On Diversity, Teaming, and Hierarchical Policies: Observations from the Keepaway Soccer task

Stephen Kelly[1] and Malcolm I. Heywood[1]

[1]*Faculty of Computer Science, Dalhousie University, Halifax, NS. Canada*

Article originally appears at EuroGP'14 under Springer copyright 2014
http://link.springer.com/chapter/10.1007\%2F978-3-662-44303-3_7

Abstract

The 3-versus-2 Keepaway soccer task represents a widely used benchmark appropriate for evaluating approaches to reinforcement learning, multi-agent systems, and evolutionary robotics. To date most research on this task has been described in terms of developments to reinforcement learning with function approximation or frameworks for neuro-evolution. This work performs an initial study using a recently proposed algorithm for evolving teams of programs hierarchically using two phases of evolution: one to build a library of candidate meta policies and a second to learn how to deploy the library consistently. Particular attention is paid to diversity maintenance, where this has been demonstrated as a critical component in neuro-evolutionary approaches. A new formulation is proposed for fitness sharing appropriate to the Keepaway task. The resulting policies are observed to benefit from the use of diversity and perform significantly better than previously reported. Moreover, champion individuals evolved and selected under one field size generalize to multiple field sizes without any additional training.

1 Introduction

Keepaway soccer was conceived as a simplification of the full RoboCup simulated soccer task in which the objective is for K 'keepers' to maintain possession of the ball for as long as possible from $K-1$ 'takers' [21, 20]. The takers assume a pre-specified policy whereas the keepers need to learn an appropriate policy. Keepaway is implemented using the same RoboCup simulator as used for the full game of soccer, but with additional constraints on the rules, boundary of the field, and number of players. The Keepaway task is known to be non-Markovian and has a wide range of results from reinforcement learning and neuro-evolution (Section 2). However, to date, there has been little interest in applying genetic programming (GP) to this task. The goal of this work is to make an initial assessment of the capability of GP and the role of diversity maintenance in identifying effective keeper policies under this domain. Previous results using neuro-evolution have made use of genotypic diversity measures [16, 27]. While genotypic diversity metrics have been proposed for canonical forms of GP (e.g., [3]), their design is not necessarily obvious for the case of GP that supports task decomposition through teaming. Thus, in this work we introduce a novel formulation for phenotypic fitness sharing and empirically evaluate its effect on hierarchical GP under the Keepaway task.

The GP framework assumed in this work takes the form of symbiotic bid-based GP (hereafter SBB), where this has previously been illustrated under various reinforcement learning tasks e.g., Rubik cube [15], Pin-ball [12] and Acrobot handstand [8]. In SBB, a control policy is defined by a team of simple programs that are coevolved, each specializing on a subcomponent of the task. Adopting a teaming approach to policy search implies that it is possible to start from simple policies at initialization and incrementally introduce more complexity as the task warrants. The utility of task decomposition in general has been demonstrated under supervised [14] and reinforcement learning [4]. Previous results with SBB have also indicated that if an initial run does not identify suitably general policies, then the contents of the initial population can be 'cashed' and referred to as a library of 'meta actions' [15, 12, 8]. Individuals evolved in a second independent run learn the context for deploying the previously evolved policies. That is to say, SBB supports a mechanism through which hierarchical policies may be incrementally constructed.

In this work, we are specifically interested in the role of diversity maintenance in the development of such hierarchical policies. The underlying assumption is that such diversity is necessary during the development

of the initial population of SBB individuals (meta actions), but not necessary when constructing policies from meta actions. More generally, the need for diversity and modularity is frequently acknowledged, particularly in environments with dynamic properties [26, 7]. Thus, under the guise of ensemble methods as applied to streaming data tasks, diversity is seen to provide faster reaction times to a change, but does not necessarily facilitate faster convergence to the new concept [17]. Under a neuro-evolutionary setting in which solutions to the iterated prisoner's dilemma are coevolved, diversity is shown to support the development of a broader range of policies across the population as a whole, but it is difficult to integrate this diversity into a single individual [5]. Finally, the field of evolutionary robotics frequently reports better performance when reward is given for both novel solutions as well as optimizing fitness [6, 18]. However, it is also clear that defining an appropriate diversity metric is as an open ended activity.

In addition to investigating the role of diversity, we are also interested in discovering how a GP approach compares to the current state of the art under the most widely considered $K = 3$ (3-versus-2) configuration of the Keepaway task. As will be established in Section 2, progress on the Keepaway task has been dominated by reinforcement learning and neuro-evolutionary methods.

2 Related work

As noted in the introduction, the Keepaway task represents a benchmark for both multi-agent and reinforcement learning / policy search in general [21, 20, 16, 27]. Given that a decision maker is necessary for each keeper, reinforcement learning approaches have adopted a heterogeneous assignment of learners to keepers, where this is a function of the overhead in attempting to update a single function approximator w.r.t. multiple keepers [27]. Conversely, (evolutionary) policy search generally assumes a homogeneous assignment, where this is a reflection of the lack of specialization required in the keeper policies [25].

The original development of Keepaway defines the task of the decision maker in terms of a pre-specified decision tree in which, should another keeper be in possession of the ball, the free keepers assumes a "get open" behaviour. Otherwise if the keeper is not in possession of the ball but can get there faster than any other teammate, then the keeper approaches the ball. The task of the learning algorithm is to discover the appropriate strategy for the case of a keeper in possession of the ball [21, 20]. When in possession of the ball there are a total of K atomic actions, a; or $a \in \{\text{HoldBall, Pass2ThenReceive, ..., Pass}k\text{ThenReceive}\}$; where the Pass$k$ThenReceive action defines which keeper to pass the ball to, with k indexing the nearest ($k = 2$) to most distant ($k = K$) keeper. This is the most common formulation of the keepaway task, and will be assumed in the work here.

Under reinforcement learning, the first obstacle to be addressed was how to formulate the task such that credit assignment mechanisms such as SMDP Sarsa(λ) could be applied [21, 20]. With this achieved, most emphasis has been on the type of function approximation used to model Q-values. Thus, function approximation based on tile coding [21, 20] has been superseded by the use of Radial Basis Functions [27] or kernel methods [11].

Several approaches to neuro-evolution have been applied to the Keepaway task, including NEAT [22, 27], EANT [16] and HyperNEAT [24]. All schemes make extensive use of genotypic diversity for maintaining multiple species during a run. These studies also adopt the result reporting framework established under the reinforcement learning approaches cited above, making comparison between different algorithms possible. The same approach is assumed here.

In the case of GP, we note that a layered learning approach has been adopted in the past to facilitate the incremental evolution of tree structured GP, with and without ADFs [9, 10]. However, these results are reported for a different soccer simulator (TeamBots) and hence different atomic actions. Layered learning assumes that the task undergoes some prior decomposition with training performed relative to the simpler tasks first. It was also necessary to enforce a prior discretization of the state variables (i.e., a simplification of the task) and limit the number of takers to 1 i.e., 3-versus-1 keepaway.

3 Hierarchical symbiotic policy search

Frameworks for evolving teams of programs represent an alternative approach for deriving modular solutions under GP. Bremeier and Banzhaf assumed a representation in which a fixed number of programs were grouped (per team) and evaluated collectively, with variation operators switching programs between teams as well as modifying individual programs [1]. Thompson and Soule also assumed fixed sized teams, but introduced orthogonal selection operators i.e., building teams from the perspective of the program or team [23]. SBB explicitly supports 'incremental complexification' through the use of a symbiotic framework for coevolving team membership (host)

and programs (symbionts) cooperatively [14]. Thus, host individuals define a team (host membership) by indexing some subset of the available symbionts (programs). Assuming a variable length representation for the hosts implies that the size of a team is free to evolve. Symbiosis appears because host and symbiont individuals exist in independent populations with fitness only evaluated at the host population.[1]

3.1 Symbiont

Symbiont programs take the form of bid-based GP [13]. Each symbiont represents a tuple consisting of a task specific discrete action, a, and a program, p. Without loss of generality, we assume a linear GP representation [2]. The role of a program is to define a bidding strategy. Thus, consider the case of a host consisting of two symbionts $\langle a_1, p_1 \rangle$ and $\langle a_2, p_2 \rangle$. Given a set of state variables describing the current state of the task domain, each symbiont associated with the host executes its program. The symbiont with largest output, say p_2, 'wins' the right to suggest its action at the current time step, or a_2 in the case of this example. Evaluation for the current host continues until an episodic end state is encountered i.e., for each new state of the task, the symbiont programs are executed and the winning symbiont suggests its action, in each case potentially updating the state of the task.

3.2 Variation operators

Variation operators are asexual and take the form of a set of mutation operators applied to a host *after* it is cloned. Two mutation operators are used: remove symbiont from the host, add a symbiont to the host. In addition, a third operator can initiate the creation of a new symbiont. In this case a symbiont that is currently a member of the host is cloned and the cloned symbiont's action and / or program is modified (inserting / deleting instructions). For further details of the variation operators see [14].

3.3 Selection operator

Evolution is conducted under a breeding metaphor, thus post fitness evaluation, the worst performing H_{gap} individuals are deleted. Any symbionts that are not part at least one remaining team are assumed to be ineffective and therefore also deleted. No attempt is made to derive symbiont fitness through, say, the average of the host fitness in which it is a member. In effect we are assuming multi-level selection in which an organism is only evaluated as a whole as opposed to the sum of its parts [19]. One implication of this process is that the host population is a fixed size, whereas the symbiont population 'floats' under the action of the selection and variation operators.

3.4 Constructing Hierarchical Policies

Evolution will be performed in two distinct phases. During phase 1 symbionts assume atomic actions taken from the task domain. This lasts for a fixed number of generations and establishes a population of meta actions for use as actions by the second phase of evolution. The goal of phase 2 is to discover under what conditions to switch between different meta actions as identified during phase 1. No further modification of individuals from phase 1 tasks place. As per phase 1, evaluating a host from phase 2 (h_i^2) results in the identification of a winning symbiont (Section 3.1). However, at this point the action is a previously evolved host, h_j^1, as discovered during phase 1 i.e., a meta action. Thus, host h_j^1 is now executed for the current state variables, with the winning symbiont this time selecting an atomic action which is used to update the state of the game. Further details regarding the evolution of hierarchical policies is available from [15, 12, 8].

3.5 Fitness and Diversity

In the specific case of the 3-versus-2 Keepaway task, the soccer simulator defines the location of keepers and takers such that each keeper is stochastically initialized in one of the three corners of a square field and all takers are initialized in the fourth corner. The corners associated with keepers and takers do not vary, but the precise initial location does. Likewise, the ball is initialized 'near' one of the keepers, but this also varies. Unlike most episodic tasks, this means that it is not possible to precisely control the initial configuration of the task. Hence, during fitness evaluation, each host plays multiple games, but it is not possible to replicate the initial conditions, or for that matter other stochastic events that potentially occur during a game i.e., the soccer simulator adds noise to actions and state variables. Each host will play P games per generation, thus fitness of a host is merely the average

[1] Hereafter host / team and symbiont / program will be used interchangeably.

total duration for a game as measured by the simulator over *at least* P games. We naturally hope this averaging smooths out some of the variation that is not explicitly due to the host's policy.

In terms of diversity maintenance, previous researchers have assumed linear combinations of fitness and novelty (e.g., [6]), Pareto multi-objective formulations (e.g., [18]), or fitness sharing based on genotypic diversity (e.g., [22, 16]). Each method varies with respect to the number of user-specified parameters and their sensitivity. Furthermore, a function for measuring genotypic or phenotypic distance between policies is typically required, where this can be task-specific or generic. In the following we will adopt a phenotypic fitness sharing methodology that incorporates a task-specific distance metric to select the most 'similar' game as played by other hosts. This is a significant departure from earlier formulations for diversity (e.g., [15, 12, 8]). The motivation for this approach stems from the observation that start states in the keepaway task are stochastic and both sensor readings and actuators are noisy. Thus, a policy's phenotype cannot be characterized solely by the reward received relative to a particular initial task configuration. Instead, the property captured by fitness sharing will summarize the configuration of the *failure state*. Thus, we reward diversity in failure, where this is taken as an indicator of policy behaviour. With this in mind, the following approach is taken to fitness sharing.

Let each host maintain a history of the end state and reward for the most recent P_{hist} games. State variables in keepaway are ego-centric, hence translation and rotation independent. This reduces the number of trivial differences that might appear in the end state of a game. Thus, for each of the last P_{hist} games we record the ego-centric state variables relative to the keeper initialized in the upper-left corner of the field. Our use of homogeneous keepers is taken to imply that a single keeper perspective is sufficient to characterize team behaviour. Each host plays P games per generation, overwriting the P oldest entries in that host's history. The actual size of a host's historical record will therefore range between P and P_{hist} depending on age. A host's shared fitness score, s_i, discounts the reward for each historical game as follows:

$$s_i = \frac{1}{h_{hist}} \sum_{j \in h_{hist}} \left(\frac{G(h_i, e_j)}{\sum_{k \neq i} (1 - dist(e_j, e_{hist})) G(h_k, e_{hist})} \right) \quad (1)$$

where $G(h_i, e_j)$ is the reward host i received for game j, the denominator summation $\sum_{k \neq i}$ is over all other hosts in the same population, e_{hist} is the game failure state from the historical record of host k that most closely matches e_j, with the corresponding reward $G(h_k, e_{hist})$, and $dist(e_j, e_{hist})$ is a Euclidean distance metric, normalized to the unit interval, with 1 denoting least similarity. h_{hist} represents all games currently stored in the historical record for host i.

In short, Eqn. (1) re-weights the reward that host i receives on game j relative to the historical record available for other hosts on the game they played with the most similar *failure state*. We also limit the estimation of final game state to that of the keeper positions alone. Hence, under the 3-versus-2 Keepaway task a total of 5 state variables are used to characterize the final state of a game. Given that evolution is applied in two distinct phases in order to construct explicitly hierarchical SBB policies (Section 3.4), fitness sharing is only applied during the first phase. In phase 2 the objective is explicitly exploitive, hence no fitness sharing will be deployed.

4 Results

As a guide to parameterizing SBB we note from Whiteson *et al.* [27], that NEAT utilized 6,000 evaluations per generation with 50 to 60 generations (the latter was not explicitly reported), implying that between 300,000 to 360,000 evaluations were performed per run. Under the SBB parameterization from Table 1, the corresponding total evaluation count is defined as: $t_{max} \times H \times P = 200,000$. However, there are two phases of evolution in order to construct hierarchical policies (Section 3.4), thus 400,000 evaluations in total. The publicly available code distribution provided the initial implementation for SBB[2], from which the necessary modifications were made to provide the new formulation for fitness sharing and interface to the Robocup Soccer Server[3]. Supporting code from the designers of the keepaway task[4], provides a full implementation of the keepaway environment. This work uses version 1.15.1 and 0.6 of the soccer server and keepaway code respectively.

As per established practice on the Keepaway task, reporting of performance will be conducted *throughout* the evolutionary cycle. Thus, at every 125 generations we let each host play 100 games and then have the champion host play 1,000 games. No fitness sharing is deployed during champion identification. A total of 12 independent runs are performed. Results from previous research are summarized in Table 2, where these are generally the

[2] `http://web.cs.dal.ca/~mheywood/Code/SBB/`

[3] `http://sourceforge.net/apps/mediawiki/sserver`

[4] `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/`

Table 1: SBB Parameterization per phase of evolution (Section 3.4). t_{max} is the total generation limit; ω_{max} is the maximum number of symbionts a host may support under a variable length representation; H is the host population size; P is the number of games played by each each host per generation; P_{hist} is the maximum number of games stored in each host's historical record; H_{gap} is the number of individuals replaced during breeding in the host population; p_{xx} denote the frequency with which different search operators are applied; *numRegisters* and *maxProgSize* represent the number of registers and maximum instruction count for (symbiont) programs.

Host (teams)				Symbiont (programs)	
Parameter	Value	Parameter	Value	Parameter	Value
t_{max}	250	ω_{max}	15	*numRegisters*	8
H	80	H_{gap}	40	*maxProgSize*	48
P	10	P_{hist}	100	p_{delete}, p_{add}	0.5
p_{md}	0.7	p_{ma}	0.7	p_{mutate}, p_{swap}	1.0
p_{mm}	0.2	p_{mn}	0.1		

Table 2: Summary of keeper possession times for 3-versus-2 Keepaway. Averages reflect 1,000 test games for a champion policy. Simulated hours represents the total number of simulation hours necessary to return the champion policy.

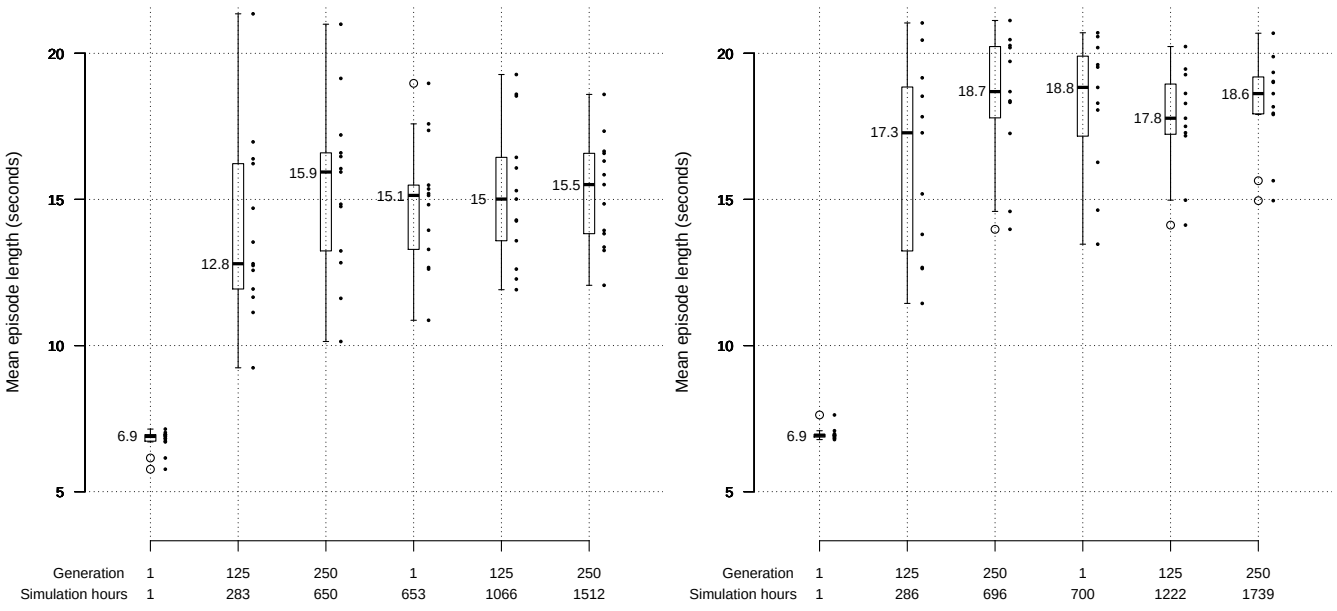
Algorithm	Avg. keeper possession time	Simulated hrs. (of play)
HyperNEAT [24]	15.4 sec	unknown
EANT [16]	14.9 sec	≈ 200
NEAT [22, 27]	14.1 sec	≈ 800
Sarsa with RBF [22, 27]	12.5 sec	≈ 50

result of 5 independent runs. Two experiments will be performed: hierarchical SBB without fitness sharing versus hierarchical SBB with fitness sharing (during phase 1 of constructing the hierarchy). In all cases a field size of 20×20 meters is assumed.

Figure 1 reports test performance of the champion individual at three points through each of the two phases necessary to develop the hierarchical SBB policy. Generation 1 corresponds to the champion individual at initialization. Generation 125 and 250 correspond to the performance of the champion individual half way and at the final generation. As there are two distinct phases of evolution the generation count cycles from 1 to 250 twice. The 'simulation hours' reflects the total accumulated number of hours of simulated play on the soccer server.

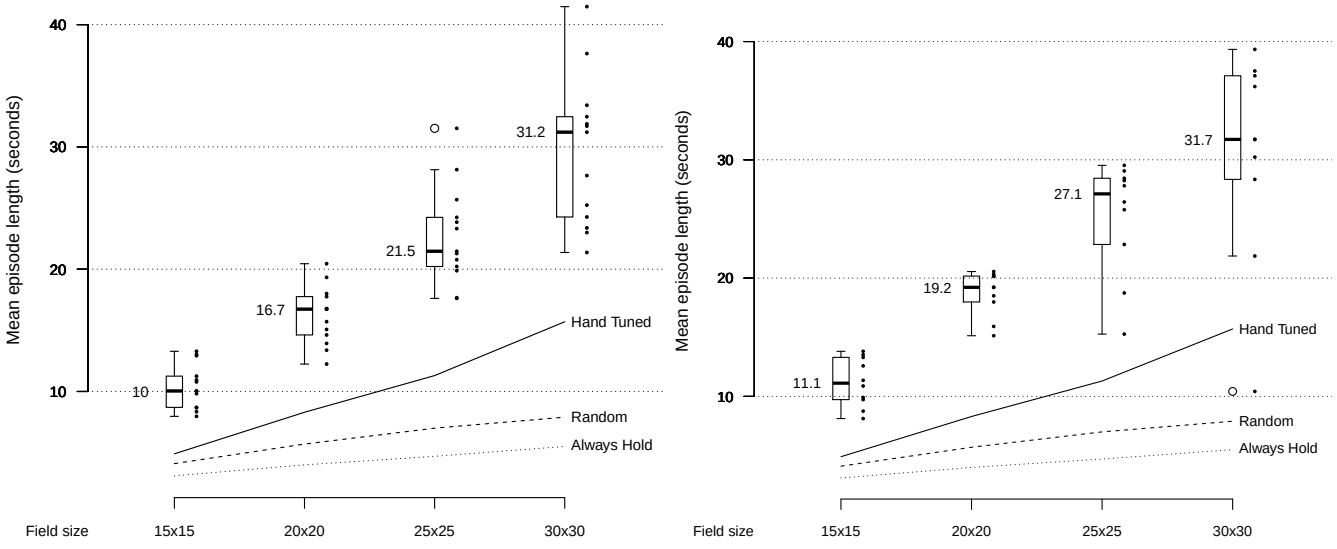
Comparing the average values reported for test performance during the hierarchical SBB runs in Figure 1 with the previously reported best test performance (Table 2) it is apparent that both forms of SBB provide significant improvements by the end of the first phase of evolution. This is achieved with 286 (with diversity) to 650 (no diversity) simulation hours, which is comparable or better than either of the neuro-evolutionary methods. The second phase of evolution, in which the second level of the hierarchical policy is built, emphasizes consistency (fitness sharing is disabled). This focuses the performance of champion individuals such that there is less variation between different runs. Additional experiments with fitness sharing applied during both phases of evolution did not reach this level of performance or consistency.

A further post training test of generalization is performed as follows. At the last generation, the champion individual is identified (as reported in the last column of Figure 1). Such a champion has been evolved and selected with respect to a field size of 20×20 meters. This *same* champion individual is then deployed on three other field sizes *without* any additional modification, Figure 2. Naturally, the task becomes easier on larger field sizes. Such an experiment was previously performed relative to Sarsa under a tile coded representation [21]. In the case of the Sarsa–tile coding result, performance was very sensitive to the configuration on which training was performed e.g., even when trained on a more difficult field size, performance on the easier field size was worse. Conversely, both SBB configurations perform better on the easier field sizes and exceed any of the default behaviours under the most difficult field size. Similar general trends were reported for HyperNEAT, albeit with a much lower overall level of performance [24]. It is also clear that for each field size, SBB with diversity provides improved generalization e.g., the difference between mean game times for each field size is: 1.1, 2.5, 5.6, 0.5 seconds respectively. In comparison, note that a total of 1.3 seconds of game time separates the ranking of all three neuro-evolutionary methods applied to the 3-versus-2 Keepaway task (Table 2).



(a) No diversity (b) With diversity

Figure 1: Average performance of champion policy against 1,000 test games. Two level hierarchical policy with 250 generations per level. Box plot reflects the quartile distribution and scatter plot the actual performance points from 12 runs. Numerical value reports the median of the 12 runs.



(a) No diversity (b) With diversity

Figure 2: Generalization of champion policy against multiple field sizes. Champions *evolved* and *identified* w.r.t. 20×20 meter field. 'Hand tuned', 'Random' and 'Always hold' represent baseline policies provided in the Keepaway code base [20].

5 Conclusion

The most popular form of the Keepaway soccer task (3-versus-2) has been revisited using a recently proposed scheme for evolving teams of programs hierarchically [15, 12, 8]. To do so, particular attention was applied to the definition of diversity maintenance. Specifically, a fitness sharing formulation was proposed, where this avoids the need to specify the relative weight of diversity to fitness. Including diversity results in the discovery of policies with an additional ≈ 3 seconds of play w.r.t. the field size for which evolution is performed and an additional 0.5 to 2.5 seconds of play when testing the same policy against games held on different field sizes. Moreover, when diversity is included, the policies provide games that last ≈ 3 seconds longer than previously published results.

Future research will investigate task-independent phenotypic *and* genotypic diversity measures as well as consider other formulations of the task domain, such as the more difficult 4-versus-3 configuration. Other properties of interest could include varying the players' field of view from 360 deg to more limited ranges or the ultimate objective of evolving entire soccer teams.

6 Acknowledgements

Authors gratefully acknowledge funding provided by NSERC Discovery (Canada).

References

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

[2] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer, 2007.

[3] E. K. 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] R. Calabretta, S. Nolfi, D. Parisi, and G. P. Wagner. Duplication of modules facilitates the evolution of functional specialization. *Artificial Life*, 6(1):69–84, 2000.

[5] S. Y. Chong, P. Tino, 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] G. Cuccu and F. Gomez. When novelty is not enough. In *EvoApplications – Part I*, volume 6624 of *LNCS*, pages 234–243. Springer, 2011.

[7] I. Dempsey, M. O'Neill, and Brabazon A. Survey of EC in dynamic environments. In *Foundations in Grammatical Evolution for Dynamic Environments*, volume 194 of *Studies in Computational Intelligence*, chapter 3, pages 25–54. Springer, 2009.

[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] S. M. Guastafson and W. H. Hsu. Layered learning in genetic programming for a cooperative robot soccer problem. In *European Conference on Genetic Prgramming*, volume 2038 of *LNCS*, pages 291–301, 2001.

[10] William H. Hsu, Scott J. Harmon, Edwin Rodriguez, and Christopher Zhong. Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In *Late Breaking Papers at the Genetic and Evolutionary Computation Conference*, 2004.

[11] T. Jung and D. Polani. Learning robocup-keepaway with kernels. In *JMLR: Workshop and Conference Proceedings – Gaussian Processes in Practice*, pages 33–57, 2007.

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

[13] 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.

[14] 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.

[15] P. Lichodzijewski and M. I. Heywood. The Rubik cube and GP temporal sequence learning: an initial study. In *Genetic Programming Theory and Practice VIII*, chapter 3, pages 35–54. Springer, 2011.

[16] Jan Hendrik Metzen, Mark Edgington, Yohannes Kassahun, and Frank Kirchner. Analysis of an evolutionary reinforcement learning method in a multiagent domain. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 291–298, 2008.

[17] L. L. Minku, A. P. White, and X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, 2010.

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

[19] S. Okasha. Multilevel selection and the major transitions in evolution. *Philosophy of Science*, 72:1013–1025, 2005.

[20] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, September 2005.

[21] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward robocup soccer. In *The Eighteenth International Conference on Machine Learning*, pages 537–544, 2001.

[22] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1321–1328, 2006.

[23] 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.

[24] Phillip Verbancsics and Kenneth O. Stanley. Evolving static representations for task transfer. *The Journal of Machine Learning Research*, 99:1737–1769, 2010.

[25] M. Waibel, L. Keller, and D. Floreano. Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3):648–660, 2009.

[26] R. A. Watson and J. B. Pollack. Modular interdependency in complex dynamical systems. *Artificial Life*, 11(4):445–458, 2005.

[27] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 21(1):1–35, July 2009.