Swarm Intelligence
https://doi.org/10.1007/s11721-024-00243-w

Resilient swarm behaviors via online evolution and behavior fusion

Aadesh Neupane[1] · **Michael A. Goodrich**[1]

Received: 29 November 2022 / Accepted: 2 August 2024

Abstract
Grammatical evolution can be used to learn bio-inspired solutions to many distributed multiagent tasks, but the programs learned by the agents often need to be 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. The BeTr-GEESE grammatical evolution algorithm presented in this paper creates agents that use both vertical and lateral gene transfer to 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 online 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 shortly (temporal locality). Finally, the paper modifies BeTr-GEESE to perform behavior fusion across multiple modular behaviors using activator and repressed conditions so that a fixed (non-evolving) population of heterogeneous agents is resilient to perturbations.

Keywords Grammatical evolution · Behavior trees · Swarm · Resilience · Modularity · 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 (Gordon, 2010; Seeley, 2010). Such

✉ Aadesh Neupane
adeshnpn@byu.edu

Michael A. Goodrich
mike@cs.byu.edu

[1] Department of Computer Science, Brigham Young University, Provo, UT, USA

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 (Linksvayer & Janssen, 2009; Perez & Aron, 2020), local interaction (Gordon, 2010), task switching (Seeley, 2009), lateral transfer (Lampe et al., 2003), and modularity (Toth & Robinson, 2009).

This paper adopts the characterization of a resilient agent as formulated by Leaf et al. (2023): "a [resilient] agent [is one] that can accomplish its goal in the presence of perturbations". Importantly, the concept of resilience is contingent upon the pursued goal and the specific perturbation. Given the inherent difficulty in quantifying resilience based solely on Leaf's characterization, we propose two resilience metrics: (a) power and (b) efficiency. The power resilience metric is defined as the maximum attainable success probability before reaching a predefined time threshold. Conversely, the efficiency resilience metric is defined by the elapsed time for an agent to fulfill a predetermined performance threshold. These metrics serve to provide quantitative measures of an agent's resilience, facilitating a deeper understanding of its adaptive capabilities in the face of uncertainties.

Evolutionary approaches are powerful tools for learning bio-inspired swarm behaviors (Doncieux et al., 2011; Eiben et al., 2010; Zahadat et al., 2015). 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 (Ferrante et al., 2013; Neupane et al., 2018; Neupane & Goodrich, 2019b), and most demonstrations first evolve solutions and then deploy those learned solutions as *fixed strategies*. Detailed experimental investigations into the performance of fixed strategies under perturbations are not consistently conducted, rendering it challenging to ascertain their resilience to specific perturbations. In particular, concerning foraging and nest-maintenance experiments delineated in Neupane et al.'s works (Neupane et al., 2018; Neupane & Goodrich, 2019b), fixed strategies frequently exhibit non-resilient behaviors. The primary objective of this paper is to discern potentially generalizable attributes capable of empowering grammatical evolution to yield resilient swarm behaviors.

Biology sometimes uses rapid adaptation to overcome performance degradation of fixed strategies. Rapid adaptation mechanisms include stress-induced mutation, lateral gene transfer, and continuous evolution in bacteria (Hall et al., 2017). 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 (Koza, 1994; Wagner & Altenberg, 1996). Unfortunately, online 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 (Ferrante et al., 2015). 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 contribute to the collective task. Carefully constructed fitness functions (e.g., intrinsic and extrinsic motivators (Neupane et al., 2018; Singh et al., 2010; Wang et al., 2018)) help solve the second problem but are unlikely to succeed in the presence of perturbations since new fitness functions are needed for each perturbation type.

The BeTr-GEESE grammatical evolution algorithm presented in this paper exhibits a curious phenomenon: BeTr-GEESE agents successfully perform collective foraging and

[1] Resilient task performance differs from ecological resilience in which population sizes show resilience to variations (Gunderson, 2000) and from stability-based definitions of resilience in which some property of a collective remains in a locally stable region (Holling, 1996).

nest maintenance *while they are evolving*, but the collective performs poorly when learning stops (Neupane & Goodrich, 2022a). 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 (Neupane & Goodrich, 2022a). Time-multiplexing is a form of *lateral gene transfer* (Ochman et al., 2000) in which genes transfer between organisms, in contrast to *vertical parent-to-child gene transfer*.

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. Rapid learning through lateral gene transfer is first demonstrated and then explained using the concepts of *modularity* and *locality*. *Modularity* in evolutionary algorithms means that geneotype-to-phenotype mappings tend to associate specific phenotypic characteristics with specific genes, in contrast to "general purpose" genes that exhibit complex phenotypes (Wagner & Altenberg, 1996). 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 of trace compression and cache design in computer architecture (Samples, 1989; Sorenson & Flanagan, 2002) 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.

We demonstrate that the rapid learning enables resilience by applying various types of perturbations during evolution and then measuring resulting performance. However, BeTr-GEESE requires online evolution to be successful. Once evolution stops, agents perform poorly. It is desirable to create an algorithm that exploits the rapid learning of modular behaviors to create fixed agents that are collectively resilient.

This paper approaches the problem of coordinating learned modular behaviors by taking inspiration from how regulatory proteins govern which gene is expressed in an organism. The results are agents that use a bio-inspired gene expression mechanism that exhibits resilient behaviors. The bio-inspired regulatory approach is modeled as a *behavior fusion problem* from the robotics literature (Goodridge & Luo, 1994). An extension of the BeTr-GEESE algorithm called Multi-GEESE is introduced that combines behaviors using activators and repressor BT nodes. These changes makes it possible for fixed strategies to be resilient after the evolution stops, thereby ensuring sustained adaptability in dynamic environments. Experimental results suggest that the fixed population of heterogeneous agents obtained from Multi-GEESE learns the conditions to activate or repress specific behaviors based on when and where the agents are in the environment.

The paper is organized as follows. Section 2 reviews related work on grammatical evolution, modular agent design, and biological inspiration for modular agent design. Section 3, which is adapted from Neupane and Goodrich (Neupane & Goodrich, 2022a), describes the BeTr-GEESE grammatical evolution algorithm and compares its performance to similar swarm-based grammatical evolution algorithms. Section 4, which is adapted from a different paper by Neupane and Goodrich (Neupane & Goodrich, 2022b), then analyzes the

[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 (Steiner, 1972).

modularity and resilience properties of the algorithm. A new algorithm Multi-GEESE is then presented that evolves resilient fixed agent strategies using a modified grammar capable of expressing different phenotypes.

2 Related work

The ability of ant, bee, and termite colonies to solve complicated decision processes with partial information has motivated researchers to mimic their behaviors in artificial agents and robots (Reid et al., 2015; Mlot et al., 2011; Noirot & Darlington, 2000; Rubenstein et al., 2014; Cheng et al., 2005). A somewhat cumbersome way to create bio-inspired swarms is to create mathematical models from carefully collected biological data (Nevai et al., 2010; Sumpter & Pratt, 2003). A complementary approach is to use evolutionary robotics techniques to evolve controllers (Doncieux et al., 2015; Kriesel et al., 2008), which often works because simple individual agents can produce complex swarm behaviors. Evolutionary robotics requires the designer to choose and aggregate from various controllers (e.g., state-machines (Ferrante et al., 2013; Brooks, 1986; Petrovic, 2008; Pintér-Bartha et al., 2012; König et al., 2009; Neupane et al., 2018), neural networks (Cliff et al., 1993; Lewis et al., 1992; Duarte et al., 2016; Trianni et al., 2003), behavior trees (Kucking et al., 2018; Kuckling et al., 2021)), evolutionary algorithms (e.g., genetic evolution (Kriesel et al., 2008; Brooks, 1986), grammatical evolution (Ferrante et al., 2013; Neupane & Goodrich, 2019a)), and fitness functions (Nelson et al., 2009).

Despite favourable design choices, the performance of evolved behaviors often degrades when tested with real robots or in presence of uncertainties (Jakobi et al., 1995). Fortunately, there is prior work on how to evolve robust behaviors. Bongard (2011) showed that morphological change during evolution accelerates the discovery of robust behaviors. They concluded that environment fluctuations, directional selection, and stabilization pressure favors the evolution of robustness.

Robustness and resilience can arise as a result of modularity (Wagner & Altenberg, 1996). Modularity can enhance an organism's capacity to evolve resilient behaviors because (a) the organization of biological system into modules may permit changes inside one module without perturbing other modules and (b) modules can be combined and reused to create new biological function. Yamashita and Tani (2008) showed that modules organized into a functional hierarchy promotes evolving complex behaviors.

Evolving resilient behaviors by evolving structural and functional modularity can be very slow if only traditional genetic operators (selection, crossover, and mutation) are used. Recent papers show that endosymbiosis or horizontal transfer are often observed when organisms are stressed (Jablonka et al., 2014; Lane, 2015; Quammen, 2018). Evidence suggests that horizontal transfer contributes to rapid evolution, presumably because horizontal transfer might be more computationally efficient than evolving complex controllers (Lee, 1999; Engebråten et al., 2018). Bongard (2008) demonstrated that robots enabled with lateral transfer of models perform better than robots that rely on shared modelling using an estimation-exploration co-evolutionary algorithm.

Evolving controllers for agents in swarm systems can be sped up by performing so-called hybrid computation (Johnson & Brown, 2016; Jones et al., 2019). Importantly, controller's higher fitness does not necessarily imply higher robustness. For example, Soule (2006) showed that the controller with highest fitness is not necessarily resilient, and genetic changes

easily disrupt the fittest individuals. Addressing this issue, Bredeche et al. (2012) developed an environment-driven distributed evolutionary algorithm MEDEA to evolved efficient and robust controllers. MEDEA operates solely on environmental selection principles and facilitates agent mobility within the environment. Genomes conducive to active agent movement are proliferated at a significantly higher rate compared to those promoting static behavior. Haasdijk et al. (2013) augmented MEDEA by introducing task credits within the MONEE algorithm framework. Task credits encapsulate an agent's task performance, and this information is leveraged during the evolutionary selection phase to guide the evolutionary process.

Agents that use multiple modular controllers can be designed using top-down or bottom-up approaches. Since this paper addresses bio-inspired solutions, it emphasizes bottom-up approaches. For example, the subsumption architecture (Brooks, 1986) is a widely cited example for how complex behaviors can emerge by decomposing complex behaviors into layered sub-behaviors. Building on this decomposition philosophy, behavior fusion (Goodridge & Luo, 1994; Li & Feng, 1994) is a bottom-up approach that learns a weight or priority for each modular behavior. Behavior fusion is similar to how bacteria use regulatory proteins as repressors and activators to express a particular gene (Browning & Busby, 2004). Similarly, the concept of hormones (Shen et al., 2000; Yim et al., 2007) has been shown to dynamically group modules. Also similar, phenotypic plasticity, wherein an organism can manifest varied phenotypes in response to distinct environmental conditions, has garnered significant attention in various biological studies. This adaptive mechanism has been explored in ants (Kelly et al., 2011), bees (Holway et al., 2002), and plants (Goulson et al., 2015). In the domain of swarm robotics, Hunt (2020) elucidated the potential utility of phenotypic plasticity for fostering resilience. Additionally, Miras et al. (2020) empirically demonstrated the efficacy of environmental regulation in enhancing adaptation. Notably, Miras et al. (2020) introduced a novel encoding method wherein a genotype encodes multiple phenotypes, exemplifying another facet of phenotypic plasticity.

This paper is interested in resilient swarm behaviors. Theoretical definitions exist for resilience metrics in multi-agent and swarm systems, but only a few experimental studies have been done using those metrics (Vistbakka & Troubitsyna, 2019; Leaf & Adams, 2022). Varughese et al. (2017) showed that a swarm system had high resilience to agent-to-agent communication failures, Canciani et al. (2019) showed that swarm agents performing the best-of-N task are not resilient to denial of service perturbation. Prasetyo et al. (2018, 2019) showed that the presence of the stubborn agents is enough to achieve resilient performance and adaptability when the quality of the site changes dynamically in best-of-N problem.

3 BeTr-GEESE algorithm

This section first presents the BeTr-GEESE grammatical evolution algorithm and then compares the algorithm to similar grammatical evolution algorithms for swarm-based agents. The results show that BeTr-GEESE agents rapidly learn how to forage or perform nest maintenance. The rapid learning is mainly due to (a) the post-condition, pre-condition, action structure, and (b) BT feedback.

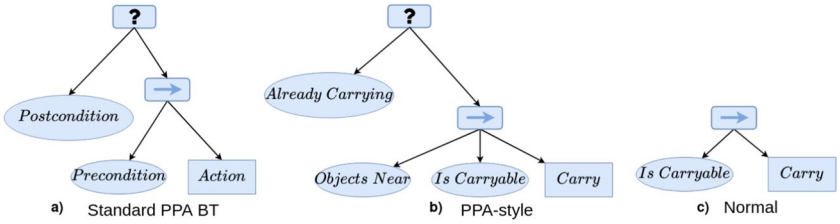

Fig. 1 BTs for primitive behaviors. **a** General PPA-style BT. **b** The *CompositeSingleCarry* primitive behavior implemented using PPA-style structure. **c** The *CompositeSingleCarry* implemented using a conventional tree structure

3.1 BeTr-GEESE description

BeTr-GEESE agents use *sense-act-update* evolution steps to learn individual behaviors or "programs" from a 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 enumerated grammar productions are used to produce the agent controller phenotype. The BeTr-GEESE grammar shown below implements a behavior tree (BT) that has a post-condition, pre-condition, action (PPA) structure (Colledanchise & Ögren, 2018), 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 (15)). The names in productions (7, 11, 15) are self-explanatory given the descriptions of foraging and nest maintenance tasks in Sect. 3.3. Note that both successful foraging and nest maintenance require each basic action in production 15. The set of productions (1–10) is intended to be general, generating an arbitrary BT (Behavior Tree). By contrast, productions (11–19) specify task-specific post-conditions, pre-conditions, and actions. Notably, in production 18, the presence of only "Food" indicates that the grammar is tailored specifically for foraging tasks; consequently, if "Food" is absent, the grammar is geared towards maintenance tasks. Each BT returns a success, failure, or running status that encodes how successful the program has been in satisfying a post-condition.

The PPA structure is integral to the success of BeTr-GEESE. Figure 1a illustrates a standard PPA BT, which is defined as a BT where the selector root node ("?") ensures that the *action* node on the right branch of the sequence control node (→) is not carried out if the *post-condition* node is already satisfied (Colledanchise & Ögren, 2018). Figure 1c illustrates a non-PPA style behavior tree produced by GEESE-BT, which requires a pre-condition (*IsCarryable*) to be satisfied `and` the task (*Carry*) successfully performed. The `and` operator is implemented as a sequence BT node (→). BeTr -GEESE uses the PPA structure in Fig. 1a for implementing all primitive behaviors (PB) in production 15. For illustration, *CompositeSingleCarry* PB is shown in Fig. 1b. The root selector node (represented as the "?") checks the post-condition (left branch, *AlreadyCarrying*) and calls the sequence node in the right child only if the post-condition is not met. The right child checks pre-conditions

and implements the action. The post-condition ensures that the planner does not re-execute when the goal has already been met.

$$\langle root \rangle ::= \langle sequence \rangle \mid \langle selector \rangle \tag{1}$$

$$\langle sequence \rangle ::= [\text{Sequence}]\langle ppa \rangle[/\text{Sequence}] \mid [\text{Sequence}]\langle root \rangle\langle root \rangle[/\text{Sequence}] \\ [\text{Sequence}]\langle sequence \rangle\langle root \rangle[/\text{Sequence}] \tag{2}$$

$$\langle selector \rangle ::= [\text{Selector}]\langle ppa \rangle[/\text{Selector}] \mid [\text{Selector}]\langle root \rangle\langle root \rangle[/\text{Selector}] \\ [\text{Selector}]\langle selector \rangle\langle root \rangle[/\text{Selector}] \tag{3}$$

$$\langle ppa \rangle ::= [\text{Selector}]\langle postconditions \rangle\langle ppasequence \rangle[/\text{Selector}] \tag{4}$$

$$\langle postconditions \rangle ::= \langle SuccessNode \rangle \mid \langle ppa \rangle \mid [\text{Sequence}]\langle postcondition \rangle[/\text{Sequence}] \tag{5}$$

$$\langle postcondition \rangle ::= \langle postcondition \rangle[\text{PostCnd}]\langle postconditiont \rangle \\ [/\text{PostCnd}] \mid [\text{PostCnd}]\langle postconditiont \rangle[/\text{PostCnd}] \tag{6}$$

$$\langle postconditiont \rangle ::= \text{NeighbourObjects\_}\langle objects \rangle \mid \text{NeighbourObjects\_}\langle sobjects \rangle \mid \\ \text{IsCarrying\_}\langle dobjects \rangle \mid \text{NeighbourObjects\_}\langle dobjects \rangle \mid \\ \text{DidAvoidedObj\_}\langle sobjects \rangle \mid \text{IsVisitedBefore\_}\langle sobjects \rangle \tag{7}$$

$$\langle ppasequence \rangle ::= [\text{Sequence}]\langle preconditions \rangle[\text{Act}]\langle action \rangle[/\text{Act}][/\text{Sequence}] \mid \\ [\text{Sequence}]\langle constraints \rangle[\text{Act}]\langle action \rangle[/\text{Act}][/\text{Sequence}] \mid [\text{Sequence}] \\ \langle preconditions \rangle\langle constraints \rangle[\text{Act}]\langle action \rangle[/\text{Act}][/\text{Sequence}] \tag{8}$$

$$\langle preconditions \rangle ::= [\text{Sequence}]\langle precondition \rangle[/\text{Sequence}] \tag{9}$$

$$\langle precondition \rangle ::= \langle precondition \rangle[\text{PreCnd}]\langle preconditiont \rangle[/\text{PreCnd}] \mid \\ [\text{PreCnd}]\langle preconditiont \rangle[/\text{PreCnd}] \tag{10}$$

$$\langle preconditiont \rangle ::= \text{IsDropable\_}\langle sobjects \rangle \mid \text{NeighbourObjects\_}\langle objects \rangle\text{\_inv} \mid \\ \text{IsVisitedBefore\_}\langle sobjects \rangle\text{\_inv} \mid \text{IsCarrying\_}\langle dobjects \rangle\text{\_inv} \mid \\ \text{IsVisitedBefore\_}\langle sobjects \rangle \mid \text{IsCarrying\_}\langle dobjects \rangle \mid \text{NeighbourObjects\_}\langle objects \rangle \tag{11}$$

$$\langle constraints \rangle ::= [\text{Sequence}]\langle constraint \rangle[/\text{Sequence}] \tag{12}$$

$$\langle constraint \rangle ::= \langle constraint \rangle[\text{Cnstr}]\langle constraintt \rangle[/\text{Cnstr}] \mid [\text{Cnstr}]\langle constraintt \rangle \\ [/\text{Cnstr}] \tag{13}$$

$$\langle constraintt \rangle ::= \text{CanMove} \mid \text{IsCarryable\_}\langle dobjects \rangle \mid \text{IsDropable\_}\langle sobjects \rangle \tag{14}$$

$$\langle action \rangle ::= \text{MoveTowards\_}\langle sobjects \rangle \mid \text{Explore} \mid \text{CompositeSingleCarry\_}\langle dobjects \rangle \mid \text{CompositeDrop\_}\langle dobjects \rangle \mid \text{MoveAway\_}\langle sobjects \rangle \tag{15}$$

$$\langle objects \rangle ::= \langle sobjects \rangle \mid \langle dobjects \rangle \tag{16}$$

$$\langle sobjects \rangle ::= \text{Hub} \mid \text{Sites} \tag{17}$$

$$\langle dobjects \rangle ::= \text{Food} \mid \text{Debris} \tag{18}$$

$$\langle SuccessNode \rangle ::= [\text{PostCnd}]\text{DummyNode}[/\text{PostCnd}] \tag{19}$$

The phenotype is a program determined by the BT, which determines the agent's behavior in the environment. During evolution, BeTr -GEESE rewards those agent behaviors that promote genetic diversity and world exploration, observe or accomplish subtasks, or avoid constraint violations. 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. Phenotype fitness, defined in Eq. (1) with $A_0 = D$, is evaluated over time, which is necessary because there is delay between acting and receiving a reward,

$$A_t = 0.1(A_{t-1}) + (E_t + B_t). \tag{1}$$

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 (Ursem, 2002; Schwander et al., 2005; Toffolo & Benini, 2003). The diversity function (type I) takes a BT as input and extracts all the nodes from the tree. Recall that the BNF grammar produces only (a) "Sequence" and "Selector" BT controls and (b) primitive and higher-level agent behaviors. The extracted nodes from the tree are stored in a dictionary structure as control nodes and behavior nodes; the number of such nodes is also stored. D is defined as the total number of unique behavior nodes divided by the total behaviors defined in the grammar. *Exploration fitness* (type II) (Črepinšek et al., 2013), E, promotes visiting new locations, and is defined as the number of unique world locations visited by the agent. GEESE-BT implements ad hoc fitness functions. Prospective fitness (type III) prioritizes "intrinsic" actions such as picking up, carrying, or dropping objects. On the other hand, task-specific fitness (type IV) employs hand-tailored fitness functions intended to incentivize collective actions, including maximizing the total food gathered at the hub and relocating debris away from the hub.

Recall that BeTr-GEESE agents evolve a PPA-style BT from the grammar, allowing easy evaluation of whether some pre-condition, post-condition, or action nodes succeeded or failed. The quality of the agent's controller is therefore a function of the status of such nodes. *BT feedback fitness*, B, is defined as the sum of post-condition, constraint, and BT root node rewards. When a post-condition node status is success, a subjectively chosen reward of $+1$ indicates that some potentially useful condition in the world holds. A subjectively chosen reward of -2 occurs when a constraint node status is failure. A subjectively chosen reward of $+1$ is returned when the root selector node status succeeds, indicating that some sub-task has been accomplished somewhere in the BT. PPA-style BT feedback B specifically does three things: (a) inhibits constraint violations, (b) promotes the use of basic actions, and (c) rewards successful sub-task completion.

Table 1 Evolution parameters used to produce Fig. 3

Parameters	BeTr-GEESE
Parent-selection	Fitness + Truncation
Mutation probability	0.01
Crossover probability	0.9
Crossover	variable_onepoint (O'neill et al., 2003; Fenton et al., 2017)
Maximum depth of derivation tree	10 levels
Agent sense range (GS)	10 environment units
No. of agents	100
Storage threshold (ST)	7
Interaction probability (IP)	0.85
Genome-selection	Diversity

3.2 Experiment design

Two GE algorithms, BeTr-GEESE and GEESE-BT are now compared. The two algorithms use the same genetic operators, the same parameter values, the same form of lateral transfer between agents, the same basic actions, and the same pre-conditions and post-conditions. Most of the common genetic parameters (selection, mutation crossover, depth) in Table 1 were identical to other GEESE algorithm variants to ensure fair comparisons. Other parameters such as agent sense range, storage threshold, interaction probability were subjectively chosen based on the results of preliminary hyper-parameters search experiments. Maximum tree depth is a practical parameter that limits the effect of recursive dependencies in the grammar. There is reason to believe that the parameter choices in Table 1 are somewhat generalizable because (a) they are similar to other papers (Fenton et al., 2017; Ferrante et al., 2013) in which swarm behaviors are evolved and (b) locality-based evolution is likely to be compatible with many divisible and additive tasks (Steiner, 1972).

The algorithms in the experiment differ 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 (Colledanchise & Ögren, 2018). Third, BeTr-GEESE used the fitness function described above, whereas GEESE-BT used a combination of diversity fitness (D), exploration fitness (E), and ad hoc motivators (types III and IV in the Fig. 3). Experiments ran on a machine with an i9 CPU, 64 GB RAM running 16 parallel threads. *PonyGE2* (Fenton et al., 2017) was used to implement GEESE-BT and BeTr-GEESE. BT controllers were created using *py_trees* (Stonier & Staniaszek, 2021), and the swarm simulation environment was created using *Mesa* (Kazil et al., 2020).

Figure 2 illustrates a simulation environment made up of 100×100 square cells wherein sites, hubs, and obstacles are randomly placed. Cells are grouped in 10×10 square units called *grids*, delineated by the darker lines. The dark pink circle surrounding the agent indicates its sensing radius. Agents are point objects delineated by (x, y) coordinate tuples, possessing the properties of speed, direction, and sensing radius. A grid environment is used to streamline nearest-neighbor computations, whereby each agent assesses the presence of objects or fellow agents within its 3-unit sensing radius. With *grid size* parameter GS = 10, computational efficiency is attained, as an agent

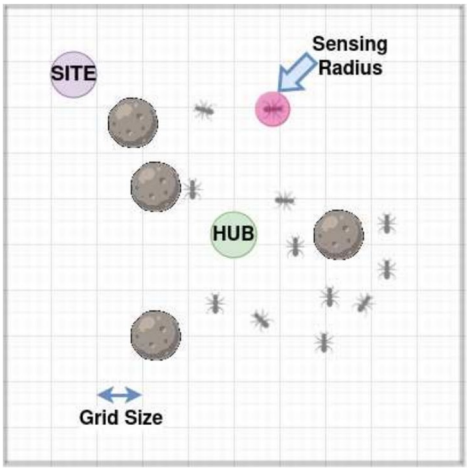

Fig. 2 The 100×100 grid world simulation environment with hub at the center, site in the upper left, and randomly placed obstacles. Agents are represented by grey ants

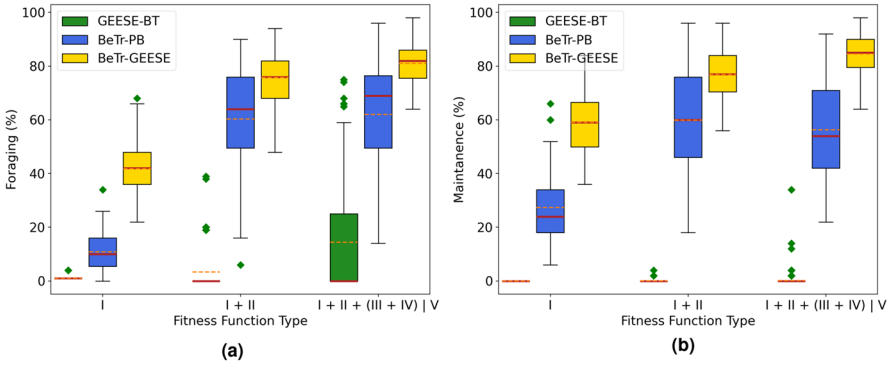

Fig. 3 Task (foraging/maintenance) percentage (%)

confines its distance computations to objects residing within the same grid. Experiments use a population of 100 agents, each advancing 2 units per time step along their designated heading angle. Each experiment starts with agents situated at a hub, which is positioned at the origin. Agents used the controllers mapped from the genotype-to-phenotype process to interact with the environment. Then the agents use the sense-act-update steps described in Sect. 3 to learn fit controllers until the evolution time limit of $T = 12{,}000$ steps. The agents sense any object within their sensing radius, which is the same as GS, i.e., they can sense objects within their same grid cell. At any instance, the agent has access to its 2D location and the location of other objects in the same grid cell. They also have access to the location of other objects (sites, obstacles, and others) once they discover them during exploration, and this information is updated on the BT blackboard.

3.3 Learning efficiency

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. 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* is defined as the ratio of the number of successful evolution trials to the total number of trials. BeTr-GEESE's average success rate was 75%, eight times higher than GEESE-BT even when GEESE-BT used the task-specific fitness functions. The success rate can be thought of as a measure of *learning efficiency* because the successes occur while the agents are evolving. High success rates indicate that agents solve the task while they are evolving their behaviors.

measured by the percentage of food/debris transported to/from the hub with respect to variations in primitive behaviors, grammar, and fitness function. **Di**versity is type (I), **E**xploration is type (II), Pr**o**spective is type (III), **T**ask-specific is type (IV), and **BT** feedback is type (V) fitness function in the x-axis. The different color boxes represent GEESE algorithm variants. The *x*-axes represent different fitness function combinations. BeTr-GEESE (yellow boxes), which produces PPA-style BTs, performs better across all different fitness function combinations

The details of Fig. 3 are available in the prior paper (Neupane & Goodrich, 2022a) and are omitted for space, but the key aspects of the figure are summarized here. The green results are for the GEESE-BT algorithm. This algorithm uses a BNF grammar that is suitable for creating behavior trees, but does not use the PPA-style behavior trees used by BeTr-GEESE. Various combinations of ad hoc fitness functions, specifically tuned to the spatially divisible tasks used in the experiment, are labeled with III and IV. The blue results are produced when the root behaviors of the agents, called primitive behaviors, are organized using PPA structures but the rest of the grammar allows other types of behavior structures. The yellow results are produced when the grammar above is used and when ad hoc fitness functions are replaced by behavior tree fitness in Eq. (1).

The main lessons from the figure are that (a) structuring behavior trees to use PPA structures rapidly evolves behaviors that successfully perform the task while learning, and (b) using behavior tree success or failure status as part of the fitness function enables efficient evolution without resorting to ad hoc fitness functions. Many multiagent tasks are neither divisible nor additive (Steiner, 1972). Such tasks typically require coordination among agents, and such coordination requires different grammatical primitives. The observations in this paper are limited to divisible and additive tasks and are not likely to apply to other task types.

3.4 Discussion

BeTr-GEESE performed better GEESE-BT because the PPA-style BT and the BT feedback. This subsection discusses (a) scalability and (b) limitations of the BNF grammar and the BT feedback function.

Good swarm evolution algorithms should scale well when the agent population size increases. Experiments were performed with different population sizes (n) from the set $n \in \{50, 100, 200, 300, 400, 500\}$. Increasing population size increased the run-time

Table 2 Size and structure metrics comparing morphological modularity of BNF grammar between GEESE-BT and BeTr-GEESE algorithms

Metric	GEESE-BT	BeTr-GEESE
(a) *Size modularity metrics*		
term	24	30
var	11	20
mcc	27	44
avs	4.09	3.75
hal	132.94	283.62
(b) *Structure modularity metrics*		
timp	15.56%	7.60%
clev	36.36%	40%
nslev	4	8
dep	6	6

substantially. Notably, success rates stayed relatively high, and the effects of various combinations of grammar type and fitness functions were consistent across all population sizes. In other words, population size affects the time the algorithm requires but not the effects of the critical algorithm parameters. Based on the results of the initial experiments, all results in the paper use a population of 100 agents because this population size allowed many experiments to be performed relatively quickly.

The structure of the BNF grammar is integral to a grammatical evolution algorithm like BeTr-GEESE. The result demonstrate that the PPA-style structure is useful, but the question remains how well the primitive behaviors apply to other multiagent problems. Recall that the BeTr-GEESE grammar includes five primitive agent behaviors in production that are combined with pre-conditions and post-conditions. For many swarm tasks, more and new primitive behaviors will likely be needed, requiring more evolution time and larger codons to allow the algorithm to learn complex controllers. Fortunately, the feedback provided by the PPA-style structures is simple and likely to scale to a larger set of primitive behaviors. The high-level grammar designs are also likely generic enough to accommodate new primitive behaviors. Scalability issues associated with new behaviors are essential for future work.

BT fitness is a simple weighted sum of various node statuses. During execution, a specific node could fail in one-time step and return success in the next step because node success or failure depends on context. The effects of context is addressed in Sect. 4, but one lesson from that section is that the local context faced by an agent is likely to persist for some period of time. Thus, only nodes that persistently fail will limit what types of behaviors can be evolved. Persistently failing nodes limit how much of the fitness landscape can be explored when behaviors are evolved. The result is a conservative exploration of the fitness landscape, where the term "conservative" means that explorations is biased against failures.

4 What enables rapid learning?

The section discusses why BeTr-GEESE agents learn so quickly, which sheds light on the how ongoing lateral transfer enables resilience.

4.1 Modularity

We first compare the modularity of the BeTr-GEESE and GEESE-BT's grammars using the modularity metrics from (Simon, 2019; Power & Malloy, 2004; Črepinšek et al., 2010). It is important to note that it is the *modularity of the grammars* being compared in this section and *not the modularity of the learned behavior trees*. Table 2 shows that existing modularity metrics are ambiguous: BeTr-GEESE derivation trees are complex but have some structural correlations that might enable learning. On one hand, the *size modularity* metrics in Table 2(a) suggest that BeTr-GEESE is less modular than GEESE-BT. The PPA structure encoded in BeTr-GEESE's grammar redundantly includes checks of constraints and post-conditions, so 30 terminals appear on the right-hand-side (RHS) of productions in contrast to 24 for GEESE-BT. The PPA structure produces "wider" trees, and this requires more non-terminals (20–11). BeTr-GEESE also has more productions and possible derivation trees, yielding a higher value of McCabe cyclomatic complexity (44–27). Finally, BeTr-GEESE averages fewer symbols on the RHS of productions (3.75–4.09) and produces programs that are more difficult to understand according to the Halstead effort metric (283.62–132.94).

On the other hand, the *structural modularity metrics* in Table 2(b) suggest that the BeTr-GEESE grammar is more modular. Specifically, derivation trees for BeTr-GEESE are more treelike according to the tree impurity metric (7.6–15.56%). Additionally, related functionalities (non-terminals) in BeTr-GEESE are more logically grouped together according to the *nslev* clustering metric (8–6) and according to the *normalized count of levels* metric (40–36.36%). Derivation trees produced by the BeTr-GEESE grammar have higher correlations between non-terminals, which theoretically makes it easier to learn syntactically correct programs.

An alternative notion of modularity is task-based, that is, how well a task can be divided into "chunks". Evolution efficiency is influenced by an algorithm's ability to learn these "chunks". Both foraging and nest maintenance are *divisible* and *additive* (Steiner, 1972). 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 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 multi-action phenotype.

BeTr-GEESE limited gene expressiveness works well with its fitness function to learn single action phenotypes. In fact, 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

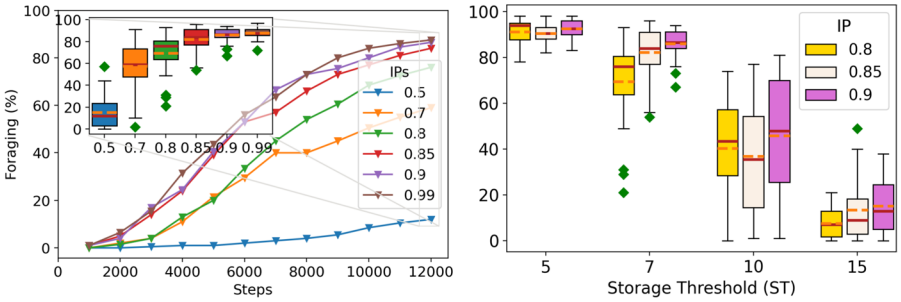

Fig. 4 **a** Foraging (%) vs IP. ST = 7. **b** Relationship between IP, ST, and foraging(%). GS = 10. An increase in IP increases the foraging performance, whereas an increase in ST decreases the foraging performance

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 leads to the learning of modular genes, meaning that fit genes typically only express a one-behavior phenotype.

4.2 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 are evaluated by describing the locality characteristics of the algorithm. Recall that all agents use the same BNF grammar to evolve BTs, the fitness of an agent's current behavior is comparable to other agents in the environment. However, the fitness function of a specific agent behavior is not directly proportional to the task's accomplishment since task accomplishment is a function of the emergent interaction of the swarm. Consequently, establishing the relationship between agent fitness and task performance is complex and beyond the scope of the paper.

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 agents are in close range cannot be controlled directly as it depends on the agent's current BT controller and environment objects. Fortunately, varying IP and ST alters how frequently an agent can perform genetic operations and evolve new controllers. Specifically, high IP and low ST are

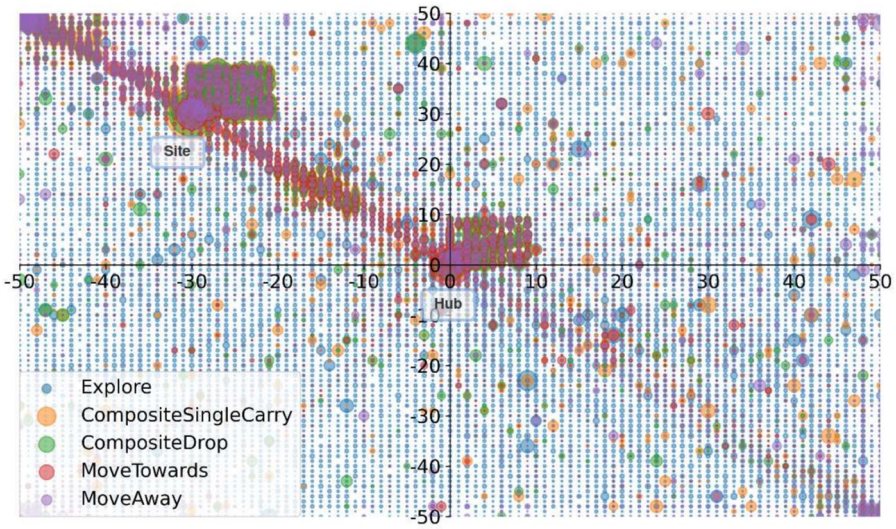

Fig. 5 Visualizing spatial locality: ST = 7, IP = 0.85, GS = 10, 3000 evolution steps. Agents cluster around vital environment objects such as Hub and Site, where most of the behavior transition occurs

directly proportional to the agent's probability to evolve new controllers because they enable rapid exchange of "genetic material" with other agents, whereas low IP and high ST evolve new controllers much more slowly.

Sixteen independent foraging runs were conducted for a range of IP and ST values, and results are summarised in Fig. 4. Figure 4a shows that with a high willingness to transfer genes to other agents (IP $>$ 0.8), the agents can change genes rapidly, boosting evolution from vertical transfer and lateral transfer. When IP $<$ 0.6, the agents persist with current behaviors too long, slowing down evolution. Figure 4b 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 persisting too long with a phenotype hinders evolution.

Figure 5 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.

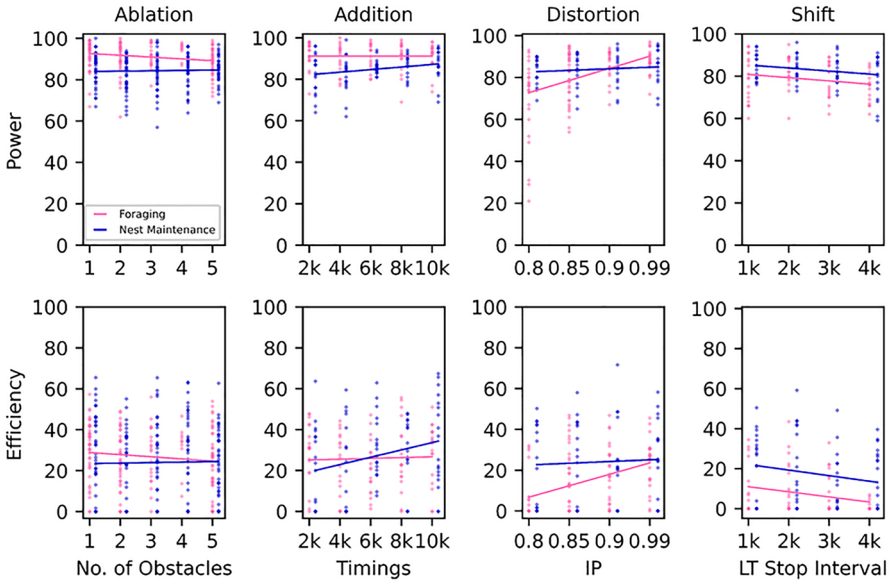

Fig. 6 Efficiency and power over a range of perturbations for foraging and next maintenance; For each experiment in each column, the variable named on the x-axis was varied and all other parameters were held constant. BeTr-GEESE agents perform both foraging and nest maintenance with high resilience power but low efficiency

4.3 Summary

BeTr-GEESE agents rapidly learn modular, one-behavior genes. Genes that express different subtask-specific phenotypes are held onto for a limited time by agents (temporal locality) and genes with the necessary phenotype are exchanged at locations in the world that determine which behavior is needed.

The GEESE-BT algorithm tried to directly evolve complete controllers with ad hoc, task-specific fitness functions. By contrast, BeTr-GEESE tried to evolve simple controllers using modular BT feedback. Each approach has strengths and weaknesses, but literature in biology suggests that it's easier to learn simple behaviors and then fuse them to form complex behaviors. Results in this section are consistent with this pattern holds for the foraging and nest maintenance tasks: learning and combining simple behaviors was more effective than learning complicated BTs.

5 BeTr-GEESE is resilient because it evolves quickly

This section is a modified version of the prior work in Neupane & Goodrich (2022b). When the BeTr-GEESE algorithm uses online evolution, agents evolve quickly enough to resiliently solve problems that arise when the world is perturbed.

5.1 Experiment design

Experiments using parameter values from the previous section were conducted when perturbations occurred. The independent variable is perturbation type, which are described below using the terminology adapted from Leaf and Adams (2022). The first dependent variable is the *power* resilience metric (Leaf & Adams, 2022), defined as the peak success probability achieved before the maximum number of allowed evolution steps $T = 12{,}000$. The second dependent variable is an affine transformation of the *time efficiency* (t_θ) resilience measure (Leaf & Adams, 2022), 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 $T_{\max} = 12{,}000$. Efficiency is set to zero for trials in which the threshold is not met.

5.2 Results

The four upper subplots show power, while the four lower plots show efficiency in Fig. 6 for each swarm task. Unless reported otherwise, all experiments reported in this section were configured with two obstacles, IP = 0.85, ST = 7, GS = 10, and $T = 12{,}000$. Table 1 specifies all other parameters. Sixteen simulations were performed for each perturbation condition.

Ablation An *ablation* perturbation reduces information, control, or possibilities (Leaf & Adams, 2022). 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 $t \in \{1000, 2000, \ldots, 11{,}000\}$. Obstacles remain in the world after their introduction. The experiment conditions are all combinations of t and obs. The first column of Fig. 6 shows mean power and efficiency for the ablation experiments.

The obstacle has impassable properties, so the agents must navigate around these objects. As the number of obstacles grows, the valid area for the agents to navigate decreases, so there are constraints on paths the agents can take. Moreover, adding obstacles changes the frequency and locations of where agents interact. So the modularity property of the agents remains constant, but the temporal and spatial locality properties vary based on the position of the obstacles in the environment. As the agents are equipped with an obstacle avoidance module that utilizes a simple Bug-follow-1 algorithm (Lumelsky and Stepanov 1987), the power and efficiency did not decrease significantly. The fitted line has a slight negative slope around -0.9 on average for both tasks, indicating that both power and efficiency slightly decrease as the number of obstacles increases. Since the power and efficiency fluctuation is small and above the 80% threshold for both tasks, BeTr-GEESE is resilient to adding obstacles.

Addition An *addition* perturbation increases the set of observable states or actions (Leaf & Adams, 2022). An experiment was performed where new actions were added to the BeTr-GEESE BNF grammar above as follows:

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

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

$$\langle motiontype \rangle ::= \text{Normal} | \text{Avoid} \tag{3}$$

$$\langle nonmotion \rangle ::= \text{CompositeSingleCarry\_}\langle dobjects \rangle | \text{CompositeDrop\_}\langle dobjects \rangle \quad (4)$$

This modification increases an agent's action set by allowing an agent to choose between locomotion behaviors with/without obstacle avoidance. The modification in the BNF grammar slightly increases the McCabe cyclomatic complexity value and has fewer symbols on the RHS of production on average. So there is a slight change in the size modularity metrics. Only two obstacles were randomly added to the environment at time $t \in \{1000, 2000, \ldots, 11{,}000\}$, so there is a change in the world that might hinder spatially local behaviors or lateral gene transfer. A fixed number of obstacle additions is impactful because the agent can use the increased action space and choose when to evolve or not evolve obstacle avoidance behaviors. Previous ablation experiments added different numbers of obstacles to reduce the set of possible paths an agent can take in the environment. In contrast, in additional experiments, the number of obstacles remains at 2, so the net effect of obstacles is constant. The second column of Fig. 6 shows mean power and efficiency for the addition experiments.

The fitted lines have small positive slopes indicating that the power increased as there was a minor change in the modularity of the BNF grammar, allowing agents to evolve resilient behaviors. Since a constant number of obstacles were added for each experiment, adding obstacles did not appear to alter how locality enabled efficient learning. Only a slight fluctuation with power and efficiency metrics indicates that the agents were capable of choosing between primitive behaviors with/without avoidance algorithm from the modified BNF grammar as needed.

Distortion A *distortion* perturbation alters the probability with which states or actions occur (Leaf & Adams, 2022). Altering interaction probability, 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. 6 shows results for the distortion experiments.

The fitted line for the foraging task has a high slope value, indicating that both power and efficiency increase as IP increases. The slope is positive for the nest maintenance task but is not as big as the foraging task. Thus, an increase in IP increases power and efficiency in both tasks, but foraging is more sensitive to distortion than nest maintenance. The reason for the increased distortion sensitivity is that the forging and nest-maintenance tasks have very different spatial structures. In foraging, the hub is in the center and food site locations vary. In nest maintenance, the debris is all over the hub and that debris needs to be removed from the hub. Thus, the sensitivity in the plots is due to those natural differences in the spatial locality of environment objects for the foraging and nest-maintenance tasks. Additionally, recall that the likelihood of agents interacting in the environment increases as IP increases. So, increasing IP values will directly impact temporal locality by increasing agents' probability of switching controllers more quickly.

Shift A *shift* perturbation combines the effects of multiple instances of ablation, addition, or distortion operations (Leaf & Adams, 2022). 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. 6 shows mean power and efficiency for the shift experiments.

The fitted lines for both tasks have a high negative slope value, indicating that both power and efficiency decreased as the duration Δ where the lateral transfer was disabled increased. When the lateral transfer is turned off, the agents cannot exchange genes with neighboring agents, directly impacting the temporal locality property. The lower efficiency indicates that agents struggled to learn successful behaviors within the time threshold T when Δ is large.

5.3 Discussion

The slopes of the fitted line for the foraging and nest-maintenance tasks differ based on the perturbation type. For the foraging task, the food is concentrated at the site. As the agents explore and interact with the environment, they evolve fit controllers, and at the end of the simulation the food will be concentrated at the hub. There is a cost of exploration for the foraging tasks because agents must first explore the environment to find the site. Consequently, the efficiency is low compared to nest maintenance. The power is high because once enough agents discover the site, agent-agent interaction will enable them to evolve behaviors that exploit the learned spatial information.

In contrast, the debris is first concentrated at the hub and then dispersed throughout the environment as the agents evolve in the nest maintenance task. Consequently, there is no exploration cost for the agents. Still, once enough debris is removed from the hub, it's harder for the agents to find debris dropped randomly in the environment that does not satisfy the nest-maintenance threshold of 30 units. Thus, nest maintenance has higher efficiency but lower power than foraging.

Overall, 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 (Doyle et al., 2013; Goh & Tan, 2007). Thus, online evolution makes BeTr-GEESE agents inefficient but with high resilience power.

6 Evolving resilient fixed strategies

In Evolutionary Robotics (ER) (Doncieux et al., 2015) simulation is a valuable tool because it makes it possible to quickly evaluate ideas, alter experiment setups, and replicate experiments. Once good solutions are evolved in simulation, the best solutions are transferred to the final robot. Despite the benefits observed with online evolution in the previous section, online evolution is (a) computationally expensive, and (b) difficult to generate satisfying explanation of agent actions do to the continuous evolution. Consequently, this section presents strategies to create resilient agents once they stop evolving.

BeTr-GEESE agents successfully perform collective foraging and nest maintenance while evolving, but they perform poorly once evolution stops. Denote an agent who has stopped applying the genetic operators and modifying their program as a *fixed agent* with a *fixed program*. In the literature, fixed programs are created by copying some of the fittest behaviors from the evolved agents. A new population of agents is created from that pool of fittest behaviors. The new population of fixed agents can be homogeneous or heterogeneous.

Conceptually, fixed-agent programs created from the learned agents could have higher efficiency since they do not have to perform evolutionary computation and rely on learned behaviors. However, prior work viclrspsBeTrspsGEESE showed that homogeneous populations of best-performing agents and heterogeneous populations of high-performing agents obtained from BeTr-GEESE performed poorly. The reason is simple: agents can typically perform only one subtask, and a population of these single-task agents is ineffective.

We now propose an evolution strategy for creating fixed agents. The resulting algorithm, called *Multi-GEESE*, has a *collect phase* and a *combine phase*. During the collect phase, the agents stores the most fit controllers corresponding to each of the five basic behaviors: MoveTowards, MoveAway, Explore, CompositeSingleCarry, and CompositeDrop. Upon completion of the collect phase, each agent possesses at most five distinct BT controllers, each tailored to one of these primitive behaviors, all structured within a PPA framework. Subsequently, in the combine phase, agents leverage a BNF grammar to assimilate the BT controllers produced during the collect phase into an Activator-Action-Repressed (AAR) structure BT. The combine phase essentially executes behavior fusion by integrating PPA BTs with condition nodes where, based on environmental cues, certain BTs become active while others remain inactive. Thus, agents must learn the activator and repressor conditions to activate or deactivate one of the PPA BTs acquired from the collect phase. A more comprehensive description of each phase is provided below.

6.1 Collect phase

A group of BeTr-GEESE agents succeeds because each basic subtask is performed: explore the world, move towards a known location, move away from a known location, pick up objects, and drop objects; see production (15) in Sect. 3. Endow each agent with a simple dictionary memory structure, denoted by `GeneDict` with five elements, one for each of the basic actions in production (15). Denote the elements of the dictionary by `GeneDict[x]` where

$$\texttt{x} \in \{\texttt{MoveTowards}, \texttt{MoveAway}, \texttt{Explore}, \texttt{CompositeSingleCarry}, \texttt{CompositeDrop}\}.$$

Each dictionary element stores a gene and the fitness of that gene, initialized to empty and negative infinity, respectively.

During the collect phase, agents evolve behavior through both vertical and lateral gene transfer using the BeTr-GEESE algorithm. Each time a gene with highest fitness is selected for use, the agent inspects this *active gene* to see which of the basic actions it contains. If there are no basic actions in the active gene, then the memory structure is not updated. If there is precisely one basic action, named `CurrentAction`, the agent compares the fitness of the current action with the fitness of the gene stored in `GeneDict[CurrentAction]`. Two conditions need to be met to update the dictionary: (a) the fitness of the active gene exceeds the fitness of `GeneDict[CurrentAction]` and (b) the return status of `CurrentAction` behavior tree node is *success*. Updating the dictionary only when the `CurrentAction` node is successful filters out low-quality behaviors explicitly. If there is more than one basic action in the active gene, the agent ignores the gene, favoring simple phenotypes over more complex phenotypes.

The collect phase runs for a fixed number of iterations. In the experiments below, the number of iterations is subjectively set to 12,000 time steps. This value was chosen because the previous sections have demonstrated that agents can perform all necessary subtasks to

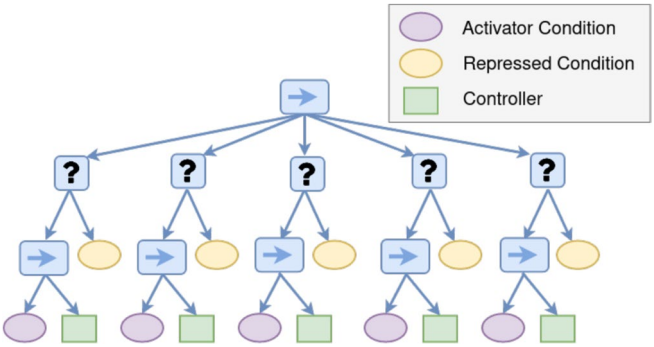

Fig. 7 Generic controller that uses the concept of activators and repressors to perform behavior fusion. Purple and yellow nodes are activator and repressed conditions respectively. There condition nodes need be learned by agents and will likely be different in each sub-trees

succeed in both foraging and in performing nest maintenance. 12,000 time steps makes it likely that most agents have a gene for each element of its dictionary.

6.2 Combine phase

After the collect phase is complete, each agent enters the *combine phase* in which genes in the *GeneDict* are organized using an evolutionary process to perform *behavior fusion*. The behavior fusion is not performed by learning behavior weights or priorities, but rather using inspiration from bacterial gene expression in which regulator proteins repress or activate genes. Temporal locality is then controlled by gene activation or repression, and the evolutionary process learns to map environment conditions to gene activation or repression.

Figure 7 shows a generalized BT that performs behavior fusion for all the phenotypes from the five genes present in the memory *GeneDict*. Note that there are five sub-trees, one for each gene. The green rectangle leaf nodes represent the programs/controllers encoded in the genes. Each green rectangle has a left neighbor represented as a purple condition node, which checks the activator condition for the controller. The activator and controller nodes have a parent that is a sequence node type, which ensures that the activator condition is true before the controller can be active.

The parent of the activator-controller sequence node is a selector node. This selector node allows control to pass to a subsequent controller in the tree if either the controller was successful or the yellow repressed condition node holds. If the yellow repressed condition node returns success, the gene is "turned off" and control flows to the next sub-tree. Otherwise, the control stays in the same sub-tree. In essence, the *activator* and *repressed* condition nodes dictate when and where a particular sub-tree (PPA controller) is active to accomplish the goal.

Observe that Fig. 7 specifies the activators and repressed conditions for each controller but not the order in which each controller executes. The root sequence node gives the controller on the left the first chance to execute, but if it is repressed then each subsequent controller is given a chance in a left-to-right sequence of controllers. The learning task is thus to identify (a) the correct left-to-right sequence of controllers and (b) activator/repressed conditions.

We propose the following grammar. Production 1 produces five sub-trees, each having activator-action-repressed (AAR) pattern. Production 2 chooses between five *basic actions*,

meaning actions not committed to a specific gene learned during the collect phase. Unique basic actions are not imposed by the grammar because Productions 1–2 make it allow each controller to same element of the gene dictionary obtained in the collect phase. The diversity function discourages controllers in which each action is the same, biasing the evolutionary process toward phenotypes that contain each basic action from the collect phase. Productions 3–7 define the AAR sub-tree structure shown in Fig. 7. The controller node is represented by the *lastSelector* non-terminal.

Productions 8–9 define the repressed conditions. Productions 10–14 match activator conditions with five unique basic actions. Productions 15–17 define activator conditions. Productions 18–22 define the five basic actions, which are the programs saved in the gene dictionary. Productions 23–25 define environment objects. The set of productions (1–15) is intended to be general, generating an arbitrary AAR structure BT, which is illustrated in Fig. 7. By contrast, productions (16–25) specify task-specific post-conditions, pre-conditions, and actions. Given the grammar, the task is to sequence the genes obtained in the collect phase and pair them with environmental signals so that the agents learn to activate and repress the basic actions to complete the task.

$$\langle root \rangle ::= [\text{Sequence}]\langle aar \rangle\langle aar \rangle\langle aar \rangle\langle aar \rangle\langle aar \rangle[/\text{Sequence}] \tag{1}$$

$$\langle aar \rangle ::= \langle aarA \rangle \mid \langle aarB \rangle \mid \langle aarD \rangle \mid \langle aarE \rangle \mid \langle aarC \rangle \tag{2}$$

$$\langle aarA \rangle ::= [\text{Selector}]\langle lastSelectorA \rangle\langle repressor \rangle[/\text{Selector}] \tag{3}$$

$$\langle aarB \rangle ::= [\text{Selector}]\langle lastSelectorB \rangle\langle repressor \rangle[/\text{Selector}] \tag{4}$$

$$\langle aarC \rangle ::= [\text{Selector}]\langle lastSelectorC \rangle\langle repressor \rangle[/\text{Selector}] \tag{5}$$

$$\langle aarD \rangle ::= [\text{Selector}]\langle lastSelectorD \rangle\langle repressor \rangle[/\text{Selector}] \tag{6}$$

$$\langle \langle aarE \rangle ::= [\text{Selector}]\langle lastSelectorE \rangle\langle repressor \rangle[/\text{Selector}] \tag{7}$$

$$\langle repressor \rangle ::= \langle repressor \rangle[\text{RepCnd}]\langle repressort \rangle[/\text{RepCnd}] \mid [\text{RepCnd}]\langle repressort \rangle[/\text{RepCnd}] \tag{8}$$

$$\langle lastSelectorA \rangle ::= [\text{Selector}]\langle activators \rangle[\text{Act}]\langle actiona \rangle[/\text{Act}][/\text{Selector}] \tag{9}$$

$$\langle lastSelectorB \rangle ::= [\text{Selector}]\langle activators \rangle[\text{Act}]\langle actionb \rangle[/\text{Act}][/\text{Selector}] \tag{10}$$

$$\langle lastSelectorC \rangle ::= [\text{Selector}]\langle activators \rangle[\text{Act}]\langle actionc \rangle[/\text{Act}][/\text{Selector}] \tag{11}$$

$$\langle lastSelectorD \rangle ::= [\text{Selector}]\langle activators \rangle[\text{Act}]\langle actiond \rangle[/\text{Act}][/\text{Selector}] \tag{12}$$

$$\langle lastSelectorE \rangle ::= [\text{Selector}]\langle activators \rangle[\text{Act}]\langle actione \rangle[/\text{Act}][/\text{Selector}] \tag{13}$$

$$\langle activators \rangle ::= [\text{Sequence}]\langle activator \rangle[/\text{Sequence}] \tag{14}$$

$$\langle activator \rangle ::= \langle activator \rangle[\text{ActiveCnd}]\langle preconditiont \rangle[/\text{ActiveCnd}] \mid [\text{ActiveCnd}]\langle preconditiont \rangle[/\text{ActiveCnd}] \tag{15}$$

$$\langle repressort \rangle ::= \text{NeighbourObjects\_}\langle objects \rangle \mid \text{NeighbourObjects\_}\langle sobjects \rangle \mid \text{NeighbourObjects\_}\langle dobjects \rangle \mid \text{DidAvoidedObj\_}\langle sobjects \rangle \mid \text{IsCarrying\_}\langle dobjects \rangle \mid \text{IsVisitedBefore\_}\langle sobjects \rangle \tag{16}$$

$$\langle preconditiont \rangle ::= \text{IsDropable\_}\langle sobjects \rangle \mid \text{NeighbourObjects\_}\langle objects \rangle \mid \text{NeighbourObjects\_}\langle objects \rangle\_invert \mid \text{IsVisitedBefore\_}\langle sobjects \rangle \mid \text{IsVisitedBefore\_}\langle sobjects \rangle\_invert \mid \text{IsCarrying\_}\langle dobjects \rangle \mid \text{IsCarrying\_}\langle dobjects \rangle\_invert \tag{17}$$

$$\langle actiona \rangle ::= \text{Explore} \tag{18}$$

$$\langle actionb \rangle ::= \text{MoveTowards} \tag{19}$$

$$\langle actionc \rangle ::= \text{MoveAway} \tag{20}$$

$$\langle actiond \rangle ::= \text{CompositeSingleCarry} \tag{21}$$

$$\langle actione \rangle ::= \text{CompositeDrop} \tag{22}$$

$$\langle sobjects \rangle ::= \text{Hub} \mid \text{Sites} \tag{23}$$

$$\langle dobjects \rangle ::= \text{Food} \mid \text{Debris} \tag{24}$$

$$\langle objects \rangle ::= \langle sobjects \rangle \mid \langle dobjects \rangle \tag{25}$$

6.3 Learning in the combine phase

The agents use the same *sense-act-update* as BeTr-GEESE with the same fitness functions, but with three major differences. First, the programs learned in the collect phase do not change during the connect phase. Thus, the codon used in Multi-GEESE uses productions 18–22 and does not use the productions used in the BeTr-GEESE grammar. Second, the Multi-GEESE agent's grammar has controller terminals that encode the programs learned in the collect phase rather than primitive behaviors in the BeTr-GEESE grammar. Third, the genotype-to-phenotype process maps the Multi-GEESE gene codon into a program in which productions 18–22 are replaced with the programs learned in the collect phase.

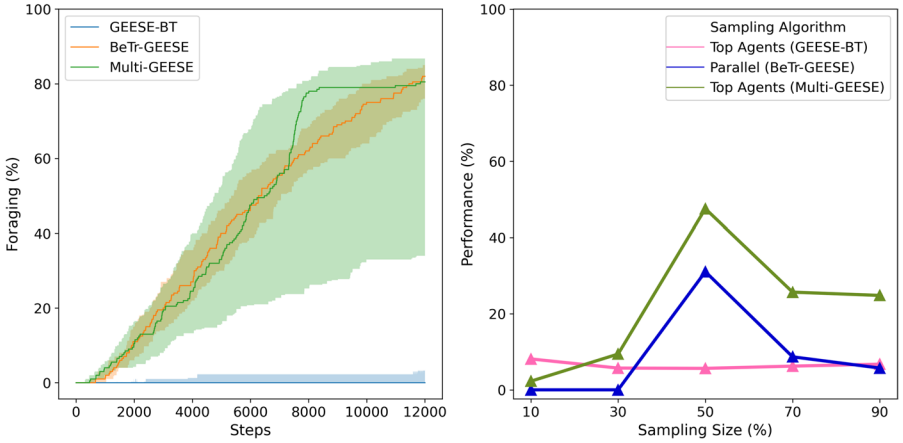

Fig. 8 **a** Comparing different GEESE algorithms based on foraging performance. **b** Population quality for populations created by sampling the top n% of agents for GEESE-BT, BeTr-GEESE, and Multi-GEESE agents. Multi-GEESE has a wide inter-quartile range denoting sub-optimal performance but has practical advantages over other methods

6.4 Learning efficiency

The first question to be answered is how well do Multi-GEESE agents perform each task while they are evolving. Recall that *learning efficiency* is defined as the task performance percentage as a function of time. Figure 8a compares the learning efficiency of various grammatical evolution algorithms on the foraging task. Thirty-two independent experiments are done for each GEESE algorithm. The solid line represents the median, and the shaded region represents the interquartile range. Note that only the *combine* phase of Multi-GEESE is plotted in the graph as *collect* phase is the same as of BeTr-GEESE.

The learning efficiency curve for Multi-GEESE has a steep slope at the beginning of the simulation and then flattens out towards the end. Recall that the *combine* phase of Multi-GEESE is learning activator and repressed conditions for the genes obtained during the *collect* phase. Once those conditions are learned, the task performance shoots up. The plateau happens when the site is almost empty of food, and the remaining food items are scattered around the environment or already at the hub.

One reason for the wide interquartile range is that learning the activator and repressed conditions using the same fitness function used by BeTr-GEESE might be sub-optimal. Since the new grammar has changed considerably from BeTr-GEESE, a new fitness function might be needed. Fortunately, the successful learning in the *combine* phase suggests that the fitness function is somewhat general, as predicted in Neupane and Goodrich (2022a). Future work should explore other fitness functions.

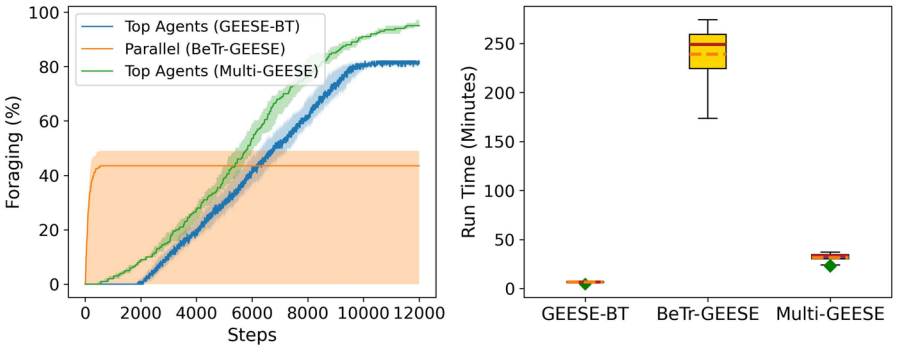

Fig. 9 **a** Foraging performance of best performing heterogeneous agents. **b** Run-time (cpu) time for the heterogeneous agents

6.5 Heterogeneous populations of fixed agents

The fitness of an individual agent can be deceiving because the agent might be fit only when other agents in a heterogenous population are performing necessary supporting tasks. Recall that *fixed agents* are agents who have stopped applying genetic operators and modifying their programs. A heterogeneous population of fixed agents was formed for GEESE-BT and Multi-GEESE by sorting the agents at the end of evolution by their fitness value, identifying the top n% of agents, and then cloning them to create 100 agents. This strategy, which is denoted the *Top Agents* strategy, suits both GEESE-BT and Multi-GEESE because the agents have complex controllers capable of completing the task independently.

A different strategy for selecting fixed BeTr-GEESE agents is needed since the learned programs have a modular controller with just one action node. Consequently, a heterogeneous population from BeTr-GEESE agents was formed by first sorting the agents at the end of evolution by their fitness value and identifying the top n% of agents. Second, all the top n% of the programs were "*OR*ed" together by forming a root BT node with a parallel node, which loosely acts as logical *or*. Finally, 100 agents were created by randomizing the order of sub-trees. More details of this hybrid approach can be found in Section 5 of the earlier work (Neupane & Goodrich, 2022a). This strategy is called the *Parallel* strategy.

Figure 8b shows the performance of a heterogeneous population of high-performing agents for the foraging task. The bold lines are the median performance values across sixteen independent experiments for each sampling size. We initially believed that a small number of fixed agents would perform the best with decreasing performance as more agents were allowed. However, results show that the Multi-GEESE population's is similar to GEESE-BT: performance slowly increases initially, peaks at 50%, and then decreases. One reason that Multi-GEESE performs poorly at 10% sampling size is the fitness function. Empirical analysis showed that, on average, only five learned agents had all five primitive actions, and these agents' fitness was not the highest like we assumed. The Multi-GEESE fitness function is a discounted sum of diversity fitness, exploration fitness, and BT feedback, but does not reward agents that successfully perform all five required subtasks. When a sampling size of 50% is used, two types of agents are selected: the most fit "partially able" agents and agents capable of all five primitive behaviors.

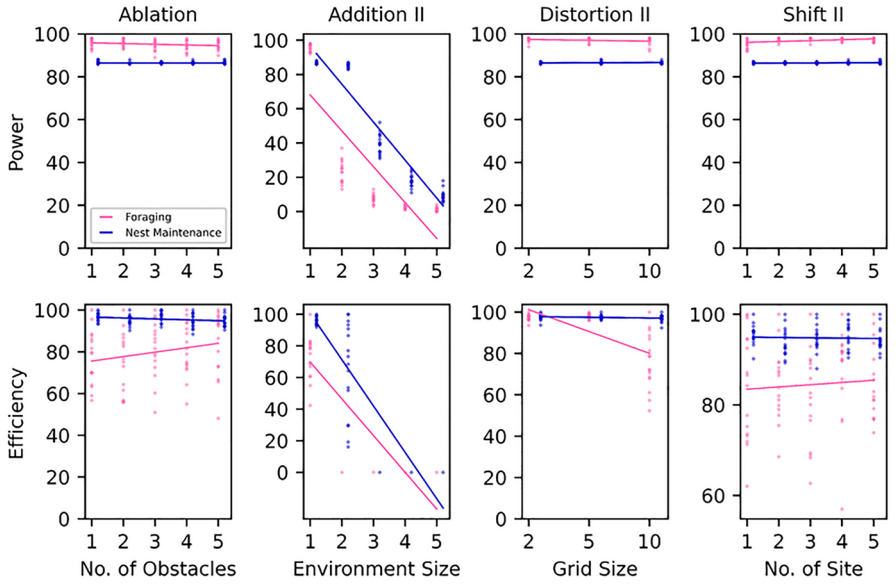

Fig. 10 Efficiency and power over a range of perturbations for foraging and nest maintenance. For each column, the variable named on the x-axis was varied, and all other parameters were held constant

6.6 Performance of best-performing populations

For each geese algorithm (GEESE-BT, BeTr-GEESE, Multi-GEESE), the best sampling size (0.1, 0.5, 0.5) is picked from Fig. 8b). The best-performing population of heterogeneous agents is then selected, and 16 independent foraging experiments are performed with standard environment parameters. Figure 9 compares the foraging performance and run-time. Multi-GEESE heterogeneous agents perform much better than BeTr-GEESE and GEESE-BT. GEESE-BT agents have the lowest run-time, but Multi-GEESE agents also execute quickly. These results suggest that Multi-GEESE agents strike a powerful balance of efficient learning, high performance, and low run-time.

7 Resilience experiments with multi-GEESE agents

BeTr-GEESE accomplished foraging and nest maintenance with high power when different perturbations were introduced as long as they continued learning; see Fig. 8a. This section examines the resilience power and efficiency of a heterogeneous fixed population of Multi-GEESE agents. The same experiment parameters are used as described in Sect. 5.1. The set of perturbations differs from the the previous resilience experiments since the previous experiments perturbed how learning occurred.

Ablation One or more obstacles, $obs \in \{1, 2, \ldots, 5\}$ are added and remain in the world. The first column of Fig. 10 shows mean power and efficiency. This ablation perturbation is the same as used in the leftmost column of Fig. 6 so the two can be directly compared.

The fixed population of heterogeneous Multi-GEESE agents has higher efficiency than BeTr-GEESE.

Addition II Recall that an *addition* perturbation increase the set of observable states or actions. Additional states were created by varying the width×height of the environment size. Label $envSize \in \{100 \times 100, 200 \times 200, \ldots, 500 \times 500\}$ using $\{1, 2, \ldots, 5\}$ in the figure for readability. The second column of Fig. 10 shows mean power and efficiency. Both power and efficiency drop linearly as the size of the environment increases. As environment size increases, a lot of space needs to be explored to find where the environment objects (sites, debris) are. When environment size increases, either the simulation time needs to increase so that the agents have more time to explore the environment, or the population size needs to increase so there are more agents to cover more ground. A fixed population of agents does not appear to be resilient, but that is not because the population is not capable but rather because more agents are needed to perform the task.

Distortion II Recall that a *distortion* perturbation alters the probability with which state or actions occur. Altering grid size (GS) changes the probability of detecting objects since the agent sensing range equals grid size. Note that an agent must be in the grid to sense an object. Experiment conditions used $GS \in \{2, 5, 10\}$. The third column of Fig. 10 shows mean power and efficiency. The power metric is approximately constant but efficiency drops as the grid size increases. This is an artifact of the sensing method. For small grid size, a site resides in many grids so exploration to find the site is more efficient. Thus, there is a slight decrease in efficiency as the grid size increase for the foraging task.

Shift II Recall that a *shift* perturbation combines the effect of multiple instances of ablation, addition, or distortion operations. For shift experiments, $site \in \{1, 2, \ldots, 5\}$ multiple sites are added, and food was distributed among those sites. The fourth column of Fig. 10 shows mean power and efficiency. Sites have no positive or negative influence on the nest maintenance task; thus, the addition of the sites does not affect the power or efficiency of the nest maintenance task as expected. For foraging, agents have a slightly higher probability of finding a site when there are more sites, which yields a slight positive slope for the efficiency metric.

7.1 Discussion

A heterogeneous population of fixed Multi-GEESE agents shows high resilience with both power and efficiency metrics for ablation, distortion II, and shift II perturbations. The minimal power and efficiency for addition II experiments are due to an exponential increase in the state space size. The behaviors learned by the agents do not encode environment size information, so as the environment increases, agents spend more time exploring, which results in lower performance. This performance issue can be addressed by modifying the BNF grammar with appropriate environmental information or increasing the simulation time, which would be interesting to explore in future studies. Since the fitness function used during the collect phase of Multi-GEESE does not include any task-specific rewards, the agents learn to perform the tasks in diverse ways, which enables the fixed population of Multi-GEESE agents to be resilient.

8 Future work

Note that BeTr-GEESE agents should theoretically able to be resilient to changes in the nature of the task, rapidly relearning behaviors that allow the collective to switch from foraging to nest maintenance behaviors. This suggests that combining *collect* and *connect* phases of Multi-GEESE might make learning more efficient while simultaneously keeping runtime low and enabling resilience to changes in the nature of the task. Additionally, it would be interesting to explore the dynamic adaptation of the mutation rate of genes when the agent's learning efficiency goes low, which is suggested by stressed-induced mutations in bacteria. Multi-GEESE has high resilience power and efficiency on divisible and additive tasks like foraging and nest-maintenance tasks, but future work should explore the generalizability of Multi-GEESE with other swarm tasks, reward structures, and primitive behaviors.

Moreover, it would be of considerable interest to investigate the applicability of the evolved behaviors to real robotic systems, aiming to ascertain if the resilience metrics such as locality and modularity are good indicators of resilience. Furthermore, an examination of the scalability of perturbations, algorithms, and resilience metrics to accommodate thousands of agents would provide valuable insights into the feasibility of implementing these strategies on a larger swarm systems. Additionally, exploring the potential generalization of the algorithm to alternative controllers, such as Deep Neural Networks (DNNs) as opposed to Behavior Tree controllers, presents an intriguing avenue for further research.

9 Conclusion

The BeTr-GEESE grammatical evolution algorithm resiliently responds to environment perturbations by enabling online evolution. Rapid online 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 perform all required subtasks sequentially, producing resilient performance in divisible and additive group tasks like foraging and nest maintenance. Switching between subtasks is enabled by lateral gene transfer. However, the behaviors of successful groups must exhibit temporal locality, meaning that an agent must persist in behavior long enough to perform essential functions but also means that agents cannot persist too long or evolution is too slow. Lateral transfer occurs in spatially local regions where agents are likely to meet, allowing location-specific behaviors to be adopted by neighboring agents. Online evolution through lateral transfer of simple modules exhibits resilience because agents can adapt to perturbations and succeed in their tasks, but this adaptation might be inefficient. A biologically inspired enhancement of using activators and repressors with BeTr-GEESE allowed a fixed population of heterogeneous Multi-GEESE agents to accomplish tasks with high resilience power and efficiency. The fitness function used in BeTr-GEESE is not tied to specific tasks and has standard functions like diversity, exploration, and BT feedback, which made it general enough to be used for both the collect and connect phases in Multi-GEESE despite there being significant changes in the grammar and optimization objective.

Acknowledgements Not applicable.

Author contributions Equal contributions from the authors. Not applicable.

Funding This work was supported by the U.S. Office of Naval Research (N00014-18-1-2831).

Data availibility statement Available upon request.

Declarations

Conflict of interest No.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Yes.

References

Bongard, J. (2011). Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences, 108*(4), 1234–1239.

Bongard, J. C. (2008). Accelerating self-modeling in cooperative robot teams. *IEEE Transactions on Evolutionary Computation, 13*(2), 321–332.

Bredeche, N., Montanier, J. M., Liu, W., & Winfield, A. F. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems, 18*(1), 101–129.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation, 2*(1), 14–23.

Browning, D. F., & Busby, S. J. (2004). The regulation of bacterial transcription initiation. *Nature Reviews Microbiology, 2*(1), 57–65.

Canciani, F., Talamali, M. S., Marshall, J. A., Bose, T., & Reina, A. (2019). 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).

Cheng, J., Cheng, W., & Nagpal, R. (2005). Robust and self-repairing formation control for swarms of mobile agents. In *AAAI* (vol. 5).

Cliff, D., Husbands, P., Harvey, I., et al. (1993). Evolving visually guided robots. *From Animals to Animats, 2*, 374–383.

Colledanchise, M., & Ögren, P. (2018). Behavior trees in robotics and al: An introduction.

Črepinšek, M., Kosar, T., Mernik, M., Cervelle, J., Forax, R., & Roussel, G. (2010). On automata and language based grammar metrics. *Computer Science and Information Systems, 14*, 309–329.

Črepinšek, M., Liu, S. H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR), 45*(3), 1–33.

Doncieux, S., Bredeche, N., Mouret, J. B., & Eiben, A. E. G. (2015). Evolutionary robotics: What, why, and where to. *Frontiers in Robotics and AI, 2*, 4.

Doncieux, S., Mouret, J. B., Bredeche, N., & Padois, V. (2011). Evolutionary robotics: Exploring new horizons. In *New horizons in evolutionary robotics* (pp. 3–25). New York: Springer.

Doyle, J. C., Francis, B. A., & Tannenbaum, A. R. (2013). *Feedback control theory*. North Chelmsford, MA: Courier Corporation.

Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S. M., & Christensen, A. L. (2016). Evolution of collective behaviors for a real swarm of aquatic surface robots. *PloS ONE, 11*(3), e0151834.

Eiben, A. E., Haasdijk, E., & Bredeche, N. (2010). Embodied, on-line, on-board evolution for autonomous robotics

Engebråten, S. A., Moen, J., Yakimenko, O., & Glette, K. (2018). Evolving a repertoire of controllers for a multi-function swarm. In *International conference on the applications of evolutionary computation* (pp. 734–749). New York: Springer.

Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., & O'Neill, M. (2017). Ponyge2: Grammatical evolution in python. In *Proceedings of the genetic and evolutionary computation conference companion* (pp. 1194–1201).

Ferrante, E., Duéñez-Guzmán, E., Turgut, A. E., & Wenseleers, T. (2013). Geswarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proceedings of the 15th annual GECCO conference* (pp. 17–24). ACM.

Ferrante, E., Turgut, A. E., Duéñez-Guzmán, E., Dorigo, M., & Wenseleers, T. (2015). Evolution of self-organized task specialization in robot swarms. *PLoS Computational Biology, 11*(8), e1004273.

Goh, C. K., & Tan, K. C.(2007). Evolving the tradeoffs between pareto-optimality and robustness in multi-objective evolutionary algorithms. In *Evolutionary computation in dynamic and uncertain environments* (pp. 457–478). Berlin: Springer.

Goodridge, S. G., & Luo, R. C. (1994). Fuzzy behavior fusion for reactive control of an autonomous mobile robot: Marge. In *Proceedings of the 1994 IEEE international conference on robotics and automation* (pp. 1622–1627). IEEE.

Gordon, D. M. (2010). *Ant encounters*. Princeton: Princeton University Press.

Goulson, D., Nicholls, E., Botías, C., & Rotheray, E. L. (2015). Bee declines driven by combined stress from parasites, pesticides, and lack of flowers. *Science, 347*(6229), 1255957.

Gunderson, L. H. (2000). Ecological resilience-in theory and application. *Annual Review of Ecology and Systematics, 31*(1), 425–439.

Haasdijk, E., Weel, B., & Eiben, A. E. (2013). Right on the MONEE: Combining task-and environment-driven evolution. In *Proceedings of the 15th annual conference on genetic and evolutionary computation* (pp. 207–214).

Hall, J. P., Brockhurst, M. A., & Harrison, E. (2017). Sampling the mobile gene pool: Innovation via horizontal gene transfer in bacteria. *Philosophical Transactions of the Royal Society B: Biological Sciences, 372*(1735), 20160424.

Holling, C. S. (1996). Engineering resilience versus ecological resilience. *Engineering Within Ecological Constraints, 31*(1996), 32.

Holway, D. A., Lach, L., Suarez, A. V., Tsutsui, N. D., & Case, T. J. (2002). The causes and consequences of ant invasions. *Annual Review of Ecology and Systematics, 33*(1), 181–233.

Hunt, E. R. (2020). Phenotypic plasticity provides a bioinspiration framework for minimal field swarm robotics. *Frontiers in Robotics and AI, 7*, 23.

Jablonka, E., & Lamb, M. J. (2014). *Evolution in four dimensions, revised edition: Genetic, epigenetic, behavioral, and symbolic variation in the history of life*. Cambridge: MIT Press.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *European conference on artificial life* (pp. 704–720). Berlin: Springer.

Johnson, M., & Brown, D. S. (2016). *Evolving and controlling perimeter, rendezvous, and foraging behaviors in a computation-free robot swarm*. Technical report, Air Force Research Laboratory/ RISC Rome United States.

Jones, S., Winfield, A. F., Hauert, S., & Studley, M. (2019). Onboard evolution of understandable swarm behaviors. *Advanced Intelligent Systems, 1*(6), 1900031.

Kazil, J., Masad, D., & Crooks, A. (2020). Utilizing python for agent-based modeling: The mesa framework. In R. Thomson, H. Bisgin, C. Dancy, A. Hyder, & M. Hussain (Eds.), *Social, Cultural, and Behavioral Modeling* (pp. 308–317). Cham: Springer.

Kelly, S. A., Panhuis, T. M., & Stoehr, A. M. (2011). Phenotypic plasticity: Molecular mechanisms and adaptive significance. *Comprehensive Physiology, 2*(2), 1417–1439.

König, L., Mostaghim, S., & Schmeck, H. (2009). Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics, 2*(4), 695–723.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing, 4*(2), 87–112.

Kriesel, D. M. M., Cheung, E., Sitti, M., & Lipson, H. (2008). Beanbag robotics: Robotic swarms with 1-DOF units. In *International conference on ant colony optimization and swarm intelligence* (pp. 267–274). Berlin: Springer.

Kucking, J., Ligot, A., Bozhinoski, D., & Birattari, M. (2018). Behavior trees as a control architecture in the automatic design of robot swarms. In *ANTS 2018*. IEEE

Kuckling, J., Vincent Van P., & Birattari, M. (2021). Automatic modular design of behavior trees for robot swarms with communication capabilites. In *EvoApplications* (pp. 130–145).

Lampe, D. J., Witherspoon, D. J., Soto-Adames, F. N., & Robertson, H. M. (2003). 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.

Lane, N. (2015). *The vital question: Energy, evolution, and the origins of complex life*. New York: WW Norton & Company.

Leaf, J., & Adams, J. A. (2022). Measuring resilience in collective robotic algorithms. In *Proceedings of the 21st international conference on autonomous agents and multiagent systems* (pp. 1666–1668).

Leaf, J., Adams, J. A., Scheutz, M., & Goodrich, M. A. (2023). Resilience for goal-based agents: Formalism, metrics, and case studies. *IEEE Access, 11*, 121999–122015.

Lee, W. P. (1999). Evolving complex robot behaviors. *Information Sciences, 121*(1–2), 1–25.

Lewis, M. A., Fagg, A. H., & Solidum, A. (1992). Genetic programming approach to the construction of a neural network for control of a walking robot. In *Proceedings 1992 IEEE international conference on robotics and automation* (pp. 2618–2623). IEEE

Li, W., & Feng, X. (1994). Behavior fusion for robot navigation in uncertain environments using fuzzy logic. In *Proceedings of IEEE international conference on systems, man and cybernetics* (Vol. 2, pp. 1790–1796). IEEE

Linksvayer, T. A., & Janssen, M. A. (2009). 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.

Lumelsky, V. J., & Stepanov, A. A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica, 2*(1), 403–430.

Miras, K., Ferrante, E., & Eiben, A. (2020). Environmental regulation using plasticoding for the evolution of robots. *Frontiers in Robotics and AI, 7*, 107.

Mlot, N. J., Tovey, C. A., & Hu, D. L. (2011). Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences, 108*(19), 7669–7673.

Nelson, A. L., Barlow, G. J., & Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems, 57*(4), 345–370.

Neupane, A., & Goodrich, M. A. (2019a). Designing emergent swarm behaviors using behavior trees and grammatical evolution. In *Proceedings of the 18th AAMAS conference* (pp. 2138–2140).

Neupane, A., & Goodrich, M. A. (2019b). Learning swarm behaviors using grammatical evolution and behavior trees. In *IJCAI* (pp. 513–520).

Neupane, A., Goodrich, M. A., & Mercer, E. G. (2018). Geese: Grammatical evolution algorithm for evolution of swarm behaviors. In *Proceedings of the 20th annual GECCO conference* (pp. 999–1006).

Neupane, A., & Goodrich, M. (2022a). Efficiently evolving swarm behaviors using grammatical evolution with PPA-style behavior trees. In *From cells to societies: Collective learning across scales*.

Neupane, A., & Goodrich, M. A. (2022b). Learning resilient swarm behaviors via ongoing evolution. In *International conference on swarm intelligence* (pp. 155–170). Berlin: Springer.

Nevai, A. L., Passino, K. M., & Srinivasan, P. (2010). Stability of choice in the honey bee nest-site selection process. *Journal of Theoretical Biology, 263*(1), 93–107.

Noirot, C., & Darlington, J. P. (2000). Termite nests: Architecture, regulation and defence. In *Termites: Evolution, sociality, symbioses, ecology* (pp. 121–139). Berlin: Springer.

Ochman, H., Lawrence, J. G., & Groisman, E. A. (2000). Lateral gene transfer and the nature of bacterial innovation. *Nature, 405*(6784), 299–304.

Oneill, M., Ryan, C., Keijzer, M., & Cattolico, M. (2003). Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines, 4*(1), 67–93.

Perez, R., & Aron, S. (2020). Adaptations to thermal stress in social insects: Recent advances and future directions. *Biological Reviews, 95*(6), 1535–1553.

Petrovic, P. (2008). Evolving behavior coordination for mobile robots using distributed finite-state automata. In *Frontiers in evolutionary robotics*. InTech.

Pintér-Bartha, A., Sobe, A., & Elmenreich, W. (2012). Towards the light-comparing evolved neural network controllers and finite state machine controllers. In *Proceedings of the tenth workshop on intelligent solutions in embedded systems* (pp. 83–87). IEEE.

Power, J. F., & Malloy, B. A. (2004). A metrics suite for grammar-based software. *Journal of Software Maintenance and Evolution: Research and Practice, 16*(6), 405–426.

Prasetyo, J., De Masi, G., & Ferrante, E. (2019). Collective decision making in dynamic environments. *Swarm Intelligence, 13*(3), 217–243.

Prasetyo, J., Masi, G. D., Ranjan, P., & Ferrante, E. (2018). The best-of-n problem with dynamic site qualities: Achieving adaptability with stubborn individuals. In *International conference on swarm intelligence* (pp. 239–251). Berlin: Springer.

Quammen, D. (2018). *The tangled tree: A radical new history of life*. New York, NY: Simon and Schuster.

Reid, C. R., Lutz, M. J., Powell, S., Kao, A. B., Couzin, I. D., & Garnier, S. (2015). 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.

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science, 345*(6198), 795–799.
Samples, A. D. (1989). Mache: No-loss trace compaction. In *Proceedings of the 1989 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (pp. 89–97).
Schwander, T., Rosset, H., & Chapuisat, M. (2005). 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.
Seeley, T. D. (2009). *The wisdom of the hive: The social physiology of honey bee colonies*. Cambridge: Harvard University Press.
Seeley, T. D. (2010). *Honeybee democracy*. Princeton University Press: Princeton.
Shen, W. M., Lu, Y., & Will, P. (2000). Hormone-based control for self-reconfigurable robots. In *Proceedings of the fourth international conference on autonomous agents* (pp. 1–8).
Simon, H. A. (2019). *The sciences of the artificial, reissue of the third edition with a new introduction by John Laird*. Cambridge, MA: MIT Press.
Singh, S., Lewis, R. L., Barto, A. G., & Sorg, J. (2010). Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development, 2*(2), 70–82.
Sorenson, E. S., & Flanagan, J. K. (2002). Evaluating synthetic trace models using locality surfaces. In *Proceedings of the IEEE international workshop on workload characterization* (pp. 23–33).
Soule, T. (2006). Resilient individuals improve evolutionary search. *Artificial Life, 12*(1), 17–34.
Steiner, D. I. (1972). *Group process and productivity*. New York, NY: Academic Press.
Stonier, D., & Staniaszek, M. (2021). Behavior Tree implementation in Python. https://github.com/splintered-reality/py_trees/
Sumpter, D., & Pratt, S. (2003). A modelling framework for understanding social insect foraging. *Behavioral Ecology and Sociobiology, 53*(3), 131–144.
Toffolo, A., & Benini, E. (2003). Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation, 11*(2), 151–167.
Toth, A., & Robinson, G. (2009). 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). New York: Cold Spring Harbor Laboratory Press.
Trianni, V., Groß, R., Labella, T. H., Şahin, E., & Dorigo, M. (2003). Evolving aggregation behaviors in a swarm of robots. In *European conference on artificial life* (pp. 865–874). Berlin: Springer.
Ursem, R. K. (2002). Diversity-guided evolutionary algorithms. In *International conference on parallel problem solving from nature* (pp. 462–471). Berlin: Springer.
Varughese, J. C., Thenius, R., Schmickl, T., & Wotawa, F. (2017). Quantification and analysis of the resilience of two swarm intelligent algorithms. In *GCAI* (pp. 148–161).
Vistbakka, I., & Troubitsyna, E. (2019). Modelling autonomous resilient multi-robotic systems. In *International workshop on software engineering for resilient systems* (pp. 29–45). Berlin: Springer.
Wagner, G. P., & Altenberg, L. (1996). Perspective: Complex adaptations and the evolution of evolvability. *Evolution, 50*(3), 967–976.
Wang, J. X., Hughes, E., Fernando, C., Czarnecki, W. M., Duéñez-Guzmán, E. A., & Leibo, J. Z. (2018). Evolving intrinsic motivations for altruistic behavior. arXiv preprint arXiv:1811.05931
Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS Computational Biology, 4*(11), e1000220.
Yim, M., Shen, W. M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., & Chirikjian, G. S. (2007). Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine, 14*(1), 43–52.
Zahadat, P., Hamann, H., & Schmickl, T. (2015). 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).