On Synergies between Diversity and Task Decomposition in Constructing Complex Systems with GP

Jessica P. C. Bonson
Dalhousie University,
Halifax, NS. Canada
jpbonson@gmail.com

Stephen Kelly
Dalhousie University,
Halifax, NS. Canada
skelly@cs.dal.ca

Andrew R. McIntyre
Dalhousie University,
Halifax, NS. Canada
armcnty@cs.dal.ca

Malcolm I. Heywood
Dalhousie University,
Halifax, NS. Canada
mheywood@cs.dal.ca

ABSTRACT

Complexity in genetic programming is unfortunately often associated with undesirable properties such as code bloat. In this work, we review developments in which complex systems are promoted through: 1) the evolution of teams of programs, and then 2) the context specific reuse of previously evolved code. To do so, two classes of diversity are identified: intra-team diversity and inter-team diversity. Intra-team diversity promotes task decomposition/cooperative coevolution between multiple programs, i.e. teams of programs. A fundamental requirement is that programs can learn context. Inter-team diversity is promoted through maintaining model and task diversity during evolution. The combination of both result in the ability to identify teams of programs and associate them with specific contexts, and then organize teams of programs hierarchically so solve multiple tasks. Finally, the concept of cumulative population wide performance is used to illustrate how inter model diversity in particular introduces useful biases into the types of solutions evolved.

CCS Concepts

•Computing methodologies → Sequential decision making; Genetic programming;

Keywords

Genetic Programming; Task Decomposition; Diversity maintenance; Coevolution

1. INTRODUCTION

Complex systems demonstrate emergent properties in which interactions between multiple interacting components and an environment aggregate in non-additive ways. Many aspects of evolutionary computation could potentially contribute to realizing such a goal, e.g. coevolution, multi-agent systems, swarm intelligence, neuro-evolution. In this work we are particularly interested in supporting the development of complex systems while assuming a program based representation, i.e. genetic programming (GP). Although various 'genotypic' diversity mechanisms have certainly been proposed for GP [3], their utility is ultimately rather specific and reflect a desire to identify correlation between diversity and fitness. Conversely, recent developments have proposed rewarding (behavioural) diversity alone [26], promote diversity in a multi-objective space [31], or switch between different objectives [16, 7].

In order to address diversity maintenance under the specific context of GP, three factors are pursued simultaneously. Firstly, we desire solutions to take the form a set of cooperating programs as opposed to a single monolithic piece of code. At the very least this enables us to provide more clarity to credit assignment, i.e. variation is limited to specific modules [42] whereas each decomposed task is easier to solve [34] (providing that a 'good' decomposition is found). Secondly, we do not believe that it will necessarily be possible to solve tasks from a single run of evolution. As a consequence, we wish to maintain the diversity of the teams of programs such that they solve different aspects/parts of the overall task. Indeed, policies solving such parts might evolve to be mutually independent, as has been observed under the competitive coevolution of game strategies [5]. Thus, given a population of teams that represent different policies for solving different aspects of the task, it is now potentially possible to organize the previously evolved policies into a policy (decision) tree such that switching and/or blending of the original policies takes place, resulting in a more general overall policy.

In this work, we conduct a review of approaches for coevolving teams of programs to solve tasks collectively (Section 2). The basic goal of which is to let team complement be an emergent property. We comment on the degree to which cooperative coevolution and context learning has to play in addressing the challenge of constructing effective teams or *intra-team diversity*. Mechanisms for code reuse in GP are also reviewed, where this potentially supports teaming behaviour and facilitates the construction of complex program structures. The second part of the review considers diversity

GECCO'16 Companion, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4323-7/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908961.2931655

maintenance in general from the perspective of behavioural diversity and test case diversity or *inter-team diversity*.

Section 3 provides a more detailed overview to the properties of the symbiotic bid based (SBB) framework as used to construct complex teams of programs for classification and reinforcement learning tasks. In particular, we highlight properties that lead to the ability to construct policy trees. Policy trees represent a structure that has independent of program representation, but facilitates a bottom-up process for organizing programs in complex ways. Finally, an illustrative example of diversity maintenance in the construction of policy trees is given in Section 4.

2. BACKGROUND

In the following we provide a review of two distinct themes pertinent to the evolution of complex systems in GP: task decomposition and diversity maintenance. Task decomposition emphasizes the ability to decompose a task at multiple levels within a model/policy. Thus, different code fragments might be cooperatively coevolved in parallel, or code evolved for different (sub-)tasks might be used to leverage the capacity to solve more complex tasks. Naturally, diversity maintenance also has a role to play in promoting better developmental paths. We will consider this issue from the perspective of behavioural diversity and task diversity.

2.1 Task decomposition in GP

Two specific perspectives will be noted with regards to GP task decomposition: cooperative coevolution and code reuse. **Cooperative coevolution** (e.g., [34]) is a 'meta' concept applicable to both optimization[1] and model building, where it is the latter that we are interested in here. The distinction is significant because from the perspective of optimization tasks, a priori decisions are frequently made to ensure that a common context is enforced. Specifically, optimization tasks often assume a prior decomposition in which an independent population is initialized for each dimension present in the coevolved (fixed length) genotype (e.g., [32]). In short, the relation between each of the coevolved populations is fixed a priori (Figure 1). Research efforts then attempt to address issues such as how to best minimize the noise when evaluating the fitness of a particular gene given a sampling of 'collaborators' from the remaining gene locations [33]. Conversely, under frameworks targeting the evolution of models – such as genetic programming, neuro-evolution and learning classifier systems – context is potentially an emergent property. Thus, rather than a solution to a three class classification task always consisting of, say, three GP programs, the process of learning context (coevolution) might resolve in favour of using five programs; thus more closely reflecting the underlying complexity of classifying each class [28, 29].

Under a GP context, solutions taking the form of multiple programs are generally referred to as 'teaming'. Each individual would be expressed as a matrix consisting of a predefined number of programs (team size (T) $\times$ instructions). Early work by Brameier and Banzhaf placed specific emphasis on assessing different mechanisms for combining the outcome from a team of programs into a single scalar 'prediction', e.g. averaging, error weighting, majority voting, weighted voting [2]. Fitness was equated with the overall team behaviour. Conversely, Imamura *et al.* assumed an Island model of T independent populations each producing a single team member [14]. A selection heuristic attempted to sample the T populations for team participation such that the errors of each team member were uncorrelated. This proved difficult to achieve in practice. However, it was also observed that individuals from Brameier and Banzhaf's algorithm tended to be individually weak, but collectively strong, whereas the Island model tended to produce strong individuals, but less strong teams.

[1]Solutions take the form of a point in a multidimensional space.

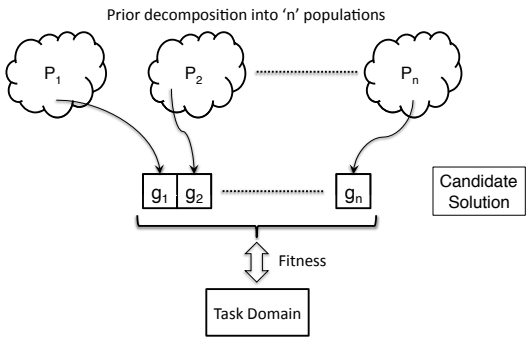


Figure 1: Generic cooperative coevolutionary framework [34]. Prior knowledge is used to establish the number of cooperating components. Implies that a candidate solution is a fixed length representation of n genes.

Thomason and Soule introduced a matrix representation (team size (T) $\times$ population size) in which selection could be individual-wise and replacement team-wise or vice versa [39]. Again, each team always consist of T individuals (programs), one individual from each column (columns are independent populations). Program context was enforced by limiting selection/replacement to be column specific. It was necessary to specify fitness for individual programs and teams; team fitness typically taking the form of the overall task performance. Defining individual (as opposed to team) fitness, however, required the identification of a suitable task specific heuristic, limiting the applicability of the method.

Recent developments in teaming algorithms for GP assume some form of bidding metaphor [28, 29, 43]. The work of Lichodzijewski utilizes a two population model based on a symbiotic metaphor (Figure 2). One population uses a variable length representation to determine which programs appear in what teams (a program may appear in more than one team). Programs define a bidding strategy (described in Section 3) to determine context, with only the winning program from a team declaring an action. Inter-team diversity was initially promoted through the use of both fitness sharing and (competitive) coevolution of training scenarios (Section 2.2). Thus, team content was cooperatively coevolved, and teams were rewarded for solving different subsets of training cases. Wu and Banzhaf went on to investigate whether teams could be evolved from individuals. This again required that fitness measures be specifically crafted for team and individual, with specific penalty terms being introduced to reward teams of an 'ideal' size [43]. Conversely, Lichodzijewski only required fitness to be defined at the level of a team, thus purely a factor of the task domain.

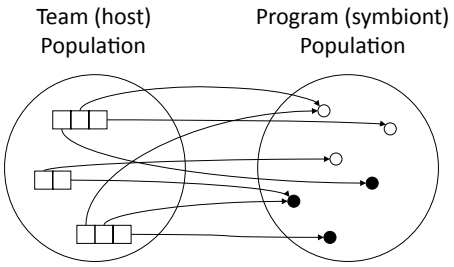


Figure 2: Generic SBB architecture. Fitness is evaluated for each member of the host population. Each host indexes between 2 and ω symbionts. Host population size is a constant (H_{size}) whereas symbiont population size 'floats', depending on the size and uniqueness of host symbiont complement.

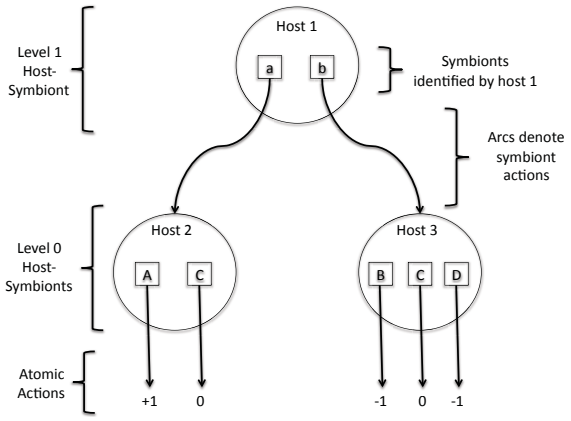


Figure 3: Example of an SBB policy tree. Host–symbionts in level 0 were previously evolved in an earlier cycle of evolution and rewarded for discovering different behaviours. Symbionts in level 0 assume actions from a task specific set of atomic actions. Host-symbionts at level 1 learn how to mix/switch between previously evolved policies. The only difference between evolution at level 0 and 1 is that a symbiont action in level 1 is a pointer to a previously evolved host.

Direct comparisons between non-teaming GP[2] versus team based GP formulations, indicate that with regards to stationary tasks[3] then task decomposition generally results in better quality solutions [2, 14, 39]; while providing better insight into the dependencies between attributes and outcomes [29]. Moreover, as the attribute space increases, the capacity to associate unique subsets of attributes with specific outcomes increases significantly [29]. In addition, when the underlying task is non-stationary (as potentially the case under streaming data applications), then GP formulated to support teaming is able to much more effectively track changes in the underlying task [41].

In the case of **code reuse**, we identify two general themes: that associated with emergent detection of 'automatically defined functions' (ADF) and more recent attempts to explicitly reuse previously evolved code. The central issue with the ADF approach was again the need to a priori define how much modularity was necessary [23]. Latter versions of this approach attempted to incrementally identify such modules using stochastic and/or frequency heuristics (e.g., [21]), thus avoiding the need to begin evolution from maximal complexity (as with ADFs). Most recently, the concept of tags has been revisited, where tags are used to establish approximate matches between calling and called code [38]. It is apparent, however, that such emergent approaches have sensitivity to the conditions under which they are able to usefully identify modules (see also 'tag addressable functions' below).

An alternative approach to code reuse is to make use of solutions as evolved in an *earlier run*. Naturally, if the task is directly 'solvable' without reuse, the initial run would have identified a solution. If not, a new run is performed where individuals from the earlier run can be resampled as a parent for recombination with an individual from the population currently under evolution. Early attempts to formulate such a scheme attempted to use an earlier run to seed the population of a later run, but were sensitive to premature convergence (as reviewed in [17]).

Under tree structured GP the the approach of Jaśkowski *et al.* adopts a blending mechanism through a special form of crossover [15]. Specifically, one parent is selected from the previously evolved population (random selection with uniform p.d.f.) and a parent identified in the population actively under evolution (selection incorporating the concept of fitness). Crossover points are identified in both parents to identify a single child which results in the introduction of genotypic material from the earlier run. Such a process naturally does not attempt to explicitly learn context for transferring material, but emphasizes importing previously evolved genetic material.

Conversely, the approach of Keijzer *et al.* make use of tags and arity to actively evolve 'tag addressable functions' (TAF) – a form of pointer – between the (tree structured GP) population under evolution and code fragments in a library of previously evolved code [17]. To do so, the previously evolved code is a priori organized into code fragments of different arity, and given tag values. Conversely, the population actively under evolution assumes a representation in which all nodes of a GP individual take the form of TAFs covering a range of arity. Thus, individuals in the evolved population are *solely* constructed from previously evolved code fragments. Various mechanisms were proposed for updating the frequency with which TAFs are referenced. Both works recognize that the library of code for reuse need not be associated with the same task. Thus, 'transfer' of material from previous runs might reflect solutions to different, but related tasks.

Finally, we note that the generic SBB architecture of Figure 2 provides a natural mechanism for learning how to deploy previously evolved code. The actions associated with individuals in the symbiont population merely take the form of a pointer to previously evolved team [18, 8]. Symbionts now explicitly learn under what circumstance to execute previously evolved teams. This then results in the bottom-up

[2]GP limited to constructing solutions consisting of monolithic code alone.

[3]The underlying process is stationary, but could include sources of noise. Many classification and/or regression tasks fall into this category.

construction of a policy tree in which different branches of the tree are executed depending on which symbiont programs 'win' at the root (Figure 3). It is now possible to organize code hierarchically to solve complex reinforcement learning tasks [19, 37] or transfer policies between multiple different reinforcement learning tasks [20]. Indeed, the capacity to construct hierarchical architectures has long been associated with the development of robust complex systems, e.g. the watch maker parable of Herbert Simon [36].

2.2 Diversity maintenance

We recognize two basic sources of diversity that potentially have an impact on the construction of complex systems with GP (or evolving complex models in general): model diversity (behavioural or genotypic), and task diversity (variation in training scenarios). We provide examples of each and note specific application highlights as well as emphasizing mechanisms that might be assumed to combine multiple diversity mechanisms.

Model diversity: and the maintenance thereof has increasingly benefited from being able to combine the effect of multiple performance metrics. Thus, in addition to a specific performance objective (e.g., number of correct classifications), mechanisms can be adopted for discounting an individual's outcome relative to the number of other individuals that also have the same outcome. Several authors have proposed some form of (implicit)[4] fitness sharing [35, 30, 28], a general form for which is:

$$s_i = \sum_k \left(\frac{G(tm_i, p_k)}{\sum_j G(tm_j, p_k)} \right)^{\alpha} \quad (1)$$

where $G(tm_i, p_k)$ is the task dependent reward (for maximization) defining the quality of individual tm_i on training scenario p_k and α is the relative weight given to uniqueness.

Such a metric has been utilized under classification [30, 28, 29] and various reinforcement learning tasks [35, 18, 8, 37]. Under reinforcement learning domains in particular, this might provide the basis for identifying a cross section of policies that can then be reused under a second round of evolution, resulting in a more general ultimate policy (see the tutorial example in Section 4). Underlying assumptions / constraints include that complete control exists over the training cases, so that the outcome of each candidate solution can be assessed under exactly the same conditions.

Novelty as an objective initially assumed the perspective that individuals should be rewarded for ultimate outcomes that were different [25], such a difference is most often characterized behaviourally. At some level this was equivalent to having a common starting point to a task (say, a common maze entry), but rewarding the discovery of different end points (within a finite exploratory budget). The approach was then refined to reward different behavioural *trajectories* [26], where trajectories are described in terms of a quantized set/subset of the task's state variables or state and action pairs. Naturally, too much quantization might preclude the recognition of novelty, introducing aliasing effects [26, 22]. Multiple metrics have been proposed for measuring novelty [10, 9], there potentially being a tradeoff between the utility of task-independent versus task-specific metrics. Moreover, it is also necessary to make use of an archive of behaviours, or those judged to represent more significant differences.[5] Thus, the novelty of a new genotype is compared against the k nearest behaviours w.r.t. the archived individuals.

One opportunity that specifically appears when assuming a teaming GP approach is that novelty can also be expressed in terms of a comparison of team member complement (a genotypic property) in addition to behavioural properties [20]. Assuming bidding metaphor for team cooperation, the distance between two teams tm_i and tm_j is summarized as the ratio of active GP individuals common to both teams.[6] Thus, the distance between teams i and j is

$$dist(tm_i, tm_j) = 1 - \frac{Tm_{active}(tm_i) \cap Tm_{active}(tm_j)}{Tm_{active}(tm_i) \cup Tm_{active}(tm_j)} \quad (2)$$

where $Tm_{active}(tm_x)$ represents the set of active programs in team x. Naturally, such a diversity metric is task independent and explicitly discounts hitchhiking programs.

Most recently, combined approaches have been assumed in which combinations of metrics have been utilized. Naturally, multiple performance and/or behavioural objectives could be recombined through a weighted sum of product formulation (objective scalarization). However, this always raises issues about how to find 'good' weight values. Conversely, adopting a Pareto multi-objective formulation provides a much cleaner framework for comparing multiple criteria under GP [1].

From the perspective of constructing complex systems in general, particular highlights include the use of Pareto multi-objective formulations for evolutionary robotics [31, 40]. Specific benefits include recognition of proxy objectives, capacity for addressing the bootstrap problem, and increased resilience to premature convergence. Proxy objectives recognize that it might not be possible to explicitly measure the target behaviour, but when all proxy objectives are satisfied, then the overall goal might be assumed. Likewise, the bootstrap problem might also be satisfied sequentially, leading to the concept of incremental evolution [11]. In essence, the ultimate objective is too difficult to be solved immediately, but might be reached through the satisfaction of a sequence of objectives. When adopting a Pareto multi-objective approach it is not necessary to specify the order with which different objectives need to be satisfied.

Assuming a Pareto multi-objective formulation naturally implies that it is then possible to simultaneously assume multiple diversity measures and/or performance objectives. However, there is at least one practical caveat: the cost of continually measuring multiple diversity/performance objectives. Conversely, a multi-objective approach can be maintained by switching between diversity measures, thus at any generation a fixed performance objective might be assumed with a novelty metric chosen stochastically from a set of behavioural diversity metrics [7]. Indeed, there is evidence that switching between objectives that are 'related' is beneficial for promoting modularity [16]. Evidence of this in GP was

[4]Explicit fitness sharing requires the definition of an appropriate distance function, thus limiting its applicability to tasks where an appropriate distance function can be designed.

[5]Archives also appear as a mechanism for supporting competitive coevolution as in noteworthy opponents to compete against [35].

[6]Active team members are those that successfully suggest an action.

found when applying GP teaming to non-stationary streaming tasks [41].

Task diversity: implies that the training scenarios (the environment) can be manipulated to encourage diversity in the evolution of models. Competitive coevolution attempts to discover informative orderings of training scenarios while minimizing the impact of pathologies such as disengagement (bootstrapping) or forgetting (e.g., [4, 6]). Ultimately, a population of independent policies might emerge in which different programs solve entirely independent aspects of a task [5]. However, given support for program reuse, it might then be possible to compose a policy tree with a capability that exceeds the sum of its reused programs. Moreover, even supporting stochastic variation in the sampling of training scenarios has been shown to lead to more general programs [24].

Recently, a lexicase selection operator was proposed that appears to promote behavioural diversity without having to explicitly formulate a diversity metric [12]. To do so, tournaments for parental selection are formulated in which: 1) a training exemplar is randomly selected (without replacement) and, 2) all individuals are evaluated on the training case. Only those individuals with the correct outcome are allowed to progress to evaluation on the next randomly sampled training exemplar. The underlying insight being that individuals will fail in different ways, and this 'diversity' is being rewarded by the lexicase selection process. Naturally, it must be possible to exert sufficient control on the training partition, both in terms of testing all individuals on exactly the same training case, and in terms of an equal representation of classes of exemplar (under classification tasks).

Finally, we also note that a point is reached where prior decomposition of a task might be necessary before suitable progress can be made. Thus, under layered learning, training scenarios are explicitly constructed to represent skills for which policies are evolved. For example, as in learning different aspects of playing soccer [13]. Each skill policy is then slotted into a prior task decomposition – decision tree – in order to build an overall policy for solving a task. Conversely, policies evolved to solve easier 'source tasks' (skills) might be adapted to solving a new 'target task' or task transfer. An example of this from GP was recently demonstrated in which policies took the form of two soccer (source) skills – keepaway and shooting – with a target task of playing half-field offense [20]. Of particular interest here, from a complex systems perspective, was that although the overall system required more than 10 to 15 programs to cooperate, only half of them need be executed to make any single decision (discussed further below).

3. SBB FRAMEWORK

As noted above, the SBB framework provides a flexible architecture for promoting the evolution of complex models for classification and reinforcement learning tasks. In the following we highlight some of the useful properties of the framework.[7]

Symbiotic two population architecture: Teams and programs are represented by two independent populations and evolved under a symbiotic relationship (Figure 2). Specifically, each member of the host population represents a potential team, whereas members of the symbiont population represent candidate programs for appearing within a team. Fitness is only explicitly expressed at the level of team. After each generation the *Gap* worst performing teams are deleted, and any programs failing to be indexed by at least one team are deleted. Variation operators act on the remaining content generating new individuals by: 1) cloning a team, 2) adding/deleting pointers, 3) cloning one or more program and then adding/deleting/modifying instructions [28, 29].

Bid-based GP: A 'bidding' metaphor is used to explicitly separate the issue of learning a *context* (for an action) and suggesting the action itself [27, 28, 29, 43]. Given the current state from the task domain and a team under evaluation, execute each program participating in this team. The program with maximum output 'wins' the right to suggest its corresponding action. Under classification tasks the actions represent class labels. Under reinforcement domains, actions take the form of atomic task specific actions. Each program may only assume a single action. Thus, a team must index programs with at least two different actions, however, the compliment of programs in any given team represents an emergent property. Different teams may have different numbers of programs, programs can appear in more than one host, and teams may have multiple programs with the same action. All this freedom in how programs can be deployed by teams provides a wide range of mechanisms for discovering task specific decompositions [28, 29].

Inter-team diversity and policy trees: Despite the use of a teaming metaphor there is no guarantee that a single champion host will emerge that solves all of a task at the end of evolution. As a consequence, mechanisms to encourage inter-team diversity are utilized (Section 2.2). This means that a population of teams with a range of behaviours may emerge that potentially covers the total set of policies necessary to solve a task (e.g., [5]). However, it is not possible to know from the initial state of a task which policy to deploy when/where. This question can be addressed by initiating a new host–symbiont population, with actions taking the form of pointers to previously evolved teams [18, 8, 19, 37]. Evaluation of any given team is now a process of descending a policy tree until a specific atomic action is recommended (Figure 3). Section 4 will illustrate this property under a simple reinforcement learning task.

4. LEVERAGING DIVERSITY FROM RANDOM INITIAL POLICIES

In the following we will assume the task of learning to play tic-tac-toe. Such a task is used here to provide an illustration for what multi-level policies (aka policy trees) are attempting to achieve and the role of diversity maintenance in achieving this. The basic goal is to evolve a policy for playing tic-tac-toe, as in a reinforcement learning environment in which a reward of $[1, 0.5, 0]$ is received for winning, drawing or loosing each game. Results are reported for 20 initializations for each level of the hierarchy and an independent set of 1,000 random test policies.

4.1 Level 0 policies

We begin by merely initializing the first population of SBB host–symbionts and assume these as the result of the first round of evolution. Figure 4 illustrates the strength of play across the entire population of host–symbiont teams, i.e. the

[7]Several code bases are publicly available http://web.cs.dal.ca/~mheywood/Code/

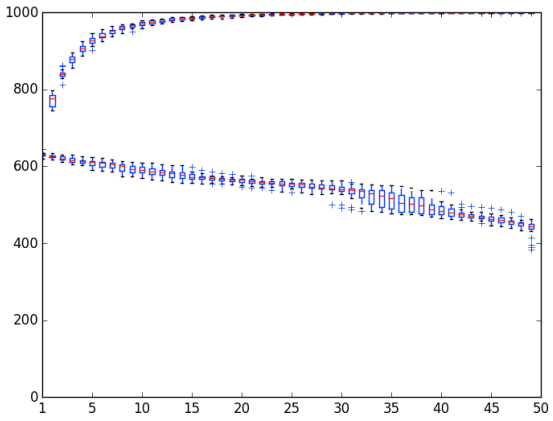


Figure 4: Case of *population wide* performance of 50 teams at level 1 for random initial host-symbionts. 1000 test games and evaluation repeated over 20 initial populations.

quality of play of the random initial policies. To do so, each team is ordered in descending rank order and then a cumulative curve constructed representing the combined performance of teams through the population (ascending curve). The latter curve represents how much of the task is *collectively solved* as opposed to how much each team is able to solve.

A particular feature of interest is the steep rise in the cumulative curve for the first 10 (x-axis) policies, implying that there is considerable diversity in the games won by (the initial random) teams. Naturally, this information is only available through an exhaustive evaluation of policies against games, a process that is not possible during evolution. However, we consider the properties captured in Figure 4 to be indicative of the kind of outcome we might want to see on a task after evolution at level 0. That is to say, if we cannot find a team (aka policy) that solves the task outright, we would like to be able to identify a population of teams that addressed different aspects of the underlying task. Including appropriate inter-team diversity measures during evolution at level 0 would be central to achieving this, for example, SBB as applied to keepaway soccer [19].

4.2 Evolving policy trees

Evolution at level 1 results in the development of policy trees (Figure 3). Teams at level 1 learn under what conditions to switch/blend previously evolved teams as identified by the content of the Host–symbiont population from level 0. In order to illustrate the impact of introducing evolution with varying amounts of diversity across level 0 and 1, we consider the following three cases:

1. Evolution at both level 0 and level 1 (50 generations each) without diversity maintenance; hereafter the *no diversity case*.

2. Population at Level 0 assumes the content of the initial population (Figure 4). Population at Level 1 is evolved for 50 generations *without* diversity; hereafter the *medium diversity case*.

3. Population at Level 0 assumes the content of the initial population (Figure 4), but the population at Level 1 is evolved for 50 generations *with* the diversity measure of Eq. (2); hereafter the *maximum diversity case*.

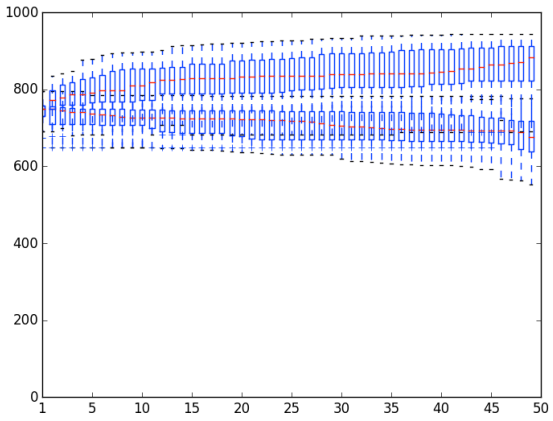
Figure 5 illustrates the performance at the end of evolution at level 1, i.e. each team represents the root node of a policy tree (Figure 3) that attempts to identify under what conditions to switch between policies present in level 0. In case 1, SBB populations are evolved at both level 0 and 1 for 50 generations each, but without any form of diversity maintenance (subplot 5(a)). The performance of the champion policy is now around 75%, moreover, all individuals in the population (at the final generation of evolution at level 1) solve a similar subset of games, as illustrated by the relatively flat cumulative curve. In short, conducting evolution at both level 0 and level 1 *without* any form of diversity maintenance results in a worse cumulative performance than the initial population at level 0.

In case 2, diversity is introduced by assuming the content of level 0 at initialization (Figure 4), but performing evolution at level 1 under fitness alone for 50 generations (no additional diversity mechanisms). Naturally, evolution at level 1 does not have direct access to the information of Figure 4. Instead, the correct subset of level 0 policies for incorporating into a policy tree needs to be learnt, as does the context for deploying such level 0 policies. Figure 5(b) illustrates the resulting post training test performance. The single best solution is as strong as when policy trees are evolved without any form of diversity. However, the cumulative curve is now also stronger, implying that although evolution was performed without explicit diversity maintenance at level 1, the use of a diverse range of behaviours from level 0 was a better basis for constructing policies at level 1.

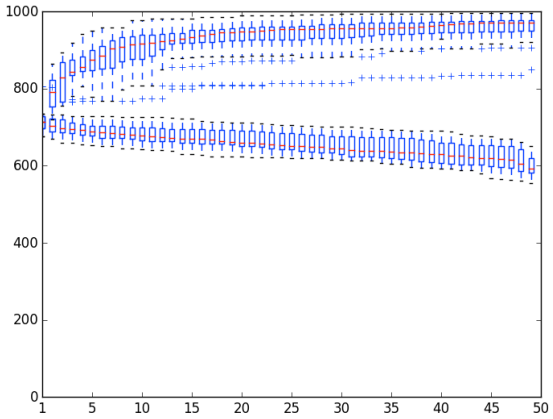
Finally, the full diversity case is illustrated in Figure 5(c). Two general trends appear: 1) the cumulative curve is again stronger; and, 2) although the strength of the champion individual is equivalent to that under case 1 and 2, the population retains a greater proportion of individuals that are significantly weaker than the champion (aka specialists).

5. CONCLUSIONS

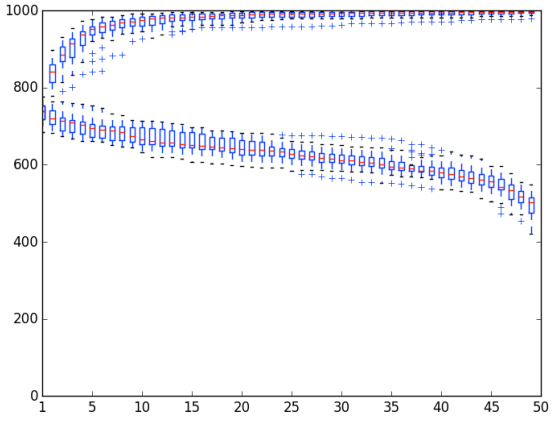
A review has been made of architectures proposed to facilitate code reuse and teaming GP as emergent properties. One central enabling theme in this respect was providing the capacity to learn context. GP teaming was then related to more general developments in diversity maintenance. In adopting forms of GP capable of context learning it was then possible to make use of teams previously evolved under different forms of diversity maintenance (model diversity and/or task diversity) and explicitly learn how to construct hierarchical policies (of teams) capable of generalizing to broader ranges of tasks, all as emergent properties. Central to doing so is being able to learn the context for deploying previously evolved teams. We motivated this in terms of the potential for coverage in the simple task of tic-tac-toe. In this case, the diversity available at initialization is such that some subset of the initial teams are actually capable of 'covering' this task. However, evolution tends to promote



(a) Case 1 – No diversity



(b) Case 2 – Medium diversity



(c) Case 3 – Maximum diversity

Figure 5: Number of games won at level 1 evolution of 50 teams with varying amounts of diversity. Ascending (descending) curve represents cumulative (individual) performance. Distribution estimated over 20 runs.

the survival of the stronger individual(s), at the expense of no longer being able to cover the task in terms of the overall population behaviour. Specific application examples where evolution of GP teams has been shown to explicitly make use of earlier populations to construct policy trees include: Acrobot [8] and pinball [18] reinforcement learning benchmarks, various subtasks of multi-agent soccer (keepaway [19] and half field offense [20]) and discovering transforms between subgroups in the Rubik's Cube [37].

6. ACKNOWLEDGEMENTS

The authors are grateful for support from the (Canadian) NSERC Discovery and CFI programs.

7. REFERENCES

[1] K. M. S. Badran and P. Rockett. The influence of mutation on population dynamics in multiobjective genetic programming. *Genetic Programming and Evolvable Machines*, 11(1):5–33, 2010.

[2] M. Brameier and W. Banzhaf. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines*, 2(4):381–407, 2001.

[3] E. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.

[4] J. Cartlidge and S. Bullock. Combating cevolutionary disengagement by reducing parasite virulence. *Evolutionary Computation*, 12(2):193–222, 2004.

[5] S. Y. Chong, P. Tiňo, and X. Yao. Relationship between generalization and diversity in coevolutionary learning. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):214–232, 2009.

[6] E. D. de Jong. A monotonic archive for Pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[7] S. Doncieux and J.-B. Mouret. Behavioral diversity with multiple behavioral distances. In *IEEE Congress on Evolutionary Computation*, pages 1427–1434, 2013.

[8] J. A. Doucette, P. Lichodzijewski, and M. I. Heywood. Hierarchical task decomposition through symbiosis in reinforcement learning. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 97–104, 2012.

[9] J. Gomes and A. L. Christensen. Generic behavioral similarity measures for evolutionary swarm robotics. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 199–206, 2013.

[10] F. J. Gomez. Sustaining diversity using behavioral information distance. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 113–120, 2009.

[11] F. J. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[12] T. Helmuth, L. Spector, and J. Matheson. Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, 2015.

[13] W. H. Hsu, S. J. Harmon, E. Rodriguez, and C. Zhong. Empirical comparison of incremental reuse

strategies in genetic programming for keep-away soccer. In *Genetic and Evolutionary Computation Conference – Late Breaking Papers*, 2004.

[14] K. Imamura, R. B. Heckendorn, T. Soule, and J. A. Foster. Behavioral diversity and a probabilistically optimal GP ensemble. *Genetic Programming and Evolvable Machines*, 4:235–253, 2004.

[15] W. Jaśkowski, K. Krawiec, and B. Wieloch. Knowledge reuse in genetic programming applied to visual learning. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1790–1797, 2007.

[16] N. Kashtan, E. Noor, and U. Alon. Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences of the USA*, 104(34):13711–13716, 2007.

[17] M. Keijzer, C. Ryan, and M. Cattolico. Run transferable libraries – learning functional bias in problem domains. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 3103 of *LNCS*, pages 531–542, 2004.

[18] S. Kelly and M. I. Heywood. On run time libraries and hierarchical symbiosis. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3245–3252, 2012.

[19] S. Kelly and M. I. Heywood. On diversity, teaming, and hierarchical policies: Observations from the Keepaway soccer task. In *European Conference on Genetic Programming*, pages 75–86, 2014.

[20] S. Kelly and M. I. Heywood. Knowledge transfer from keepaway soccer to half-field offense through program symbiosis. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1143–1150, 2015.

[21] K. E. Kinnear. Alternatives in automatic function definition: A comparison of performance. In *Advances in Genetic Programming*, chapter 6, pages 119–141. MIT Press, 1994.

[22] S. Kistemaker and S. Whiteson. Critical factors in the performance of novelty search. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 965–972, 2011.

[23] J. R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1992.

[24] I. Kushchu. Genetic programming and evolutionary generalization. *IEEE Transactions on Evolutionary Computation*, 6(5):431–442, 2002.

[25] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the XI International Conference on Artificial Life*. MIT Press, 2008.

[26] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[27] P. Lichodzijewski and M. I. Heywood. Pareto-coevolutionary genetic programming for problem decomposition in multi-class classification. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 464–471, 2007.

[28] P. Lichodzijewski and M. I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 863–870, 2008.

[29] P. Lichodzijewski and M. I. Heywood. Symbiosis, complexification and simplicity under GP. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 853–860, 2010.

[30] R. I. McKay. Fitness sharing in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 435–442. Morgan Kaufmann, 2000.

[31] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

[32] L. Panait, S. Luke, and R. P. Wiegand. Biasing coevolutionary search for optimal multiagent behaviours. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.

[33] L. Panait, K. Tuyls, and S. Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research*, 9:423–457, 2008.

[34] M. A. Potter and K. A. DeJong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[35] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[36] H. A. Simon. *The Sciences of the Artificial*. MIT Press, 1969.

[37] R. J. Smith, S. Kelly, and M. I. Heywood. Discovering Rubik's Cube subgroups using coevolutionary GP – A five twist experiment. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, 2016.

[38] L. Spector, K. Harrington, and T. Helmuth. Tag-based modularity in tree-based genetic programming. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 815–826, 2012.

[39] R. Thomason and T. Soule. Novel ways of improving cooperation and performance in ensemble classifiers. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1708–1715, 2007.

[40] V. Trianni and M. López-Ibáñez. Advantages of task-specific multi-objective optimization in evolutionary robotics. *PLoS ONE*, 10(8):DOI:10.1371/journal.pone.0136406, 2015.

[41] A. Vahdat, J. Morgan, A. R. McIntyre, M. I. Heywood, and N. Zincir-Heywood. Evolving GP classifiers for streaming data tasks with concept change and label budgets: A benchmarking study. In A. H. Gandomi et al., editor, *Handbook of Genetic Programming Applications*, chapter 18, pages 451–480. Springer, 2015.

[42] G. P. Wagner and L. Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50:967–976, 1996.

[43] S. Wu and W. Banzhaf. Rethinking multilevel selection in genetic programming. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1403–1410, 2011.