Learning Resilient Swarm Behaviors via Ongoing Evolution

Aadesh Neupane and Michael A. Goodrich

Department of Computer Science, College of Physical and Mathematical Sciences,
Brigham Young University, Provo, UT, USA
adeshnpn@byu.edu, mike@cs.byu.edu

Abstract. Grammatical evolution can be used to learn bio-inspired solutions to many distributed mulitagent tasks, but the programs learned by the agents are often not resilient to perturbations in the world. Biological inspiration from bacteria suggests that ongoing evolution can enable resilience, but traditional grammatical evolution algorithms learn too slowly to mimic rapid evolution because they utilize only vertical, parent-to-child genetic variation. Prior work with the BeTr-GEESE grammatical evolution algorithm showed that individual agents who use both vertical and lateral gene transfer rapidly learn programs that perform one step in a multi-step problem even though the programs cannot perform all required subtasks. This paper shows that BeTr-GEESE can use ongoing evolution to produce resilient collective behaviors on two goal-oriented spatial tasks, foraging and nest maintenance, in the presence of different types of perturbation. The paper then explores when and why BeTr-GEESE succeeds, emphasizing two potentially generalizable properties: modularity and locality. Modular programs enable real-time lateral transfer, leading to resilience. Locality means that the appropriate phenotypic behaviors are local to specific regions of the world (spatial locality) and that recently useful behaviors are likely to be useful again in the near future (temporal locality).

1 Introduction

Bees, ants, termites, and other biological collectives efficiently solve complex problems without centralized control like finding a new site, foraging, nest-building, and protecting the colony, even when the environment fluctuates [21,60]. Such biological collectives *resiliently* accomplish tasks[1] in the presence of various perturbations that arise in the environment. Research has identified various resilience mechanisms including stress-induced adaptation [39,50], local interaction [21], task switching [59], lateral transfer [34], and modularity [71].

The purpose of this paper is to identify potentially generalizable properties that can enable grammatical evolution to use ongoing evolution to produce resilient swarm behaviors. Evolutionary approaches are powerful tools for learning

[1] Resilient task performance differs from ecological resilience in which population sizes show resilience to variations [22] and from stability-based definitions of resilience in which some property of a collective remains in a locally stable region [24].

bio-inspired swarm behaviors [12,15,79]. Grammatical evolution (GE) is a type of algorithmic evolution where evolutionary operators act on a given grammar to learn individual agent programs from the grammar. GE has been used to evolve swarm behaviors [18,44,43], and most demonstrations first evolve solutions and then deploy those learned solutions as fixed strategies. These fixed strategies can fail or degrade when perturbations are introduced into the environment.

Biology suggests solutions to overcome performance degradation of fixed strategies including stressed induced mutation, lateral gene transfer, and continuous evolution in bacteria [23]. A simple view of rapid adaptation is (a) that individual agents learn modular, circumstance-specific behaviors, and (b) that collective diversity allows suitable module exchange when circumstances change [30,76]. Ongoing evolution is unlikely to increase the resilience of many GE algorithms for two reasons. First, many GE algorithms learn too slowly to rapidly adapt, as demonstrated by the low rate of learning successful behaviors [19]. Second, the fitness of many collective behaviors requires significant coordination among agents, making it difficult to apportion fitness to the individual agents trying to learn how to contributed to the collective task. Carefully constructed fitness functions (e.g., intrinsic and extrinsic motivators [44,62,77]) help solve the second problem but are unlikely to be as useful in the presence of perturbations since new fitness functions might be needed for each perturbation type.

A curious phenomenon in prior work on the BeTr-GEESE grammatical evolution algorithm suggests that the algorithm can be adapted to successfully apply lateral gene transfer to produce collective resilience. Specifically, BeTr-GEESE agents successfully perform collective foraging and nest maintenance *while they are evolving*, but when learning stops the collective performs poorly [45]. Individual BeTr-GEESE agents do not learn programs that are sophisticated enough to perform all required subtasks but instead rapidly learn modular behaviors that perform only one subtask. The collective succeeds by using "time-multiplexing" in which agents switch behaviors by laterally exchanging modules, allowing all subtasks to be performed [45]. Time-multiplexing is a form of *lateral gene transfer* [48] in which genetic material transfers between organisms, in contrast to *vertical gene transfer* from parent to child.

This paper explores how BeTr-GEESE uses lateral gene transfer to produce resilient swarm behaviors in two distributed, *divisible and additive*[2] spatial tasks: foraging and nest maintenance. Resilience is first demonstrated by applying various types of perturbations during evolution and then measuring resulting performance. The concepts of *modularity* and *locality* are then used to explain how resilience emerges. *Modularity* in evolutionary algorithms means geneotype-to-phenotype mappings tend to associate specific phenotypic characteristics with specific genes, in contrast to "general purpose" genes that exhibit complex phenotypes [76]. The *divisible and additive* nature of foraging and nest maintenance mean that individual agents can evaluate the fitness of modular behaviors without requiring the cooperation of many agents. *Locality* is a concept from the field

[2] *Divisible* and *additive* multiagent tasks can be broken into subtasks achievable by individual programs that each contribute to the group problem to be solved [65].

of trace compression and cache design in computer architecture [57,63] in which useful bytes of data cluster in time (temporal locality) and in adjacent memory cells (spatial locality). In a multiagent collective solving a spatial task, *temporal locality* means that a (modular) behavior that has been useful in the recent past is likely to be useful again soon, and *spatial locality* means that successful (modular) behaviors are likely to be localized to certain regions of the world.

2 Related Work

The ability of biological collectives to solve problems with partial, uncertain information has motivated AI researchers to mimic their behaviors [55,40,47,56,6]. One way to create collective algorithms is to collect data from natural colonies and then create mathematical models that can be used in algorithms [46,67,68]. Another way is to use evolutionary techniques to evolve agent behaviors [11].

Evolutionary robot systems require the creation of agent controllers: (i) state-machines [18,51,52,29], (ii) neural networks [7,38,14,72], (iii) behavior trees [32,33], and (iv) controllers learned through genetic or grammatical evolution [31,4,18,42]. Individuals in a swarm do not need to possess complex capabilities to evolve effective swarm behaviors [31]. Even with favorable controller choices, evolutionary algorithms, and fitness functions [41], evolved collective behaviors often degrade when tested with real robots or in presence of uncertainties [26].

Modularity contributes to evolving resilient behaviors because (i) modular organizations permit changes to one module without perturbing other modules and (ii) modules can be combined and reused to create new functions [76,2,1,78]. In a modular system, a module has more frequent interaction within the subsystem than outside the subsystem [61], so *modularity* is the measure of interaction between different components in a system [61,53]. For example, a highly modular grammar enabled a GE algorithm to evolve better multiagent solutions [69]. Evolving resilience might require additional evolutionary operations, such as lateral gene transfer [25,35,54]. Lateral transfer allows single agents to efficiently evolve complex behaviors when rewards are sparse and delayed [37,16].

There are many evolutionary algorithms designed to learn resilient swarm and multiagent behaviors [3,27,74,5,75,36]. Unfortunately, high fitness is not equivalent to resilient behaviors and fittest individuals are easily disrupted by genetic changes [64]. To the best of our knowledge, this paper is the first to evaluate the resilience of a grammatical evolution algorithm using ongoing evolution.

3 BeTr-GEESE Overview

BeTr-GEESE agents [45] use *sense-act-update* evolution steps to learn individual behaviors or "programs" from a bio-inspired task grammar. During the *sense* phase, agents exchange genes with (nearby) agents. The definition of "nearby" is controlled by the *grid size* (GS) parameter, and the willingness to transfer genes is controlled by the *interaction probability* (IP). During the *act* phase, an agent queries its storage pool to determine whether the pool size exceeds its *storage*

threshold (ST) parameter. If the threshold is exceeded, agents apply the select-crossover-mutate genetic operations to the gene pool. During the *update* phase, an agent replaces its current gene if there is a new gene with higher fitness. BeTr-GEESE agents discard all other genes after updating and begin again.

Like other GE algorithms, BeTr-GEESE encodes genes as a sequence of integer *codons*. The codon sequence specifies the order in which grammar productions are used to produce the agent controller phenotype. The BeTr-GEESE grammar (see appendix) implements a behavior tree (BT) that has a postcondition, precondition, action (PPA) structure [8], with leaf nodes that either test basic properties of the environment (productions (7,11)) or perform basic actions like moving or picking up objects (production (14)). The names in productions (7,11,15) are self-explanatory given the descriptions of foraging and nest maintenance in Section 4.1. Each BT returns a success, failure, or running status that encodes how successful the program has been in satisfying a post-condition.

Each agent acts in the environment using its phenotype program. BeTr - GEESE rewards behaviors that promote genetic diversity and world exploration, observe or accomplish subtasks, or avoid constraint violations. Phenotype fitness is subjectively defined as $A_t = 0.1(A_{t-1}) + (E_t + B_t)$, where $A_0 = D$. Phenotype fitness is evaluated over time, which is necessary because there is delay between acting and receiving a reward. When agents exchange genes, they also exchange the genes' fitness values, making it possible for an agent to avoid "testing" the phenotype because its fitness is known. *Diversity fitness*, D, promotes gene diversity and is used when a gene is first created ($t = 0$) from either the initial random population or through mutation and crossover of an existing gene pool [73,58,70]; D is defined as the total number of unique behavior nodes in the BT divided by the total possible behaviors. *Exploration fitness* [10], E, promotes visiting new locations, and is defined as the number of unique world locations visited by the agent. *BT feedback fitness*, B, is defined as the sum of post-condition, constraint, and BT root node rewards. For each post-condition or root node status returning *success*, a subjectively chosen reward of $+1$ occurs, and -2 reward occurs if a constraint node returns *failure*.

4 Resilience Experiments

This section demonstrates that, when the BeTr-GEESE algorithm uses ongoing evolution, agents are capable of solving problems that arise when the world or the agents are perturbed. The next section explores why.

4.1 Experiment Design

Experiments were conducted using two tasks: *foraging* and *nest maintenance*. Experiments use a population of 100 agents that move with speed of 2 units per time step in a 100x100 grid environment with agent neighborhood sensing defined by the *grid size* parameter GS=10. A hub of radius ten is placed at the origin. A maximum of 12,000 evolution steps are allowed. Quoting from [45],

foraging requires agents to retrieve food from a source to a hub. A single foraging site of radius ten with 100 "food" units is randomly placed at 30 units from the hub. Task performance is the percentage of food at the hub. *Nest maintenance* requires agents to move debris near the hub to anywhere farther than 30 units from the hub. 100 "debris" objects are placed within ten units of the hub.

Parameters	BeTr-GEESE
Parent-Selection	Most fit 50%
Mutation Probability	0.01
Crossover Probability	0.9
Crossover	variable_onepoint [49,17]
Maximum Depth of Derivation Tree	10 levels

The parameters in the table above are used in the experiments. These parameters were subjectively selected from choices made in prior work on grammatical evolution. Maximum tree depth is a practical parameter that limits the effect of recursive dependencies in the grammar. *PonyGE2* [17] was used to implement GEESE-BT and BeTr-GEESE. BT controllers were created using *py_trees* [66], and the swarm simulation environment was created using *Mesa* [28]. Experiments ran on a machine with an i9 CPU, 64 GB RAM running 16 parallel threads.

The independent variable is perturbation type, described below [36]. The first dependent variable is the *power* resilience metric [36], defined as the peak success probability achieved before the maximum number of allowed evolution steps $T = 12000$. The second dependent variable is an affine transformation of the *time efficiency* (t_θ) resilience measure [36], which is defined as the time required for an algorithm to satisfy a given performance threshold $\theta = 80\%$. *Efficiency* is defined as $e = ((T_{\max} - t_\theta)/T_{\max}) * 100$ where $T_{\max} = 12000$. Efficiency is set to zero for trials in which the threshold is not met.

4.2 Results

The four upper (respectively, lower) sub-plots in Figure 1 show power (respectively, efficiency) for each swarm task. Sixteen independent simulations were performed for each experiment condition described below.

Ablation. An *ablation* perturbation reduces information, control, or possibilities [36]. Adding obstacles is an ablation perturbation since obstacles reduce the navigable space for the agents. In experiments, $obs \in \{1, 2, \ldots, 5\}$ obstacles are added to the world at time $t \in \{1000, 2000, \ldots, 11000\}$. Obstacles remain in the world after their introduction. The experiment conditions are all combinations of t and obs. The first column of Fig. 1 shows mean power and efficiency.

Addition. An *addition* perturbation increases the set of observable states or actions [36]. An experiment was performed where new actions are added to the BeTr-GEESE BNF grammar in appendix A as follows:

$$\langle action \rangle ::= \langle motion \rangle | \langle nonmotion \rangle \tag{1}$$

$$\langle motion \rangle ::= \text{MoveTowards\_}\langle sobjects \rangle\text{\_}\langle motiontype \rangle | \text{Explore\_0\_}\langle motiontype \rangle | \text{MoveAway\_}\langle sobjects \rangle\text{\_}\langle motiontype \rangle \tag{2}$$

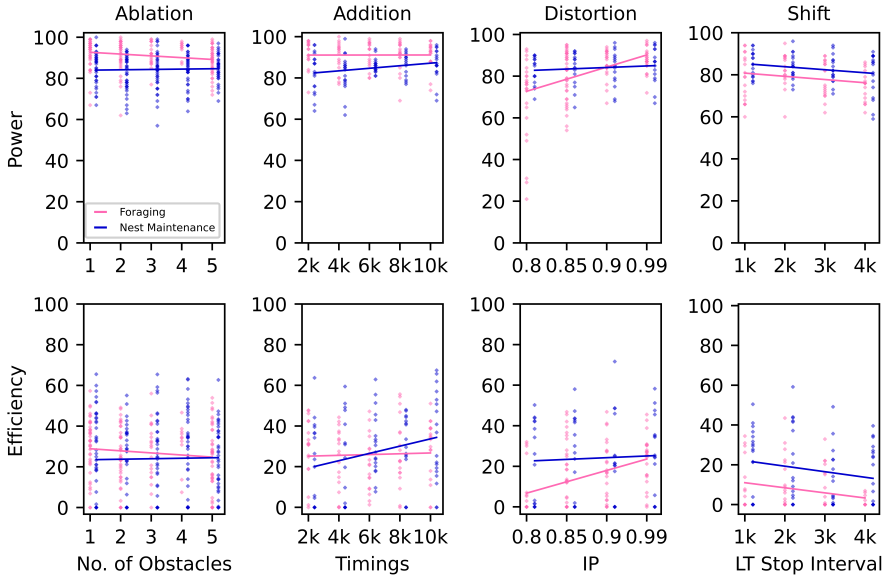

Fig. 1: Efficiency and power over a range of perturbations for foraging and next maintenance; IP=0.85, ST=7, and GS=10.

$$\langle motiontype \rangle ::= \text{Normal}|\text{Avoid} \tag{3}$$
$$\langle nonmotion \rangle ::= \text{CompositeSingleCarry\_}\langle dobjects \rangle|\text{CompositeDrop\_}\langle dobjects \rangle \tag{4}$$

This modification increases an agent's action set by allowing an agent to choose between locomotion behaviors with/without obstacle avoidance. Two obstacles were randomly added to the environment at time $t \in \{1000, 2000, \ldots, 11000\}$. The second column of Fig. 1 shows mean power and efficiency.

Distortion. A *distortion* perturbation alters the probability with which states or actions occur [36]. Altering IP changes how frequently agents evolve, distorting probable states and actions. Experiment conditions used IP values in $\{0.8, 0.85, 0.9, 0.99\}$. The third column of Fig. 1 shows mean power and efficiency.

Shift. A *shift* perturbation combines the effects of multiple instances of ablation, addition, or distortion operations [36]. For the shift experiments, lateral transfer is initially turned *on* but turned *off* at time step 1000 for a duration of $\Delta \in \{1000, 2000, \ldots, 4000\}$ time steps, preventing an agent from collecting genes from neighbors. The fourth column of Fig. 1 shows mean power and efficiency.

4.3 Discussion

BeTr-GEESE agents show high resilience according to the power metric. Given sufficient time, agents evolve solutions when perturbations occur. High power persists across a range of perturbation types and parameters. The behaviors

are inefficient because evolving revised solutions rarely occurs quickly. A power-efficiency tradeoff is observed, similar to optimality-robustness tradeoffs in control theory, where robust systems are often suboptimal [13,20]. Thus, ongoing evolution makes BeTr-GEESE agents inefficient but with high resilience power.

5 What Enables Resilience?

The section's goal is to identify properties of the BeTr-GEESE algorithm that could potentially generalize to other problems and GE swarm algorithms. Understanding these properties also sheds light on the limitations of using ongoing lateral transfer in GE to enable resilience.

5.1 Modularity

This subsection evaluates modularity properties of BeTr-GEESE. In prior work [45], two GE algorithms were compared: BeTr-GEESE and GEESE-BT. The two algorithms used the same genetic operators, the same parameter values (IP=0.85, ST=7, and GS=10), the same form of lateral transfer between agents, the same basic actions, and the same preconditions and postconditions. The algorithms differed in three ways. First, BeTr-GEESE's grammar had a *CanMove* constraint necessary when obstacles are present in the world. Second, GEESE-BT's grammar produced traditional BTs and BeTr-GEESE's grammar produced PPA-style BTs [8]. Third, BeTr-GEESE used the fitness function described above, and GEESE-BT used both the fitness function above and task-specific motivators.

In the prior work, foraging (respectively, nest maintenance) was considered *successful* if more than 80% of the food is collected (respectively, debris is removed) during the time period when agents were evolving. *Success rate* was defined as the ratio of the number of successful evolution trials to the total number of trials. BeTr-GEESE's success rate was 75%, more than eight times higher than GEESE-BT even when GEESE-BT used the task-specific fitness functions. Note that success requires each basic action (production 15) in the grammar.

Having established that BeTr-GEESE performs successfully while learning, we now present new work that addresses whether the BeTr-GEESE grammar is more modular than GEESE-BT's grammar according to existing modularity metrics [61,53,9]. The BeTr-GEESE and GEESE-BT grammars have the same terminals with one exception: the *CanMove* constraint. The other terminals encode the basic actions, preconditions, postconditions, and constraints. The PPA structure encoded in BeTr-GEESE's grammar redundantly includes checks of constraints and postconditions, so 30 terminals appear on the right-hand-side (RHS) of productions in contrast to 24 for GEESE-BT. The PPA structure also produces "wider" trees, and this requires more non-terminals (20 to 11). BeTr-GEESE also has more productions and possible derivation trees, yielding a higher value of McCabe cyclomatic complexity (44 to 27). Finally, BeTr-GEESE averages fewer symbols on the RHS of productions (3.75 to 4.09) and produces programs that are more difficult to understand according to the Halstead effort

metric (283.62 to 132.94). Thus, on one hand, these *size modularity metrics* suggest that BeTr-GEESE's grammar is less modular than GEESE-BT.

On the other hand, *structural modularity metrics* suggest that the BeTr-GEESE grammar is more modular. Specifically, derivation trees for BeTr-GEESE are more treelike (as opposed to more closely representing graphs) according to the tree impurity metric (7.6% to 15.56%). Additionally, related functionalities (non-terminals) in BeTr-GEESE are more logically grouped together according to the *nslev* clustering metric (8 to 6) and according to the *normalized count of levels* metric (40% to 36.36%). Derivation trees produced by the BeTr-GEESE grammar have higher correlations between non-terminals, and these correlations theoretically make it easier to learn syntactically correct programs.

Existing modularity metrics are ambiguous: BeTr-GEESE derivation trees are complex but have some structural correlations that might enable learning. An alternative notion of modularity is how well the task can be divided into "chunks". Both foraging and nest maintenance are *divisible* and *additive* [65]. They are divisible because the multistep mission of finding, moving, and dropping objects can be broken into subtasks. They are additive because individual agents can independently contribute to the cumulative success of the group. Agents need not all be coordinating to succeed, and no single agent has to perform all subtasks. Thus, for example, an agent can (incorrectly) move an object to an undesirable location, and another agent can move it to a desired location.

BeTr-GEESE uses the divisibility and additive properties to produce modular behaviors wherein genes only express simple actions. Each codon in a gene represents a production number in the grammar, so the sequence of codons in the gene encodes the derivation tree as a sequence of productions used to produce a valid PPA-style BT. The size modularity metrics indicate an important property of the derivation trees: many productions are needed for each simple action in the tree. The limited gene size, no more than 100 codons per gene, inhibits including all of the productions necessary for a valid, multi-action phenotype.

BeTr-GEESE limited gene expressiveness works well with its fitness function to learn single action phenotypes. Indeed, the prior work reported that 98% of the programs created by BeTr-GEESE had only one of the basic actions from production (15), while successful programs produced by GEESE-BT included all four. BeTr-GEESE fitness function includes feedback from the PPA-style BT phenotype, inhibiting constraint violations, promoting the use of basic actions, and rewarding successful subtask completion. Thus, even though BeTr-GEESE's derivation trees can be complex, the BT provides feedback that inhibit trees that do not perform any subtasks and promote trees that can perform single subtasks. Thus, BeTr-GEESE is modular in the sense that subtask-specific trees receive rapid feedback, which works well with divisible and additive collective tasks.

5.2 Locality

BeTr-GEESE allows modular behaviors to be quickly learned, but agents still need to be able to perform all subtasks to successfully forage or maintain the nest. This subsection shows that lateral transfer allows modular behaviors to

be changed so that individual agents can find, carry, and drop objects, thus performing all necessary subtasks. The properties of lateral transfer is evaluated by describing the locality characteristics of the algorithm.

We begin with temporal locality. *Temporal locality* is the notion that a gene, and its associated phenotype, has a time window in which it is useful. A phenotype capable of performing a subtask must persist long enough for the subtask to be accomplished (e.g., explore until a site is found, travel from site to hub). But if an agent "holds onto" the gene too long then the agent cannot switch to the next needed subtask. Recall that after a BeTr-GEESE agent has received a sufficiently large number of genes through inter-agent interactions, it performs the standard genetic operators, selects the most fit, and then discards all but the most fit gene. Thus, how long a gene persists is determined by how frequently agents meet and exchange genes through lateral transfer. The lower bound on how long a gene persists is therefore controlled by: (i) how often agents are within close enough range to exchange genes (GS), (ii) how often agents with range exchange genetic information (IP), and (iii) the number of genes required before an agent applies the genetic operators (ST). How often the agent are in close range cannot be controlled directly, but IP and ST can be varied to control for how often agents meet each other.

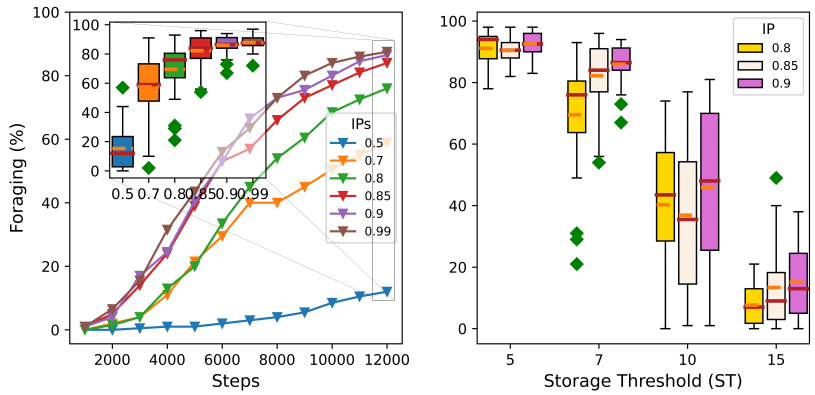

Fig. 2: a) Foraging (%) vs IP. ST=7. b) Relationship between IP, ST, and foraging(%).

Sixteen independent foraging runs were conducted for each value of IP and ST, and the experiment results are summarised with box and whisker plots in Fig. 2. Fig. 2a) shows that with a high willingness to transfer genes to other agents (IP>0.8), the agents can change genes rapidly, promoting evolution through lateral and vertical transfer. When IP<0.6, the agents persist with current behaviors too long, slowing down evolution. Fig. 2b) shows that when ST is high, which means that agents must meet many other agents before evolving,

agents are not able to change controllers quickly, and their performance goes down. Both figures show that too much persistence hinders evolution.

Fig. 3 shows that BeTr-GEESE agents exhibit *spatial locality*. The colors in the figure indicate the most fit gene when agents perform the genetic operators. The figure is constructed from the first 3000 evolution steps in one successful simulation, but all successful simulations exhibit similar *locality* patterns. The most fit gene selected by BeTr-GEESE agents depends on the location of the environment. For example, the figure shows a uniform distribution of blue explore-the-world behaviors. The figure also shows yellow clusters of carry-an-object behaviors, green clusters of drop-an-object behaviors, and linear clouds of move-towards and move-away behaviors. Clusters and clouds form around and between the hub and food sites, enabling agents to meet and evolve relevant controllers to solve particular sub-tasks at particular locations. The meeting locations enable lateral transfer of useful genes, which tend to localize around those regions of the world where specific subtasks are needed.

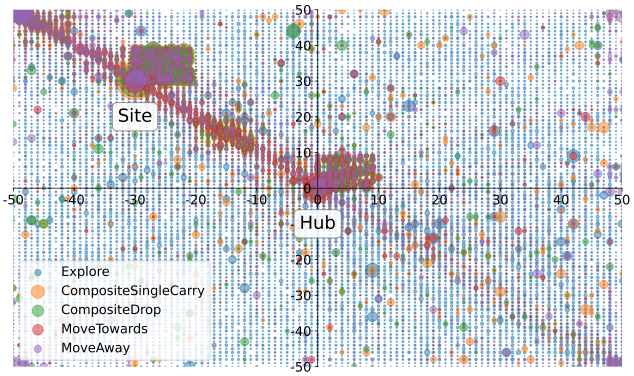

Fig. 3: Visualizing spatial locality: ST=7, IP=0.85, GS=10, 3000 evolution steps.

6 Conclusion

The BeTr-GEESE grammatical evolution algorithm resiliently responds to environment perturbations by enabling ongoing evolution. Rapid ongoing evolution is possible because the algorithm uses a limited gene size, thereby producing agent programs that are modular in the sense that they can only perform single subtasks. These modular, subtask-specific programs can be exchanged through lateral transfer to sequentially perform all required subtasks, which produces resilient performance in divisible and additive group tasks like foraging and nest maintenance. Switching between subtasks is enabled by lateral gene transfer, but the behaviors of successful groups must exhibit temporal locality, meaning that

an agent must persist in a behavior long enough to perform basic functions but also meaning that agents cannot persist too long or else evolution is too slow. Lateral transfer occurs at spatially local regions of the world where agents are likely to meet, allowing location-specific behaviors to be adopted by neighboring agents. Ongoing evolution through lateral transfer of simple modules exhibits resilience in the sense that agents can adapt to perturbations and still succeed in their tasks, but this adaptation might be inefficient. Future work should explore how modularity and locality in BeTr-GEESE can be applied not only to other GE algorithms and other types of multiagent problems, but also to designing efficient, fixed behaviors that produce resilient collective behaviors.

A PPA Grammar

⟨*root*⟩ ::= ⟨*sequence*⟩ | ⟨*selector*⟩ (1)

⟨*sequence*⟩ ::= [Sequence]⟨*ppa*⟩[/Sequence]| [Sequence]⟨*root*⟩⟨*root*⟩[/Sequence] [Sequence]⟨*sequence*⟩⟨*root*⟩[/Sequence] (2)

⟨*selector*⟩ ::= [Selector]⟨*ppa*⟩[/Selector]|[Selector]⟨*root*⟩⟨*root*⟩[/Selector] [Selector]⟨*selector*⟩⟨*root*⟩[/Selector] (3)

⟨*ppa*⟩ ::= [Selector]⟨*postconditions*⟩⟨*ppasequence*⟩[/Selector] (4)

⟨*postconditions*⟩ ::= ⟨*SuccessNode*⟩ | ⟨*ppa*⟩ |[Sequence]⟨*postcondition*⟩[/Sequence] (5)

⟨*postcondition*⟩ ::= ⟨*postcondition*⟩[PostCnd]⟨*postconditiont*⟩ [/PostCnd]|[PostCnd]⟨*postconditiont*⟩[/PostCnd] (6)

⟨*postconditiont*⟩ ::= NeighbourObjects_⟨*objects*⟩|NeighbourObjects_⟨*sobjects*⟩| IsCarrying_⟨*dobjects*⟩|NeighbourObjects_⟨*dobjects*⟩| DidAvoidedObj_⟨*sobjects*⟩|IsVisitedBefore_⟨*sobjects*⟩ (7)

⟨*ppasequence*⟩ ::= [Sequence]⟨*preconditions*⟩[Act]⟨*action*⟩ [/Act][/Sequence]| [Sequence]⟨*constraints*⟩[Act]⟨*action*⟩[/Act][/Sequence]|[Sequence] ⟨*preconditions*⟩⟨*constraints*⟩[Act]⟨*action*⟩[/Act][/Sequence] (8)

⟨*preconditions*⟩ ::= [Sequence]⟨*precondition*⟩[/Sequence] (9)

⟨*precondition*⟩ ::= ⟨*precondition*⟩[PreCnd]⟨*preconditiont*⟩ [/PreCnd]| [PreCnd]⟨*preconditiont*⟩[/PreCnd] (10)

⟨*preconditiont*⟩ ::= IsDropable_⟨*sobjects*⟩|NeighbourObjects_⟨*objects*⟩_inv| IsVisitedBefore_⟨*sobjects*⟩_inv|IsCarrying_⟨*dobjects*⟩_inv| IsVisitedBefore_⟨*sobjects*⟩| IsCarrying_⟨*dobjects*⟩|NeighbourObjects_⟨*objects*⟩ (11)

⟨*constraints*⟩ ::= [Sequence]⟨*constraint*⟩[/Sequence] (12)

⟨*constraint*⟩ ::= ⟨*constraint*⟩[Cnstr]⟨*constraintt*⟩[/Cnstr]|[Cnstr]⟨*constraintt*⟩ [/Cnstr] (13)

⟨*constraintt*⟩ ::= CanMove|IsCarryable_⟨*dobjects*⟩| IsDropable_⟨*sobjects*⟩ (14)

⟨*action*⟩ ::= MoveTowards_⟨*sobjects*⟩|Explore|CompositeSingleCarry_⟨*dobjects*⟩ |CompositeDrop_⟨*dobjects*⟩| MoveAway_⟨*sobjects*⟩ (15)

⟨*objects*⟩ ::= ⟨*sobjects*⟩|⟨*dobjects*⟩ (16)

⟨*sobjects*⟩ ::= Hub|Sites (17)

⟨*dobjects*⟩ ::= Food|Debris (18)

⟨*SuccessNode*⟩ :: = [PostCnd]DummyNode[/PostCnd] (19)

References

1. Bongard, J.: Morphological change in machines accelerates the evolution of robust behavior. Proceedings of the National Academy of Sciences **108**(4), 1234–1239 (2011)
2. Bongard, J.C.: Accelerating self-modeling in cooperative robot teams. IEEE Transactions on Evolutionary Computation **13**(2), 321–332 (2008)
3. Bredeche, N., Montanier, J.M., Liu, W., Winfield, A.F.: Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. Mathematical and Computer Modelling of Dynamical Systems **18**(1), 101–129 (2012)
4. Brooks, R.: A robust layered control system for a mobile robot. IEEE Jnl. on Robotics and Automation **2**(1), 14–23 (1986)
5. Canciani, F., Talamali, M.S., Marshall, J.A., Bose, T., Reina, A.: Keep calm and vote on: Swarm resiliency in collective decision making. In: Proceedings of Workshop Resilient Robot Teams of the 2019 IEEE International Conference on Robotics and Automation (ICRA 2019). p. 4 (2019)
6. Cheng, J., Cheng, W., Nagpal, R.: Robust and self-repairing formation control for swarms of mobile agents. In: AAAI. vol. 5 (2005)
7. Cliff, D., Husbands, P., Harvey, I., et al.: Evolving visually guided robots. From animals to animats **2**, 374–383 (1993)
8. Colledanchise, M., Ögren, P.: Behavior trees in robotics and al: An introduction (2018)
9. Črepinšek, M., Kosar, T., Mernik, M., Cervelle, J., Forax, R., Roussel, G.: On automata and language based grammar metrics. Computer Science and Information Systems (14), 309–329 (2010)
10. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR) **45**(3), 1–33 (2013)
11. Doncieux, S., Bredeche, N., Mouret, J.B., Eiben, A.E.G.: Evolutionary robotics: what, why, and where to. Frontiers in Robotics and AI **2**, 4 (2015)
12. Doncieux, S., Mouret, J.B., Bredeche, N., Padois, V.: Evolutionary robotics: Exploring new horizons. In: New horizons in evolutionary robotics, pp. 3–25. Springer (2011)
13. Doyle, J.C., Francis, B.A., Tannenbaum, A.R.: Feedback control theory. Courier Corporation (2013)
14. Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S.M., Christensen, A.L.: Evolution of collective behaviors for a real swarm of aquatic surface robots. PloS One **11**(3), e0151834 (2016)
15. Eiben, A.E., Haasdijk, E., Bredeche, N.: Embodied, on-line, on-board evolution for autonomous robotics (2010)
16. Engebråten, S.A., Moen, J., Yakimenko, O., Glette, K.: Evolving a repertoire of controllers for a multi-function swarm. In: International Conference on the Applications of Evolutionary Computation. pp. 734–749. Springer (2018)
17. Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., O'Neill, M.: Ponyge2: Grammatical evolution in python. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1194–1201 (2017)
18. Ferrante, E., Duéñez-Guzmán, E., Turgut, A.E., Wenseleers, T.: Geswarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In: Proc. of the 15th annual GECCO conference. pp. 17–24. ACM (2013)

19. Ferrante, E., Turgut, A.E., Duéñez-Guzmán, E., Dorigo, M., Wenseleers, T.: Evolution of self-organized task specialization in robot swarms. PLoS computational biology **11**(8), e1004273 (2015)
20. Goh, C.K., Tan, K.C.: Evolving the tradeoffs between pareto-optimality and robustness in multi-objective evolutionary algorithms. In: Evolutionary computation in dynamic and uncertain environments, pp. 457–478. Springer (2007)
21. Gordon, D.M.: Ant encounters. Princeton University Press (2010)
22. Gunderson, L.H.: Ecological resilience—in theory and application. Annual review of ecology and systematics **31**(1), 425–439 (2000)
23. Hall, J.P., Brockhurst, M.A., Harrison, E.: Sampling the mobile gene pool: innovation via horizontal gene transfer in bacteria. Philosophical Transactions of the Royal Society B: Biological Sciences **372**(1735), 20160424 (2017)
24. Holling, C.S.: Engineering resilience versus ecological resilience. Engineering within ecological constraints **31**(1996), 32 (1996)
25. Jablonka, E., Lamb, M.J.: Evolution in four dimensions, revised edition: Genetic, epigenetic, behavioral, and symbolic variation in the history of life. MIT press (2014)
26. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: European Conference on Artificial Life. pp. 704–720. Springer (1995)
27. Johnson, M., Brown, D.S.: Evolving and controlling perimeter, rendezvous, and foraging behaviors in a computation-free robot swarm. Tech. rep., Air Force Research Laboratory/RISC Rome United States (2016)
28. Kazil, J., Masad, D., Crooks, A.: Utilizing python for agent-based modeling: The mesa framework. In: Thomson, R., Bisgin, H., Dancy, C., Hyder, A., Hussain, M. (eds.) Social, Cultural, and Behavioral Modeling. pp. 308–317. Springer International Publishing, Cham (2020)
29. König, L., Mostaghim, S., Schmeck, H.: Decentralized evolution of robotic behavior using finite state machines. Intl. Jnl. of Intelligent Computing and Cybernetics **2**(4), 695–723 (2009)
30. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and computing **4**(2), 87–112 (1994)
31. Kriesel, D.M.M., Cheung, E., Sitti, M., Lipson, H.: Beanbag robotics: Robotic swarms with 1-DOF units. In: Intl. Conf. on Ant Colony Optimization and Swarm Intelligence. pp. 267–274. Springer (2008)
32. Kucking, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic design of robot swarms. In: ANTS 2018. IEEE (2018)
33. Kuckling, J., Van P., V., Birattari, M.: Automatic modular design of behavior trees for robot swarms with communication capabilites. In: EvoApplications. pp. 130–145 (2021)
34. Lampe, D.J., Witherspoon, D.J., Soto-Adames, F.N., Robertson, H.M.: Recent Horizontal Transfer of Mellifera Subfamily Mariner Transposons into Insect Lineages Representing Four Different Orders Shows that Selection Acts Only During Horizontal Transfer. Molecular Biology and Evolution **20**(4), 554–562 (04 2003)
35. Lane, N.: The vital question: Energy, evolution, and the origins of complex life. WW Norton & Company (2015)
36. Leaf, J., Adams, J.A.: Measuring resilience in collective robotic algorithms. In: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems. pp. 1666–1668 (2022)
37. Lee, W.P.: Evolving complex robot behaviors. Information Sciences **121**(1-2), 1–25 (1999)

38. Lewis, M.A., Fagg, A.H., Solidum, A.: Genetic programming approach to the construction of a neural network for control of a walking robot. In: Robotics and Automation, 1992. Proceedings., 1992 IEEE Intl. Conf. on. pp. 2618–2623. IEEE (1992)
39. Linksvayer, T.A., Janssen, M.A.: Traits underlying the capacity of ant colonies to adapt to disturbance and stress regimes. Systems Research and Behavioral Science: The Official Journal of the International Federation for Systems Research **26**(3), 315–329 (2009)
40. Mlot, N.J., Tovey, C.A., Hu, D.L.: Fire ants self-assemble into waterproof rafts to survive floods. Proceedings of the National Academy of Sciences **108**(19), 7669–7673 (2011)
41. Nelson, A.L., Barlow, G.J., Doitsidis, L.: Fitness functions in evolutionary robotics: A survey and analysis. Robotics and Autonomous Systems **57**(4), 345–370 (2009)
42. Neupane, A., Goodrich, M.A.: Designing emergent swarm behaviors using behavior trees and grammatical evolution. In: Proc. of the 18th AAMAS conference. pp. 2138–2140 (2019)
43. Neupane, A., Goodrich, M.A.: Learning swarm behaviors using grammatical evolution and behavior trees. In: IJCAI. pp. 513–520 (2019)
44. Neupane, A., Goodrich, M.A., Mercer, E.G.: Geese: grammatical evolution algorithm for evolution of swarm behaviors. In: Proc. of the 20th annual GECCO conference. pp. 999–1006 (2018)
45. Neupane, A., Goodrich, M.: Efficiently evolving swarm behaviors using grammatical evolution with ppa-style behavior trees. In: From Cells to Societies: Collective Learning across Scales (2022)
46. Nevai, A.L., Passino, K.M., Srinivasan, P.: Stability of choice in the honey bee nest-site selection process. Journal of theoretical biology **263**(1), 93–107 (2010)
47. Noirot, C., Darlington, J.P.: Termite nests: architecture, regulation and defence. In: Termites: evolution, sociality, symbioses, ecology, pp. 121–139. Springer (2000)
48. Ochman, H., Lawrence, J.G., Groisman, E.A.: Lateral gene transfer and the nature of bacterial innovation. nature **405**(6784), 299–304 (2000)
49. O'neill, M., Ryan, C., Keijzer, M., Cattolico, M.: Crossover in grammatical evolution. Genetic programming and evolvable machines **4**(1), 67–93 (2003)
50. Perez, R., Aron, S.: Adaptations to thermal stress in social insects: recent advances and future directions. Biological Reviews **95**(6), 1535–1553 (2020)
51. Petrovic, P.: Evolving behavior coordination for mobile robots using distributed finite-state automata. In: Frontiers in Evolutionary Robotics. InTech (2008)
52. Pintér-Bartha, A., Sobe, A., Elmenreich, W.: Towards the light—comparing evolved neural network controllers and finite state machine controllers. In: Proc. of the Tenth Workshop on Intelligent Solutions in Embedded Systems. pp. 83–87. IEEE (2012)
53. Power, J.F., Malloy, B.A.: A metrics suite for grammar-based software. Journal of Software Maintenance and Evolution: Research and Practice **16**(6), 405–426 (2004)
54. Quammen, D.: The tangled tree: A radical new history of life. Simon and Schuster (2018)
55. Reid, C.R., Lutz, M.J., Powell, S., Kao, A.B., Couzin, I.D., Garnier, S.: Army ants dynamically adjust living bridges in response to a cost–benefit trade-off. Proceedings of the National Academy of Sciences **112**(49), 15113–15118 (2015)
56. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014)

57. Samples, A.D.: Mache: No-loss trace compaction. In: Proceedings of the 1989 ACM SIGMETRICS international conference on Measurement and modeling of computer systems. pp. 89–97 (1989)
58. Schwander, T., Rosset, H., Chapuisat, M.: Division of labour and worker size polymorphism in ant colonies: the impact of social and genetic factors. Behavioral Ecology and Sociobiology **59**(2), 215–221 (2005)
59. Seeley, T.D.: The wisdom of the hive: the social physiology of honey bee colonies. Harvard University Press (2009)
60. Seeley, T.D.: Honeybee democracy. Princeton University Press (2010)
61. Simon, H.A.: The Sciences of the Artificial, reissue of the third edition with a new introduction by John Laird. MIT press (2019)
62. Singh, S., Lewis, R.L., Barto, A.G., Sorg, J.: Intrinsically motivated reinforcement learning: An evolutionary perspective. IEEE Transactions on Autonomous Mental Development **2**(2), 70–82 (2010)
63. Sorenson, E.S., Flanagan, J.K.: Evaluating synthetic trace models using locality surfaces. In: Proceedings of the IEEE International Workshop on Workload Characterization. pp. 23–33 (2002)
64. Soule, T.: Resilient individuals improve evolutionary search. Artificial Life **12**(1), 17–34 (2006)
65. Steiner, D.I.: Group process and productivity. Academic Press (1972)
66. Stonier, D., Staniaszek, M.: Behavior Tree implementation in Python (12 2021), https://github.com/splintered-reality/py_trees/
67. Sumpter, D., Pratt, S.: A modelling framework for understanding social insect foraging. Behavioral Ecology and Sociobiology **53**(3), 131–144 (2003)
68. Sumpter, D.J.: Collective animal behavior. In: Collective Animal Behavior. Princeton University Press (2010)
69. Swafford, J.M., O'Neill, M.: An examination on the modularity of grammars in grammatical evolutionary design. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2010)
70. Toffolo, A., Benini, E.: Genetic diversity as an objective in multi-objective evolutionary algorithms. Evolutionary computation **11**(2), 151–167 (2003)
71. Toth, A., Robinson, G.: Evo-devo and the evolution of social behavior: brain gene expression analyses in social insects. In: Cold Spring Harbor symposia on quantitative biology. vol. 74, pp. 419–426. Cold Spring Harbor Laboratory Press (2009)
72. Trianni, V., Groß, R., Labella, T.H., Şahin, E., Dorigo, M.: Evolving aggregation behaviors in a swarm of robots. In: European Conference on Artificial Life. pp. 865–874. Springer (2003)
73. Ursem, R.K.: Diversity-guided evolutionary algorithms. In: International Conference on Parallel Problem Solving from Nature. pp. 462–471. Springer (2002)
74. Varughese, J.C., Thenius, R., Schmickl, T., Wotawa, F.: Quantification and analysis of the resilience of two swarm intelligent algorithms. In: GCAI. pp. 148–161 (2017)
75. Vistbakka, I., Troubitsyna, E.: Modelling autonomous resilient multi-robotic systems. In: International Workshop on Software Engineering for Resilient Systems. pp. 29–45. Springer (2019)
76. Wagner, G.P., Altenberg, L.: Perspective: complex adaptations and the evolution of evolvability. Evolution **50**(3), 967–976 (1996)
77. Wang, J.X., Hughes, E., Fernando, C., Czarnecki, W.M., Duéñez-Guzmán, E.A., Leibo, J.Z.: Evolving intrinsic motivations for altruistic behavior. arXiv preprint arXiv:1811.05931 (2018)

78. Yamashita, Y., Tani, J.: Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. PLoS computational biology **4**(11), e1000220 (2008)
79. Zahadat, P., Hamann, H., Schmickl, T.: Evolving diverse collective behaviors independent of swarm density. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1245–1246 (2015)