A Quality-Diversity Approach to Evolving a Repertoire of Diverse Behaviour-Trees in Robot Swarms

Kirsty Montague[1], Emma Hart[1][0000−0002−5405−4413], Geoff Nitschke[2][0000−0001−9058−852X], and Ben Paechter[1][0000−0002−4841−0805]

[1] Edinburgh Napier University,
{k.montague,e.hart,b.paechter}@napier.ac.uk,
[2] University of Capetown
{geoff.nitschke}@uct.ac.za

Abstract. Designing controllers for a swarm of robots such that collaborative behaviour emerges at the swarm level is known to be challenging. Evolutionary approaches have proved promising, with attention turning more recently to evolving repertoires of diverse behaviours that can be used to compose heterogeneous swarms or mitigate against faults. Here we extend existing work by combining a Quality-Diversity algorithm (MAP-Elites) with a Genetic-Programming (GP) algorithm to evolve repertoires of *behaviour-trees* that define the robot controllers. We compare this approach with two variants of GP, one of which uses an implicit diversity method. Our results show that the QD approach results in larger and more diverse repertoires than the other methods with no loss in quality with respect to the best solutions found. Given that behaviour-trees have the added advantage of being human-readable compared to neural controllers that are typically evolved, the results provide a solid platform for future work in composing heterogeneous swarms.

Keywords: Swarm-robotics · Quality-Diversity · Genetic-Programming.

1 Introduction

Evolutionary techniques were first proposed to design controllers for swarms of robots as far back as 1992 [17]. A steadily increasing amount of work since then has shown that EAs are capable of designing controllers for a range of tasks, demonstrating that self-organising behaviours can emerge from a swarm in which each member executes the same controller. Recently, attention has begun to turn towards evolving repertoires of controllers which exhibit behavioural diversity. This is particularly prevalent in the literature relating to evolution of controllers for individual robots, for example, Cully et al. [7], but has also begun to permeate the swarm robotics field. Having access to a repertoire of behaviourally diverse controllers has multiple potential advantages. For example, it can facilitate fault-tolerance by providing alternative methods of control when there are faulty sensors or actuators; it raises the possibility of adapting to unseen

situations in novel environments not known when the controllers were evolved; it enables composition of a *heterogeneous* swarm with respect to control. The latter is pertinent given the breadth of literature across disciplines suggesting that groups of diverse problem-solvers can outperform groups of high-ability problem-solvers [16]. Thus, there is considerable need for a library of diverse behavioural repertoires that can be used to improve swarm task-performance.

Existing work relating to evolution of diverse repertoires for swarms has tended to use neural network or parameterised controllers in conjunction with a *quality-diversity* algorithm. The latter are a family of methods first introduced in [27] that evolve multiple solutions that are diverse with respect to one or more user-defined characteristics. This study evaluates a method to evolve diverse behaviours on various swarm-robotic tasks, where behaviours result from controllers defined by *behaviour-trees* (BTs). BTs are structures that describe switching between a finite set of tasks [4, 23], enabling complex behaviours to be described from a series of simple tasks. In addition, they have the added advantage of being human-readable: this allows them to be analysed, extended and verified [29]. A broad range of recent work has investigated ways in which Behaviour Trees can be leveraged for fully-autonomous robotic control [1, 2, 5, 12, 23] in both individual and swarm robotics.

The paper makes two contributions to the field of BTs and swarm-robotics. Firstly, we show that genetic-programming (GP) can be used to evolve diverse BTs representing *primitives* defining general skills required to realise foraging behaviours. Evolving primitives rather than a single BT that realises an overall foraging behaviour is advantageous in that primitives can be combined in multiple ways in a higher-level control strategy to achieve multiple tasks. We compare two GP methods for evolving primitives. The first requires separate runs of a GP algorithm to produce each primitive. The second simultaneously evolves all three primitives using an implicit population diversity mechanism. We then develop a third method that combines the quality-diversity algorithm MAP-Elites [25] with GP to explicitly evolve a repertoire of behaviours that are diverse with respect to three axes describing behavioural characteristics. We demonstrate our QD-GP algorithm significantly outperforms the two GP approaches in terms of metrics that quantify both the quality and diversity of the generated behaviours, although the GP method using implicit diversity is surprisingly competitive in terms of both quality and diversity metrics.

2 Background

In the context of swarm robotics, evolutionary algorithms have been used to evolve a range of controller *types*: most commonly, neural controllers are learned, with an EA learning the network topology, weights or both [30]. The parameters of parametric controllers have also been evolved [8]. However, another line of enquiry has focused on the evolution of *behaviour-tree* (BT) controllers. They are often presented as a viable architecture for robotics on the basis that they remove the trade-off between reactivity and modularity inherent in finite state machines

[6], allowing complex behaviours to be encoded while still maintaining independence between components. In addition they have the added advantage of being human-readable; this allows them to be analysed, extended and verified [29]. Successful examples include evolution of behaviour tree controllers for swarms of kilobot robots foraging in a simulated arena, developed in a series of papers [18–20]. Behaviour trees for swarms of e-puck robots [24] were also evolved by [21] and compared to finite-state machine controllers generated by AutoMoDe Chocolate [10] and neural networks generated by an approach dubbed Evostick [11]. The authors found that both of the modular design approaches transferred to reality more successfully than the controllers evolved by Evostick. [14] introduce the Instinct Evolution Scheme to control a swarm of miniature autonomous agents with constrained resources and reconfigurable hardware. They propose a method where Grammatical Evolution is used to evolve BTs from a set of action nodes derived from a Pareto front of configurations for each hardware module which satisfy local objectives (for example precision versus power consumption).

Quality-diversity algorithms [27] are a relatively new family of evolutionary algorithms that aim to find a maximally diverse collection of individuals (often with respect to a user-defined measure of behaviour) in which each individual is as high performing as possible. While the field has rapidly expanded to include a large set of algorithm variants, most derive from two fundamental algorithms: MAP-Elites [25] and Novelty-Search with Local Competition (NSLC) [22]. Both methods return an archive of diverse, high-quality behaviors in a single run. Many successful applications of QD methods to evolve diverse behavioural repertoires for single robots exist[3]. However, applications to swarm-robotics to provide diverse behavioural repertoires are still under-represented in the literature, despite the potential benefits sketched in section 1.

Engebråten *et al* [8] evolve a repertoire of diverse controllers for a multi-function swarm using MAP-Elites. Robots were controlled using parametric or weighted controllers inspired by the use of artificial potential fields and evolved to conduct perimeter surveillance and network creation tasks. In Gomes *et. al.* [13], evolution of a diverse set of task-agnostic behaviours is tackled. Robots have neural controllers, and novelty-search with local-competition (NSLC) [22] is used to evolve a set of diverse behaviours that can be utilised in multiple tasks. The authors demonstrated that repertoires of general swarm behaviours can be generated, yielding a wide diversity of high-quality behaviours. Steyven *et. al.* [15] describe the first distributed model of the MAP-Elites algorithm for embodied evolution: each robot in a swarm runs its own MAP-Elites algorithm, with a robot exchanging or updating its individual container when coming into range of another robot. Their algorithm EDQD was demonstrated to produce a diverse range of controllers in a simple swarm following task.

Thus, despite a plethora of literature exploring QD methods to generate diverse behavioural repertoires for individual robots, such methods are under-explored in swarm-robotics. This study extends related work by combining GP

[3] For an up-to-date list of relevant papers, see https://quality-diversity.github.io

with MAP-Elites to evolve a repertoire of BT controllers, representing generic *building-block* primitives, combined to realise swarm-robotic foraging tasks.

3 Methodology

The goal of the paper is twofold. Firstly, we investigate whether GP can be used to evolve BTs representing three primitive behaviours that are important components of swarm-foraging tasks (*go-to-food*, *go-to-nest*, *increase-neighbourhood-density*). Similarly to the work described by Gomes et al. [13], the motivation behind this approach is that general low-level behaviours can in future be composed into more complex behaviours in multiple different tasks. Secondly, we investigate the extent to which a diverse repertoire of swarm controllers can be evolved. In this work, we assume that swarms are homogeneous in that all robots run the same controller, however as noted in section 1, the motivation behind this is that the evolved repertoire has broad swarm-robotic task applicability.

Two GP algorithms for generating BTs are compared. GP_1 evolves a BT to optimise the behaviour of a *single* primitive, hence needs to be run separately for each of three primitives *go_to_food*, *go_to_nest*, *increase_density*. The variants are thus labelled GP_f, GP_n, GP_d respectively. In contrast, $GP_{f,n,d}$ *simultaneously* evolves BTs for all three primitives: an *implicit* diversity maintenance mechanism is used in which the selection method compares solutions on a randomly chosen objective each time it as called. We then compare these methods to a quality-diversity approach (MAP-Elites) which evolves behaviours for each objective that are diverse with respect to three user-defined descriptors. Like GP_1, this also needs to be run once per objective (consequently labelled QD_n, QD_f, QD_d).

3.1 Environment, Robots and Simulator

The simulator and task implemented are inspired by the foraging kilobots experiment described in Jones et al. [18]. This requires robots to travel between nest and food regions in an arena as shown in figure 1. Our swarm consists of 9 robots. We consider evolution of three behavioural fragments which can be combined by a higher-level algorithm to achieve more complex swarm behaviours. The fragments considered attempt to maximise the following objectives:

- **O1: Increase neighbourhood density** maximises the difference between the density of neighbouring robots at the beginning and end of each trial by subtracting the initial density from the final density.

- **O2: Move towards the nest region** uses the distance estimated by each robot based on the shortest route by hops via neighbouring robots from its location at the start and end of each trial. The difference is maximised by subtracting the initial distance from the final distance.

– **O3: Move towards the food region** maximises the robots' estimated difference in distance to the food region in the same way as the nest region, by subtracting the initial estimated distance from food from the final estimated distance from food.

The robots used are *footbots*, shown in figure 2, simulated using ARGoS [26] and based on marXBots as described in Bonani et al. [3]. They move on a combination of tracks and wheels (treels). The sensing and actuation capabilities of the robots in this scenario are given in table 1. If a robot is within 50mm of the centre of the arena, it is considered to be within the nest region, while if it is located more than 50mm plus the width of the gap between nest and food then it is considered to be within the food region. A *blackboard* provides the interface between the evolving behaviour-trees and the footbot control software. Information received from sensors and other robots in each update cycle is condensed into the blackboard entries listed below, each of which return true or false, with two separate entries as necessary for *left* and *right*. Note that 'left' means that a detection event occurred anywhere in the left hemisphere of the robot (assuming a virtual line drawn through the centre of the robot from front to back). 'Right' is interpreted in the same manner.

– Is this robot in the food region
– Is this robot in the nest region
– Is the shortest route to food to this robot's left / right
– Is the shortest route to the nest to this robot's left / right
– Is the closest neighbouring robot to this robot's left / right

Each update cycle, the robots transmit their ID and estimated distances from the food and nest regions to their neighbours. Distances are calculated using the shortest hop distance via neighbouring robots after adding the extra distance to the robot in question. Robots currently within the nest or food regions will send a default estimate of 0mm and robots that are out of range default to 500mm. Messages arriving from robots more than 100mm away are ignored while messages within this range arrive with 95% probability (as per Jones et al. [18]). Also in keeping with Jones et al [18]), behaviour trees are ticked (executed) at 2Hz while sensing and communication takes place at 8Hz, and the range and bearing data is averaged over seven samples to estimate neighbourhood density and the distance to the nest and food regions.

A single evaluation consists of ten trials, each lasting 20 seconds (40 ticks). In each trial, the robots start in a random position within 100mm of the centre of a 500mm x 500mm arena. To encourage robust behaviours, the trials are equally split between two arena configurations which differ in the size of the gap between the nest and food regions (50mm and 70mm) while the radius of the nest region remains constant at 50mm.

3.2 Behaviour tree representation

All nodes in a typical BT return one of three strings, *success*, *failure*, or *running* (the node's operation will continue to execute until at least the next tick). This

Table 1: Footbot Reference Model

Ground sensor	Informs the robot if it is in the nest or food regions
Range	Distance to each neighbour
Bearing	Angle of each neighbour
Signal	Information payload from each robot in range containing the cumulative distance to nest and food regions derived from hops
Motors	Left and right wheels controlled independently, with ± 10% variation between robots

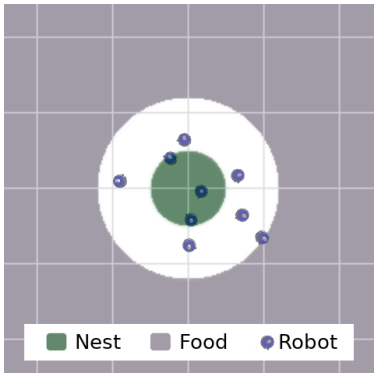


Fig. 1: The arena layout, with nine robots initialised in random starting positions.

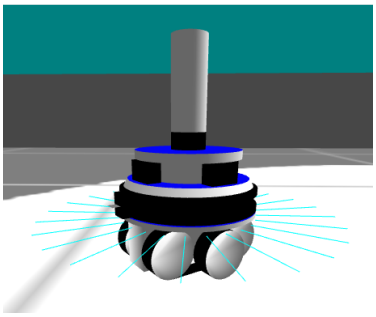


Fig. 2: A screenshot of a footbot robot in the arena taken in ARGoS.

results in a uniform interface where all nodes are compatible, removing any possibility of creating an invalid tree along with any requirement for type safety. The root node of a behaviour tree is executed (ticked) at regular intervals. If the root has child nodes, the tick propagates through the tree as each branch node which is executed ticks one or more of its children. Composition nodes form the branches and control the flow of execution through the tree by prioritising between several child nodes in different ways (table 2). The composition nodes in these experiments have memory so if a child node returns *running* the parent does likewise. Upon resuming execution they continue by ticking the same child in their list the next time they themselves are ticked.

Leaf nodes can be either conditions or actions. Condition nodes control the flow of execution by returning *success* or *failure* depending whether their condition is satisfied, which determines whether their parent can continue to tick any subsequent children. The action nodes in these experiments all represent locomotion behaviours, returning *running* initially (in effect pausing the BT for one tick) and returning *success* on the next tick. In doing so, the action node terminates and allows execution to flow back to the parent. Condition or action nodes do not have children so they can be exchanged freely by the mutation

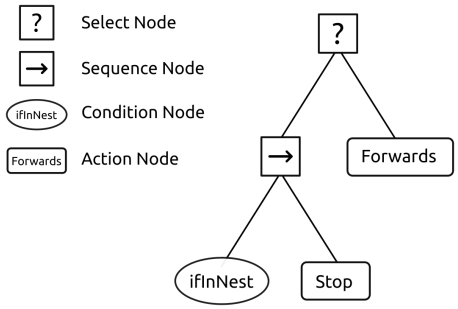


Fig. 3: An example behaviour tree.

and crossover operators. An example behaviour tree is shown in figure 3, with a select node at its root and a sequence node as the first child. The tree checks if the robot is in the nest, in which case it stops, otherwise it moves forward.

Behaviour tree nodes: BTs nodes in these experiments can fulfil one of three roles as outlined below:

- **Composition nodes** control the flow of execution through the tree. There are three types of node (sequence, select and probabilistic), described in table 2. The role of these nodes is to govern execution of child nodes, accounting for which was executed last.
- **Condition nodes** have no children and return *success* or *failure* depending whether their condition is satisfied. Nodes are listed in table 3 and determine the direction of the nearest neighbour (left/right); direction of the shortest hop distance to food or nest (left/right); whether the robot is within the nest or food region. Where necessary, the robots' immediate vicinity within 100mm is divided into left and right hemispheres and the node's return value indicates whether the given half contains the object of interest.
- **Action nodes** have no children. Their purpose is to send commands to the robots' motors to provide movement. These are listed in table 4

Table 2: Composition Nodes

Sequence node with memory and 2, 3 or 4 children	Executes each child until one fails
Select node with memory and 2, 3 or 4 children	Ticks each child until one succeeds
Probabilistic node with memory and 2, 3 or 4 children	Ticks one child chosen at random

Table 3: Condition Nodes

If on food	Returns success if the robot is within the food region
If food to left	Returns success if the shortest route to the food region is to the robot's left
If food to right	Returns success if the the shortest route to the food region is to the robot's right
If in nest	Returns success if the robot is within the nest region
If nest to left	Returns success if the shortest route to the nest region is to the robot's left
If nest to right	Returns success if the shortest route to the nest region is to the robot's right
If robot to left	Returns success if the nearest robot is to this robot's left
If robot to right	Returns success if the nearest robot is to this robot's right

Table 4: Action Nodes

Stop	No movement for one tick
Forwards	Move forwards for one tick
Forwards left	Right wheel forwards for one tick, rotating the robot anti-clockwise
Forwards right	Left wheel forwards for one tick, rotating the robot clockwise
Reverse	Move backwards for one tick
Reverse left	Right wheel in reverse for one tick, rotating the robot clockwise
Reverse right	Left wheel in reverse for one tick, rotating the robot anti-clockwise

3.3 Algorithms

Controllers for three behaviour fragments are evolved using a benchmark evolutionary algorithm and two variations which employ different strategies for promoting behavioural diversity.

Benchmark algorithms: We compare results obtained from running a classical genetic-programming algorithm with quality-diversity approach. The GP algorithm is implemented using DEAP [9] with the parameters specified in table 5. We evaluate two versions of this algorithm:

1. GP_o: this optimises a single objective o (one of the three functions specified in section 3.1), and therefore evolves a BT for one of three primitive behaviours. Experiments are conducted for each of three primitives in turn from each of the three variants (GP_f, GP_n, GP_d).

2. $GP_{f,n,d}$: in order to *implicitly* encourage diversity within the population, each time the selection method is called, one of three objectives is chosen with uniform probability, and the individuals in the tournament are compared according to the chosen objective. A list of the elite solution for each objective is maintained throughout the run. As the intention is to evolve

three separate trees representing each primitive, the population size is tripled and the maximum evaluation budget is set to three times the budget of GP_o.

All of the nodes used to define BTs return the same type (a string which is either "success", "failure" or "running"), so there is no requirement for the algorithm to be strongly typed. Pressure is applied towards smaller trees by selecting the shortest in the event that their fitness scores are exactly equal and at least one of them is more than three nodes deep.

MAP-Elites for Evolution of Diverse Behaviours: We use the QD algorithm MAP-Elites [25]. This is implemented using the Python Library QDPy; the reader is referred to the documentation for a full description of the algorithm which follows the original definition of MAP-Elites. MAP-Elites differs from classical evolutionary algorithms in that the population is stored in a container discretised into cells according to behaviour descriptors defined by a user. Each solution is mapped to a single cell according to the derived behavioural descriptors; the cell stores up to three solutions which have the best objective-values found for that cell during the run. We define three behavioural axes describing phenotypic aspects of the robots during execution of the BT controller that promote *phenotypic* (behavioural) diversity: (1) Ratio of forwards vs backwards movements; (2) Ratio of clockwise vs anti-clockwise rotations; (3) Ratio of condition nodes vs action nodes. Quality is calculated using the same three objective functions as the GP approaches described in the previous section. As with GP_o, the QD method must be run separately for each of three primitives. These methods are labelled (QD_n, QD_f & QD_d). All parameters required to run the GP algorithm defined in DEAP and the MAP-Elites algorithm defined in QDPy are listed in table 5 and were derived from [18] where possible or by preliminary empirical testing otherwise.

4 Results

We present results from two sets of experiments: (1) A comparison of the quality of the best BT controller found from each of three methods for each primitive (GP_o, $GP_{f,n,d}$, QD_o); (2) A comparison of the diversity of behaviours found by each of the three algorithms per primitive. *Quality* is determined with respect to objective fitness for each of the three primitives. We measure diversity according to the *coverage* metric. That is, the proportion of bins filled, and finally also report the QD score which captures both quality and diversity. This is obtained by summing the highest fitness values found in each grid bin (Q_i) [28] $\sum_{i=1}^{t} Q_i$. All metrics are calculated over 10 runs with a maximum budget of 25,000 evaluations per objective[4]. To evaluate whether there is statistical significance between pairs of results, we first apply a Shapiro-Wilk test to test for normality. If the null hypothesis is rejected ($p \leq 0.05$) then a Mann-Whitney test is used for comparison, otherwise a t-test is applied (again with a confidence level of 0.05).

[4] $GP_{n,f,d}$ is allocated 75000 evaluations as it simultaneously solves 3 objectives.

Table 5: Genetic Programming & Quality-Diversity Parameters

GP Parameters	
Generations	1000
Population size	25 per objective
Tournament size	3 (GP_o) and 5 ($GP_{f,n,d}$)
Elites	1 per objective
Probability of crossover	0.8
Probability of mutating by inserting a subtree	0.05
Probability of mutating by shrinking a subtree	0.1
Probability of mutating by replacing one node	0.5
Trials per evaluation	10
Arena configurations per evaluation	2
Trial length	20 seconds (40 ticks)
QD Parameters	
Bins	8 x 8 x 8
Max items per bin	3
Initial batch size	100
Batch size	50
Generations	500

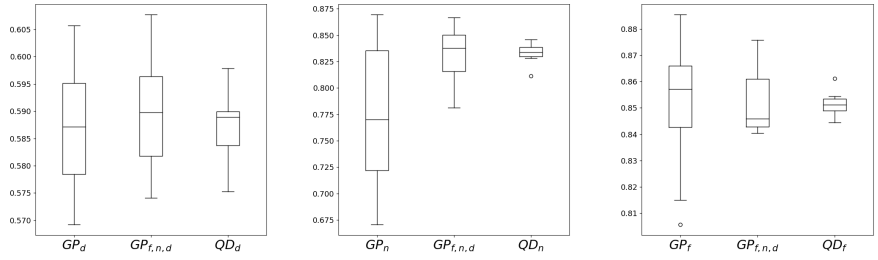
4.1 Comparison of Objective Fitness

While the main goal of the paper is to generate diverse repertoires of behaviours with respect to each of the three primitives, it is instructive to understand whether searching for diversity in addition to quality has an adverse impact on quality. Figure 4 shows box-plots over the 10 runs per objective value and algorithm. From a qualitative perspective, the QD approach appears more robust in that the variation across repeated runs is much smaller than the two GP methods. Statistical tests applied between each pair of algorithms per-objective show that the null-hypothesis cannot be rejected between any tests for the *go-to-food* and *density* objectives, whereas for the *go-to-nest* objective, a statistically significant difference is observed between GP_n and $GP_{f,n,d}$, and between GP_n and QD_n, with GP_n providing the poorer result in each case.

Our results clearly demonstrate that for two objectives (food, density), there is no evidence that the $GP_{f,n,d}$ approach, that aims to maximise all objectives simultaneously, results in any difference in quality when compared to results from the corresponding single objective GP variant. Furthermore, there is no evidence that evolving for both quality and diversity reduces the quality of the best solutions found for these two objectives.

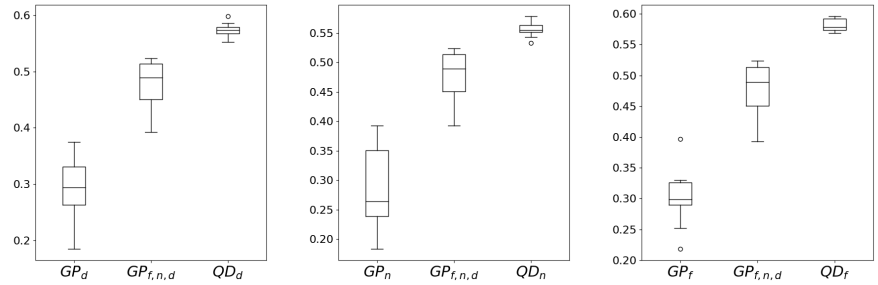
4.2 Coverage and QD Scores

Figures 5 and 6 show boxplots for the three respective metrics for each of three objectives. Results from pairwise statistical testing are also given in table 6. It is immediately clear that the QD method provides significantly better coverage



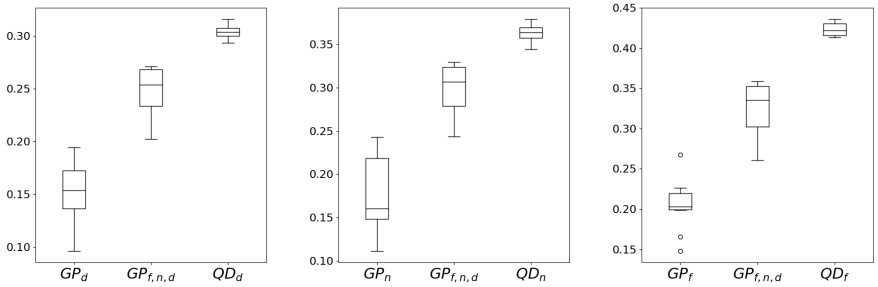
(a) Increase density (b) Move towards nest (c) Move towards food

Fig. 4: Box-plots of the best fitness obtained for all three algorithms and objectives with a design budget of 25,000 evaluations per objective and averaged over ten runs, or twenty runs where available



(a) Increase density (b) Move towards nest (c) Move towards food

Fig. 5: Coverage for each algorithm and objective with a design budget of 25,000 evaluations per objective and averaged over ten runs



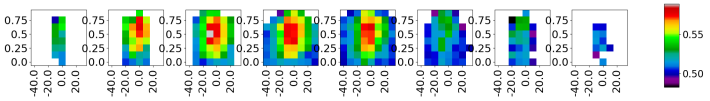
(a) Increase density (b) Move towards nest (c) Move towards food

Fig. 6: Quality diversity scores for each algorithm and objective with a design budget of 25,000 evaluations per objective and averaged over ten runs

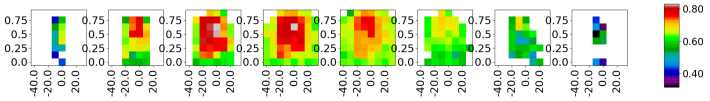
and QD score than the EA approaches, indicating that the method is able to

Table 6: Statistical testing results showing pairwise comparisons. Statistically significant results within a confidence interval of 0.05 are shown in bold.

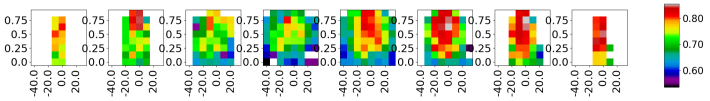
Metric	Objective	Comparison	p-value	Type of test
Best fitness	Increase density	GP_d vs $GP_{n,f,d}$	0.3505	T-test
		GP_d vs QD_d	0.9695	T-test
	Go to nest	GP_n vs $\boldsymbol{GP_{n,f,d}}$	**0.0004**	T-test
		GP_n vs $\boldsymbol{QD_n}$	**0.0066**	T-test
	Go to food	GP_f vs $GP_{n,f,d}$	0.4406	Mann-Whitney
		GP_f vs QD_f	0.9422	T-test
Coverage	Increase density	GP_d vs $\boldsymbol{GP_{n,f,d}}$	< 0.0001	T-test
		GP_d vs $\boldsymbol{QD_d}$	< 0.0001	T-test
	Go to nest	GP_n vs $\boldsymbol{GP_{n,f,d}}$	< 0.0001	T-test
		GP_n vs $\boldsymbol{QD_n}$	< 0.0001	T-test
	Go to food	GP_f vs $\boldsymbol{GP_{n,f,d}}$	< 0.0001	T-test
		GP_f vs $\boldsymbol{QD_f}$	< 0.0001	T-test
QD Score	Increase density	GP_d vs $\boldsymbol{GP_{n,f,d}}$	< 0.0001	T-test
		GP_d vs $\boldsymbol{QD_d}$	< 0.0001	T-test
	Go to nest	GP_n vs $\boldsymbol{GP_{n,f,d}}$	< 0.0001	T-test
		GP_n vs $\boldsymbol{QD_n}$	< 0.0001	T-test
	Go to food	GP_f vs $\boldsymbol{GP_{n,f,d}}$	< 0.0001	T-test
		GP_f vs $\boldsymbol{QD_f}$	< 0.0001	T-test



(a) QD_d: Increase density

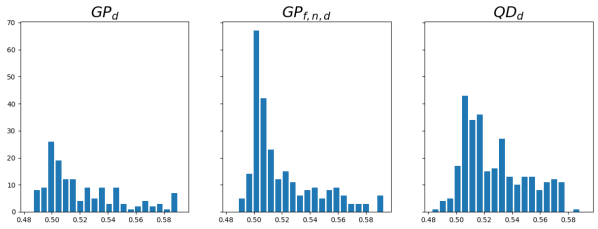


(b) QD_n: Go to nest

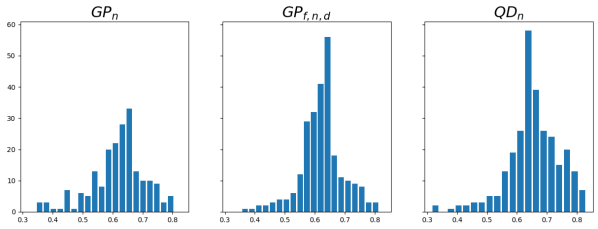


(c) QD_f: Go to food

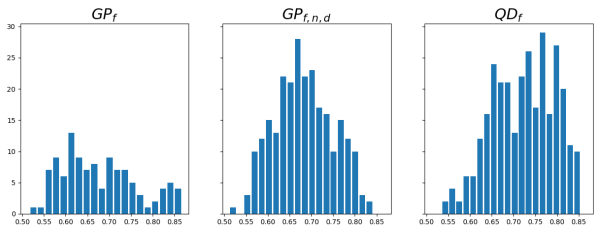
Fig. 7: Heatmaps for QD_o (three separate objectives)



(a) Increase density GP_d 148 solutions, $GP_{f,n,d}$ 255 solutions, QD_d 290 solutions



(b) Go to nest GP_n 201 solutions, $GP_{f,n,d}$ 255 solutions, QD_n 283 solutions



(c) Go to food GP_f 112 solutions, $GP_{f,n,d}$ 255 solutions, QD_f 305 solutions

Fig. 8: Histograms showing the distribution of fitness values per algorithm (one row per objective)

produce diversity in behaviour while also maintaining quality. In terms of the best solution found, the QD approach generally finds solutions that are at least on a par with the other methods. That is, there is no statistical evidence that solution task performance is different (section 4.1). Figure 7 provides a visualisation of the evolved QD container for each objective. Since the container has dimensions 8x8x8 it is presented as 8 separate 8x8 containers[5].

[5] Given the obvious difference between the QD and EA methods, we only show the QD containers given page limit constraints.

Figure 8 plots the distribution of fitness values in the repertoires generated by each algorithm for each objective from a single run of each. The figure also indicates the size of each repertoire. Although all methods produce a spread of values, the QD approaches produce repertoires that encapsulate many different behaviours that also have high quality (task performance). That is, the centre of mass of the distribution is shifted to the right in the QD case. Note that the $GP_{f,n,d}$ method finds much larger repertoires than the single objective GP method for all three objectives, and that the repertoires also contain solutions that tend to be of higher quality. One possible explanation for this is that by randomly selecting an evaluation function to score an individual during each round of tournament selection, some useful transfer of knowledge occurs which increases fitness. Similarly, Evolutionary Multi-Task Optimisation (EMTO) [31] methods have demonstrated that solving multiple tasks in combinatorial optimisation via a shared population can improve quality and is thus worthy of further investigation. Alternatively, the random choice of objective function per round of selection is likely to also encourage wider exploration of the search-space.

5 Conclusions and Further Work

The goal of this research was to compare methods for generating a repertoire of diverse behaviours for three primitive tasks that are useful components for synthesising foraging behaviours in swarms. To the best of our knowledge, this study was the first experimental comparison of various methods for generating diverse behaviours via *behaviour-tree* controllers, with respect to three different primitives. Our study compared a standard GP algorithm to a GP algorithm using an implicit diversity mechanism (to simultaneously evolve three behaviours), to a quality-diversity algorithm (MAP-Elites), using an explicit diversity mechanism.

Results showed, inline with existing literature that has evolved neural controllers, that QD is capable of evolving a diverse repertoire of high quality BT controllers. Furthermore, this study demonstrated that a GP algorithm using an implicit diversity mechanism is capable of evolving high (task-performance) quality controllers, although the repertoire size attained is smaller than that of the QD approach (figure 5). An advantage of using behaviour trees over neural networks is their readability. Using BT controllers in swarms therefore potentially offers a route to creating swarms with explainable behaviour in the future.

The evolved repertoires provide a platform for two avenues of future work. On the one hand, the diverse behaviours evolved for each primitive can be combined to deliver a broad range of complex foraging behaviours, for example, evolving a 'meta-BT' that uses the primitives themselves as nodes. Other high-level strategies might also be learned that utilise these primitives, for example, reinforcement learning. On the other hand, the existence of the evolved repertoires opens up the possibility of constructing swarms in which the robots have heterogeneous *behaviours*. This would require a search algorithm to discover useful compositions, but would provide new insights into the benefits of composing a swarm from a single high-performing behaviour or multiple behaviours that

are diverse with respect to quality. Composing swarms where each entity has a diverse form of movement also potentially offers some mitigation against faults or breakages and thus ensures swarm-robotic resilience.

References

1. Banerjee, B.: Autonomous acquisition of behavior trees for robot control. pp. 3460–3467 (10 2018). https://doi.org/10.1109/IROS.2018.8594083
2. Biggar, O., Zamani, M.: A framework for formal verification of behavior trees with linear temporal logic. IEEE Robotics and Automation Letters **5**(2), 2341–2348 (2020). https://doi.org/10.1109/LRA.2020.2970634
3. Bonani, M., Longchamp, V., Magnenat, S., Retornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., Mondada, F.: The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research pp. 4187–4193 (01 2010). https://doi.org/10.1109/IROS.2010.5649153
4. Colledanchise, M., Marzinotto, A., Ögren, P.: Performance analysis of stochastic behavior trees. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 3265–3272 (2014). https://doi.org/10.1109/ICRA.2014.6907328
5. Colledanchise, M., Natale, L.: Improving the parallel execution of behavior trees (09 2018). https://doi.org/10.1109/IROS.2018.8593504
6. Colledanchise, M., Ögren, P.: Behavior trees in robotics and ai: An introduction. CoRR **abs/1709.00084** (2017), http://arxiv.org/abs/1709.00084
7. Cully, A., Clune, J., Tarapore, D., Mouret, J.B.: Robots that can adapt like animals. Nature **521**(7553), 503–507 (2015)
8. Engebråten, S.A., Moen, J., Yakimenko, O., Glette, K.: Evolving a repertoire of controllers for a multi-function swarm. In: International Conference on the Applications of Evolutionary Computation. pp. 734–749. Springer (2018)
9. Fortin, F.A., De Rainville, F.M., Gardner, M., Parizeau, M., Gagné, C.: Deap: Evolutionary algorithms made easy. Journal of Machine Learning Research, Machine Learning Open Source Software **13**, 2171–2175 (07 2012)
10. Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Mascia, F., Trianni, V., Birattari, M.: Automode-chocolate: automatic design of control software for robot swarms. Swarm Intelligence **9** (06 2015)
11. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: Automode: A novel approach to the automatic design of control software for robot swarms. Swarm Intell **8**, 1–24 (06 2014). https://doi.org/10.1007/s11721-014-0092-4
12. Giunchiglia, E., Colledanchise, M., Natale, L., Tacchella, A.: Conditional behavior trees: Definition, executability, and applications. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). pp. 1899–1906 (2019). https://doi.org/10.1109/SMC.2019.8914358
13. Gomes, J., Christensen, A.L.: Task-agnostic evolution of diverse repertoires of swarm behaviours. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) Swarm Intelligence. pp. 225–238. Springer International Publishing, Cham (2018)
14. Hallawa, A., De Roose, J., Andraud, M., Verhelst, M., Ascheid, G.: Instinct-driven dynamic hardware reconfiguration: Evolutionary algorithm optimized compression for autonomous sensory agents. pp. 1727–1734 (07 2017). https://doi.org/10.1145/3067695.3084202

15. Hart, E., Steyven, A.S., Paechter, B.: Evolution of a functionally diverse swarm via a novel decentralised quality-diversity algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 101–108 (2018)
16. Hong, L., Page, S.E.: Groups of diverse problem solvers can outperform groups of high-ability problem solvers. Proceedings of the National Academy of Sciences **101**(46), 16385–16389 (2004)
17. Husbands, P., Harvey, I.: Evolution versus design: Controlling autonomous robots. In: Proceedings of the Third Annual Conference of AI, Simulation, and Planning in High Autonomy Systems' Integrating Perception, Planning and Action'. pp. 139–140. IEEE Computer Society (1992)
18. Jones, S., Studley, M., Hauert, S., Winfield, A.: Evolving Behaviour Trees for Swarm Robotics, pp. 487–501. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-73008-0_34, https://doi.org/10.1007/978-3-319-73008-_34
19. Jones, S., Studley, M., Hauert, S., Winfield, A.: A two teraflop swarm. Frontiers in Robotics and AI **5**, 11 (02 2018). https://doi.org/10.3389/frobt.2018.00011
20. Jones, S., Winfield, A., Hauert, S., Studley, M.: Onboard evolution of understandable swarm behaviors. Advanced Intelligent Systems **1** (07 2019). https://doi.org/10.1002/aisy.201900031, https://doi.org/10.1002/aisy.201900031
21. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: International Conference on Swarm Intelligence. pp. 30–43. Springer (2018)
22. Lehman, J., Stanley, K.O.: Evolving a diversity of virtual creatures through novelty search and local competition. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. pp. 211–218. ACM (2011)
23. Marzinotto, A., Colledanchise, M., Smith, C., Ogren, P.: Towards a unified behavior trees framework for robot control. pp. 5420–5427 (05 2014). https://doi.org/10.1109/ICRA.2014.6907656
24. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: Proceedings of the 9th conference on autonomous robot systems and competitions. vol. 1, pp. 59–65. IPCB: Instituto Politécnico de Castelo Branco (2009)
25. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909 (2015)
26. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G.A.D., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: Argos: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**, 271–295 (2012)
27. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: A new frontier for evolutionary computation. Frontiers in Robotics and AI **3**, 40 (2016)
28. Pugh, J.K., Soros, L.B., Szerlip, P.A., Stanley, K.O.: Confronting the challenge of quality diversity. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 967–974 (2015)
29. Scheper, K., Tijmons, S., De Visser, C., Croon, G.: Behavior trees for evolutionary robotics. Artificial life **22** (02 2016). https://doi.org/10.1162/ARTL_a_00192
30. Trianni, V.: Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots, vol. 108. Springer (2008)
31. Wei, T., Wang, S., Zhong, J., Liu, D., Zhang, J.: A review on evolutionary multitask optimization: Trends and challenges. IEEE Transactions on Evolutionary Computation (2021)