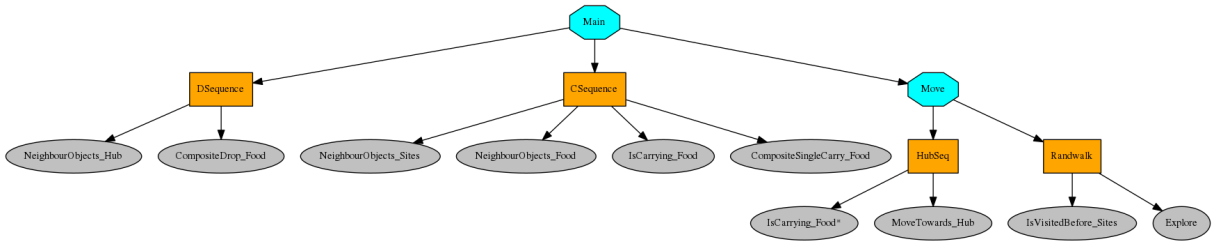

Figure 2: Hand-coded Behavior Tree for single-source foraging task.

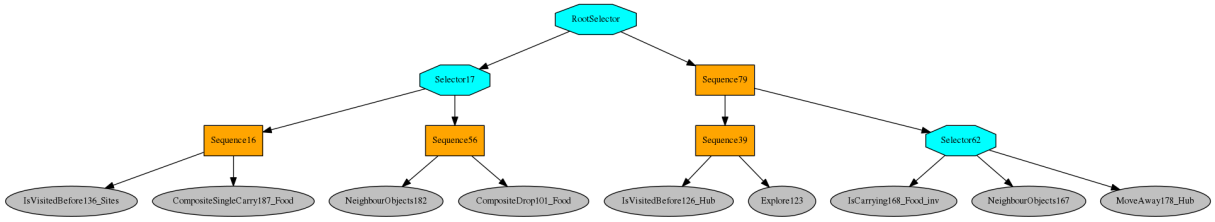

Figure 3: One of the evolved Behavior Tree for single-source foraging task.

The right-hand side of the production rule 1 defines the BT control structures. Production rules [2-5] recursively define execution nodes. Production rules [6-7] define primitive behaviors. Production rule 8 defines the *general static elements* in the swarm environment. Production rule [9-11] defines the *dynamic objects* in the environment.

Fitness Functions. Let $\mathbf{A}_t$ denote the fitness of the agent A. The overall agent fitness of a genome at time step t is given by

$$A_t = \beta(A_{t-1}) + (O_t + E_t + P_t). \quad (1)$$

where O_t is the task-specific objection function, E_t is the exploration fitness, P_t is the prospective fitness, A_{t-1} is the agent fitness in previous generation, and β is the generational discount factor. E_t and P_t are a form of "bootstrapping" to help boost learning. E_t rewards exploration to different spatial region where as P_t rewards agents for persisting in activities that may be useful.

The most fit 50% of the genomes in the agent's repository are selected to be parents. Crossover followed by mutation is then performed to create a new population of genomes. A single genome is then selected based on diversity fitness, and the agent follows the corresponding BT program. In every simulation step, each agent evaluates its fitness (A_t) using its BT controller. Three distinct class of fitness function (equation 1), each necessary given the non-episodic learning and the large search space, are used to evaluate the agent.

3 RESULTS

A single-source foraging scenario from [18] is considered. The agent's task is to collect food from a *source* region in the environment and bring the food to a *hub* region in the environment. Agents do not have prior information regarding the source location.

Figure 1 shows the average % food collected by the swarm in the task. The left and right portion shows the performance by evolved and hand-coded behaviors respectively. The evolved behaviors clearly perform better than the hand-coded behavior.

It takes about 3000 iterations before GE-agents start gathering food in the hub because agents need to explore the environment first to find the location of the source. Since there is just a single source it takes agents some time to find the source and communicate the information of the source. In contrast, the hand-coded behavior was able to collect food at roughly 500 iterations.

After analyzing the BT for both behaviors in Figures 2–3, we observe that the exploration node is independent of other nodes in hand-coded BT whereas in evolved BT, the explore node is dependent on another composite node. The independence of explore node from other nodes allows the hand-coded BT to explore the environment much faster than the evolved BT.

Subjective observations, to be quantified in future work, indicate that the diversity of agents in the evolved agents is one reason for their success; all the agents have the same behavior for the hand-coded solution. Another reason for the inferior performance of hand-coded behavior is that it is hard for humans to construct a BT tree with a cyclic nature as BT are directed trees. For foraging, there is cyclic pattern of visiting hub and other objects of interest.

4 SUMMARY

Results show that a recursively defined BT-based grammar, built from common agent behaviors, can be used by the GEESE algorithm to evolve solutions to single-source foraging task. Because of the difficulty of solving the credit assignment problem, bootstrapping methods must be added to the fitness function to find solutions in reasonable time.

ACKNOWLEDGMENTS

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported by the ONR grant number N000141613025.

REFERENCES

[1] Rodney Brooks. 1986. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation* 2, 1 (1986), 14–23.

[2] Scott Camazine, Jean-Louis Deneubourg, Nigel R Franks, James Sneyd, Eric Bonabeau, and Guy Theraula. 2003. *Self-organization in biological systems*. Vol. 7. Princeton University Press.

[3] Michele Colledanchise and Petter Ögren. 2017. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics* 33, 2 (2017), 372–389.

[4] Michele Colledanchise and Petter Ögren. 2018. Behavior Trees in Robotics and AI: An Introduction. (2018).

[5] Lucy Cooke. 2018. UNDERBUG An Obsessive Tale of Termites and Technology. (2018).

[6] Eliseo Ferrante, Edgar Duéñez-Guzmán, Ali Emre Turgut, and Tom Wenseleers. 2013. GESwarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proc. of the 15th annual conf. on Genetic and evolutionary computation*. ACM, 17–24.

[7] Deborah M Gordon. 2010. *Ant encounters: interaction networks and colony behavior*. Princeton University Press.

[8] D Isla. [n. d.]. Handling complexity in the Halo 2 AI, 2005. *URL: http://www. gamasutra. com/gdc2005/features/20050311/isla _01. shtml [21.1. 2010]* ([n. d.]).

[9] Simon Jones, Matthew Studley, Sabine Hauert, and Alan Winfield. 2018. Evolving behaviour trees for swarm robotics. In *Distributed Autonomous Robotic Systems*. Springer, 487–501.

[10] Lukas König, Sanaz Mostaghim, and Hartmut Schmeck. 2009. Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics* 2, 4 (2009), 695–723.

[11] Chong-U Lim. 2009. An AI Player for DEFCON: an evolutionary approach using behavior trees. *Imperial College, London* (2009).

[12] Aadesh Neupane, Michael A Goodrich, and Eric G Mercer. 2018. GEESE: Grammatical Evolution Algorithm for Evolution of Swarm Behaviors. In *Proceedings of the 17th Intl. Conf. on Autonomous Agents and MultiAgent Systems*. Intl. Foundation for Autonomous Agents and Multiagent Systems, 2025–2027.

[13] Aadesh Neupane, Michael A Goodrich, and Eric G Mercer. 2018. GEESE: grammatical evolution algorithm for evolution of swarm behaviors. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 999–1006.

[14] Pavel Petrovic. 2008. Evolving behavior coordination for mobile robots using distributed finite-state automata. In *Frontiers in evolutionary robotics*. InTech.

[15] Agnes Pintér-Bartha, Anita Sobe, and Wilfried Elmenreich. 2012. Towards the lightâĂŤComparing evolved neural network controllers and Finite State Machine controllers. In *Intelligent Solutions in Embedded Systems (WISES), 2012 Proceedings of the Tenth Workshop on*. IEEE, 83–87.

[16] Conor Ryan, JJ Collins, and Michael O Neill. 1998. Grammatical evolution: Evolving programs for an arbitrary language. In *European Conf. on Genetic Programming*. Springer, 83–96.

[17] Thomas D Seeley. 2009. *The wisdom of the hive: the social physiology of honey bee colonies*. Harvard University Press.

[18] John H Franks Sudd, Nigel R John H Sudd, and Nigel R Franks. 1987. *The behavioural ecology of ants*. Technical Report.

[19] David JT Sumpter. 2010. *Collective animal behavior*. Princeton University Press.

[20] Antti Valmari. 1996. The state explosion problem. In *Advanced Course on Petri Nets*. Springer, 429–528.