Discovering Agent Behaviors Through Code Reuse: Examples From Half-Field Offense and Ms. Pac-Man

Stephen Kelly and Malcolm I. Heywood, *Senior Member, IEEE*

***Abstract*—This paper demonstrates how code reuse allows genetic programming (GP) to discover strategies for difficult gaming scenarios while maintaining relatively low model complexity. Critical factors in the proposed approach are illustrated through an in-depth study in two challenging task domains: RoboCup soccer and Ms. Pac-Man. In RoboCup, we demonstrate how policies initially evolved for simple subtasks can be reused, with no additional training or transfer function, in order to improve learning in the complex half-field offense (HFO) task. We then show how the same approach to code reuse can be applied directly in Ms. Pac-Man. In the latter case, the use of task-agnostic diversity maintenance removes the need to explicitly identify suitable subtasks *a priori*. The resulting GP policies achieve state-of-the-art levels of play in HFO and surpass scores previously reported in the Ms. Pac-Man literature, while employing less domain knowledge during training. Moreover, the highly modular policies discovered by GP are shown to be significantly less complex than state-of-the-art solutions in both domains. Throughout this paper, we pay special attention to a pair of task-agnostic diversity maintenance techniques, and empirically demonstrate their importance to the development of strong policies.**

***Index Terms*—Code reuse, coevolution, genetic programming (GP), half-field offense (HFO), Ms Pac-Man, task transfer.**

I. Introduction

DISCOVERING nonplayer characters (NPCs) for games through reinforcement learning (RL) is of broad significance to game AI in general [1], with specific importance to the following three classes of end user.

1) From the perspective of the game player, highly skilled, believable agent behaviors represent interesting game content and, thus, enhance the gaming experience.
2) From the perspective of the game designer, NPCs that exhibit human-level competence can be used to simulate playthroughs in newly created game environments, automating aspects of quality control in game design. Furthermore, automated testing of game content is essential to procedural content generation (i.e., game design by AI).
3) From the perspective of the AI researcher, games represent complex and challenging benchmark environments in which to investigate topics such as domain-independent AI, where the construction of NPCs plays a central role (e.g., [2]).

RL [3] implies that the learning agent performs multiple interactions with the task environment before enough feedback is received to qualify the NPC agent's strategy. Games potentially represent a generic example of a task with delayed feedback. The goal of the NPC agent is to discover a strategy, or policy, to play the game at a suitably challenging level. However, when scaling RL to increasingly difficult task domains it is not always possible to directly solve the original task formulation. For example, the number of interactions with the task might be too costly to perform or there is complete disengagement between the learner and task, i.e., no "learning gradient."

One approach to deal with this issue is to first learn policies for some source task and then "transfer" this to the required target task [4], [5]. In this paper, we present a framework for code reuse under GP that naturally supports task transfer. Policies are constructed hierarchically—hereafter a policy tree—during independent phases of evolution.[1] First, a diverse group of lower level source policies are evolved for related but different source task(s). A second phase of evolution is conducted to learn how to reuse the source policies under the target task by evolving a higher level switching policy. Such a process enables policies from multiple source tasks to be recombined to solve the overall target task. Moreover, we note that source tasks may be "automatically" discovered through mechanisms such as behavioral diversity/novelty [6], [7]. We are interested in comparing the relative merits of the two approaches.

In this paper, the utility of evolving policy trees through code reuse is initially demonstrated under the challenging multiagent soccer domain of half-field offense (HFO). In Phase 1, policies are evolved under two independent source tasks: keepaway (which evolves policies for retaining possession of the ball) and goal scoring. Phase 2 evolves a policy for the HFO target task by learning how to reuse a subset of the previously evolved behaviors, or a switching policy. HFO is much closer to the full-scale RoboCup soccer task and requires one team to defend their goal and the other team to score (the offense team has no goal or goalie). The objective, state, and action spaces are not the same for source and target tasks, a further criterion for meaningful task transfer [4], [5]. Furthermore, the agent's sensors and actuators are noisy, making the problem highly stochastic and partially

Manuscript received January 7, 2016; revised February 10, 2017, July 18, 2016, and July 28, 2017; accepted October 13, 2017. Date of publication October 26, 2017; date of current version June 14, 2018. This work was supported by the Discovery Program of the National Science and Engineering Research Council (Canada) under Grant RGPIN-2015-06117. *(Corresponding author: Stephen Kelly.)*

The authors are with the Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 4R2 Canada (e-mail: skelly@dal.ca; mheywood@cs.dal.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2017.2766980

[1]The term "policy tree" refers to the hierarchical organization of multiple programs to solve a task and has nothing to do with the representation assumed for each program.

observable. We demonstrate that the final explicitly hierarchical policy tree achieves an equivalent level of performance as the current RL state-of-the-art, but at a fraction of the model complexity. Next, the identical framework is applied to the popular *Ms. Pac-Man* video game, where most previous research, including the current state-of-the-art learning algorithm [8], relies on a similar prior task decomposition into multiple source tasks. We demonstrate that code reuse with diversity maintenance alone is sufficient to exceed the best results published to date. Additional decomposition through prior specification of source tasks improves on this. Furthermore, GP solutions are again shown to be simpler and more efficient than previous champions developed by neuroevolution.

In summary, the scope of this paper is to demonstrate the ability to learn high-quality NPCs for challenging game environments with minimal prior knowledge. The use of policy trees in constructing NPCs through task transfer implies that source task selection can be less constrained or even automated (see results under Ms. Pac-Man, Section V-B), and multiple source tasks can be considered. As the policy tree evolves, evolution determines which source policies are of merit and which are not. Ultimately, source policies might have been originally developed from the perspective of say, goal scoring, but utilized for maintaining possession (this actually transpired in practice, see Section V-D). Thus, pursuing the proposed approach to task transfer provides a game designer and/or AI researcher with more flexibility to discover effective/novel NPC policies that would otherwise never have been possible.

This paper is a significant extension of our earlier work with task transfer and diversity [9], addressing the following additions and improvements.

1) Initial attempts at task transfer [9] required carefully selected task-specific objectives in order to identify useful source policies posttraining. The work described here avoids this requirement through improvements to population diversity maintenance.
2) Diversity maintenance in [9] was assumed, but not empirically compared against the no diversity case. This work makes such a comparison and confirms the significance of multiple diversity metrics.
3) A linear combination of fitness and novelty scores for each agent was previously employed, which required tuning a scalar weighting parameter prior to evolution. This work uses a Pareto multiobjective approach, thus avoiding this parameter, with the addition of a multipopulation model further improving diversity.
4) The RoboCup tasks were only considered in [9], where this work also evaluates the framework in Ms. Pac-Man.
5) An in-depth literature review is included herein, placing the proposed approach in context with previous findings on transfer learning and code reuse, coevolution, and diversity maintenance.

The above topics are developed as follows: Section II provides a summary of related work, after which the source and target task domains are characterized (see Section III). The formulation assumed for evolving policy trees—symbiotic bid-based GP—is then summarized (see Section IV) with emphasis on how the architecture supports task transfer, and the approach taken for defining task-agnostic diversity mechanisms. An experimental study in both HFO soccer and Ms. Pac-Man domains is detailed in Section V with concluding remarks appearing in Section VI.

Animations under both RoboCup and Ms. Pac-Man, which illustrate how the modular GP strategies support decision-making during gameplay, are available at https://github.com/skellco/SBBTD. The site also provides source code for the SBB framework, as well as links to code and tutorials for both task domains.

II. Background

Many tasks are too complex to be approached without some amount of decomposition into simpler subtasks. For the purposes of this study, we consider two basic approaches for achieving such a decomposition: 1) *a priori* crafting of specific subtasks that are either solved sequentially or independently and then recombined (Section II-A); or 2) through cooperative coevolutionary (CC) mechanisms in which decision-making agents are expressed by multiple programs (see Section II-B). In supporting the identification of effective solutions, diversity maintenance represents a reoccurring theme, with Section II-C highlighting the issues of importance to this paper.

A. Transfer Learning

Transfer learning (under RL) refers to the use of solutions from easier source tasks in an attempt to solve a more difficult target task [4], [5]. The key issues in transfer learning can be summarized in terms of the following:

1) what knowledge is transfered;
2) how to implement the knowledge transfer between tasks that may differ with respect to sensors, actions, and objectives; and
3) How to quantify the utility of transfer. As such, at least from an evolutionary computation perspective, task transfer in RL has a broad heritage.

In particular, we identify incremental evolution [10] and layered learning [11] as having similar motivation and sharing the same broad issues.

Incremental evolution assumes that the source tasks follow a sequence (or chain [12]) in which each source task is incrementally more complex than the last. Naturally, it is only possible to solve the first source task in the sequence tabula rasa.[2] Moreover, the issue of what to transfer was defined in terms of a single champion individual that is then used to seed the population for the next source task. Thus, variation operators applied to the genotype of the champion are deemed to provide a suitable starting point for generating content for the next population. Variations on this theme might carry the content of the population "as is" between consecutive source tasks (e.g., [13], [14]) or transfer multiple champion individuals [15]. Implicit in this process is the approach for addressing the issue of "how to transfer," i.e., the *a priori* sequence of source tasks. Moreover, there is sufficient similarity between sensors and actions

[2]Solving a task "tabula rasa" implies the direct application of RL to a task without any attempt to provide any *a priori* decomposition into simpler task(s).

for the champion individual(s) to directly carry over between consecutive source tasks.

Layered learning [11] is a more recent development and as such captures a wider range of mechanisms in which task transfer might appear. Specifically, rather than assume that source tasks represent a sequence of explicitly related tasks of incremental complexity, layered learning may define source tasks that are initially independent. For each source task a separate training environment with unique states and/or actions may appear. At some point, transfer is accomplished through a gating or switching policy that joins independent solutions to each source task, where such a switching policy might take the form of a prior decision tree (no learning involved) or could also be evolved [16].

The final development that task transfer brings is to consider what happens when some source task action(s) and/or sensors are not present or have altered meaning in the target task [4], [5]. Layered learning assumes that solutions to source tasks do not require any reinterpretation by a later target task. Conversely, policies from source tasks may have incomplete partial alignment, each covering a portion of the sensors and actions available, but jointly still not covering the complete sensor/actuator space of the target task. A learned intertask mapping is one method of addressing this issue [17], [18]. In addition, Hyper-NEAT with a "Bird's-eye view" [19] has been proposed as a way to evolve generalized state representations such that an agent can transfer to more complex tasks without additional learning, assuming the objective remains the same.

B. Cooperative Coevolution

The principal advantage offered by adopting a CC approach is automatic problem decomposition, i.e., evolving solutions in the form of interacting, coadapted subcomponents [20]. It is then possible for credit assignment to identify useful "modules" and associate them with specific contexts, i.e., the interaction between specialization and modularity [21]. Moreover, the impact of variation operators is further subscribed to particular modules, resulting in less likelihood of disruption to all the functional properties of a policy [22].

Early formulations for CC made use of knowledge regarding the number of subcomponents. Each subcomponent was associated with a separate population. For example, given a prior specification for a neural network topology, then different populations might be associated with each weight in the network [23], [24]. A heuristic is now necessary for choosing representatives from each population, potentially differentiating between group fitness (e.g., fitness of the resulting neural network) versus fitness of individual subcomponents.

Within the context of GP, cooperative coevolution implies that each subcomponent is a program and a policy is represented as a team (of programs) [25], i.e., a group behavior. Assuming that knowledge is available for defining how many programs should participate within a team, then separate populations can be associated with each program [26]. The issue of selection heuristics again appears, where this can be performed at the group or subcomponent (program) level. Indeed, alternating between the two appears to provide robust teams under multiclass classification tasks [26].

Bidding mechanisms have been used to provide the basis for the evolution of the number of subcomponents. Both [27] and [28] utilize such a formulation. In this paper, we assume the former as it does not rely on task specific definitions for fitness at both the individual and group levels. Indeed, we subscribe to the idea that organisms must be evaluated as a whole, rather than independent modules, in order to evolve cooperation [26]–[28]. Moreover, balancing the need to develop specialization (a property of a subcomponent) and cooperation (a property of a collection of subcomponents) is where diversity maintenance and neutrality also potentially have parts to play.

One interesting development that will later be employed is that of symbiosis [29]. Symbiosis explicitly represents the group and subcomponent using independent populations (host and symbiont, respectively). Two works in particular have used such a framework to support CC: 1) Symbiotic, Adaptive Neuro-Evolution (SANE), an architecture for evolving weights (symbiont) and neural networks (host) [30]; and 2) SBB, where programs are symbionts and teams of programs are hosts [27]. In both cases the host population is where a complete organism is expressed (and where fitness evaluation takes place), but in terms of pointers to members of the symbiont population. In effect a combinatorial search is being performed by the host population for "good" symbiont combinations. If host individuals assume a variable length representation (as in [27]), then the number of subcomponents is also evolved.

C. Diversity Maintenance

Promoting diversity is motivated by an underlying desire to avoid premature convergence and/or produce individuals that succeed/fail in different ways. We identify three generic approaches (that can potentially be used in combination): competitive coevolution, regularization, and multiple populations, of which the later two methods pertain to this work.

Diversity maintenance through regularization implies that the fitness function incorporates discounting for properties other than goal directed task performance. Indeed, at one extreme is novelty as the objective [6], [7], [31]. In this paper, we assume two task-agnostic diversity mechanisms (see Section IV-C). These are motivated in part by our intuition that both structural diversity and behavioral diversity are likely to promote specialization. Indeed, it has been previously suggested that GP applied to complex problems is likely to benefit from multiple diversity mechanisms [32]. Our specific method for maintaining structural diversity is inspired by the approach taken in [33] for promoting diversity under a neuroevolutionary setting. In addition, promoting behavioral diversity, or diversity in the observable characteristics of policies as they interact with the environment, has been shown to significantly improve development in evolutionary robotics [34]. Moreover, we note that the approach taken to combining multiple diversity measures also has an impact on the development of modularity. Thus, instead of all forms of regularization being present all the time, switch-

ing between different objectives can potentially be beneficial [35], [36].

Finally, in the case of Island formulations for multipopulation frameworks, we note that the underlying motivation is to provide programs with "independent errors" (in their behavioral traits) [37]. However, the issue of how to identify the optimal group of individuals post training needs to be explicitly addressed. Moreover, it has been noted that providing more explicit mechanisms to maintain diversity (such as regularization, different performance objectives or CC) may provide a more direct method for maintaining diversity (e.g., [26]).

III. Task Domains

The algorithm presented in this paper is empirically evaluated in two challenging game environments: RoboCup two-dimensional (2-D) simulated soccer and Ms. Pac-Man. Both environments are widely used for evaluating machine learning in sequential decision-making tasks. Both environments also present interesting opportunities for task transfer because each can be decomposed into a variety of simpler source tasks, in which the learning agent can gain experience prior to attempting the more complex target task.

A. RoboCup Soccer

In RoboCup 2-D soccer, researchers have primarily focused on subtasks of the full game, such as keepaway [38], in order to more directly assess the specific contributions of various innovations while maintaining essential aspects of the soccer task domain [19], [33], [39]–[43]. These tasks are of general relevance to RL because they have the following properties:

1) multiagent, implying that different agents will need to decide on which role they will need to prioritize at different parts of the game;
2) stochastic, where this affects both sensor information and actions suggested by agents;
3) utilizing real-valued state variables;
4) can be parameterized to represent a family of incrementally more difficult tasks.

It is the latter property that has seen their recommendation as a benchmark for assessing the utility of RL algorithms in general and task transfer in particular [4], [40], [41]. We focus on three specific subtasks: goal scoring, keepaway, and HFO.

In keepaway, a team of K keepers tries to maintain possession of the ball for as long as possible while an opposing team of $K-1$ takers attempt to gain possession [40]. The game ends if the ball is kicked out of bounds or captured by the takers, at which point each keeper receives the game duration in milliseconds as a reward signal. Thus, the task objective is for keepers to learn a policy to maximize the length of play against the takers, which follow an *a priori* strategy.[3]

In this paper, we are specifically interested in a 4 versus 3 (4v3), or $K=4$, version of the task. Our version of keepaway is played on a 60×60 m field, instead of the more common 25×25 m bounding box, as our ultimate objective is the HFO task as opposed to keepaway.[4] The game is initialized with keepers in four corners of a 15×15 m^2 at center field, the ball placed near one keeper, and the takers in the centre. Keepers learn to deploy a set of domain-specific macroactions $\{hold, getOpen, pass(k)\}$, at discrete time steps of 100 ms. Each macroaction may last more than one time step, and control is returned to the player when the macroaction terminates. A keeper in possession of the ball has the option of either $hold$ or $pass(k)$, where k indexes one of its $K-1$ teammates. Keepers not in possession of the ball assume the *getOpen* macroaction.

At each time step, keepers receive state information in the form of 11 real-valued distance and angle sensor inputs that describe the location of other players on the field. The sensors are noisy, thus players must cope with inaccurate state information. Furthermore, the players' actuators are unreliable, often resulting in inaccurate passes and fumbled attempts to hold the ball. The presence of noisy sensors and actuators makes credit assignment particularly difficult and represents one of the issues specifically addressed by the proposed approach.

HFO is another subtask of RoboCup soccer with significantly more complexity than keepaway [41]. In HFO, a team of offense players tries to manoeuvre the ball past a defending team and around the goalie in order to score. A game ends if the following holds:

1) the ball is kicked out of bounds;
2) the ball is captured by an outfield defense player;
3) the ball is intercepted by the goalie; or
4) a goal is scored.

The defense team in HFO follow a prespecified behavior defined by the original task [41]. Each offense player receives one reward signal of either $\{0.8, 0.8, 0.9, 2\}$[5] for each of the above end-game conditions.

We are specifically interested in 4 versus 4 (4v4) HFO, in which the offense team has four players while the defense team is made up of three outfield defenders plus a goalie. HFO is played on one half of the full soccer field. All outfield players are initialized within a 15×15 m^2 as in keepaway, but in HFO the centre of the initialization box is stochastically placed somewhere on the field roughly 55 m from the goal. Naturally, the goalie is initialized in the goal region.

Offense players in HFO learn to deploy a somewhat different set of macroactions than in the keepaway task. Their options at each time step are $shoot, dribble, getOpen, pass(k)$. Thus, an offense player in possession of the ball has the option to either dribble towards the goal, pass to a teammate, or shoot. As per the keepaway task, each macroaction may last more than one time step and control is returned to the player when the macroaction terminates. The HFO task has the same 11 sensors available in keepaway, plus six additional sensor inputs that summarize distance to the goal for each offense player and the angle of

[3] Use of an *a priori* strategy leads to consistent benchmarking practices.

[4] Generalization between different keepaway field sizes has been demonstrated [19], [44]. However, it is more difficult to evolve behaviors for smaller fields, where this comes at a considerable computational cost.

[5] This reward structure was obtained by adding 1 to the rewards used by Sarsa under the same task [41], thus defining reward in terms of positive values alone. We do not claim any optimality for these values from an evolutionary computation perspective.

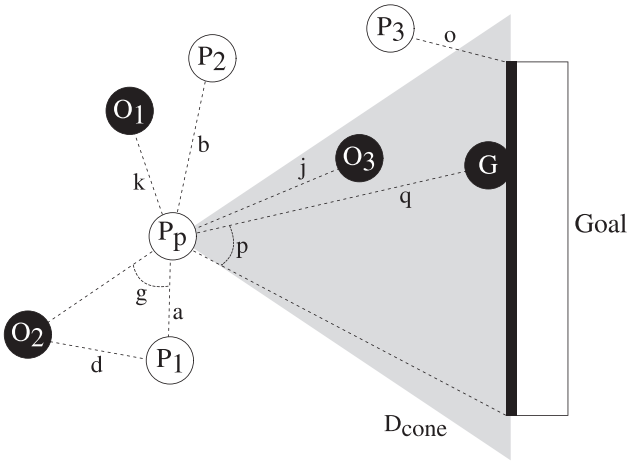

Fig. 1. Sample sensor inputs for the 4v4 half field offense task. P_j denotes players where P_p is the special case of the player from which egocentric measurements are made. G is the special case of the "goalie" opponent, while O_i denotes outfield opponents (defense players). Equivalent measurements are repeated for each player.

the maximum scoring window for the player with the ball, i.e., how open the net is (see Fig. 1). As in keepaway, all sensor inputs and actuators are unreliable. Readers are refereed to [9] for a detailed list of all sensors, whereas the original HFO paper provided a complete definition of the task [41].

The goal scoring task is similar to the HFO environment except that the initial position of players is restricted to within 15 m of the goal. While some passing and ball handling will still be necessary from this position, offense players no longer need to manoeuvre the ball down field and can, therefore, concentrate on scoring goals.

In this paper, agents will begin by learning keepaway and scoring source tasks separately, and then transfer this experience to HFO. keepaway and goal scoring are clearly subtasks of HFO. For example, from the stochastic start position in HFO, offensive players essentially have to maintain possession and dribble towards the goal before trying to score. However, the ball-handling skills under HFO are different from keepaway, requiring the players to maintain possession of the ball *and* create goal scoring opportunities. Furthermore, players that only have experience in the scoring task, having only observed the environment near the goal region, will need to cope with larger distance and angle sensor readings under the HFO task. Thus, success in the HFO environment requires additional capabilities and reinterpretation of skills learned from the source tasks.

B. Ms. Pac-Man

The Ms. Pac-Man simulation assumed in this paper is a close approximation of the 1982 arcade version of the game, which is one of the most popular video games of all time. Best results on this task to date have all benefited from incorporating some form of task-specific bias. Specific examples include assumptions regarding the optimal amount of modularity to include under modular neuroevolution [8], or providing appropriate end game tactics for use under Monte Carlo tree search (MCTS) [45].

The objective is to identify a policy for guiding Ms. Pac-Man through a series of four mazes, each scattered with multiple regular pills and four power pills. When Ms. Pac-Man finds a pill, she eats it, gaining ten points per regular pill and 50 points per power pill. When all pills have been eaten, Ms. Pac-Man advances to the next maze. However, the game includes four opponents, or ghosts, which emerge sequentially from a lair in the centre of each maze and pursue Ms. Pac-Man. If a ghost touches her, she loses a life. Ghosts behave nondeterministically, necessitating a responsive strategy to out-maneuver them, rather than simply memorizing an effective trajectory through the maze. Normally, Ms. Pac-Man and the ghosts move at the same speed. However, when Ms. Pac-Man eats a power pill, all ghosts become edible and reduce their speed by 50%, making it possible for Ms. Pac-Man to catch and eat them. The first, second, third, and fourth ghosts eaten are worth 200, 400, 800, and 1600 points, respectively. Ms. Pac-Man is, therefore, a predator-prey scenario that requires the agent to switch between multiple modes of behavior throughout an episode. The agent must assume the role of predator and prey at different times, actively pursuing ghosts when they are edible and evading ghosts when they are threats [8]. Multimodal behavior of this nature is similar to the ball handling and scoring requirement in the HFO subtask of RoboCup soccer. However, unlike HFO, multiple mode transitions are explicitly required in Ms. Pac-Man. This, coupled with the fact that a single game in Ms. Pac-Man requires thousands of decisions form the agent, places additional burden on credit assignment and makes the task particularly difficult.

Schrum and Miikkulainen [8] provide two key insights into learning Ms. Pac-Man agents: 1) Ms. Pac-Man can be formulated as a multiobjective problem in which maximizing the number of pills and ghosts eaten are treated as distinct objectives; and 2) Explicitly encouraging modularity in the learning representation facilitates the discovery of multimodal behavior. The prior task decomposition required by the former insight suggests that task transfer may be applicable to this domain. The requirement for modularity is perfectly suited to our proposed method, an explicitly modular framework for cooperative coevolution of GP teams, detailed in Section IV.

Similar to decomposing the objective space *a priori*, various GP methods have made effective use of prior game knowledge in configuring the environment. Alhejali and Lucas [46] designed multiple tightly-constrained game scenarios, or training camps, in which the agent was trained relative to specific objectives prior to attempting to learn a more general controller. Their GP also employed high-level sensors, such as those directly alerting Ms. Pac-Man to dangerous states, and high-level actions that, for example, may assume control for multiple time steps and lead Ms. Pac-Man out of dangerous states. However, GP restricted to low-level actions and sensors has also been successful in this task [47]. Low-level sensors include only general information like "distance to the nearest pill," and thus do not imply a specific interpretation by the agent, i.e., "good" and "bad" states. Low-level actions imply that the Ms. Pac-Man controller simply selects a direction to move (UP, DOWN, LEFT, or RIGHT) in each time step of the game. Brandstetter and Ahmadi [47]

demonstrated how a GP individual with low-level actions and sensors can be designed to evaluate and rate each direction, then simply move in the direction with maximum rating. Our work extends this to the case in which polices consist of multiple cooperative GP individuals, or teams. As such, the experiments in this paper assume low-level actions and sensors for Ms. Pac-Man, as outlined in [8].

In each time step, there are a total of 95 sensors: seven nondirectional sensors (e.g., num. of regular pills left in maze, num. of edible ghosts, remaining edible ghost time) and 22 directed sensors for each of the four directions (e.g., distances to the first, second, third, and fourth closest ghosts, whether each ghost is approaching, distance to nearest power pill and maze junction.). For a complete list of sensors and detailed description of each sensor, see 'conflict sensors" in [8].

We frame Ms. Pac-Man as a suitable case for task transfer in which the two broad objectives of the game, eating pills and eating ghosts, are treated as source tasks, with overall game score as the target task. While both source objectives must be maximized to achieve high scores, the relative benefit in pursuing one objective over the other needs to be managed effectively by the agent under the target task. For example, eating pills too aggressively would result in clearing the maze before all the ghosts are eaten, missing out on significant scoring opportunities, while eating ghosts too aggressively leaves Ms. Pac-Man vulnerable to threat ghosts as she clears pills from difficult regions of the maze. As such, behaviors developed under the source task(s) must be adapted, or recontextualized under the target environment. Furthermore, Ms. Pac-Man requires specific skills for evading threat ghosts and luring both threat and edible ghosts into disadvantaged positions, neither of which are explicitly rewarded in either the source or target objective functions.

IV. Symbiotic Bid-Based GP

Symbiotic bid-based (SBB) GP is a hierarchical framework for coevolving teams of simple programs over two distinct phases of evolution. The first phase produces a library of diverse, specialist teams of limited capability. The second phase builds more general and robust policies by reusing the library, essentially building generalist strategies from multiple specialists (see Fig. 2). Thus, diversity maintenance is critical during the first phase of evolution to ensure the identification of a wide range of specialist behaviors. See [44] and [48] for an example of this within the context of the keepaway task and [49]–[51] for additional applications of SBB under RL tasks. Relative to previous work, the SBB algorithm described in this paper adopts a more general, task-agnostic approach to diversity maintenance, and explicitly addresses how the incremental development of a policy tree facilitates knowledge transfer between different tasks. Three critical components of the algorithm are summarized as follows:

1) building teams of programs for sequential decision-making tasks;
2) constructing policy trees through code reuse;
3) task-agnostic diversity mechanisms.

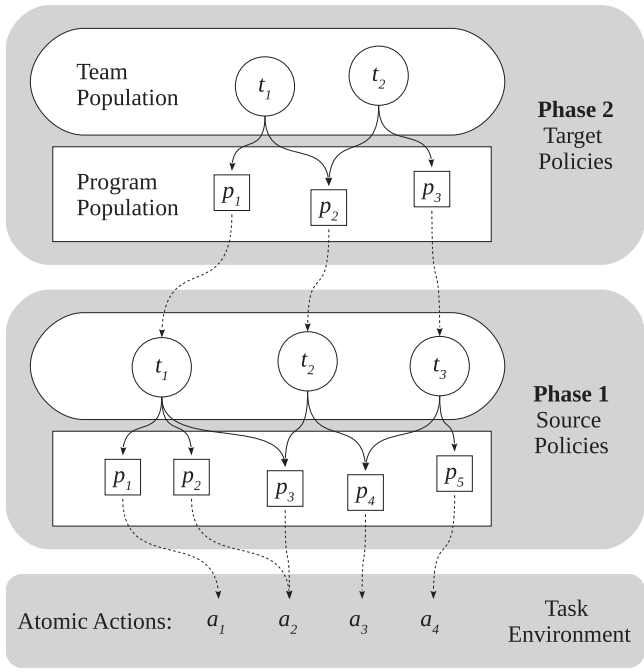

Fig. 2. Constructing policy trees through symbiotic bid-based (SBB). Two independent phases are assumed. Each phase defines policies in terms of teams of programs (host and symbiont). Phase 1 identifies a diverse set of relatively simple "source" policies. Phase 2 learns how to reuse Phase 1 source policies for solving a different, more complex target task.

Algorithm 1: Example program in which execution is sequential. Programs may include two-argument instructions of the form $R[x] \leftarrow R[x] \circ R[y]$ in which $\circ \in +, -, x, \div$; single-argument instructions of the form $R[x] \leftarrow \circ(R[y])$ in which $\circ \in \cos, \ln, \exp$; and a conditional statement of the form if $(R[x] < R[y])$ then $R[x] \leftarrow -R[x]$. $R[x]$ is a reference to an internal register, while $R[y]$ may reference internal registers or state variables (sensor inputs). Determining which of the available sensor inputs are actually used in the program, as well as the number of instructions and their operations, are both emergent properties of the evolutionary process.

1) $R[0] \leftarrow R[0] - R[3]$
2) $R[0] \leftarrow R[0] \div R[7]$
3) $R[1] \leftarrow Log(R[0])$
4) IF $(R[0] < R[1])$ THEN $R[0] \leftarrow -R[0]$
5) RETURN $R[0]$

A. Teams of Programs

Teams in SBB are comprised of multiple linear genetic programs [25], or register machines. Each program returns a single real number, the result of executing a sequence of instructions that operate on sensor inputs or internal registers (Algorithm 1). Each program is also associated with one action, which refers to either: 1) an atomic action[6] from the task domain; or 2) a previously evolved team, i.e., code reuse. In sequential

[6]A discrete set of task specific "atomic" actions are identified *a priori* (see Section III).

decision-making tasks, all programs in the team will execute their code relative to the current state variables at each time step. The team then deploys the action of the program with the highest output, or the winning bid. Thus, the role of each program is to define a unique context for deploying its action given the current state of the environment.

In its simplest form, SBB maintains one population of programs and one population of teams (Phase 1, Fig. 2). Team development is driven by a (variable length) genetic algorithm (GA) such that a fixed number of the least desirable teams (Mgap) are deleted in each generation and replaced by the offspring of surviving teams.[7] Thus, the GA is driven by group-level selection, i.e., the team is judged as a whole rather than by the performance of individual components, which is a critical factor in the evolution of cooperation [26]–[28]. As such, programs have no individual fitness. At each generation, orphaned programs—those that are no longer a member of any team—are deleted.

B. Code Reuse

In order to support code reuse (with or without task transfer), multiple initializations of the basic two-population model (see Section IV-A) can be vertically stacked and developed bottom-up over sequential phases of evolution (see Fig. 2). Specifically, the team and program populations developed in Phase 1 are cached and treated as a library of reusable code. Thus, Phase 1 teams now represent code, which is reused as actions for evolution during Phase 2. One contribution of this study is to show that such a bottom up hierarchical approach to policy development provides natural opportunities for task transfer since each level of the hierarchy can be developed relative to different components of the task, e.g., different environments or objectives.

C. Specialization and Diversity Maintenance

In order to mitigate the effect of noisy fitness evaluations in stochastic tasks, the fitness of a policy is often defined by the mean reward across multiple evaluations in the task environment. However, in order to promote population diversity, each policy's novelty must also factor into the selection process, where novelty refers to the degree of similarity between a policy and all other members of the same population. The intuition behind explicit diversity maintenance is that searching for good things as well as different kinds of things has two key benefits: 1) diversity helps prevent premature (team) convergence; and 2) when developing a library of reusable code, a diverse population represents a versatile toolbox for subsequent reuse [51] (see Phase 1, Fig. 2). As such, two broad approaches to diversity maintenance are assumed in this paper: regularization and multiple (source task) populations.

1) Regularization: Diversity through regularization implies that the teams are selected for policies that produce high rewards *and* exhibit unique qualities relative to all other members of the *same* population. Rather than assuming a single metric to quantify the similarity of two policies, we adopt two different metrics and switch between them, selecting either metric with equal probability at each generation. Switching between dissimilar distance metrics:

1) avoids introducing yet another scalar weighting parameter;
2) has been shown to be as effective as combining metrics in a multiobjective formulation [7];
3) actively encourages the development of modularity [35], [36]. Such metrics are also ideally task-agnostic.

A program-utility distance metric is used to characterize diversity across the programs that are "active" within a team, and as such defines diversity as a group property. A program is considered active if it contributes an action at least once during the life of a team. The distance between teams is summarized as the ratio of active programs common to both teams. Thus, the program-utility distance between teams i and k is

$$\text{dist}(t_i, t_k) = 1 - \frac{\text{Prog}_{\text{active}}(t_i) \cap \text{Prog}_{\text{active}}(t_k)}{\text{Prog}_{\text{active}}(t_i) \cup \text{Prog}_{\text{active}}(t_k)} \quad (1)$$

where $\text{Prog}_{\text{active}}(t_x)$ represents the set of active programs in team x. There is nothing task specific in this metric.

A behavioral distance metric characterizes a team by how it interacts with the environment. At each decision point in a game, the state and subsequent action taken by the team are recorded. Each state variable is discretized to $[0, 1, 2]$, or low, medium, high. Thus, for each training game a *profile* vector is recorded, $\vec{p} = [\{a(t_s), \vec{s}_d(t_s)\}, t_s \in T]$, where $a(t_s)$ is the atomic action taken, $\vec{s}_d(t_s)$ is the discretized state observation, and T represents every decision point in the associated game. Teams maintain a historical record of the profile $\vec{p}$ for every game played during training. We define $\vec{P}$ to be the concatenation of all profile vectors $\vec{p}$ in a team's historical record. Let $Z(\vec{P})$ be the compressed length[8] (in bytes) of profile vector $\vec{P}$. The behavioral distance between a pair of teams can now be summarized as the normalized compression distance (NCD) [52] between their corresponding profiles

$$\text{NCD}(\vec{P}_i, \vec{P}_k) = \frac{Z(\vec{P}_i\vec{P}_k) - \min(Z(\vec{P}_i), Z(\vec{P}_k))}{\max(Z(\vec{P}_i), Z(\vec{P}_k))} \quad (2)$$

where $Z(\vec{P}_i\vec{P}_k)$ is the compressed length of the two profile vectors, $\vec{P}_i$ and $\vec{P}_k$, concatenated. NCD returns a real number between 0 and ≈ 1.2 that characterizes the difference between profile vectors. Equation (2) leverages the ability of compression algorithms to filter redundancies in data. For example, if $\vec{P}_i$ and $\vec{P}_k$ are very similar, then $Z(\vec{P}_i\vec{P}_k) \approx Z(\vec{P}_i) \approx Z(\vec{P}_k)$, in which case $\text{NCD}(\vec{P}_i, \vec{P}_k) \approx 0$. NCD is informative even when comparing vectors that differ in length, which is important because each team's profile will contain a variety of episodes with different outcomes.

Balancing fitness and novelty can be achieved in several ways, including fitness sharing [49], linear weighted sum of fitness and

[7]Team offspring are created by cloning the team along with all its programs, and then applying mutation-based variation operators to the cloned team and programs, as parameterized in Table II.

[8]This paper uses the bzip2 data compression library for calculating the length (in bytes) of a compressed sequence of integers.

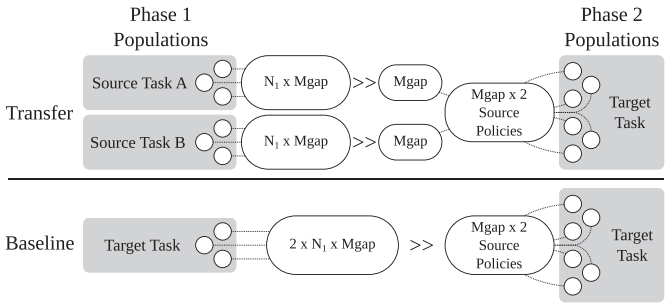

Fig. 3. Selecting "source" policies from Phase 1 of evolution for recombination at Phase 2. N_1 is the number of independent populations evolved in Phase 1 for each source task. M gap is the final number of policies produced from each population. $\gg$ symbolizes the filtering process in which policies are ranked and selected based on average training reward. Phase 1 is parameterized such that transfer and baseline cases are allocated equivalent computational resources.

novelty [9], [31], or multiobjective optimization [7]. In this paper, we adopt a simple two-objective approach based on a Pareto dominance relation which combines task fitness with novelty as measured using one of the above diversity measures, or $\text{Fit}(t_i)$ is the mean reward over all games in which t_i has been evaluated, and $\text{Nov}(t_i)$ is the mean distance, as measured with either diversity metric [see (1) or (2)], between t_i and the $k\text{nn} = 15$ nearest neighbors in the same population.[9] At the start of each generation we choose which diversity metric to assume throughout that generation with equal probability.

A team t_i is said to dominate another team t_j, if t_i is better that t_j in at least one objective and no worse in the others.[10] Finally, each team is ranked prior to selection, where we assume that more desirable teams will be dominated by fewer individuals

$$\text{Rank}(t_i) = 1 - \frac{\text{Dom}(t_i)}{M\text{size}} \tag{3}$$

where $\text{Dom}(t_i)$ is the number of teams in the same population that dominate t_i.

2) Multiple Populations: A user may have multiple candidate source tasks in mind, but no way of *a priori* prioritizing or selecting which source tasks to employ in practice. We, therefore, assume that all candidate source tasks will be developed in parallel during independent instances of Phase 1 evolution. Phase 2 will then identify which specific source policies to deploy and how. As shown in Fig. 3, the case of transfer" implies that multiple different source tasks are learned in Phase 1, where the alternative is to simply assume the target task throughout, or "baseline" in Fig. 3. In effect, the transfer case requires prior information to inform the identification of code for reuse. Conversely, the baseline scenario assumes that diversity maintenance is sufficient for this purpose.

V. Results

A representative comparator algorithm establishes a comparative measure of performance for each task domain as follows.

[9] $k\text{nn} = 15$ was selected as a reasonable value based on empirical tuning prior to the experiments described in this paper.

[10] Formally, a solution a_i dominates another solution a_j if $\forall k[f_k(a_i) \geq f_k(a_j)] \wedge \exists k[f_k(a_i) > f_k(a_j)]$.

TABLE I
Algorithms Benchmarked in Section V

Label	Description
SBB.TD	SBB with transfer and switching diversity maintenance
SBB.T	SBB with transfer, no diversity maintenance
SBB.D	SBB no transfer, with switching diversity maintenance
SBB.D.B	SBB no transfer, Behavioral diversity maintenance only
SBB.D.P	SBB no transfer, Prog-Utility diversity maintenance only
SBB	SBB no transfer, no diversity maintenance
SarsaRBF	Sarsa with Radial Basis Function Approximator
HAND	Hand-crafted HFO policy
MM-NEAT	Modular Multiobjective NEAT with 2 Modules

SarsaRBF and HAND are comparison algorithms for HFO soccer. MM-NEAT is a comparison algorithm for Ms. Pac-Man. Shaded rows indicate experiments conducted only in HFO soccer, All other SBB treatments are tested in both domains

TABLE II
Parameterization of Team and Program Populations

Team (GA) population			
Parameter	Value	Parameter	Value
N_1	10	N_2	20 (10)
$e_{\max}$ Phase 1	250 650 (181 800)	T	50 (3)
$e_{\max}$ Phase 2	113 400 (90 900)	e_{test}	1000 (100)
M size	180	M gap	50%
$p_{m\,d}, p_{m\,a}$	0.7	ω	30
$p_{m\,m}$	0.2	$p_{m\,n}$	0.1
Program population			
numRegisters	8	maxProgSize	96
$p_{\text{delete}}, p_{\text{add}}$	0.5	$p_{\text{mutate}}, p_{\text{swap}}$	1.0

$p_{m\,x}$ denotes a mutation operator in which: $x \in \{d, a\}$ are the prob. of deleting or adding a program, respectively; $x \in \{m, n\}$ are the prob. of creating a new program or changing the program action, respectively. $e_{\max}$ is the maximum number of evaluations over all populations (See Fig. 3) per phase. Parameters for Ms. Pac-Man are in parentheses.

1) SarsaRBF: The Sarsa temporal difference method with radial basis functions. SarsaRBF represents the current state-of-the-art in the HFO task [41].
2) MM-NEAT: Modular multiobjective NEAT with two modules, representing the current state-of-the-art in the Ms. Pac-Man task [8].

In both cases, we assume the parameterization/source code from the original implementations.

Under SBB, we are particularly interested in the utility of diversity maintenance (see Section IV-C), thus all SBB cases are evaluated with and without diversity maintenance. When no diversity is enforced, teams are selected based on training reward alone, or $\text{Fit}(h_i)$ in Section IV-C (balancing fitness and novelty). Table I summarizes all experimental cases.

Table II summarizes the parameterization assumed for SBB.T. Values specific to Ms. Pac-Man are in parentheses, while all other parameters are common to both domains. The maximum number of training evaluations $(e_{\max})$ is a function of the population size, number of generations, and number of evaluations for each individual per generation. These parameters can be modified based on the nature of the task. For example, the goal scoring source task was easier to learn than the keepaway source

task, and thus required fewer generations. The number of generations per phase and the number of evaluations per generation for each individual (e_{gen}) were as follows:

1) scoring: 60 generations, $e_{\text{gen}} = 25$;
2) keepaway: 125 generations, $e_{\text{gen}} = 10$;
3) HFO: 125 generations, $e_{\text{gen}} = 10$;
4) all Ms. Pac-Man tasks: 100 generations, $e_{\text{gen}} = 10$.

In order to keep the maximum number of evaluations constant over all experiments, SBB assumes the same parameterization as SBB.T *except* in the case of the Phase 1 generation limit. When multiple source tasks are not employed at phase 1 (baseline in Fig. 3), the generation limit is increased such that $e_{\max}$ for phase 1 is equivalent across all experiments in the respective task domain. In HFO, SarsaRBF learning curves appeared to plateau after roughly 20 000 episodes, at which point training was stopped and policies saved for evaluation under test conditions.[11] MM-NEAT evolved a population of 100 individuals with 10 evaluations each over 200 generations, for a total of $100 \times 10 \times 200 = 200\,000$ evaluations.

Due to the abundant sources of noise in RoboCup[12] and Ms. Pac-Man,[13] posttraining test games (e_{test}, Table II) are required to provide an accurate measure of performance for the single champion policy from each run. However, the computational cost of evaluations precludes testing an entire population over a large number of games in order to identify such a champion. Thus, the best policy in any population is identified by first identifying the single policy with the highest training reward in each of the final T generations. If the policy with highest training reward has already been saved, the next-best policy is identified. These T unique cached policies are then tested in e_{test} games, and the highest scoring policy represents the population champion assumed for test.

A. HFO Test Performance

Fig. 4 reports test performance for the four SBB configurations under the HFO task (first six rows of Table I), along with SarsaRBF and the hand-crafted policy. It is apparent that both SarsaRBF and SBB.T reach similar levels of performance. Indeed, there is no statistical difference between SarsaRBF and either of the SBB.T distributions in Fig. 4 (the Mann–Whitney rank test, $P < 0.05$), as indicated by the Gray bracket across the top of the figure grouping these distributions. Diversity maintenance does not provide a significant benefit when task-transfer is used in the HFO task, nor is it harmful. The fact that diversity does not make a significant difference here is likely a further testament to the utility of knowledge transfer. Source policy reuse provides enough leverage to overshadow the contribution of diversity maintenance, with all SBB.T cases reaching state-of-the-art performance. Indeed, neither SBB case without transfer is able to reach a competitive level with SarsaRBF (SBB and SBB.D in Fig. 4). However, diversity maintenance becomes important in the absence of task transfer, where only the SBB case with diversity maintenance (SBB.D) was able to outperform the hand-crafted policy. Finally, in order to test the utility of stochastically switching distance metrics within the diversity mechanism (see Section IV-C), we performed one run with strictly behavioral diversity (SBB.D.B) and one run with strictly program-utility diversity (SBB.D.P). While standalone behavioral distance worked better than program utility, stochastically switching between metrics proved important, as neither metric worked well enough alone to outperform the hand-crafted baseline.

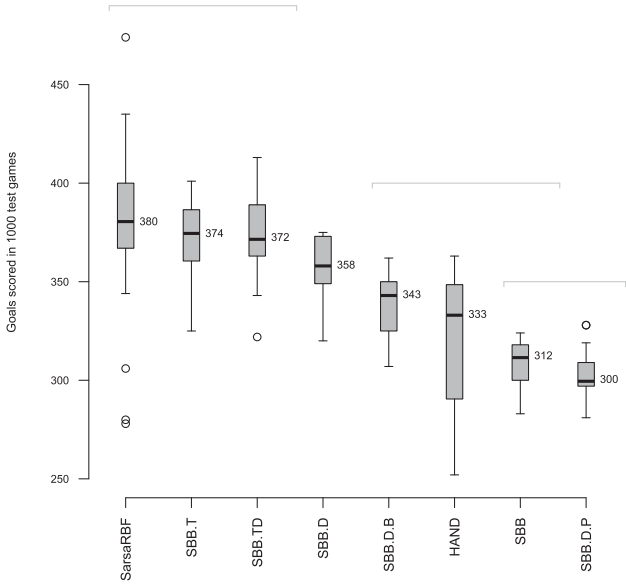

Fig. 4. Posttraining test results over 1000 games. SarsaRBF represents a current state-of-the-art reinforcement learner for the HFO task; SBB denotes hierarchical SBB cases as outlined in Table I. HAND represents a third party hand crafted (i.e., human designed) HFO policy native to the HFO environment. Box plots summarize the quartile distribution over 20 independent runs. Grey bracket groups denote distributions with no statistically significant difference, $P > 0.05$ from the Mann–Whitney rank test.

B. Ms. Pac-Man Test Performance

Several constraints are placed on the Ms. Pac-Man environment in order to minimize the computational cost of evaluations during training. As such, Ms. Pac-Man is limited to a single life, a single visit to each of four mazes, and a maximum of 8000 time steps to complete each maze. A game ends when any of these limits expire. However, a much less-constrained version of Ms. Pac-Man is used to test the quality of learned behaviors post training. This version closely matches that used in the Ms. Pac-Man versus ghosts competition (MPMvsG). Additional rules under the MPMvsG task are the following.

1) Ms. Pac-Man starts with three lives and gains an extra life after earning 10 000 points.
2) Completing the fourth maze returns Ms. Pac-Man to the first maze until each maze is visited four times, for a total of 16 levels.

[11] Under test, SarsaRBF selects actions strictly according to its value function (see [42]).

[12] Sensor and actuator noise, as well as random start conditions for takers, keepers, and ball in each game.

[13] Stochasticity in ghost behaviors and path-finding algorithms from which sensors are derived.

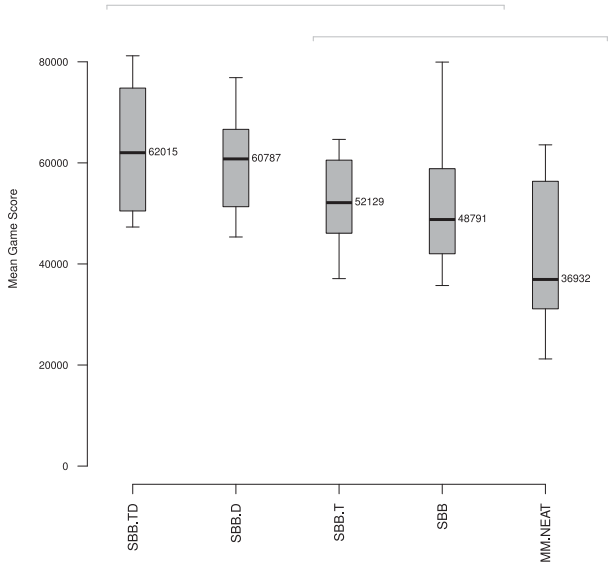

Fig. 5. Mean posttraining test scores over 100 games in the MPMvsG task. MM-NEAT represents a current state-of-the-art learner for the Ms. Pac-Man task; SBB denotes hierarchical SBB cases as outlined in Table I. Box plots summarize the quartile distribution over ten independent runs. Grey bracket groups distributions with no statistically significant difference, $P > 0.05$ from the Mann–Whitney rank test.

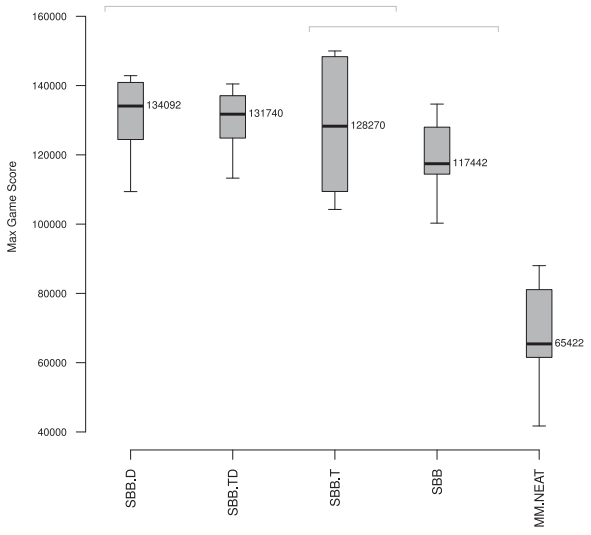

Fig. 6. Max posttraining test scores over 100 games in the MPMvsG task. MM-NEAT represents a current state-of-the-art learner for the Ms. Pac-Man task; SBB denotes hierarchical SBB cases as outlined in Table I. Box plots summarize the quartile distribution over ten independent runs. Grey bracket groups distributions with no statistically significant difference, $P > 0.05$ from the Mann–Whitney rank test.

3) the perlevel time limit is 3000 time steps, but rather than being killed when time runs out, she receives half the score from the remaining pills and advances to the next level.
4) Ms. Pac-Man has a time limit of 40 ms to return an action in each time step. If an action is not returned in time, the action form the previous time step is assumed.

In keeping with the Ms. Pac-Man literature, we report the mean and max champion test scores for the four SBB configurations (first four rows of Table I) under the MPMvsG task along with MM-NEAT, Figs. 5 and 6, respectively. There is no statistical difference between the mean score for any of the SBB configurations (see the Gray bracket extending over the first four columns of Fig. 5). However, both SBB cases with diversity maintenance significantly outperform MM-NEAT.[14] The fact that SBB manages to exceed the performance of MM-NEAT even without task transfer implies that prior task decomposition, or the separation of pill score and ghost score employed under SBB.T and in the multiobjective component of MM-NEAT, is unnecessary. However, diversity maintenance is critical for effective code reuse. That said, the combined task transfer and switched diversity maintenance scheme is still the most consistent performing model.

The maximum scores for all SBB cases are significantly better than MM-NEAT (see Fig. 6). Again, diversity maintenance appears to be more important than task transfer in this domain, as only the cases with diversity (SBB.D and SBB.TD) produce better results than the SBB baseline. We note that the SBB.TD and SBB.D results are also competitive with policy discovery through MCTS, i.e., a search process that requires considerably more task specific information than any formulation of SBB. Specifically, the corresponding average and max. posttraining performance for MCTS are 107 561 and 127 945, respectively [45]. Attempting to use GP to identify heuristics to guide MCTS resulted in average and best performance metrics of 32 641 and 62 630 [53] (i.e., significantly lower than any SBB configuration). In contrast, the best score yet achieved by an expert human player under the Ms. Pac-Man task varies between 15, 693 when given a two hour practice period [54] and 211, 480 when given an open ended number of practice games [55].

[14]The lower Gray bracket in Fig. 5 indicates no statistical difference between MM-NEAT and SBB treatments without diversity maintenance, but does not include SBB cases with diversity.

TABLE III
RANK-BASED SUMMARY OF SBB.TD, SBB.D, SBB.T, SBB POLICY TREE CONFIGURATIONS OVER EACH TASK

Domain	SBB.TD	SBB.D	SBB.T	SBB
HFO	1	3	2	4
Ms.Pac-Man	1	2	3	4
Avg. Rank	1	2.5	2.5	4

C. Significance of Policy Tree Variants

There are a total of four policy tree configurations considered across both HFO and Ms. Pac-Man task domains (SBB.TD, SBB.D, SBB.T, SBB). The Friedman rank based nonparametric test may be employed to summarize the relative significance of the configurations [56]. Table III summarizes the initial ranks. The Friedman statistic for $k = 4$, $N = 2$ is $\chi^2 = 5.4$. Renormalizing for the F-distribution provides $F_F = 9$. This is greater than the corresponding critical value of $F(3, 3) = 5.391$ at a p-value of 0.1; hence we are able to reject the null hypothesis. Finally, applying the Nemenyi post hoc test defines the performance of any two classifiers as being significantly different if the average ranks differ by a critical difference defined as $q_\alpha(k(k+1)/6N)^{1/2} = 2.96$ for $q_{0.1} = 2.291$. Thus, policy

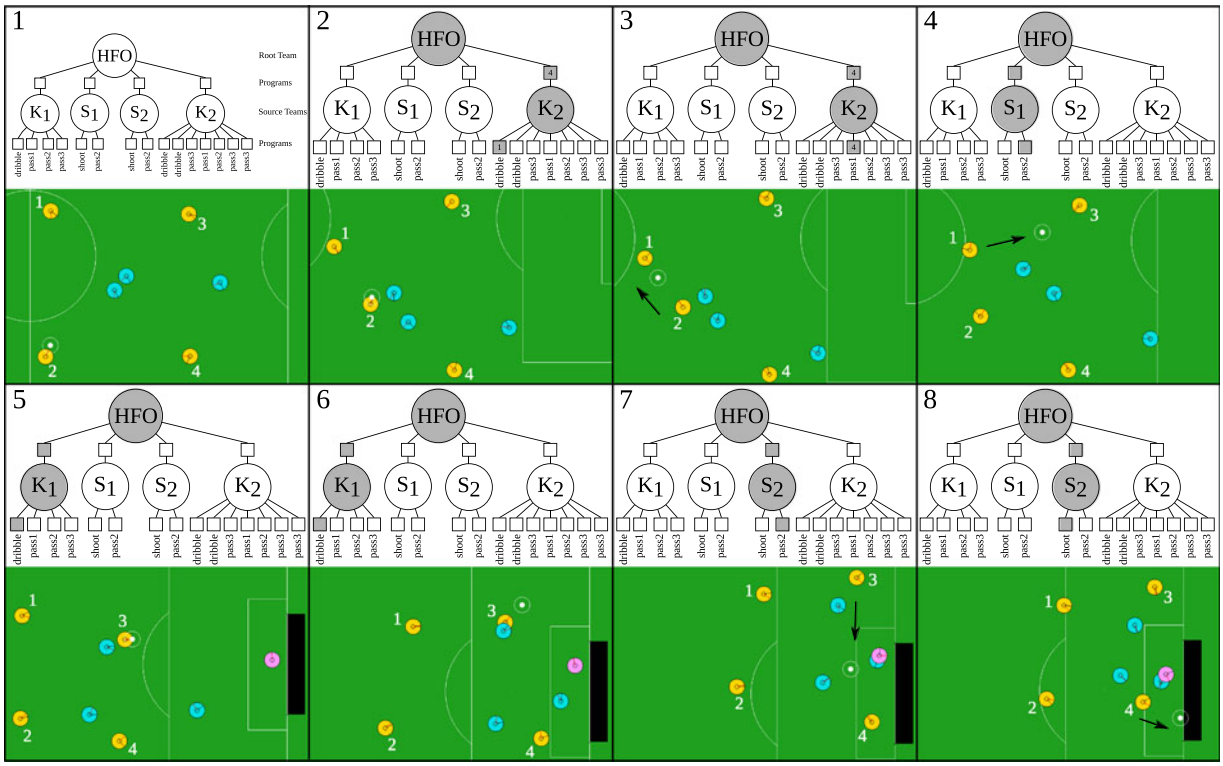

Fig. 7. Example HFO game played by a champion policy tree from the SBB.TD experiment. This policy was able to win the game after only seven "moves." The policy controls yellow players (offense team) only. Opponents (defense team) are blue, and goalie is pink.

trees configured without transfer or diversity maintenance is consistently inferior to including both diversity and transfer.

D. HFO Solution Analysis

In order to provide some insight into the behavior of policy trees in the HFO task, we can analyze a single champion policy tree from the SBB.TD experiment (see Fig. 7). This individual scored in 394 of 1000 test games. The policy tree has four active programs at the root of the policy tree, two of which index keepaway source policies (K_1, K_2) and two index scoring source policies (S_1, S_2). There are a total of 15 active programs across all source policies, covering the full scope of domain-specific atomic actions. We evolved homogeneous teams, thus any offense player in possession of the ball is controlled by the same policy tree.

To understand the behavior of this policy tree, we can trace the sequence of source policies and atomic actions deployed at each decision point during a specific HFO game, as shown in Fig. 7. This particular game proceeded as per the following play-by-play, in which the components of the policy tree are explicitly annotated.

1) The game is initialized with offense player 2 in possession of the ball.
2) Program 4 has the highest bid in the top level HFO switching policy and thus deploys source policy K_2, where program 1 outbids the others to deploy the dribble atomic action.
3) Program 4 is again selected by the HFO switching policy, deploying K_2, where program 4 now has the highest bid. Offense player 2 thus passes to its closest teammate (pass1).
4) Player 1 immediately passes to its second closest teammate, this time using scoring source policy S_1.
5) Player 3 dribbles toward the goal using K_1.
6) Player 3 loses control due to actuator noise, and is forced to chase the ball up to the top right corner of the field (with an opponent in close pursuit).
7) Player 3 passes down to player 4 (i.e., the second closest teammate).
8) Player 4 shoots and scores.

Clearly, this HFO policy interleaves different keepaway and scoring source policies while approaching the goal and achieving the winning shot. The high-level task decomposition does not follow the simplistic division of labor (ball handling and scoring) suggested by the source task configurations. Interestingly, at the lower level, K_1 and K_2 developed different contexts for the dribble atomic action, both of which proved useful. Similarly, S_1 and S_2 developed different contexts for the pass2 atomic action. Indeed, all programs in this policy produce a winning bid at least once over 25 games.

E. Ms. Pac-Man Solution Analysis

Fig. 8 depicts an SBB policy tree from the SBB.TD experimental case, which yielded the highest median score in the MPMvsG task (see Fig. 5). This policy includes four programs as part of the root team, two of which index source teams trained relative to the pill score objective (P_0 and P_1) while the other

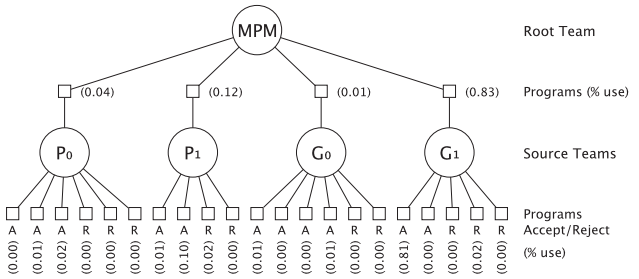

Fig. 8. Example SBB policy tree from the SBB.TD (Ms. Pac-Man) experimental case, which yielded the highest median scores in the MPMvsG task (See Fig. 5).

two index source teams trained on the ghost score objective (G_0 and G_1).

The process for an SBB policy to follow a root-to-leaf path and ultimately select an action for Ms. Pac-Man is the same as under HFO (see Section V-D). However, the domain-specific atomic actions are different and the team needs to evaluate each direction separately then make a decision [8]. Thus, programs in a team collectively decide to accept or reject each direction, where each decision also includes a bid value. The policy then moves in the accepted direction with the highest bid. If no direction is accepted, the policy moves in the rejected direction with the lowest bid. In other words, the policy either moves in the direction that is most accepted, or least rejected.

All source teams in the policy of Fig. 8 have roughly a 50/50 mix of programs with the accept and reject atomic action. The frequency of use for each program, relative to a single game of MPMvsG, is shown in parentheses. Clearly, this policy favors source policy G_1 (used in 83% of decisions) while the other source teams are switched in to achieve specific behaviors. Interestingly, P_0 is often used when eating power pills and P_1 is often used at the moment ghosts are eaten. Both these behaviors are intuitively associated with the ghost objective, rather than the pill objective under which these source teams were developed. The highest degree of switching occurs in two specific circumstances: 1) when Ms. Pac-Man is oscillating back and forth to remain in the same position[15] while luring threat ghosts; and 2) when Ms. Pac-Man is in a dangerous situation, for example, with threat ghosts approaching from multiple directions. Frequent source team switching indicates a high degree of interaction and cooperation among source teams, which collectively achieve luring and escape behaviors for Ms. Pac-Man in this case. An animation of this policy playing the MPMvsG game is available from the link at the end of Section I, in which Ms. Pac-Man survives to level 15 (of 16) and achieves a score of 100 480.

1) Comparison of SBB Solution Complexity With SarsaRBF and MM-NEAT: All SBB policies are teams of programs working cooperatively within a hierarchical policy tree. The number of programs per team and specific complement of programs forming a team are all identified during evolution. Programs

[15]Ms. Pac-Man is required to move in every time step, thus oscillating back and forth is the only way to remain in the same maze location over multiple time steps.

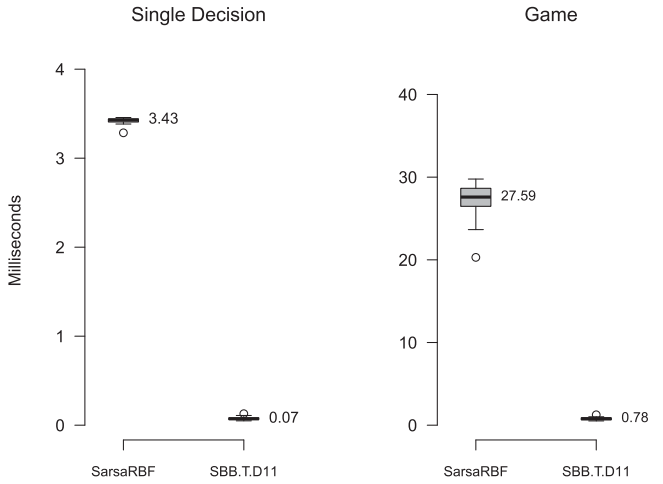

Fig. 9. CPU time for the champion/final policy to select an action in a single time step (single decision) and over an entire game of HFO (Game), averaged from 1000 test games. Box plots give the distribution over 20 independent runs for each algorithm. The nature of the RoboCup socket interface, as well as the fact that both SBB and SarsaRBF are programmed in c++, makes this direct comparison possible. A similar empirical comparison with MM-NEAT is not possible as MM-NEAT assumes a Java code base.

are simple linear register machines in which each instruction consists of a single arithmetic operator, function, or conditional statement (see Algorithm 1). Thus, we can describe the complexity of a solution by counting the total number of program instructions over all programs referenced by a policy tree.

The policy tree in Fig. 7 has a total of 19 programs (4 at the top level and 15 in the lower level), with a total of 880 instructions across all programs. However, the policy tree only follows one path from root to leaf node in order to suggest an action relative to the current state observation, which can be clearly seen in the policy animation (see the link in Section I). Indeed, while there was a median of 414 instructions in each SBB.TD policy for HFO, an average of only 116 instructions were executed in each time step during test games. In contrast, each SarsaRBF policy contains 16 320 weight/RBF pairs, all of which require execution at every time step.

MM-NEAT solutions under Ms. Pac-Man assume a neural network representation with an average of ≈ 47 nodes and ≈ 90 links in the champion networks. Each node computes the sum of products over some portion of the links, followed by the tanh activation function.[16] If we assume five operations for the activation function at each node, and a total of two operations per link for the sum of products, then the number of operations at each time step is links $\times\, 2 +$ nodes $\times\, 5 = 415$. In short, SBB policies are more efficient to implement posttraining, with the modular nature of the policy tree providing the basis for this simplicity. Fig. 9 places this into the context by reporting the average CPU time required for SarsaRBF and SBB.TD to make a decision at each time-step over 1000 test games in the HFO environment. Indeed, decision-making under SBB is several orders of magnitude faster.

[16]$\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$

VI. Conclusion

The SBB framework for evolving policy trees has been extended to facilitate the utilization of task transfer, which represents a methodology for taking policies identified under source tasks and then learning how to extend or redeploy them to provide policies under a more difficult target task. Two task domains are considered. In the first case, the HFO subtask of RoboCup soccer is used as the target task, where HFO represents a benchmark known to be more difficult than the keepaway task previously employed under task transfer. We show that SBB as originally formulated is not able to solve this task directly. In order to formulate task transfer for SBB, we make the following recommendations: 1) use multiple source tasks; and 2) evolve source task policies with diversity maintenance. Providing multiple source tasks enables the user to cover a "range" of potentially useful starting points to construct solutions to the target task policy. The policy trees that were identified using task transfer achieved HFO performance competitive with the current state-of-the-art SarsaRBF, and were significantly simpler, resulting in execution times several orders of magnitude faster. This can be particularly important in real-time games as more computational resources can be dedicated to other issues, such as strategic decision making which are typically given a lower computational priority.

The second empirical study, in which we consider the game of Ms. Pac-Man, reinforces this observation. Previous learning agents in Ms. Pac-Man have relied on prior knowledge to achieve state-of-the-art performance. However, the process for incrementally constructing policy trees adopted in this paper is able to discover reusable code both with and without support for source tasks identified with prior information. Indeed, results are competitive with schemes that assume much more *a priori* information, such as MCTS. Finally, defining solutions in the form of policy trees provides a very efficient scheme for posttraining deployment.

From the perspective of constructing NPC in general, the SBB framework enables a designer to incorporate prior biases through task transfer, while simultaneously supporting diversity maintenance. Task transfer may make the difference between discovering effective agent behaviors or not (with or without diversity). Diversity maintenance is never detrimental in our experience, and is a must when no prior information is available for task transfer. Moreover, the diversity mechanisms adopted here are task agnostic, thus independent of the particular application.

Acknowledgment

The authors would like to thank computational facilities provided by the NSERC CRD program and ACENET. ACENET is funded by the Canada Foundation for Innovation, the Atlantic Canada Opportunities Agency, and the provinces of Newfoundland and Labrador, Nova Scotia, and New Brunswick.

References

[1] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 4, pp. 317–335, Dec. 2015.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2012.

[3] R. R. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. Cambridge, MA, USA: MIT Press, 1998.

[4] M. E. Taylor and P. Stone, "An introduction to inter-task transfer for reinforcement learning," *AI Mag.*, vol. 32, no. 1, pp. 15–34, 2011.

[5] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 1, pp. 1633–1685, 2009.

[6] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evol. Comput.*, vol. 19, no. 2, pp. 189–223, 2011.

[7] S. Doncieux and J.-B. Mouret, "Behavioral diversity with multiple behavioral distances," in *Proc. IEEE Congr. Evol. Comput.*, 2013, pp. 1427–1434.

[8] J. Schrum and R. Miikkulainen, "Discovering multimodal behavior in Ms. Pac-Man through evolution of modular neural networks," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 1, pp. 67–81, Mar. 2016.

[9] S. Kelly and M. I. Heywood, "Knowledge transfer from Keepaway soccer to half-field offense through program symbiosis: Building simple programs for a complex task," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2015, pp. 1143–1150.

[10] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adapt. Behav.*, vol. 5, no. 3/4, pp. 317–342, 1997.

[11] P. Stone, *Layered Learning in Multiagent Systems*. Cambridge, MA, USA: MIT Press, 2000.

[12] J. Bongard, "Behavior chaining—Incremental behavior integration for evolutionary robotics," in *Proc. Artif. Life 11th Int. Conf. Synth. Simul. Living Syst.*, 2008, pp. 64–71.

[13] A. Agapitos, J. Togelius, and S. M. Lucas, "Evolving controllers for simulated car racing using object orientated genetic programming," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2007, pp. 1543–1550.

[14] L. Graham, R. Cattral, and F. Oppacher, "Beneficial preadaptation in the evolution of a 2D agent control system with genetic programming," in *Proc. Eur. Conf. Genetic Program.*, 2009, vol. 5481, pp. 303–314.

[15] W. H. Hsu, S. J. Harmon, E. Rodriguez, and C. Zhong, "Empirical comparison of incremental reuse strategies in genetic programming for Keepaway soccer," in *Proc. Late Breaking Papers Genetic Evol. Comput. Conf.*, 2004, Available only on CD.

[16] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone, "Evolving Keepaway soccer players through task decomposition," *Mach. Learn.*, vol. 59, no. 1, pp. 5–30, 2005.

[17] M. E. Taylor, S. Whiteson, and P. Stone, "Transfer via inter-task mappings in policy search reinforcement learning," in *Proc. ACM Int. Joint Conf. Auton. Agents Ultiagent Syst.*, 2007, Art. no. 37.

[18] M. E. Taylor, G. Kuhlmann, and P. Stone, "Autonomous transfer for reinforcement learning," in *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2008, pp. 283–290.

[19] P. Verbancsics and K. O. Stanley, "Evolving static representations for task transfer," *J. Mach. Learn. Res.*, vol. 11, pp. 1737–1769, 2010.

[20] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.

[21] C. Espinosa-Soto and A. Wagner, "Specialization can drive the evolution of modularity," *PLoS Comput. Biol.*, vol. 6, no. 3, 2010, paper e1000719.

[22] G. P. Wagner and L. Altenberg, "Perspective: Complex adaptations and the evolution of evolvability," *Evol.*, vol. 50, pp. 967–976, 1996.

[23] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Accelerated neural evolution through cooperatively coevolved synapses," *J. Mach. Learn. Res.*, vol. 9, pp. 937–965, 2008.

[24] K. Krawiec and B. Bhanu, "Visual learning by evolutionary and coevolutionary feature synthesis," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 635–650, Oct. 2007.

[25] M. Brameier and W. Banzhaf, *Linear Genetic Programming*. New York, NY, USA: Springer-Verlag, 2007.

[26] R. Thomason and T. Soule, "Novel ways of improving cooperation and performance in ensemble classifiers," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2007, pp. 1708–1715.

[27] P. Lichodzijewski and M. I. Heywood, "Symbiosis, complexification and simplicity under GP," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2010, pp. 853–860.

[28] S. X. Wu and W. Banzhaf, "Rethinking multilevel selection in genetic programming," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2011, pp. 1403–1410.

[29] M. I. Heywood and P. Lichodzijewski, "Symbiogenesis as a mechanism for building complex adaptive systems: A review," *Appl. Evol. Comput. I*, vol. 6024, 2010, pp. 51–60.
[30] D. E. Moriarty and R. Miikkulainen, "Forming neural networks through efficient and adaptive coevolution," *Evol. Comput.*, vol. 5, no. 4, pp. 373–399.
[31] G. Cuccu and F. Gomez, "When novelty is not enough," *Appl. Evol. Comput. I*, vol. 6624, pp. 234–243, 2011.
[32] E. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 47–62, Feb. 2004.
[33] J. H. Metzen, M. Edgington, Y. Kassahun, and F. Kirchner, "Performance evaluation of EANT in the robocup Keepaway benchmark," in *Proc. IEEE Int. Conf. Mach. Learn. Appl.*, 2007, pp. 342–347.
[34] J.-B. Mouret and S. Doncieux, "Encouraging behavioral diversity in evolutionary robotics: An empirical study," *Evol. Comput.*, vol. 20, no. 1, pp. 91–133, 2012.
[35] N. Kashtan, E. Noor, and U. Alon, "Varying environments can speed up evolution," *Proc. Nat. Acad. Sci.*, vol. 104, no. 34, pp. 13711–13716, 2007.
[36] M. Parter, N. Kashtan, and U. Alon, "Facilitated variation: How evolution learns from past environments to generalize to new environments," *PLoS Comput. Biol.*, vol. 4, no. 11, 2008, Paper e1000206.
[37] K. Imamura, T. Soule, R. B. Heckendorn, and J. A. Foster, "Behavioural diversity and probabilistically optimal GP ensemble," *Genetic Program. Evol. Mach.*, vol. 4, no. 3, pp. 235–254, 2003.
[38] P. Stone and R. S. Sutton, "Scaling reinforcement learning toward RoboCup soccer," in *Proc. Int. Conf. Mach. Learn.*, 2001, pp. 537–544.
[39] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for RoboCup-soccer Keepaway," *Adapt. Behav.*, vol. 13, no. 3, pp. 165–188, 2005.
[40] P. Stone, G. Kuhlmann, M. Taylor, and Y. Liu, "Keepaway soccer: From machine learning testbed to benchmark," in *RoboCup 2005: Robot Soccer World Cup IX*. New York, NY, USA: Springer-Verlag, 2006, pp. 93–105.
[41] S. Kalyanakrishnan, Y. Liu, and P. Stone, "Half-field offense in RoboCup soccer: A multiagent reinforcement learning case study," in *RoboCup-2006: Robot Soccer World Cup X*, vol. 4434 (Lecture Notes in Artificial Intelligence), G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, Eds. New York, NY, USA: Springer-Verlag, 2007, pp. 72–85.
[42] S. Whiteson, M. E. Taylor, and P. Stone, "Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning," *J. Auton. Agents Multi-Agent Syst.*, vol. 21, no. 1, pp. 1–27, 2009.
[43] S. Kalyanakrishnan and P. Stone, "An empirical analysis of value function-based and policy search reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2009, pp. 749–756.
[44] S. Kelly and M. I. Heywood, "On diversity, teaming, and hierarchical policies: Observations from the Keepaway soccer task," in *Proc. Eur. Conf. Genetic Program.*, 2014, vol. 8599, pp. 75–86.
[45] T. Pepels and M. H. M. Winands, "Enhancements for Monte-Carlo tree search in Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 265–272.
[46] A. M. Alhejali and S. M. Lucas, "Using a training camp with genetic programming to evolve Ms Pac-Man agents," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 118–125.
[47] M. F. Brandstetter and S. Ahmadi, "Reactive control of Ms. Pac-Man using information retrieval based on genetic programming," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 250–256.
[48] S. Kelly and M. I. Heywood, "Genotypic versus behavioural diversity for teams of programs under the 4-v-3 Keepaway soccer task," in *Proc. AAAI Conf. Artif. Intell.*, 2014, pp. 3110–3111.
[49] P. Lichodzijewski and M. I. Heywood, "The rubik cube and GP temporal sequence learning: An initial study," in *Genetic Programming Theory and Practice VIII*. New York, NY, USA: Springer-Verlag, 2011, ch. 3, pp. 35–54.
[50] J. A. Doucette, P. Lichodzijewski, and M. I. Heywood, "Hierarchical task decomposition through symbiosis in reinforcement learning," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2012, pp. 97–104.
[51] S. Kelly, P. Lichodzijewski, and M. I. Heywood, "On run time libraries and hierarchical symbiosis," in *Proc. IEEE Congr. Evol. Comput.*, 2012, pp. 3245–3252.
[52] F. Gomez, "Sustaining diversity using behavioral information distance," in *Proc. ACM Genetic Evol. Comput. Conf.*, 2009, pp. 113–120.
[53] A. M. Alhejali and S. M. Lucas, "Using genetic programming to evolve heuristics for a Monte Carlo tree search Ms Pac-Man agent," in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 1–8.
[54] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[55] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general Atari game playing," *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 4, pp. 355–366, Dec. 2014.
[56] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, 2006.

Stephen Kelly received the Bachelor of Fine Arts degree from Nova Scotia College of Art and Design, Halifax, NS, Canada, in 2001 and the Master of Computer Science degree, for his work on evolutionary reinforcement learning from Dalhousie University, Halifax, NS, Canada, in 2012. He is currently working toward the Ph.D. degree in the Faculty of Computer Science, Dalhousie University.

His current research interests include on machine learning in video games, with particular emphasis on building agent behaviors with modular genetic programming.

Mr. Kelly received the two best paper awards in 2017 for an approach to evolving agents in the Atari Arcade Learning Environment for general game AI (European Conference on Genetic Programming) and multitask learning (ACM Genetic and Evolutionary Computation Conference).

Malcolm I. Heywood (M'95–SM'06) received the Ph.D. degree, his work on movement invariant pattern recognition using neural networks from the University of Essex, Colchester, U.K., in 1994.

He is currently a Professor of Computer Science at Dalhousie University, Halifax, NS, Canada. His current research investigates the application of co-evolutionary methods to reinforcement learning tasks as encountered in computer games (Rubik's Cube, Arcade Learning Environment, FPS), and streaming data applications (Intrusion Detection and Financial Services).

Dr. Heywood is a member of the Editorial Board for Genetic Programming and Evolvable Machines (Springer). He was a Track Co-Chair for the GECCO GP track in 2014 and the Co-Chair for European Conference on Genetic Programming in 2015 and 2016.