Designing Emergent Swarm Behaviors using Behavior Trees and Grammatical Evolution

Aadesh Neupane
Brigham Young University
aadeshmpn@byu.edu

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

KEYWORDS
Behavior Trees; Swarms; Grammatical Evolution

ACM Reference Format:

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.

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.

\begin{align}
\langle s \rangle &::= \langle sequence \rangle | \langle selector \rangle \\
\langle sequence \rangle &::= \langle execution \rangle | \langle s \rangle \langle s \rangle | \langle sequence \rangle \langle s \rangle \\
\langle selector \rangle &::= \langle execution \rangle | \langle s \rangle \langle s \rangle | \langle selector \rangle \langle s \rangle \\
\langle execution \rangle &::= \langle conditions \rangle \langle action \rangle \\
\langle conditions \rangle &::= \langle condition \rangle \langle conditions \rangle | \langle condition \rangle \\
\langle condition \rangle &::= \langle NeighbourObjects \rangle | \langle IsDroppable\_\langle subjects \rangle \rangle | \langle isNear\_\langle objects \rangle \rangle | \langle isDebris\_\langle objects \rangle \rangle | \langle isType\_\langle objects \rangle \rangle | \langle isObstacles\_\langle objects \rangle \rangle | \langle isSources\_\langle objects \rangle \rangle | \langle isCues\_\langle objects \rangle \rangle | \langle isSignal\_\langle objects \rangle \rangle | \langle isCue\_\langle objects \rangle \rangle | \langle isSignal\_\langle objects \rangle \rangle \\
\langle action \rangle &::= \langle MoveTowards\_\langle objects \rangle \rangle | \langle Explore \rangle | \langle Explore \rangle \\
\langle subjects \rangle &::= \langle Hub \rangle | \langle Sites \rangle | \langle Obstacles \rangle \\
\langle objects \rangle &::= \langle Food \rangle | \langle Debris \rangle \\
\end{align}

Fitness Functions. Let \( A_t \) denote the fitness of the agent \( A \). The overall agent fitness of a genome at time step \( t \) is given by

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

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 \( \beta \) 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 whereas \( P_t \) 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 GEASE 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