Knowledge Transfer from Keepaway Soccer to Half-field Offense through Program Symbiosis: Building Simple Programs for a Complex Task

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

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

Article originally appears at GECCO'15 under ACM copyright 2015
http://dl.acm.org/citation.cfm?doid=2739480.2754798

Abstract

Half-field Offense (HFO) is a sub-task of Robocup 2D Simulated Soccer. HFO is a challenging, multi-agent machine learning problem in which a team of offense players attempt to manoeuvre the ball past a defending team and around the goalie in order to score. The agent's sensors and actuators are noisy, making the problem highly stochastic and partially observable. These same real-world characteristics have made Keepaway soccer, which represents one sub-task of HFO, a popular testbed in the reinforcement learning and task-transfer literature in particular. We demonstrate how policies initially evolved for Keepaway can be reused within a symbiotic framework for coevolving policies in genetic programming (GP), with no additional training or transfer function, in order to improve learning in the HFO task. Moreover, the highly modular policies discovered by GP are shown to be significantly less complex than solutions based on traditional value-function optimization while achieving the same level of play in HFO.

1 Introduction

Task domains in which credit assignment is delayed imply that the learning agent is required to perform multiple interactions with the environment before feedback is received, or reinforcement learning (RL) [19]. Games potentially represent a generic example of a task with delayed feedback. The goal of the learning agent is to discover a 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 [21, 20]. In this work we are interested in a framework for task transfer under GP in which policies are constructed hierarchically, or a policy tree. First, lower level source / leaf policies are evolved for related but different source task(s). A second phase of evolution is conducted to learn how to reuse the source / leaf policies under the target task by evolving a higher level target policy.

We empirically test this 2-phase approach for hierarchical policy reuse under the challenging multi-agent 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 shooting on the goal. Phase 2 evolves a policy for the *target task* of HFO by learning how to reuse a subset of the previously evolved behaviours. 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 criteria for meaningful task transfer [21, 20]. 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.

The architecture used to evolve policies is both highly modular and hierarchical. Specifically, a symbiotic bid-based (SBB) framework is used in which genetic algorithm (GA) and GP populations simultaneously coevolve to identify policies, i.e. programs are coevolved to collectively decompose the task and / or reuse previously evolved policies. Such a framework was applied to tasks that could be solved without task transfer [13, 4, 11]. Indeed,

the SBB framework has been shown to be capable of strong performance under Keepaway [10], but appears to be comparatively poor at HFO. This work represents the first instance in which task transfer is used in combination with SBB to address a task in which SBB would otherwise fail to reach state-of-the-art levels of performance.

Section 2 details the nature of the source and target task domains. The formulation assumed for SBB is described in Section 3. The approach to combining SBB with task transfer is established in Section 4 and the experimental study detailed in Section 5. Section 6 concludes the paper.

2 Task characterization

Robocup 2D Simulated Soccer is a widely-used environment for evaluating multi-agent machine learning systems. However, the full game of soccer is often too complex for a tabula rasa approach, i.e. learning agents try to master the entire task with a single monolithic policy and no prior task decomposition [17]. Thus, researchers have focused on sub-tasks of the full game [18, 16, 8, 14, 22, 7]. Keepaway and HFO soccer represent interesting reinforcement learning tasks in that they are: 1) multi-agent; 2) stochastic and non-Markovian; 3) utilize real-valued state variables; and 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 reinforcement learning algorithms [8, 21].

2.1 Keepaway Task

Keepaway represents one sub-task of Robocup Soccer in which a team of K *keepers* tries to maintain possession of the ball while an opposing team of $K-1$ *takers* attempt to gain possession [16]. The goal is for keepers to learn a policy that maximizes the length of play against the takers, which follow a pre-specified behaviour. The game ends if the ball is kicked out of bounds or captured by the takers. When a game ends, each keeper receives the game duration in milliseconds as a reward signal.

In this work we are specifically interested in a 4v3, or $K = 4$, version of the task. Our version of Keepaway is played on a 60×60 meter field, instead of the more common 25×25 meter bounding box, in order to increase the compatibility with the HFO task. The game is initialized with keepers in four corners of a 15×15 meter square at centre field, the ball placed near one keeper, and the takers in the centre. Keepers learn to deploy a set of domain-specific macro-actions $\{hold, getOpen, pass(k)\}$, at discrete timesteps of 100 milliseconds. Each macro-action may last more than one timestep, and control is returned to the player when the macro-action 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 automatically select the *getOpen* macro-action.

At each timestep, 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 (Table 1). 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.

2.2 Half-field Offense Task

Half-field offense is another sub-task of Robocup Soccer with significantly more complexity than Keepaway [8]. 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 ball is kicked out of bounds, captured by an outfield defense player or the goalie, or a goal is scored. The defense team in HFO follow a pre-specified behaviour defined by the original task [8]. Each offense player receives one reward signal of either $\{0.8, 0.9, 2\}$ for end-game conditions {*captured by outfield defense* or *kicked out of bounds*, *caught by goalie*, *goal scored*}.

We are specifically interested in 4v4 HFO, in which the offense team has 4 players while the defense team is made up of 3 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 meter square as in Keepaway, but in HFO the centre of the initialization box is stochastically placed somewhere on the field roughly 55 meters from the goal. Naturally, the goalie is initialized in the goal region.

Offense players in HFO learn to deploy a slightly different set of macro-actions than in the Keepaway task. Their options at each timestep 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 macro-action may last more than one timestep and control is returned to the player when the macro-action terminates. In addition to the modified action space, the HFO task has 6 *additional* sensor inputs relative to the

Table 1: Sensor inputs for the 4v3 Half-field Keepaway task and 4v4 Half-field Offense task. P_p is the player in possession of the ball. The rest of the players on P_p's team are numbered relative to their distance from P_p with P_1 being the closest. 'Opponent' refers to any taker in the Keepaway task and any 'defense' player in the HFO task. All sensor inputs are real-valued. Angle values range from 0° to 360° and distance values range from 0 to approximately 85 meters.

			Sensor Input	**Description**
Keepaway	Half-field Offense	a	$dist(P_p, P_1)$	Distance from P_p to closest teammate.
		b	$dist(P_p, P_2)$	Distance from P_p to second-closest teammate.
		c	$dist(P_p, P_3)$	Distance from P_p to third-closest teammate
		d	$dist(P_1, O_{min})$	Distance from P_1 to closest opponent.
		e	$dist(P_2, O_{min})$	Distance from P_2 to closest opponent.
		f	$dist(P_3, O_{min})$	Distance from P_3 to closest opponent.
		g	$min_{angle}(P_1, O_p, O_{min})$	Min. angle, with vertex at P_p, between P_1 and any opponent.
		h	$min_{angle}(P_2, O_p, O_{min})$	Min. angle, with vertex at P_p, between P_2 and any opponent.
		i	$min_{angle}(P_3, O_p, O_{min})$	Min. angle, with vertex at P_p, between P_3 and any opponent.
		j	$min_{dist}(P_p, O_{Dcone})$	Min. distance from P_p to any player in the dribble cone. The dribble cone is a cone with half angle 60° with its vertex at P_p and axis passing through the centre of the goal. D_{cone} is the set of defenders in the dribble cone.
		k	$dist(P_p, O_{min})$	Distance from P_p to closest opponent.
		l	$dist(P_p, GL)$	Distance from P_p to goal line.
		m	$dist(P_1, GL)$	Distance from P_1 to goal line.
		n	$dist(P_2, GL)$	Distance from P_2 to goal line.
		o	$dist(P_3, GL)$	Distance from P_3 to goal line.
		p	$max_{angle}(P_p, Goal)$	Max. angle with vertex at P_p, formed by rays connecting P_p and goalposts or opponents in the goal cone, which is the triangle formed by P_p and the two goalposts.
		q	$dist(P_p, Goalie)$	Distance from P_p to the goalie.

Keepaway task (Table 1). The additional sensors summarize distance to the goal for each offense player and the angle of the maximum scoring window for the player with the ball, i.e. how open the net is (Figure 1). As in Keepaway, all sensor inputs and actuators are unreliable. Readers are refereed to the original HFO paper for a complete definition of the task [8].

3 Symbiotic Bid-Based GP

Symbiotic Bid-Based GP (SBB) is a hierarchical framework for symbiotically coevolving policies, or decomposing the task, over two distinct phases of evolution (Figure 2).[13, 4, 11, 10]. The first phase produces a library of diverse, specialist policies of limited capability. The second phase builds more general and robust policies by reusing a subset of policies from the first phase, essentially building generalist strategies from multiple specialists. Thus, diversity maintenance is critical during the first phase of evolution to ensure the identification of a wide range of specialist behaviours. See [10, 9] for an example of this within the context of the Keepaway task. Policies are represented by two distinct populations within the SBB algorithm: a host GA population defines group membership; and a symbiont population defines a pool of GP individuals that can potentially appear in a host group.

This section gives an overview of the components of SBB that are critical to understanding this work. Relative to previous instances of the SBB framework for reinforcement learning tasks ([13, 4, 11, 10]), this work adopts a more general approach to diversity maintenance as well as having to explicitly address the issue of knowledge transfer between different tasks.

3.1 Programs

The basic building blocks in SBB are linear genetic programs [1], or register machines, consisting of a sequence of instructions that operate on sensor inputs or internal registers. We refer to these programs as symbionts, because their role in the larger algorithm is inspired by biological symbiosis [12]. Each symbiont is associated with *one*

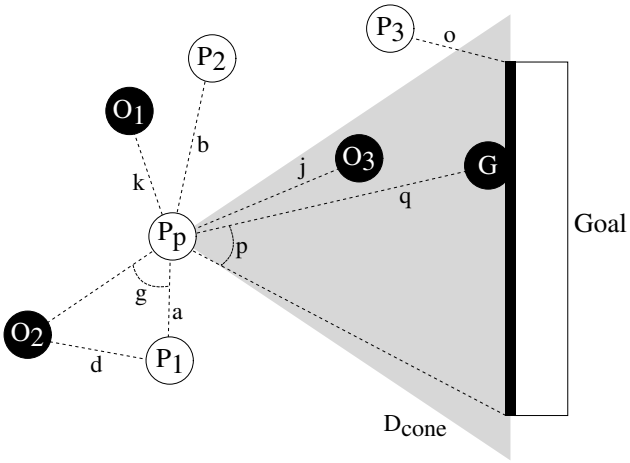

Figure 1: Sample sensor inputs for the 4v4 Half-field Offense task. See Table 1 for full description.

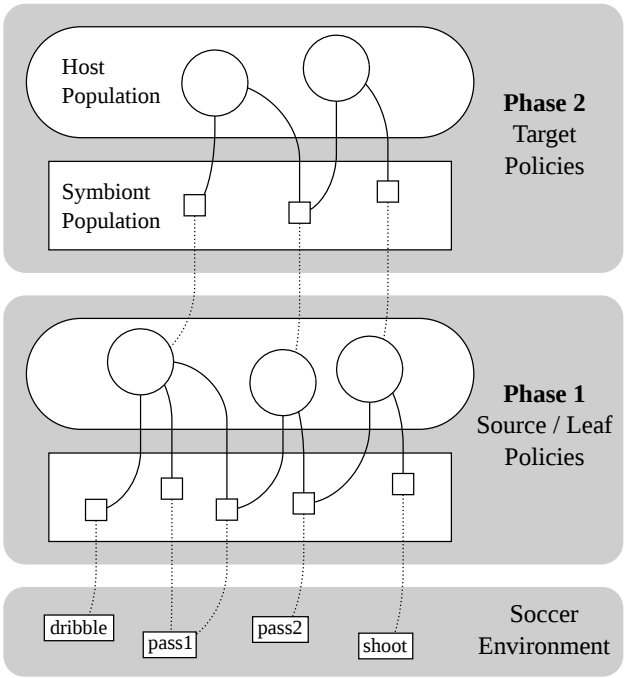

Figure 2: Constructing policy trees through SBB. Two independent phases are assumed. Each phase defines policies in terms of groups of programs (host and symbiont). Phase 1 identifies a diverse set of relatively simple 'source / leaf' policies. Phase 2 learns how to reuse phase 1 (source / leaf) policies for solving a different target task.

domain-specific action, or **atomic action**, from the task domain.[1] For example $\{hold, getOpen, pass(k)\}$ in the Keepaway task (Section 2.1). Given all sensor inputs in the current timestep, the program outputs a single real valued 'bid', which expresses how confident this symbiont is about using its particular action, given the current state of the environment. 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. Variation operators modify symbionts by adding and/or removing instructions from the program, flipping a single bit in an instruction, or changing the symbiont's atomic action [12].

3.2 Policies

A single symbiont cannot represent a policy for any task because symbionts are only associated with a *single* action (Section 3.1). Therefore, a policy is constructed from a cooperative group of symbionts. These groups are defined by members of the host population, h_i, Figure 2. For a policy to make a decision at any point in time, each symbiont identified by the host ($sym \in h_i$) must observe the current sensor inputs and execute its program, resulting in a 'bid'. The policy will then deploy the action of the symbiont with the highest bid. Algorithm 1 summarizes this process under a generic Robocup soccer task. Note that policy h_i is deployed homogeneously across all learning agents.

Algorithm 1 Generic process for SBB to determine an action for the player in possession of the ball in Robocup Soccer, given a host policy h_i. *task* denotes the soccer simulator. The 'fail' condition may correspond, for example, to an opponent intercepting the ball or the ball exceeding the field boundary. Step 1f represents a reference to a previously evolved (source) policy, as encountered in phase 2.

1. DO
   (a) Present the state variables describing the current state for the player p in possession of the ball, or $\vec{s}_p(t_s)$ to host h_i;
   (b) $\forall sym \in h_i$, execute the symbiont programs to establish the corresponding symbiont bid, or $sym(bid(\vec{s}_p(t_s)))$;
   (c) Identify the 'winning' symbiont as that with the maximum bid from host h_i, $sym^* = \arg_{sym \in h_i} \max[sym(bid(\vec{s}_p(t_s)))]$;
   (d) IF $(sym^*(action) == atomic\ action)$
   (e) THEN Present the action from the winning symbiont to the soccer simulator, or $\vec{s}(t_s = t_s + 1) \leftarrow task \leftarrow sym^*(action)$
   (f) ELSE $h_i = sym^*(action)$ *Call* [Algorithm 1]
2. WHILE (!fail)

There is no reason why a host (policy) cannot contain two symbionts with the same action. This actually occurs frequently because a symbiont essentially learns a context, relative to sensor inputs, for deploying its action. This context is by definition specific to a particular region of the policy space, thus two symbionts may develop different contexts for the same atomic action. The bidding framework is the mechanism for facilitating task decomposition between symbionts in a host, and the nature of this decomposition defines the policy's behaviour in the environment.

The same symbiont may function effectively in multiple hosts, each with a different compliment of group members. Hosts are initialized with between 2 and 5 symbionts. Variation operators may remove symbionts from the host, add an existing symbiont from the symbiont population to the host, or create a new symbiont by cloning an existing host-member and applying the symbiont variation operators to ensure that it is unique [12]. Thus, the number of symbionts in a host varies over time subject to a minimum of 2 and maximum of ω. In each generation, the selection operator deletes the worst performing $Mgap$ hosts. Symbionts no longer associated with any host are assumed to be ineffective and are also deleted.

There is no concept of symbiont fitness, only the knowledge of whether or not it is useful in any host. In effect we are assuming multi-level selection in which a policy is only evaluated as a whole as opposed to the sum of its

[1] Section 3.4 generalizes this to previously evolved SBB individuals.

parts [15]. The host population size, *Msize*, remains fixed throughout evolution while the symbiont population size varies. In order to encourage the development of a variety of policies by the host population, special attention is given to diversity maintenance (Section 3.3). Previous research has established the significance of diversity maintenance under multiple RL tasks [14, 6, 4, 3, 10, 9].

3.3 Diversity Mechanisms

In Robocup Soccer, the fitness of a policy is the mean reward across all games played. Fitness is only assigned to the host under evaluation. However, in order to promote population diversity, each host's *novelty* must also factor into the selection process, where novelty refers to the degree of similarity between a host and all other members of the *same* population. Several methods to balance fitness and novelty exist, including fitness sharing, crowding, or multi-objective optimization. In this work we adopt a simple linear combination of fitness and novelty [2], thus each host's performance is defined prior to selection:

$$\mathcal{F}(h_i) = (1-p)\cdot\overline{Fit(h_i)} + p\cdot\overline{Nov(h_i)} \tag{1}$$

where $\overline{Fit}$ and $\overline{Nov}$ are the normalized fitness and novelty of host i, and p is a parameter to control the relative weight of novelty. The novelty component requires a method for measuring the 'distance' between each pair of hosts, discussed below. Moreover, rather than assuming a single measure of diversity, we switch between two different distance measures every $\tau = 10$ generations. Switching between dissimilar distance metrics avoids introducing yet another scalar weighting parameter, and has been shown to be as effective as explicitly multi-objective formulations [3].

3.3.1 Symbiont-utility Distance Metric

A symbiont-utility distance metric is used to characterize which symbionts are active within a host.[2] A symbiont is considered active if it has scored a winning bid, and thus suggested an action for the task domain, at least once during the life of the host. The distance between hosts is summarized as the ratio of active symbionts common to both hosts. Thus, the distance between hosts i and k is

$$dist(h_i, h_k) = 1 - \frac{Sym_{active}(h_i) \cap Sym_{active}(h_k)}{Sym_{active}(h_i) \cup Sym_{active}(h_k)} \tag{2}$$

where $Sym_{active}(h_x)$ represents the set of active symbionts in host x. $p = 0.2$ when using this diversity metric in Equation 1. Note that this diversity metric is task independent.

3.3.2 Behavioural Distance Metric

A behavioural distance metric characterizes a host by how it interacts with the environment. At each decision point in a game, the state (11 real-valued state variables in the case of 4v3 Keepaway) and subsequent action taken by the host 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, $\vec{p} = [\{a(t), s(t)\}, t \in T]$, where $a(t)$ is the atomic action taken, $s(t)$ is the discretized state observation, and T represents every decision point in the associated game. Note that this method of characterizing behaviour is also task-independent. Hosts maintain a historical record of the profile $\vec{p}$ for every game played. We define $\vec{P}$ to be the concatenation of all profile vectors $\vec{p}$ in a host's historical record. Behavioural distance between a pair of hosts can now be summarized as the Normalized Compression Distance [6] between their corresponding concatenated profile vectors: $dist(h_i, h_k) = ncd(\vec{P}_i, \vec{P}_k)$. $p = 0.4$ when using behavioural diversity in Equation 1.

3.4 Policy Trees

After a number of generations, the host population may not contain any suitable solutions [13, 4, 11, 10, 9]. However, diversity maintenance encourages a variety of *different* behaviours to be present (Section 3.3). These policies can be reused as building blocks within larger policy trees. Thus, a new phase of evolution begins with an entirely new SBB initialization (Figure 2 'Phase 2'). The only difference being that a symbiont in phase 2 references a policy previously evolved during phase 1 instead of an atomic action. Thus, symbionts at phase 2 are learning the appropriate context(s) in which to deploy previously evolved *source* policies which are selected and cached

[2]The metric adopted is motivated by the approach taken in [14] for promoting diversity under a neuro-evolutionary setting.

from the independent evolutionary cycle of 'phase 1'. The selection process is detailed in Section 4.1.2. At present only one level of the policy tree undergoes development at a time, thus policies identified during phase 1 are 'frozen' during phase 2.

When evaluating a policy in phase 2, Algorithm 1 is again called. However, this time the winning action at Step 1c is a reference to a previously evolved source / leaf policy from phase 1. Algorithm 1 is then recursively called at Step 1f in order to resolve which symbiont from the source / leaf policy is selected and therefore what atomic action is deployed (Step 1e).

Conceptually, policy trees learn to combine multiple, diverse *specialist* policies into a single *generalist* policy, a process which has been demonstrated in several previous works [13, 4, 11, 10]. The major contribution of this study is to show that policy trees can be used to transfer knowledge from a source task to a related, but more complex target task. Section 4 details how this is achieved with 3 sub-tasks of Robocup Soccer.

4 Knowledge Transfer in SBB

Inter-task knowledge transfer refers to the ability of a learning algorithm to leverage experience gained by interacting with a *source* task environment in order to improve some aspect of learning in a *target* task [21, 20]. Keepaway soccer is clearly a sub-task of HFO. For example, from the stochastic start position in HFO (Section 2.2), offensive players essentially have to play Keepaway and dribble towards the goal before trying to score. It follows that knowledge gained while learning to play Keepaway should be useful in the context of HFO.

The problem of transfer from 4v3 Keepaway to HFO does not require an independent mapping interface between feature or action spaces. This is possible because most Keepaway inputs and actions have direct equivalents within the HFO task (Table 1). The one exception being the hold action in Keepaway, which is interpreted as a dribble-towards-goal action under HFO. It is therefore reasonable to deploy a policy trained in the 4v3 Keepaway task directly under HFO without modification. However, it will not represent a successful HFO policy on its own because in Keepaway there is no concept of goal scoring, or even a macro-action to support scoring goals. Furthermore, the ball-handling skills under HFO are different, requiring the players to maintain possession of the ball *and* move toward the goal region. Thus, additional capabilities are necessary in order to attempt HFO. Sections 4.1 and 4.2 describe how SBB policy trees can discover these policies over 2 phases of evolution.

4.1 Phase 1: Multiple Source Tasks

In addition to ball handling, which we aim to 'transfer' from Keepaway as a source task, the other basic component to HFO is scoring goals. In order to design an appropriate source task scenario to focus on goal scoring, we modify the HFO environment to restrict the initial position of players to within 15 meters of the goal. From this position, offense players no longer need to manoeuvre the ball down field and can therefore concentrate on scoring goals. During phase 1, multiple policies are evolved independently in each of the two *different* source tasks: Keepaway and Scoring. From these two groups, a subset of policies are selected to represent the *source task* policies, which will be combined in a policy tree for learning the target task, HFO. The sequence of events promoting this development takes the form of the following 3 steps.

4.1.1 Phase 1 Policy Development

First, we conduct $N = 10$ independent, single-phase runs of SBB in the Keepaway task. Each run is parameterized as per Table 2 with diversity maintenance (Section 3.3). After removing $Mgap$ policies at the final generation[3], each run produces 90 unique policies for a combined total over N runs of 900 Keepaway policies. The primary use for Keepaway policies will be to maintain possession of the ball. Successful transfer to the HFO task will generalize this to moving to the goal region. Likewise, $N = 10$ independent, single-phase runs of SBB are performed in the Scoring task, also returning a total of 900 candidate policies.

4.1.2 Pareto Ranking of Phase 1 Policies

Having established a pool of Keepaway and goal scoring behaviours (source tasks) we now adopt the following process to filter the number of policies before phase 2 of SBB (Figure 2). Each of the 900 *Keepaway* behaviours are deployed for 10 trial games in the full HFO environment. Information from two sensors are then used to characterize the utility of candidate Keepaway policies based on the following 2 objectives. **Distance to Goal**

[3]SBB is a breeder model where the worst $Mgap$ individuals are replaced at each generation [13, 4, 11, 10].

(Table 1 – mean of sensor input l): better policies should minimize the mean of this input over 10 games. **Distance to closest player in dribble cone** (Table 1 – mean of sensor input j): provides a sense of how congested the player in possession of the ball is, and how much space it has to dribble towards the goal without loosing possession. Hence, better policies should maximize this property.

Given these two objectives, we find the set of Pareto non-dominated policies[4] from the pool of 900 candidates, which in this case returns a subset of 11 Keepaway policies.

In the case of *Scoring* behaviours, candidate Scoring policies are also deployed for 10 trial games in the HFO environment (modified for scoring-only as per Section 4.1) and ranked based on 3 objectives: 1) maximize sensor input j to reduce congestion (see above); 2) maximize sensor input p, which indicates the degree to which the goal is 'open'; and 3) maximize the main HFO objective, or number of goals scored. Given these 3 objectives, taking the set of Pareto non-dominated policies from the pool of 900 candidates returns a subset of 7 goal scoring policies.

4.1.3 Recombination of Source Task Policies

A pool of policies developed w.r.t. the source tasks have now been identified. These represent the 'actions' available during a second phase of evolution (Figure 2). SBB phase 2 constructs a 'root node' that learns to switch between source / leaf policies to solve the HFO task.

4.2 Phase 2: Target Task

In SBB phase 2, policy trees will be evolved for the HFO task, where each policy tree may combine any of the 18 source policies identified in phase 1 (Section 4.1). However, such a simplistic decomposition of HFO into two source tasks is just a starting point. Each source policy will have a unique specialization care of diversity maintenance (see Section 3.3) and the Pareto selection process of Section 4.1. The policy identified by SBB during phase 2 essentially evolves a root node in a policy tree. Such a root node identifies under what conditions to switch between two or more source policies (as previously evolved in phase 1). Thus, the potential exists to extend and recombine existing source policies into a new behaviour, or conversely associate a previous policy with a new action as in the case of learning to dribble the ball.

Phase 2 is conducted without diversity maintenance as the objective is strictly exploitative, i.e. $p = 0$ in Eqn. 1. Each run produces $Msize/2$ policy tree root nodes. A post-training validation phase is then conducted, in which each policy tree is evaluated over an additional 25 games. The policy with the highest average reward (Section 2.2) on validation is selected as champion of the run. Finally, due to the stochasticity of the task[5], 1000 post-training test games are required to provide an accurate measure of performance for the single champion policy from each run.

5 Results

Three algorithms will be benchmarked: 1) **SBB-T**: Hierarchical SBB *with* transfer as per Section 4; 2) **SBB**: Hierarchical SBB *without* transfer or prior task decomposition. In this case the full HFO task is assumed during phase 1 and 2 of SBB solutions (Figure 2). The 3-objective Pareto ranking process, identical to that used for the Scoring source task, is used to filter phase 1 policies down to 14 source / leaf policies; and 3) **SarsaRBF**: The Sarsa Temporal Difference Method with Radial Basis Functions. SarsaRBF represents the current state-of-the-art in the HFO task [8].

As per previous research, the Robocup Soccer Server[6] simulates the 2D soccer environment. Code from the designers of Keepaway[7] and HFO[8] provides full implementation of the tasks.

Table 2 summarizes the parameterization assumed for SBB-T. In the case of the number of generations (t_{max}), two limits appear at phase 1 on account of the goal scoring task being easier than Keepaway. Parameterizing SBB assumes the same parameterization as SBB-T *except* in the case of the phase 1 population size and generation limit. SBB-T evolves Keepaway and Scoring behaviours using independent populations of size 180 for 250 and 60 generations respectively. Thus, SBB evolves a single population of size $Msize = 2 \times 180$ for a total of $t_{max} = 250 + 60$ generations.

[4] A solution a_i is said to dominate another solution a_j, if both of the following conditions are true: $a_i \succ a_j \leftrightarrow \forall k[f_k(a_i) \geq f_k(a_j)] \wedge \exists m[f_m(a_i) > f_m(a_j)]$.

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

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

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

[8] `http://www.cs.utexas.edu/~AustinVilla/sim/halffieldoffense`

Table 2: Parameterization of Host and Symbiont populations. p_{mx} denotes a mutation operator in which: $x \in \{d, a\}$ are the prob. of deleting or adding a symbiont respectively; $x \in \{m, n\}$ are the prob. of creating a new symbiont or changing the symbiont action respectively.

Host (GA) population			
Parameter	Value	Parameter	Value
t_{max} phase 1	250, 60	ω	30
t_{max} phase 2	125	t_{eval}	10
$Msize$	180	$Mgap$	50%
p_{md}	0.7	p_{ma}	0.7
p_{mm}	0.2	p_{mn}	0.1
Symbiont (GP) population			
$numRegisters$	8	$maxProgSize$	96
p_{delete}, p_{add}	0.5	p_{mutate}, p_{swap}	1.0

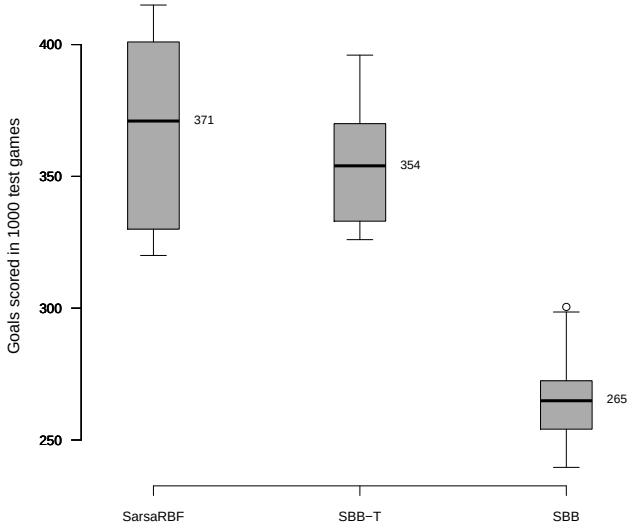

Figure 3: Post-training test results for each algorithm over 1000 games. Box plots summarize the quartile distribution over 10 independent runs. SarsaRBF represents a current state-of-the-art reinforcement learner for the HFO task; SBB-T is hierarchical SBB *with* task transfer; SBB is hierarchical SBB *without* task transfer.

The fitness of each individual is determined following play against the predefined HFO defenders in at most t_{eval} stochastically initialized games. Such a constraint is enforced on account of the computational overhead in estimating fitness in this task domain. The total generations performed at *each* phase represent a computational cost of $\approx 24hrs$. During phase 2 a total of 20 runs are performed for each SBB experiment, of which results are reported on the top 10 as defined by validation performance post training (but before test). In short, the computational cost of the task precludes fitness evaluation over larger populations or more games per fitness evaluation. Other researchers have adopted similar schemes for identifying champion solutions for evaluating test performance [5]. Moreover, on the easier Keepaway task, neuro-evolutionary solutions have been limited to performing 5 runs alone, again on account of the computational cost of performing runs [14, 22].

5.1 Test Performance

Figure 3 reports test performance for the three approaches to learning the HFO task. It is apparent that both SarsaRBF and SBB-T reach similar levels of performance. Indeed, there is no statistical difference between their distributions (Mann-Whitney rank test, $P < 0.05$). SBB is unable to reach a competitive level, $P > 0.05$ relative to both SBB-T and SarsaRBF.

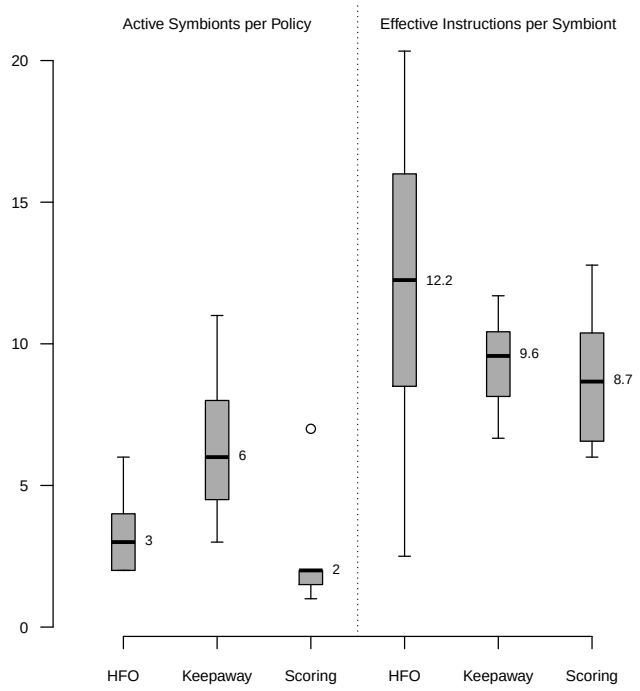

Figure 4: SBB-T solution complexity. Quartile statistics for number of symbionts (programs) per policy and instructions per program. 'HFO' denotes the SBB policy evolved against the target task (phase 2); 'Keepaway' and 'Scoring' denote the SBB policies evolved against source tasks (phase 1).

5.2 Complexity

All SBB policies are groups of symbionts working cooperatively (Section 3), and symbionts are essentially linear genetic programs. Thus, we can describe the complexity of a solution by counting the number of symbionts / instructions per symbiont referenced by a champion policy tree. Recall that two types of source task are employed during task transfer, denoted 'Keepaway' and 'Scoring' (Section 4.1). Figure 4 gives the distributions for the median number of symbionts and median number of instructions per symbiont, where both counts are post-intron removal [1], over all 10 runs of SBB-T.

For example, an HFO policy consisting of 3 symbionts, which each contain a median instruction count of ≈ 12.2, indexes three source / leaf policies. If two of these correspond to the more complex Keepaway leaf policy, each consisting of 6 programs with 9.6 instructions each, and the third is a Scoring leaf policy with 2 programs and 8.7 instructions each, then the total instruction count would be 169 instructions. This is in contrast to a SarsaRBF solution, which contains $16,320$ weight / RBF pairs that need to be calculated at each decision point. Note also that only one HFO solution ($3 \times 12.2 \approx 36$ instructions) and one Keepaway policy ($6 \times 9.6 \approx 58$ instructions) or Scoring policy ($2 \times 8.7 \approx 17$ instructions) require evaluation to produce a decision. In short, SBB policies are more efficient to implement post training. Empirically, the average CPU time required for SarsaRBF to make a decision over 1000 timesteps is 2.04 ms whereas SBB-T is an order of magnitude faster, requiring only 0.16 ms.

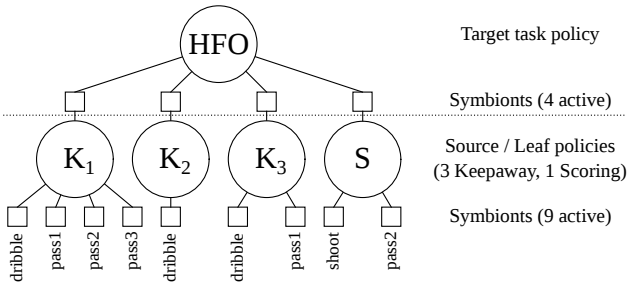

Figure 5: Example policy tree from the SBB-T experiment. Source / leaf policy 'K_2' is actually redundant, i.e. can be replaced by the 'dribble' atomic action.

5.3 Solution Analysis

In order to understand the behaviour of policy trees in the HFO task, we can analyze a single champion tree from the SBB-T experiment. Figure 5 describes one tree that managed to score in 395 of 1000 test games. This policy tree has 4 active symbionts at the upper level, 3 of which index Keepaway source policies and 1 indexes a Scoring source policy. There are a total of 9 active symbionts across all source / leaf policies, covering the full scope of domain-specific atomic actions. To understand the strategy of this policy tree, we can trace the sequence of source / leaf policies deployed at each decision point during two HFO games that each ended with a goal for this policy. Game 1:[Start K_3 K_3 K_3 K_1 K_3 K_1 K_3 K_3 K_1 K_1 K_1 S S S S S S Goal]. Game 2: [Start K_1 K_2 K_3 K_1 K_1 K_1 K_1 K_1 S K_3 S Goal]. It is clear from these traces that the strategy is to interleave different Keepaway source policies while approaching the goal, then introduce the Scoring source policy shortly before achieving the winning shot. In the case of Game 1, multiple calls to the Scoring meta-action indicate that some passing occurred near the goal region before the winning kick on goal. Interestingly, the policy has made use of K_2, which is essentially a degenerate dribble policy, implying that it can be replaced with a dribble atomic action at the symbiont in the upper policy level.

6 Conclusion

The SBB framework for evolving policy trees has been refined to facilitate the utilization of task transfer. Task transfer represents a methodology for taking policies identified under source tasks and learning how to extend or redeploy them to provide policies under a more difficult target task. In this case, the Half-field Offense sub-task 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; 2) evolve source task policies with diversity maintenance; and, 3) use Pareto dominance to refine the number of source task policies before attempting the target task.

The policy trees that were identified using task transfer achieved HFO performance competitive with the current state-of-the-art, and were significantly simpler. Conversely, it was not possible to evolve SBB policy trees with comparative performance without task transfer. Future work will continue to investigate mechanisms for making use of task transfer under SBB policy tree structures.

7 Acknowledgments

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

References

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

[2] G. Cuccu and F. Gomez. When novelty is not enough. In *EvoApp – Part I*, volume 6624 of *LNCS*, pages 234–243. Springer, 2011.

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

[4] 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.

[5] C. Gagne, M. Schoenauer, M. Sebag, and M. Tomassini. Genetic programming for kernel-based learning with co-evolving subsets selection. In *PPSN*, volume 4193 of *LNCS*, pages 1008–1017, 2006.

[6] F. Gomez. Sustaining diversity using behavioral information distance. In *ACM Conference on Genetic and Evolutionary Computation*, pages 113–120, 2009.

[7] S. Kalyanakrishnan and P. Stone. An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 749–756, 2009.

[8] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pages 72–85. Springer, Berlin, 2007.

[9] S. Kelly and M. I. Heywood. Genotypic versus behavioural diversity for teams of programs under the 4-v-3 keepaway soccer task. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014.

[10] S. Kelly and M. I. Heywood. On diversity, teaming, and hierarchical policies: Observations from the keepaway soccer task. In *European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 75–86. Springer, 2014.

[11] 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.

[12] 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.

[13] 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.

[14] Jan Hendrik Metzen, Mark Edgington, Yohannes Kassahun, and Frank Kirchner. Performance evaluation of EANT in the robocup keepaway benchmark. In *IEEE International Conference on Machine Learning and Applications*, pages 342–347, 2007.

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

[16] 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*, pages 93–105. 2006.

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

[18] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[19] R. R. Sutton and A. G. Barto. *Reinforcement Learning: An introduction*. MIT Press, 1998.

[20] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[21] Matthew E. Taylor and Peter Stone. An introduction to inter-task transfer for reinforcement learning. *AI Magazine*, 32(1):15–34, 2011.

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