A Hierarchical Approach to Evolving Behaviour-Trees for Swarm Control

Kirsty Montague[1], Emma Hart[1][0000−0002−5405−4413], and Ben Paechter[1][0000−0002−4841−0805]

Edinburgh Napier University,
{k.montague,e.hart,b.paechter}@napier.ac.uk

Abstract. Behaviour trees (BTs) are commonly used as controllers in robotic swarms due their modular composition and to the fact that they can be easily interpreted by humans. From an algorithmic perspective, an additional advantage is that extra modules can easily be introduced and incorporated into new trees. Genetic Programming (GP) has already been shown to be capable of evolving BTs to achieve a variety of sub-tasks (primitives) of a higher-level goal. In this work we show that a hierarchical controller can be evolved that first uses GP to evolve a repertoire of primitives expressed as BTs, and then to evolve a high-level BT controller that leverages the evolved repertoire for a foraging task. We show that the hierarchical approach that uses BTs at two levels outperforms a baseline in which the BTs are evolved using only low-level nodes. In addition, we propose a method to improve the quality of the primitive repertoire, which in turn results in improved high-level BTs.

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

1 Introduction

Collective intelligence arises in a swarm via the interaction of multiple agents that act individually according to their current perception of the environment. Many approaches to designing a control mechanism that results in a desired behaviour at the swarm level exist in the literature. A common means of control is to use a *hierarchical controller* in which a set of low-level control modules referred to as *primitives* are combined into a complex controller, referred to as an *arbitrator* [11, 6, 14, 21]. The low-level modules (primitives) can either be hand-designed [11] or auto-designed, e.g. using evolution to evolve a neural-network controller [6, 14] or a behaviour-tree (BT) [24]. Quality-diversity approaches such as MAP-Elites and Novelty search are increasingly being used to generate repertoires of primitives: Montague *et. al.* [24] use genetic programming combined with MAP-Elites to generate BT primitives for a foraging task, while in [8, 15] a repertoire of neural controllers is generated using novelty-search and neuro-evolution. Arbitrators can also take many different forms: the AutoMoDe family of controllers mainly uses probabilistic finite-state machines (PFSMs) [15], while

BTs are used in [17–19, 21]. Neural networks can also be evolved as arbitrators, e.g. [13].

It is clear from the above that in developing a hierarchical controller, there are many choices in terms of the representation of both primitives and arbitrators. We suggest that BTs are an obvious choice for describing both primitives and arbitrators. A BT is itself a hierarchical model which consists of actions, conditions, and operators connected by directed edges [4]. They are modular in the sense that the set of actions available to the tree can be easily modified, any part of the tree can be extracted and reused, and the modules themselves can be generated by multiple means. In addition, although neural controllers are more common, they are not well-suited to crossing the reality-gap (i.e. obtaining consistent performance between a simulated experiment and a physical experiment) due to the fine-tuned precision obtained in simulation. Conversely, Francesca *et. al.* [11] propose that increasing bias by constraining the evolutionary search to pre-defined behaviour modules reduces variance and therefore sensitivity to the reality gap. Furthermore, a neural network can also be difficult to analyse or modify whereas BTs are human-readable and therefore go some way towards being explainable [16].

In this paper, we build on an existing line of work in hierarchical controller development by using evolutionary methods to develop a hierarchical control system for a swarm which has an *interpretable* controller. Specifically, we use two evolutionary methods to learn a hierarchical controller whose primitives and arbitrator *are both* represented as BTs: to the best of our knowledge, there is no existing hierarchical controller of this form. We leverage a set of primitives which are evolved to fulfil several manually defined objectives for this purpose, using (1) a multi-task version of GP (MTGP) and (2) MAP-Elites, both described by [24]. We first extend the work of [24] in improving the quality of the primitive set learned by MTGP by introducing the notion of *compatible objectives* (see section 3.2). We then use GP to evolve a BT arbitrator for a foraging task, comparing the use of the different primitives' repertoires evolved in the previous step as input. We find that the arbitrator using primitives as nodes significantly outperforms a learned controller that uses low-level behaviours directly. Secondly, our results demonstrate that using a repertoire that contains multiple, diverse versions of each primitive leads to higher performing arbitrators than using a repertoire containing only a single high-performing primitive for each of the desired sub-behaviours.

2 Background

Hierarchical forms of control in which a desired task is decomposed into a set of simpler sub-tasks are common in many areas of robotics. Typically, an *arbitrator* is designed that selects *primitives* that execute individual behaviours, where the arbitrator can be informed by inputs from the environment or the robot's perceptions.

A long line of work in developing hierarchical controllers was spawned with AutoMoDe [11] and its sequence of successors [3, 10, 19–22]. This family of methods almost all generate probabilistic finite-state machine (PFSM) arbitrators, optimised using iterated F-race [23], with some exceptions e.g. IcePop [20] which uses Simulated Annealing as the optimiser.

In another example Cully *et. al.* [6] evolve a large set of walking gaits using a quality-diversity algorithm (MAP-Elites) for a legged robot, and an arbitrator (using Bayesian optimisation) selects the most appropriate primitive given the state of the robot and environment. Duarte *et. al.* [7] propose EvoRBC which also uses a QD algorithm (Novelty-Search) to first evolve a repertoire of low-level locomotion patterns (represented as vectors of parameters supplied to the robot's actuation system), and then to evolve a neural-network which acts as an arbitrator. EvoRBC-II extended the EvoRBC approach to include the use of closed-loop primitives. In the previously mentioned work, the *goal* of each primitive is hand-designed: that is, the decomposition of the desired high-level behaviour into primitives is performed by a human with knowledge of the desired task, e.g. for a foraging task, specifying primitives such as 'go-to-food' or 'go-to-nest'. In [15], a new approach is proposed which is mission agnostic, i.e. does not rely on the definition of task specific primitives. Their framework 'Nata' automatically generates probabilistic finite-state machines (arbitrators) in which states are selected from a repertoire of neural networks, and transition conditions are selected from a set of rules based on the sensory capabilities of the robotic platform considered [15]. A QD method is again used to generate a repertoire, after which Iterated F-Race [23] is used to assemble them into PFSMs.

Many of the methods just mentioned use neural network controllers either to create the repertoire of primitives or as the arbitrator. However, some concerns have been raised that such finely tuned precision is not suited to crossing the reality gap [11] while an additional concern is that a neural-network is a black-box, i.e. it is difficult to analyse or modify. PFSMs go someway towards addressing this criticism but require a compromise between reactivity and modularity: they cannot easily be broken down into their constituent parts because of dependencies between components and they do not scale well as the number of states grows [5]. On the other hand, Behaviour Trees (BTs) have an inherent capacity to reproduce the same functionality as PFSMs [4], but they maintain independence between components which removes these trade-offs and constraints. As noted by [16], they are also human-readable and therefore can be useful in explaining behaviours. Montague *et. al.* [24] proposed a method of evolving BT primitives using GP, exploring a multi-task GP method as well as MAP-Elites, but did not extend this to evolving an arbitrator. MAPLE [18] and Cedrata [19], both from the AutoMoDe family, use iterated F-Race to evolve BT arbitrators but not primitives. In Kuckling *et. al.* [21], two new variants of Cedrata are proposed, Cedrata-GP and Cedrata-GE which are based on genetic programming and grammatical evolution, respectively. The performance of the evolved BTs is compared against the performance of solutions created by a human designer, showing that Cedrata finds solutions that are also reliably found

by human designers. However, the automatic design methods fail to discover the same communication strategies as the human designers.

In this paper we propose an approach in which for the first time both the primitives and arbitrator are represented as BTs, leading to increased transparency in interpreting them. We first extend the work described in [24] that uses GP and QD methods to evolve a repertoire of primitives to improve the quality of the repertoire. Then we evolve a BT arbitrator that leverages this repertoire using GP, evaluating it on a foraging task that is common in swarm robotics.

.

3 Methodology

The goal of this work is to evolve a hierarchical controller for a foraging task which is composed of BTs at both the primitive and arbitrator level, evolved by GP in both cases. We build directly on previous work by Montague *et. al.* [24] which demonstrated that GP could be used to generate BT primitives for a foraging task but stopped short of generating the high-level arbitrator. We make the following contributions:

- Evolve an extended set of primitives to enlarge the repertoire available to the arbitrator using (1) MAP-Elites in conjunction with GP (a Quality Diversity algorithm denoted QD) and (2) a multi-task GP method (denoted MTGP[1]) that simultaneously evolves for multiple task fitnesses using an implicit diversity mechanism. Specifically, we extend the set of primitives from *increase-neighbourhood-density*, *go-to-nest* and *go-to-food* to include *reduce-neighbourhood-density*, *go-away-from-nest* and *go-away-from-food*.
- Propose an approach to improve the quality of the repertoires generated by the multi-task method MTGP that only considers *compatible* objectives in its task set, i.e. does not include for example *go-to-food* and *go-away-from-food* which cannot be satisfied by the same controller.
- Use GP to evolve a high-level BT controller for a foraging task leveraging the primitive repertoires as input, comparing repertoires created by the different methods outlined above.

3.1 Setup

We consider a foraging task in which the objective is for each robot in the swarm to visit the food region and then the nest region as many times as possible over the course of each simulated trial. A more detailed description can be found in section 4.2.

As per [24], a swarm is composed of nine footbot robots (figure 1, [1]) deployed in the arena shown in figure 2. Controllers are evaluated using the ARGoS

[1] Note that the authors in [24] referred to this method as e.g. $GP_{O1,O2,O3}$ however we believe it is more correctly described as a multi-task algorithm, e.g. [26]

simulator [25]. We use the same set up as described in [24], where the robots navigate by estimating the distance and direction of points of interest using information from their neighbours, while a blackboard provides an interface between the BT controllers and the footbots' sensor data. The reader is referred to the publication of [24] for full details.

In each set of experiments (to evolve primitives or arbitrators) each controller is evaluated over ten trials with randomised starting positions divided between two predefined arena configurations. The only difference in the way that arbitrators are evaluated compared with the primitives is that the length of each trial is increased from 20 seconds to 100 seconds.

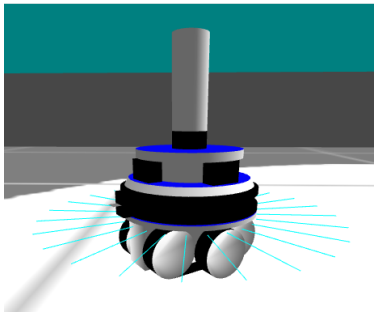


Fig. 1: A screenshot of a footbot robot in the arena taken in ARGoS.

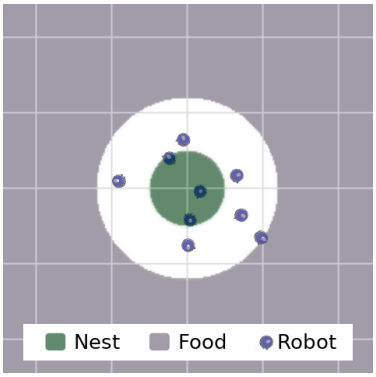


Fig. 2: The arena layout, with nine robots initialised in random starting positions.

In all of our experiments we evolve BT controllers with GP implemented using DEAP [9]. For the quality-diversity approach we combine GP with MAP-Elites using QDpy[2]. We use the same evolutionary parameters and BT implementation as described in [24], except that we add a new condition - *ifGotFood* - which indicates whether the robot has visited the food region since its last visit to the nest region. In doing so, we introduce a new internal state. Tables 1 and 2 list the nodes for evolving primitives for ease of reference. The reader is referred to [24] for detailed descriptions of the algorithms and GP implementation.

Each algorithm is run with ten different random seeds for each objective (or combination of objectives) for 1000 generations. The GP population size and the MAP-Elites batch size are both set to 25, while MTGP is assigned a population of 75 to reflect that it generates controllers for three objectives at once (therefore does not have to be run 3 times as with the other methods). All parameter settings are taken from [24]. Performance for each primitive is defined according to the median over 10 runs of the metrics described in [24] for the three

[2] https://pypi.org/project/qdpy/

primitive behaviours defined in [24] and for the three new primitive behaviours as defined in Section 3.2 which we introduce in this paper. The fitness of a BT arbitrator is defined in Section 4.2.

3.2 Evolving new primitives

As noted above, we first use Map-Elites (denoted QD from herein) and MTGP to evolve three new primitives which provide the opposite behaviour to the original primitives, *increase density*, *go to nest* and *go to food*. The motivation behind this is to increase the number of options available to the arbitrator which in turn might find better behaviours. We opted for 'obvious' objectives at this stage, but there is clearly room for considering either further hand-crafted ones or auto-generating them in future work. These primitives are described in detail below:

- **Decrease neighbourhood density** maximises the difference between the density of neighbouring robots at the beginning and end of each trial, which we calculate by subtracting the final density from the initial density.

- **Move away from the nest region** maximises the difference in distance estimated by each robot at the start and end of each trial based on the shortest route by hops via neighbouring robots. The difference is calculated by subtracting the final distance from the initial distance.

- **Move away from the food region** maximises the difference in the robots' absolute distance to the food region at the beginning and end of each trial, calculated by subtracting the initial distance from the final distance.

These primitives are evolved using the same nodes as in [24], as shown in tables 1 and 2.

Table 1: 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 2: 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

The choice of primitives to be evolved can cause issues for MTGP: at each iteration, this algorithm randomly selects one of the objective functions and assigns a fitness based on the chosen function. This encourages generalisation and was shown by [24] to improve performance for some objectives compared to a GP algorithm that evolved for each objective individually. However, it should be clear that some objectives are incompatible as previously mentioned. We therefore evaluate two versions of MTGP: one in which only compatible objectives are used, and another which includes objectives which are mutually exclusive. Hence, the following algorithms for evolving primitives are compared:

- **GP**: A baseline GP algorithm that evolves controllers for one objective at a time and is repeated for each primitive.

- **MTGP**: An algorithm that evolves controllers for multiple objectives at once, selecting one objective at random as the fitness function for each tournament used to select parents. We compare its performance using both *incompatible* (dubbed MTI) and *compatible* (dubbed MTC) combinations of objectives.

- **QD**: A MAP-Elites algorithm that evolves a collection of solutions for one objective at a time whose behaviours are diverse with respect to a set of user-defined characteristics. The characteristics which distinguish them are taken from [24] and are: difference in the ratios of forwards and backwards movement; ratios of clockwise and anti-clockwise rotations; the ratio of condition nodes and action nodes which are executed during simulation.

We define (*increase density, go to nest* and *go away from food*) as one set of compatible objectives, and (*reduce density, go away from nest* and *go to food*) as another[3]. We also define two sets of incompatible objectives: (*increase density, go to nest* and *go to food*) and *(reduce density, go away from nest* and *go away from food).*

[3] Obviously objectives such as *increase density* and *decrease density* are mutually exclusive and therefore are never considered together

3.3 Evolving an Arbitrator

To evolve a high-level foraging behaviour that leverages a repertoire of primitives evolved above, we use the single objective GP algorithm denoted GP above. This is exactly the same algorithm proposed by [24] except that the low-level action nodes are replaced by the primitives in the chosen repertoire and the set of condition nodes is restricted. We use the same evolutionary parameters as used to evolve the primitive repertoire, maintaining the population size of 25 individuals and running the algorithm for 1000 generations. Foraging also requires longer simulations so we increase the length of each trial in the arena from 20 seconds for primitives to 100 seconds for the arbitrator.

The objective function for the arbitrator rewards robots for each visit to the nest region which follows a visit to the food region. Upon visiting the food region, a robot is considered to be carrying food. If it then enters the nest region it reverts back to its default state and its score is incremented by one.

The fitness score S is defined as the number of times any robot carrying food arrives in the nest f divided by the number of robots in the arena r (which is nine in these experiments), i.e. $S = f/r$

We compare the following repertoires as input to this method:

- R1: the highest performing behaviour for each primitive found by QD.
- R2: the highest performing behaviour for each primitive returned by MTGP using compatible behaviours[4].
- R3: a repertoire containing multiple diverse BTs for each primitive. This is obtained by dividing the whole container returned by MAP-Elites into two equally sized bins along each axis, selecting the best controller from each of the eight resulting bins. This results in 8 behaviours for each of the 6 objectives, i.e. a total of 48 action nodes in the repertoire.
- R4: a repertoire containing eight diverse BTs for each primitive obtained by casting all individuals found by MTGP to the same MAP-Elites grid and dividing the axes in the same way, retrieving the best BT from each of the resulting eight bins.
- R5: a baseline experiment which uses the low-level action nodes used to evolve the primitives.

Note that using the repertoires of evolved primitives makes several of the condition nodes used by [24] obsolete. For example, condition nodes concerned only with navigation such as 'ifRobotToLeft/Right' are irrelevant at the arbitrator's level of abstraction and are therefore removed from the condition lists. This leaves just three condition nodes: (1) Is this robot in the food region; (2) Is this robot in the nest region; (3) Is this robot carrying food. The mutation operators insert condition or action nodes with equal probability.

[4] Experiments in Section 4.1 showed that the performance of MTGP using compatible objectives was often better than using incompatible objectives

4 Results

We first evaluate the proposed approaches for improving the primitive repertoires, i.e. by adding additional objectives, and using two versions of MTGP with compatible or incompatible subsets of objectives.

4.1 Extending and improving the primitive repertoire

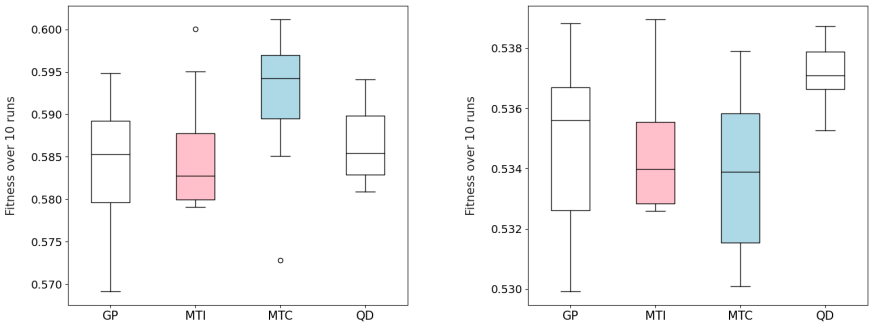
Figure 3 shows boxplots of fitness over 10 repetitions of the performance of each algorithm listed in Section 3.2 for each of the six primitive behaviours evolved. To compare pairs of algorithms, a Shapiro-Wilk test was performed to check for normality, after which either a Student t-test if the data was judged to be normal or a Mann-Whitney test otherwise. The results of these tests are shown in table 3: a confidence level of 0.05 is used to test for significance.

Table 3: Statistical testing results showing pairwise comparisons for different combinations of objectives. Statistically significant results within a confidence interval of 0.05 are shown in bold. The type of test applied is shown in italics: italicised = Mann-Whitney, non-italics=T-test

	GP vs MTI	*GP vs MTC*	*GP vs QD*	*MTI vs MTC*	*MTI vs QD*	*MTC vs QD*
Increase density	*0.7913*	0.0580	0.5575	*0.0640*	*0.3847*	0.0760
Go to nest	*0.4727*	***0.0452***	*0.5708*	0.2204	*0.0757*	***0.0257***
Go to food	*0.3217*	0.3240	**0.0270**	*0.0526*	***0.0168***	0.2199
Reduce density	0.8067	0.3838	**0.0352**	0.4535	**0.0037**	**0.0018**
Go away from nest	***0.0172***	***0.0211***	***0.0017***	0.6980	0.9482	0.6372
Go away from food	**0.0009**	**0.0259**	0.2002	**<0.0001**	**<0.0001**	0.1414

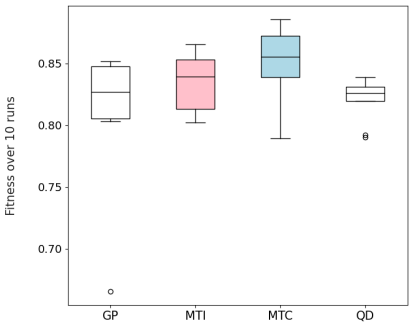
Table 4: Median values for each algorithm with the highest median in bold

	GP	*MTI*	*MTC*	*QD*
Increase density	0.585309	0.582788	**0.594218**	0.585442
Go to nest	0.826672	0.839357	**0.855242**	0.826128
Go to food	0.847747	0.842908	0.850078	**0.856015**
Reduce density	0.535612	0.533996	0.533903	**0.537092**
Go away from nest	0.792973	0.811753	0.803499	**0.813578**
Go away from food	0.624641	0.609507	**0.638132**	0.629626

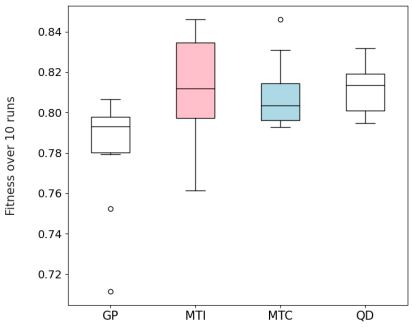


(a) Increase density

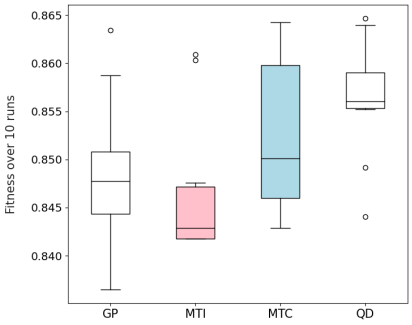
(b) Reduce density



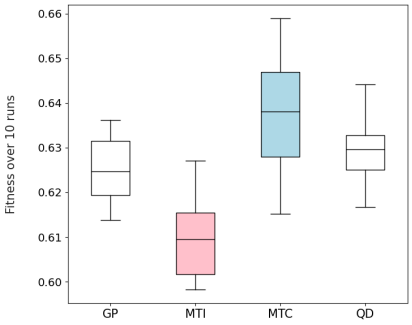
(c) Go to nest



(d) Go away from nest



(e) Go to food



(f) Go away from food

Fig. 3: Box-plots of the best fitness obtained for all three algorithms including two variations of the multi-task algorithm (MTI/MTC). Where its three objectives are compatible the performance is shown in blue; subsets of incompatible objectives are shown in red.

Figure 3 shows that the highest median performance is obtained by MTC for three objectives (*increase density*, *go to nest*, *go away from food*) and by QD for the remaining three (*reduce density*, *go away from nest*, *go to food*). However, as shown in table 3, the result is not always significant[5]. For four objectives, the median performance obtained from the compatible version of MTGP is higher than that of the incompatible version, although again the difference is not significant except in the case of *go away from food*. Surprisingly, for the *go away from nest* and *reduce density* objectives, the incompatible version of MTGP produces a higher median than its compatible counterpart although its variance is much higher. For *go away from food* and *go away from nest*, the incompatible version of MTGP performs significantly worse than the baseline.

Based on the results just described, we decide to discard the repertoires obtained with the incompatible version of MTGP and proceed to evolve a high-level arbitrator using only repertoires obtained from MTC and QD. These results are described in the next section.

4.2 Foraging Experiments

We compare BT arbitrators evolved using GP from the four different repertoires obtained by the methods described in Section 4.2. Recall that the goal is to determine: (1) if the hierarchical approach outperforms a baseline that evolves a single BT using the low-level nodes in table 2; (2) which repertoire of primitives results in the best performing arbitrator.

Figure 4 shows boxplots of results over 10 repeated experiments. Statistical test results are presented in table 5. It is immediately clear from Fig. 4 that all experiments using repertoires of evolved primitives outperform the baseline[6] (the first four entries in table 5). This confirms that a hierarchical approach which leverages a repertoire of pre-evolved primitives is preferable to directly evolving an arbitrator using low-level actions. The best median fitness is obtained using a repertoire containing 8 diverse behaviours per objective (QD_8). MT_8 provides similar performance, suggesting that having a diverse repository of primitives including multiple behaviours that optimise the same primitive is preferable to simply using the single best primitive available for an objective in the repertoire. Recall that QD produces the highest median fitness for a behaviour for 3 primitives, and MTC for the remaining three primitives. Hence it is unsurprising that QD_8 and MT_8 have similar performance, as they are both able to exploit good repertoires. In the same vein, QD_1 and MT_1 have similar performance in terms of the quality of the primitives in the repertoire, leading to similar quality arbitrators.

[5] Further work should increase the number of runs from the 10 performed to ascertain whether we should be confident in this result

[6] All experiments were run for the same amount of computational time taking into account the time taken to evolve the primitives: thus the baseline experiments are run for more generations than the arbitrator

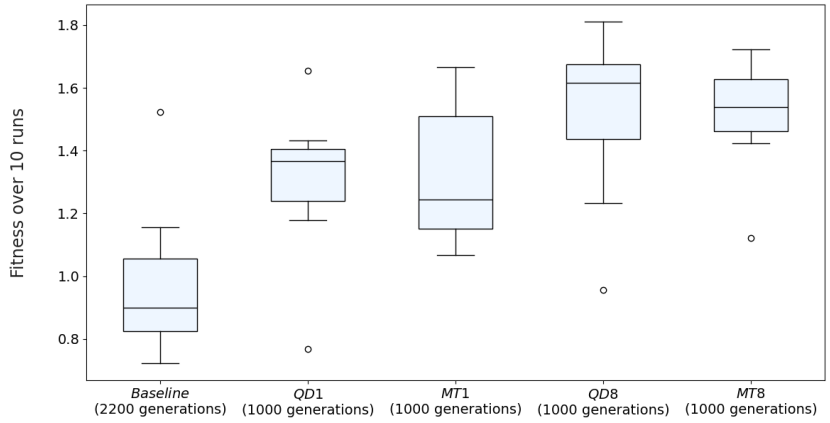


Fig. 4: A baseline algorithm which evolves a foraging behaviour from primitive actions nodes (go forwards, etc) compared with ones that use the best of each of the sub-behaviours from the QD and MTGP repertoires instead, and ones which use eight versions of each sub-behaviour from each of those repertoires.

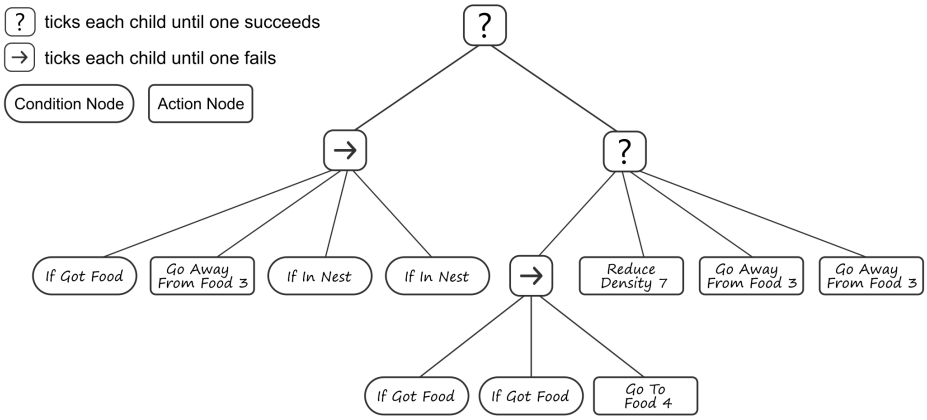
Table 5: Statistical testing results showing pairwise comparisons for foraging. Statistically significant results within a confidence interval of 0.05 are shown in bold.

Comparison	p-value	Type of test
Baseline vs QD repertoires of one	**0.0049**	**T-test**
Baseline vs MT repertoires of one	**0.0032**	**T-test**
Baseline vs QD repertoires of eight	**0.0001**	**T-test**
Baseline vs MT repertoires of eight	**<0.0001**	**T-test**
QD repertoires of one vs MT repertoires of one	0.9302	T-test
QD repertoires of eight vs MT repertoires of eight	0.9822	T-test
QD repertoires of one vs QD repertoires of eight	0.0665	T-test
MT repertoires of one vs MT repertoires of eight	**0.0316**	**T-test**
QD repertoires of one vs MT repertoires of eight	**0.0312**	**T-test**
MT repertoires of one vs QD repertoires of eight	0.0701	T-test

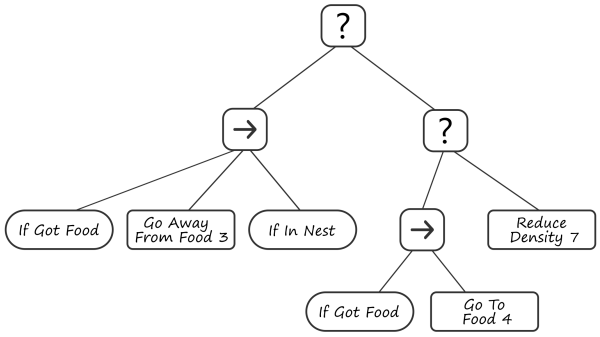
4.3 Readability

One of the main advantages of using BTs as opposed to NNs is that they are amenable to being understood by humans. Figure 5b shows an example of one of the high fitness BTs evolved, first in the full form return by the GP algorithm and then with its redundant nodes pruned by hand. The latter can be interpreted as follows: 'If you have food *go away from food*, and then if you are not in the nest, or you did not have food, check again if you have food. If you do then *go to food*, otherwise *reduce density*'.

A cursory examination will reveal that the use of *go to food* seems nonsensical. However, there is no requirement for the arbitrator to use the sub-behaviours



(a) A verbose tree for foraging. Some of the condition nodes are duplicated and all action nodes return success, so any subsequent children of a select node (denoted "?") will never be reached.



(b) The same tree with redundant nodes removed.

Fig. 5: Generated with repertoires of eight BTs per sub-behaviour from the QD repertoire.

for the purpose imagined by the designer or rewarded by the fitness function: for example we could speculate that *go to food* in figure 5b is being used simply to propel the robot forwards or backwards, since this is often what 'going to food' amounts to. This in itself is a useful behaviour.

5 Conclusions and Further Work

This paper builds on a line of work that uses a hierarchical method of developing a control system for a swarm of robots. At the primitive level, a set of controllers are created that optimise sub-tasks of the desired goal. A higher-level controller

known as an arbitrator then combines the previously generated primitives into a controller that executes the defined goal. Although the use of hierarchical methods is well-known (particularly regarding the AutoMoDe [12] series of control software), previous methods have tended to use neural-networks or PFSMs as arbitrators, with a small number of recent papers proposing BTs [21]. In this paper, we propose a method that uses BTs at both levels of the hierarchy, i.e. to evolve the primitives and then the arbitrator. As noted by [16], BTs offer considerably more explanatory power than neural-networks.

Building on previous work by Montague *et. al.* [24] that proposed using BTs to evolve primitives, we extend this work in several ways. First, we extended the set of sub-tasks described in [24] to provide new primitives that could be useful in a foraging task. Secondly we proposed an amendment to the multi-task GP approach proposed in [24] that only considers compatible behaviours when generating multiple primitives simultaneously. Finally, we evolved an arbitrator as a BT using GP that exploits the new evolved repertoires, showing that repertoires that contain multiple BTs per primitive that achieve the same objective in different ways produce the highest performing controllers. We provide an example of a BT to illustrate that it can be easily read and analysed to understand the evolved behaviour.

There is much potential for future work. Rather than evolving primitives then an arbitrator sequentially, a meta-evolutionary algorithm could be used to search for the set of primitives that produce the best arbitrator, following a similar process to [2]. While BTs are inherently readable, it would be interesting to investigate the trade-off between readability and performance: replacing the BT arbitrator with a neural-network or PFSM and repeating the experiment would illustrate any such trade-off. The function of each of the primitives at the lower level of the hierarchy is human-designed, as are the action and condition nodes used by the GP algorithm to evolve primitives. A first step in removing the need for human expertise has recently been described by Hasslemann *et. al.* [15] which tries to automatically define primitives. This type of approach could also be integrated with our proposed methodology. Finally, we proposed a first naïve approach to selecting primitives for a repertoire from the much larger container of solutions generated by both the QD and MTGP algorithms. An approach that tried to maximise diversity might yield better results, or could itself be subject to a search process, given that large containers are generated. Finally, repeating the experiments in other collective tasks would provide further insights into the generality of the approach.

References

1. 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
2. Bossens, D.M., Mouret, J.B., Tarapore, D.: Learning behaviour-performance maps with meta-evolution. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference. p. 49–57. GECCO '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3377930.3390181, https://doi.org/10.1145/3377930.3390181
3. Cambier, N., Ferrante, E.: Automode-pomodoro: an evolutionary class of modular designs. pp. 100–103 (07 2022). https://doi.org/10.1145/3520304.3529031
4. Colledanchise, M., Ögren, P.: How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. IEEE Transactions on Robotics **33**(2), 372–389 (2017). https://doi.org/10.1109/TRO.2016.2633567
5. Colledanchise, M., Ögren, P.: Behavior trees in robotics and ai: An introduction. CoRR **abs/1709.00084** (2017), http://arxiv.org/abs/1709.00084
6. Cully, A., Clune, J., Tarapore, D., Mouret, J.B.: Robots that can adapt like animals. Nature **521**(7553), 503–507 (2015)
7. Duarte, M., Gomes, J., Oliveira, S., Christensen, A.: Evorbc: Evolutionary repertoire-based control for robots with arbitrary locomotion complexity (07 2016). https://doi.org/10.1145/2908812.2908855
8. Duarte, M., Gomes, J., Oliveira, S.M., Christensen, A.L.: Evolution of repertoire-based control for robots with complex locomotor systems. IEEE Transactions on Evolutionary Computation **22**(2), 314–328 (2018). https://doi.org/10.1109/TEVC.2017.2722101
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. 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 Intelligence **8**, 89–112 (2014)
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. Gomes, J., Oliveira, S.M., Christensen, A.L.: An approach to evolve and exploit repertoires of general robot behaviours. Swarm and Evolutionary Computation **43**, 265–283 (2018)
15. Hasselmann, K., Ligot, A., Birattari, M.: Automatic modular design of robot swarms based on repertoires of behaviors generated via nov-

elty search. Swarm and Evolutionary Computation **83**, 101395 (08 2023). https://doi.org/10.1016/j.swevo.2023.101395
16. Hogg, E., Hauert, S., Harvey, D., Richards, A.: Evolving behaviour trees for supervisory control of robot swarms. Artificial Life and Robotics **25**, 569–577 (2020)
17. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) Swarm Intelligence. pp. 30–43. Springer International Publishing, Cham (2018)
18. 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)
19. Kuckling, J., van Pelt, V., Birattari, M.: Automatic modular design of behavior trees for robot swarms with communication capabilites. In: Castillo, P.A., Jiménez Laredo, J.L. (eds.) Applications of Evolutionary Computation. pp. 130–145. Springer International Publishing, Cham (2021)
20. Kuckling, J., Ubeda Arriaza, K., Birattari, M.: Automode-icepop: Automatic modular design of control software for robot swarms using simulated annealing. In: Bogaerts, B., Bontempi, G., Geurts, P., Harley, N., Lebichot, B., Lenaerts, T., Louppe, G. (eds.) Artificial Intelligence and Machine Learning. pp. 3–17. Springer International Publishing, Cham (2020)
21. Kuckling, J., Van Pelt, V., Birattari, M.: Automode-cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities. SN Computer Science **3**(2), 136 (2022)
22. Ligot, A., Hasselmann, K., Birattari, M.: Automode-arlequin: Neural networks as behavioral modules for the automatic design of probabilistic finite-state machines. In: Dorigo, M., Stützle, T., Blesa, M.J., Blum, C., Hamann, H., Heinrich, M.K., Strobel, V. (eds.) Swarm Intelligence. pp. 271–281. Springer International Publishing, Cham (2020)
23. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). https://doi.org/https://doi.org/10.1016/j.orp.2016.09.002, https://www.sciencedirect.com/science/article/pii/S2214716015300270
24. Montague, K., Hart, E., Nitschke, G., Paechter, B.: A quality-diversity approach to evolving a repertoire of diverse behaviour-trees in robot swarms. In: Correia, J., Smith, S., Qaddoura, R. (eds.) Applications of Evolutionary Computation. pp. 145–160. Springer Nature Switzerland, Cham (2023)
25. 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)
26. 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)