Hierarchical Task Decomposition through Symbiosis in Reinforcement Learning

John A. Doucette[1], Peter Lichodzijewski[1], and Malcolm I. Heywood[2]

[1]*David R. Cheriton School of Computer Science, University of Waterloo, ON. Canada**
[2]*Faculty of Computer Science, Dalhousie University, Halifax, NS. Canada*

Article originally appears at GECCO under ACM copyright 2012
http://dl.acm.org/citation.cfm?doid=2330163.2330178

Abstract

Adopting a symbiotic model of evolution separates context for deploying an action from the action itself. Such a separation provides a mechanism for task decomposition in temporal sequence learning. Moreover, previously learnt policies are taken to be synonymous with meta actions (actions that are themselves policies). Should solutions to the task not be forthcoming in an initial round of evolution, then solutions from the earlier round represent the 'meta' actions for a new round of evolution. This provides the basis for evolving policy trees. A benchmarking study is performed using the Acrobot handstand task. Solutions to date from reinforcement learning have not been able to approach the performance of those established 14 years ago using an A* search and a priori knowledge regarding the Acrobot energy equations. The proposed symbiotic approach is able to match and, for the first time, better these results. Moreover, unlike previous work, solutions are tested under a broad range of Acrobot initial conditions, with hierarchical solutions providing significantly better generalization performance.

1 Introduction

Temporal sequence learning or reinforcement learning (RL) represents a generic class of decision making problems for which the state variables of the task domain change under the influence of actions suggested by the decision maker's policy. The states and actions are task specific. Naturally, the credit assignment process is subject to temporal discounting as it is not generally possible to provide definitive information regarding the impact of any one action on the eventual payoff of a policy [7], [21], [14]. Moreover, task domains of any practical interest generally encounter the 'curse of dimensionality' with regards to the enumeration of the state–action space. Various schemes have therefore been considered for addressing the 'scaling problem' of RL. That is, how to develop a decision maker that describes policies at a more abstract level.

One of the most frequently encountered metaphors is that of a hierarchical architecture for decision making in general. Such schemes have been recommended from many standpoints e.g., philosophy [18], cognition [16], [2] and complex systems [22]. Implicit in the assumption that such hierarchies are beneficial for building effective decision makers is that some form of abstraction takes place as the hierarchy is traversed. Thus, rather than always making decisions in terms of the task specific 'atomic' actions, meta actions are first developed that are appropriate for solving useful subtasks. We assume that such meta actions form the first level of the hierarchical policy learner. Decision making at the next level of the policy learner determines under what conditions to deploy the previously identified meta actions, thus potentially decoupling RL from the curse of dimensionality. Naturally, the above represents a motivation for the ensuing approach, developments in gradient based RL methods assume somewhat different solutions typically related to Markov Processes e.g., [5, 6].

In this work we present an entirely evolutionary framework for developing such a hierarchical policy learner without recourse to prior identification of appropriate subtasks and illustrate its application under the challenging domain of the Acrobot handstand task [1]. The resulting solution represents a *policy tree* in which each node represents a set of programs with decision making beginning at the root node. Leaf nodes deploy an atomic action

*This research was conducted while an undergraduate at Dalhousie University.

whereas internal nodes correspond to meta actions. Programs establish whether child arcs are taken or not, with only a single arc being active at any layer of the decision tree i.e., winner takes all resolution across programs from the same node. The resulting policy tree is evolved 'bottom up' with no prior assumption regarding the number or type of meta actions necessary. The evolutionary basis for such a process is symbiotic bid-based genetic programming (SBB) [13]. This work emphasizes how SBB is extended to provide explicitly hierarchical solutions and applied to policy search. Task generalization is shown to benefit significantly from the capacity to build hierarchical policies.

2 Symbiotic bid-based GP

The generic architecture for SBB explicitly enforces symbiosis by separating host and symbiont into independent populations [13]. Each host represents a candidate solution in the form of a set of symbionts existing independently in the symbiont population. Performance is measured relative to the interaction between training scenarios (points) and host. A breeder model of evolution is assumed, thus a fixed number of hosts and points are deleted/ introduced at each generation. The respective properties of symbiont and host population are developed below. Note that space precludes a complete description of the algorithm [13]; instead we draw the reader's attention to the specific major differences introduced here for applying SBB to the temporal difference domain i.e., constructing policy trees. Interested readers are referred to online resources for further details [10, 11].

2.1 Representation and execution

2.1.1 Symbiont Population

Members of the symbiont population assume a Bid-Based GP representation [12]. As such, each symbiont, sym, is represented as a tuple $\langle a, p \rangle$; where a is an action as selected from the set of atomic actions associated with the task domain and p is the corresponding symbiont's program. Without loss of generality, a linear representation is assumed [3]. Execution of a symbiont's program results in a corresponding real-valued outcome in the output register, $R[0]$. In order to encourage a common bidding range from the outset, this is mapped to the unit interval through the sigmoid operator, or $sym(bid) = (1 + \exp(-R[0]))^{-1}$. The linear representation leads to programs being defined by a simple register addressing language of the form: 1) Two argument instructions, or $R[x] := R[x]\langle op_2 \rangle R[y]; op_2 \in \{+, -, \div, \times\}$; 2) Single argument instructions, or $R[x] := \langle op_1 \rangle (R[y]); op_1 \in \{\cos, \ln, \exp\}$; 3) A conditional statement of the form "IF $(R[x] < R[y])$ THEN $(R[x] := -R[x])$. In addition, $R[y]$ can be either a register reference or index a state variable.

2.1.2 Host Population

Symbionts are explicitly limited to deploying a single action. Thus a host needs to identify relevant subsets of symbionts that are capable of collaborating or ***lateral task decomposition***. To do so, each host indexes a subset $[2, ..., \omega]$ of the symbionts currently existing in the symbiont population. Fitness evaluation is only performed for hosts, symbionts do not have a fitness. Thus, for each host, h_i, fitness is evaluated with respect to a set of initial configurations of the task domain, as defined by individuals from the point population, p_j. For each task initialization, p_j, a series of interactions – defining an episode – occur between task's state variables and action as suggested by the host presently under evaluation. Each training episode ends when a task specific end condition or a computational limit is encountered (see Section 3).

In the case of SBB, each interaction between task and host has the following form:

1. Present the state variables from the task domain at time step t_s, or $\vec{s}(t_s)$;
2. Execute all (symbiont) programs identified by host, h_i, resulting in a matching number of symbiont bids or $\forall sym \in h_i : sym(bid(t_s))$, where $sym(bid(t_s))$ is the bid value resulting from execution of a symbiont's program (Section 2.1.1);
3. Identify the 'winning' symbiont as that with the maximum bid from host h_i or, $sym^* = \arg_{sym \in h_i} \max(sym(bid(t_s)))$;
4. Present the action from the winning symbiont to the task domain and update the state variables accordingly or $\vec{s}(t_s + 1) \leftarrow task \leftarrow sym^*(a)$; where $sym^*(a)$ is the action of the winning symbiont identified at Step (3).

Symbionts therefore use bidding to establish the **context** for deploying their respective action. The number of symbionts per host and 'mix' of actions represented by a host are both an artifact of evolution. The relative uniqueness of the distribution of goal satisfaction across the host and point populations will be discounted under competitive fitness sharing [17], Section 2.2.2. Should a single dominant policy *not* emerge then fitness sharing represents the principal scheme for developing meta action diversity.

2.2 Selection and Replacement

2.2.1 Point Population

The role of the point population is to define a sample of initial task configurations with sufficient diversity to provide variation in the behaviours of hosts as measured through the fitness function. A tabula rasa approach is assumed in which a simple stochastic sampling heuristic is adopted. At each generation P_{gap} points are removed with uniform probability and a corresponding number of new points introduced. The process for initializing / generating points is naturally a function of the task domain itself and will be detailed within the context of the Acrobot handstand task (Section 3).

2.2.2 Host Population

As per the point population, a fixed number of hosts, H_{gap}, are removed at each generation. Host removal is applied deterministically with the worst H_{gap} hosts removed at each generation. However, assuming a competitive fitness sharing formulation [17] maintains diversity in the host population. Thus shared fitness, s_i, of host h_i takes the form:

$$s_i = \sum_k \left(\frac{G(h_i, p_k)}{\sum_j G(h_j, p_k)} \right)^3 \tag{1}$$

where $G(h_i, p_k)$ is the task dependent reward defining the quality of policy h_i on test point p_k (see Section 4).

Naturally, deleting the worst H_{gap} hosts may result in some symbionts no longer receiving (host) indexes. Given that such symbionts must have been associated with the worst performing hosts, these symbionts are also deleted. A secondary implication of this is that symbiont population size will vary whereas the host population size remains fixed.

2.3 Variation Operators

Symbiosis is an explicitly hierarchical coevolutionary process. From an exploration/ exploitation perspective it is important not to disrupt 'good' symbiont combinations while simultaneously continuing to search for better hosts. Moreover, variation needs to be maintained at the symbiont level without disrupting symbionts that are already effective. The process for maintaining exploration (diversity) without unduly disrupting the better host–symbiont relationships therefore follows an explicitly hierarchical formulation. Following the removal of H_{gap} hosts, the remaining $H_{size} - H_{gap}$ hosts are sampled for cloning with uniform probability. The resulting clones have both their host and symbiont content modified. As such, it is ensured that new symbionts are only associated with new hosts and therefore no disruption of previous host behaviour takes place. For a detailed presentation of this process see [10, 11, 13].

2.4 Evolving policy trees through SBB

The above description summarizes the process as applied to a single 'level' or round of evolution as conducted over a fixed number of generations t_{max}. However, the content of the host–symbiont population pair might not provide any outright solutions to the task domain. Rather than begin evolution afresh from a completely new host–symbiont parameterization/ initialization, the current host population is considered to represent a set of **meta actions** from which solutions can be more readily composed. That is to say, the hosts as previously evolved at level 'l' represent the set of new 'meta' actions for the symbiont population at level '$l + 1$'. Thus the only difference between evolution at each level is the declaration of the actions that symbionts may index. At level $l = 0$ symbiont actions are always defined by the task domain, or **atomic actions**. Thereafter, for symbionts at level $l > 0$ the action set is the set of all hosts from level $l - 1$ or $a \in \{H^l\}$. For simplicity we assume that evolution is only ever conducted at the highest 'level'; all earlier levels having their content 'frozen'. Moreover, there is no

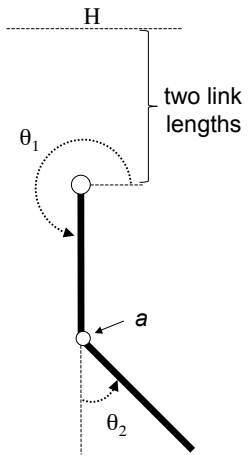

Figure 1: Acrobot 'handstand' domain. The controller is provided with state variable information in the form of the angular position θ_1, θ_2 and velocities $\dot{\theta}_1, \dot{\theta}_2$. The controller supplies the torque a to the acrobot 'waist,' the feet are at the free end of the second link whereas the 'hands' are connected to the rotating bar of the first link. Goal distance 'H' is set to two links above the axis of θ_1 in the handstand task. The force the control policy can apply is limited to one of three values: $a \in \{0, \pm 1\}$ N.

requirement for each host at level l to index all available actions from level $l-1$. The subset of actions utilized by each host is a function of policy search.

Evaluating host i at level l (or h_i^l) now has the following hierarchical form:

1. Present the state variables describing the current state of the task domain, or $\vec{s}(t_s)$;
2. $\forall sym^l \in h_i^l$, identify the corresponding level set of l symbiont bids, or $sym^l(bid(t_s))$;
3. Identify the 'winning' symbiont for host h_i^l, or $sym^* = \arg_{sym^l \in h_i^l} \max(sym^l(bid(t_s)))$;
4. IF $l == 0$ THEN Step (5) ELSE
   (a) Update level: $l = l - 1$;
   (b) Let the action from the winning symbiont, $sym^*(a)$, in Step (3) identify the new host: $h_i^l \leftarrow sym^*(a)$;
   (c) RETURN to Step (2);
5. Present the atomic action from the winning symbiont to the task domain and update any state variables accordingly, or $\vec{s}(t_s + 1) \leftarrow task \leftarrow sym^*(a)$; where $sym^*(a)$ is the action of the winning symbiont as identified at Step (3).

Such a process represents a process for the bottom-up evolution of a **policy tree**. Conversely, evaluation of a policy tree is performed top-down relative to the host–symbiont population currently under evolution. Nodes in the tree are hosts, whereas symbionts define the branching factor of a node. The path from each branch is selected as per the action of a winning bid. Nodes at deeper levels represent hosts evolved at previous generations. Only the lowest level nodes (hosts) retain symbionts with actions specified in terms of the task domain's atomic actions.

3 Handstand Acrobot

The Acrobot task domain represents a domain in which a two link 'body' has a control force applied to the 'waist' or middle link (Figure 1) [19], [21]; whereas the hands remain fixed at an axle and the feet are free to spin about the waist. The basic goal is to learn a policy that swings the feet to a goal height, H, above the hands. When this height is one link above the hands, this represents the easier *height* task [21]. An alternate definition for the goal state requires that the Acrobot perform a *handstand* [19]. Thus, a policy needs to be identified that is capable of balancing the 'feet' of the Acrobot such that the two links are aligned in the same vertical plane directly above the Acrobot's hands with a near-zero velocity. In this work we concentrate on the more difficult handstand task.

Table 1: Task Domain Parameterization. *C. of M.* denotes Center of Mass; *M. of I.* denotes Moment of Inertia.

Length of upper (lower) link	1.0m (1.0m)
Length to C. of M.: upper (lower) link	1.0m (1.0m)
M. of I.: upper (lower) link	1.0 (1.0)
Gravitational constant (g)	9.8
Control torque update freq.	5 Hz
Simulation step size	0.05 sec
Target capture region position	$\frac{\pi}{2} \pm 0.3$ rad
Target capture region velocity	± 0.3 rad/sec
Max simulation steps (D_{max})	200
Atomic actions (a)	$\{0, \pm 1N\}$

3.1 Previous Results under RL

Previous approaches to the handstand task using RL provided solutions that additionally made use of: simplifications to the goal condition (e.g., by not requiring any velocity reduction [15]); or increased the control force beyond the ± 1 N permitted (e.g., real-valued forces over the interval ± 30 N [23], [8] or ± 2 N [4]). One recent development in which the Acrobot handstand task utilizes the original task constraints incorporates the concept of 'intrinsic utility' [6]. Specifically, a Monti-Carlo approximation is used to provide a measure of intrinsic reward applicable to domains with continuous state spaces. The ensuing solution is able to provide solutions taking 17–18 seconds of simulated time (or 85 to 90 interactions).

3.2 Solution using exhaustive search

It is possible to utilize domain specific heuristics to simplify the problem to the extent that an A^* search represents a feasible approach for discovering solutions [1]. Specifically, information was utilized to establish constraints to the frequency of torque direction switches and speed–energy profiles, as well as emphasizing the use of 'singular arcs' during a trajectory. While this by no means detracted from the quality of solutions produced, it did leave several questions unanswered. First, it was still unclear whether general machine learning algorithms could discover solutions of equivalent quality. Boone's work included a comparison with the well known SARSA lambda RL algorithm (e.g., [21]), but found that it failed to converge to a solution after several days of training. Second, although Boone's work found a best case trajectory starting from the stable equilibrium position – or the **swing-up** task – the method produced no meaningful approximation of the state space. This solution is simply a sequence of torque values specific to the stable equilibrium starting point. As a consequence, varying the start position of the arm by even small amounts will require re-running the search algorithm to generate a new torque sequence. In this work we investigate the special case of the swing-up initial condition and generalizing to multiple Acrobot initializations.

3.3 Atomic action and state variables

The Acrobot parameterization explicitly follows that established in [1]. The goal of the handstand task is to identify control policies that successfully enter a ***target capture region*** while enforcing the 1 N magnitude limitation on the control force. The target capture region define specific conditions under which the goal state is satisfied i.e., a region in phase space corresponding to an (unstable) vertical equilibrium position (Table 1).[1] **State variables** take the form of angular position θ_1, θ_2 and velocities $\dot{\theta}_1, \dot{\theta}_2$ (Figure 1). Figure 1 and Table 1 also identify the available **atomic actions**.

3.4 Performance evaluation

The Acrobot handstand task will be considered from the perspective of two independent experiments: 1) rewarding solutions for the 'swing-up' formulation of the task i.e., the ultimate goal is to provide a controller capable of swinging the acrobot from the single at rest stable equilibrium position to the Acrobot entering the target capture region (defined in Section 3.3). Such an approach facilitates direct comparison with what still represents the best solution to the swing-up task (82 steps) [1]. 2) rewarding generalization properties; in which case the radii about

[1]Established practice is to deploy a LQR controller once the target capture region is encountered [19].

the Acrobot 'hands' are set to twice the link length and initial Acrobot configurations sampled at 200 equally spaced intervals. Given the differing objectives of the two assessments, the process for initializing and replacing individuals from the point population assumes different biases (see Section 3.5).

3.5 Point population routines

Acrobot configurations defined by the point population always assume alignment of the two limbs ($\theta_2 = 0$, Figure 1) and zero initial velocity. Points differ with respect to their θ_1 values. Given the a priori knowledge regarding the significance of the $< \frac{\pi}{2}, 0 >$ configuration in the swing-up runs versus a requirement for generalization in the second Acrobot runs, we introduce a bias to ensure a greater frequency of encountering the swing-up configuration during this training. Thus, for the P_{gap} points replaced at each generation, the point replacement process has the following generic form. With frequency p_{pf} initialize points to the swing-up position. The remaining $1 - p_{pf}$ points are created by selecting a 'parent point' and adding an offset of 0.05 radians to θ_1 with sign selected stochastically.

At initialization, p_{pf} points are again set to the swing-up position, with the remainder initialized with θ_1 selected with uniform probability over the interval $[0, 2\pi)$. The only difference between point initialization processes as applied to the swing-up task versus that applied during the generalization test is the value used for p_{pf}. Thus, in the swing-up (generalization) runs the corresponding value for p_{pf} is 0.5 (0.0). Naturally, points corresponding to the swing-up position benefit from fitness caching.

3.6 Concluding comments

A complete description of the system dynamics may be found in several previous works (e.g., [19], [21]), but amount to a second order differential equation. However, the Acrobot is highly sensitive to the quality of the simulation used because small errors produce cascading effects. Commonly used techniques like the Euler-Cromer method are inappropriate for use in this task domain using conventional time step sizes. With this in mind, the 4th order Runge-Kutta method of integration was used to define the new position and velocity of the Acrobot at each time step.

4 Experimental Methodology

4.1 Lexicographic cost function

A hierarchy of properties within a single scalar fitness function is established. Thus, constraints on each of the measured properties are sequentially satisfied in order or 'incremental evolution'. Three properties are considered: position, velocity and time; with position and velocity enforcing constraints associated with the target capture region employed by Boone (Section 3) and provides a measure of solution quality; summarized as follows,

1. Position (F_p): expresses the need to first reach the target capture region and was motivated by the approach taken by Munos and Moore [15] or,

$$\begin{array}{cl} \frac{\sin(\theta_1)+\sin(\theta_1+\theta_2)+2}{4} & \leftarrow \text{IF } (\theta_1 + \theta_2 \notin \{\pi/2 \pm 0.3\} \\ & \text{OR } \theta_1 \notin \{\pi/2 \pm 0.3\}) \\ 1 & \leftarrow \text{otherwise} \end{array} \tag{2}$$

2. Velocity (F_v): enforces the requirement to enter the target capture region at a sufficiently low velocity or,

$$\begin{array}{cl} 0 & \leftarrow \text{IF } (F_p < 1 \text{ OR } |\dot{\theta_1}| > 2 \text{ OR } |\dot{\theta_2}| > 2) \\ 1 & \leftarrow \text{IF } (|\dot{\theta_1}| < 0.3 \text{ OR } |\dot{\theta_2}| < 0.3) \\ \frac{4-|\dot{\theta_1}|-|\dot{\theta_2}|}{4} & \leftarrow \text{otherwise} \end{array} \tag{3}$$

3. Duration (F_d): distinguishes between the quality of solutions based on the number of steps, t_s, necessary to reach the target capture region, given a maximum duration to locate a solution, D_{max} (Table 1) or,

$$\begin{array}{cl} \frac{t_s}{D_{max}} & \leftarrow \text{IF } (F_v = 1) \\ 0 & \leftarrow \text{otherwise} \end{array} \tag{4}$$

Table 2: Parameterization of Host and Symbiont populations. As per Linear GP, a fixed number of general purpose registers are assumed (*numRegisters*) and variable length programs subject to a max. instruction count (*maxProgSize*).

Host (solution) population			
Parameter	Value	Parameter	Value
t_{max}	1 000	ω	10
P_{size}, H_{size}	120	P_{gap}, H_{gap}	20, 60
p_{md}	0.7	p_{ma}	0.7
p_{mm}	0.2	p_{mn}	0.1
Symbiont (program) population			
numRegisters	8	*maxProgSize*	48
p_{delete}, p_{add}	0.5	p_{mutate}, p_{swap}	1.0

The resulting reward function is merely the sum over contributions from each of the three properties (with the proviso that the properties are satisfied in order) or,

$$G(h_i, p_k) = \frac{1}{3} \max_{t_s} (F_p(t_s) + F_v(t_s) + F_d(t_s)) \tag{5}$$

where $0 \leq t_s \leq D_{max} - 1$ are the discrete time steps over which an interaction takes place. Thus, $G(\cdot)$ returns the best value encountered over the duration of a policy given a point–host pair (h_i, p_k). Naturally, if both position, F_p, and velocity, F_v, goals are satisfied, an interaction will terminate with the current value for the duration property, F_d. Such a reward function is of comparable complexity to functions designed by others for the swing-up task (e.g., [8], [4], [1]).

4.2 SBB Parameterization

Parameterization of SBB follows exactly the same form as used in the earlier supervised learning (classification task domain) benchmark study [13], and is summarized in Table 2. Needless to say, no claims are made regarding the optimality of these parameters.

4.3 NEAT Parameterization

The C++ distribution of NEAT [20] is employed to provide a baseline for the relative difficulty of the task domain; where NEAT represents one of the most widely utilized frameworks for policy search in RL. The original NEAT distribution is augmented with the point population to provide the same tabula rasa training dynamic as SBB receives. Thus, at each generation performance of NEAT individuals are evaluated over P_{size} points, with P_{gap} points replaced. The reward function – that is before application of NEAT specific speciation/ fitness sharing mechanisms – is the sum of rewards over all points, or Equation (5). Needless to say, both NEAT and SBB assume a common parameterization of the point population, with P_{size} and P_{gap} set to 120 and 20 respectively.

A base parameterization of NEAT was established from the original experiments with the inverted pendulum problem [20]. As per recommendations from the NEAT users page (http://www.cs.ucf.edu/~kstanley/neat.html), additional experimentation was then performed to tune the link weight mutation probabilities and assess the impact of permitting recurrent links. The resulting parameterization is summarized in Table 3. Naturally, no claims regarding the optimality of this selection is made.

Finally, the number of fitness evaluations between SBB and NEAT runs was designed to be as similar as possible, or 16,812,000 NEAT evaluations versus 16,800,000 SBB evaluations. This reflects the different population sizes for each algorithm and support for hierarchical SBB. Thus, hierarchical SBB assumes 8,400,000 evaluations per layer. In the case of each architecture 100 initializations are performed for each experiment.

5 Results

Section 3 introduced a two part framework for evaluating solutions to the Acrobot handstand task: swing-up versus generalization. Results are first reported with respect to the Acrobot 'swing-up' task under the problem configuration established by Boone, Section 5.1. We then then go on to assess the "generalization properties"

Table 3: Parameterization of NEAT.

Parameter	Value	Parameter	Value
Pop size	150	c_1	1.0
Survival thres.	0.2	c_2	1.0
weight mut. power	2.5	c_3	0.4
p(wt. link)	0.9	δ	3.0
p(toggle)	0.01	p(gene reint.)	0.001
p(add node)	0.03	p(add link)	0.05
p(recur. link)	0.1	p(mate only)	0.2
p(inter.spec.mate)	0.001	p(mutate only)	0.25
Dropoff age	15	New link tries	20

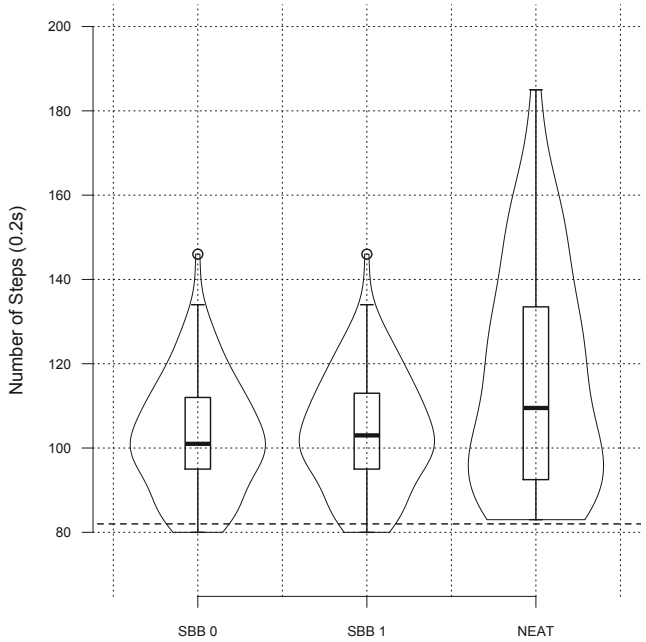

Figure 2: Number of steps necessary to solve the Acrobot swing-up task. The horizontal dashed line represents the case of the shortest previously known solution from Boone (82 steps) [1]. Box plot represents quartile statistic.

of the SBB approach by considering performance under an independent test set, as evolved under the second formulation of the point population, Section 5.2.

In all cases Shapiro-Wilk tests and visualization are used to verify the support for assuming a normal distribution in the data. Should normality be supported in all data involved in a hypothesis test, then 1-way ANOVA and student t-tests establish the corresponding statistical significance. Conversely, when data is not normally distributed then a Kruskal-Wallace test is employed. Bonferroni corrections were used to adjust p-values based on the number of hypothesis tested.

5.1 Experiment 1: Swing-up task

The basic objective is to evolve solutions to the swing-up initialization of the handstand task using the smallest number of steps as possible. NEAT found solutions on 70% of the runs, whereas SBB found solutions in 95% of runs. The following analysis is therefore performed relative to the runs that succeeded in solving the swing-up task. Figure 2 summarizes the time taken by a valid solution trajectories under the swing-up task. The distributions were not normal. No significant difference appeared between SBB levels 0 and 1, but both SBB results differed significantly from solutions identified by NEAT ($p < 0.05$ in both cases). Observed sample means were 102, 103 and 115 steps for SBB levels 0 and 1, and NEAT respectively.

In addition to constantly providing solutions to the swing-up task SBB was also able to match the previous best case 82 step trajectory on the swing-up task from Boone [1] in 3 cases and in 2 cases identified a faster solution

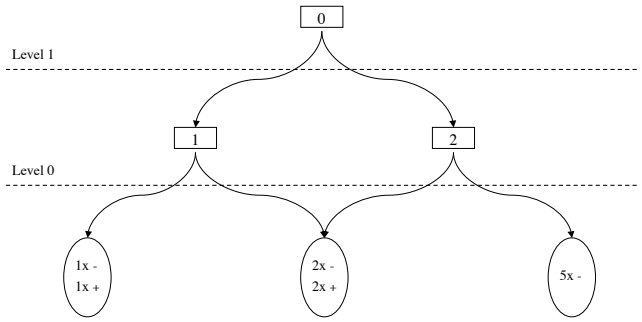

Figure 3: Breakdown of SBB architecture corresponding to the best Acrobot swing-up solution from level 1. A total of 169 instructions are distributed across 13 symbiont programs and 3 hosts.

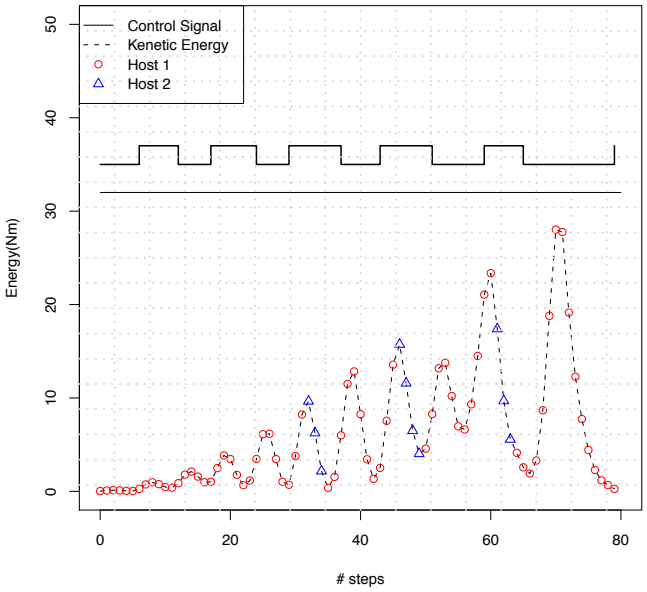

Figure 4: SBB 1 solution in 80 steps w.r.t. Acrobot swingup task . Different coloured icons differentiate between different level 0 host (meta action) deployed by level 1 symbionts.

utilizing 80 and 81 steps respectively for both non-hierarchical (SBB 0) and hierarchical policies (SBB 1) i.e., SBB 0 are reused as meta actions by SBB 1. Naturally, SBB 0 is already solving the swingup task, where this is achieved under half the evaluation count.

The principle reason for the existence of faster solutions under SBB than in the original work of Boone is that the policy is now non-greedy. Rather than always selecting actions to increase energy, actions are selected that initially result in less energy injected into the Acrobot. This actually results in more energy being pumped into the Acrobot at later time steps. In Section 5.2 we emphasize how hierarchical SBB is able to leverage this capability into better generalization performance.

The structure of the policy tree associated with the 80 step SSB 1 solution is summarized by Figure 3. Ellipses at the bottom of the figure represent sets of symbionts (and corresponding atomic action) *common* to a host. Thus, the left hand ellipse labeled '1x $-$, 1x $+$' denotes two symbionts, one with an atomic action of $-1N$ and the second with an atomic action of $+1N$. The arc connecting this pair of symbionts to the level 0 host with label '1' (left hand rectangle) implies that both symbionts are *unique* to this host. However, the two symbionts identified by the central ellipse are used by *both* hosts from level 0 (hosts carrying labels 1 and 2); as indicated by the two arcs pointing to the central set of symbionts. The last (right hand) set of symbionts consists of five symbionts all with a $-1N$ atomic action. Likewise, the single level 1 host (labeled 0) has two arcs, one to each level 0 host. Thus, there is a single level 1 symbiont associated with deploying the meta action associated with the behaviour of each level 0 host. Moreover, the solution policy is itself a combination of level 0 policies as opposed to merely the case of deploying one level 0 policy relative to a specific initial condition (Figure 4).

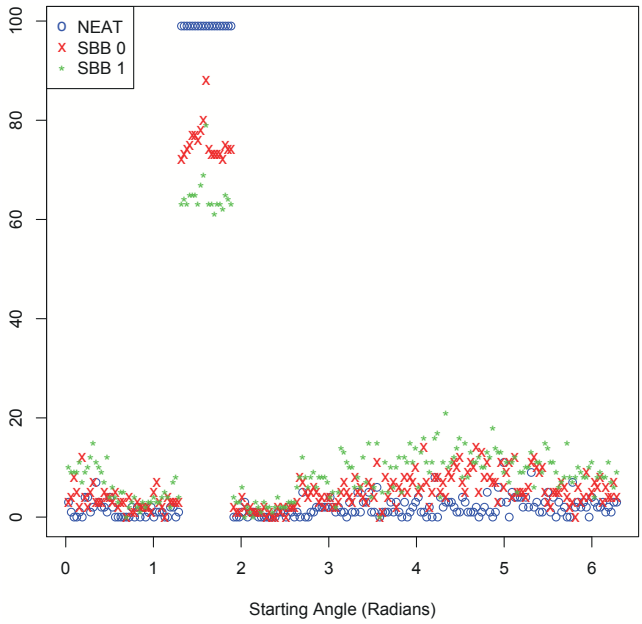

Figure 5: Scatter plot summarizes the percentage of runs for each method that solved an Acrobot initial state for a θ_1 initial condition. The plot requires viewing in colour. Test cases in the vicinity of 1.571 radians correspond to the target capture region and have a trivial solution. Cases in the vicinity of 4.712 radians correspond to the 'swing-up' task.

5.2 Experiment 2: Generalized Acrobot task

The focus of the Acrobot swing-up task of Section 5.1 was to evolve a solution with the smallest number of time steps relative to the single swing up test configuration. No advantage was demonstrated in the hierarchical formulation of SBB. In this section we are more interested in solutions that generalize, or provide solutions to as many test configurations as possible. From the perspective of the evolutionary framework, the only difference in the approach is with regard to the method for initializing/replacing members of the point population, everything else remains unchanged (Section 3.5). A total of 200 test cases are generated by sampled uniformly over the range of θ_1 or 2π. Each test case is therefore equivalent to rotating the feet to points on the circumference of a circle (radii equal to the two Acrobot links) and letting the Acrobat 'fall' under gravity. A point in the test set was deemed "solved" if the controller being tested successfully drove the Acrobot into the target capture region within 200 steps.

Naturally, test cases about the target capture region require a rather different behaviour than those elsewhere. Indeed, it transpired that such Acrobot configurations can be 'solved' through learning to do nothing. Figure 5 summarizes the frequency with which a test condition of a given initial θ_1 are solved. Given the offset in defining the 'origin' for θ_1 (Figure 1), then angles in the region of $\pi/2$ (≈ 1.571 radians) correspond to test cases in the target capture region.

In summary, NEAT solutions are dominated by those corresponding to test cases around the target capture region, in effect a mediocre stable state dominates the resulting solutions. Note also that there is no bias in the generation of training initializations in the point population to create initial Acrobot configurations in this region, all configurations are equally likely (Section 3.5). The addition of hierarchical SBB actually results in a redistribution of test cases solved away from those within the target capture region. Naturally, in the case of SBB, level 1 is limited to using behaviours previously learnt at level 0. Thus, in order for level 1 hosts to solve new test cases relative to those solved at level 0, as is the case here, then level 0 hosts must be redeployed to identify useful temporal abstractions at level 1.

The violin plot of Figure 6 summarizes the number of test cases solved when the 19 Acrobot configurations at or near the goal state are removed. The distributions are non-normal, and significant differences exist between samples ($p < 10^{-16}$). The observed average solution rates per point were 1.81%, 4.92% and 7.95% for NEAT and SBB levels 0 and 1 respectively. In short, for a given point, SBB level 1 is more than 4 times as likely as NEAT to have an individual capable of solving the test point or nearly twice as likely as a hosts at level 0.

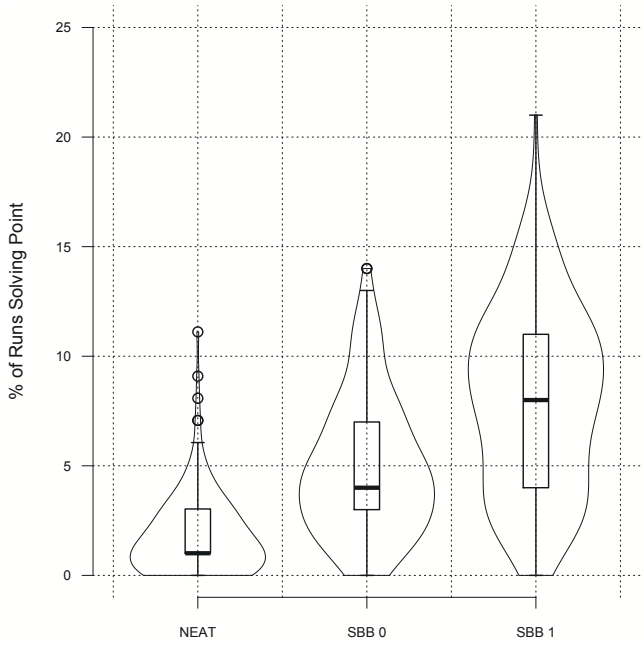

Figure 6: Percentage of individuals solving an Acrobot test configuration. The 19 test points within the vicinity of the target capture region leading to trivial policies are excluded. Box plot represents quartile statistic.

6 Conclusion

A symbiotic framework for evolving policy trees under RL is presented. Each round of evolution results in the construction of a new level to the policy tree. In the special case of the first round of SBB deployment actions take the form of the atomic actions of the task domain. Should solutions not be identified, then the content of the host population from the previous round of evolution are considered meta actions. The next round of evolution attempts to build hosts from symbionts that reference the meta actions as their actions; or policy construction at a more abstract level. A bottom up process for constructing policy trees is established where this has implications for addressing the scaling problem of RL. No a priori knowledge is necessary regarding what represent appropriate meta actions, the same generic cost function is employed throughout. Instead meta action discovery is a function of fitness sharing. The framework is demonstrated under the Acrobot handstand task. We establish for the first time solutions that match and better those identified under the A^* search heuristic and emphasize the contribution of hierarchical solutions to generalizing over multiple Acrobot configurations.

Future research will continue to benchmark the approach on additional task domains. Indeed, [10, 11] present results in a truck reversal domain[2] and [9] illustrates the approach using the 'Pinball' task. In all cases hierarchical SBB results in a significant improvement to the generalization performance. More generally a theme of particular interest is the interaction between ecology and the conditions under which new levels to the hierarchy are introduced.

Acknowledgements

The authors are grateful for support from MITACS, NSERC and CFI New Opportunities granting agencies and Killam Trust (Canada).

References

[1] G. Boone. Minimum-time control of the acrobot. In *International Conference on Robotics and Automation*, pages 3281–3287, 1997.

[2] A code distribution is available for this task at http://www.cs.dal.ca/~mheywood/Code/SBB/

[2] M. M. Botvinick, Y. Niv, and A. C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113:262–280, 2009.

[3] M. Brameier and W. Banzhaf. A comparison of linear Genetic Programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, 2001.

[4] R. Coulom. High-accuracy value-function approximation with neural networks applied to the Acrobot. In *European Symposium on Artificial Neural Networks*, pages 28–30, 2004.

[5] Y. Engel. *Gaussian process reinforcement learning*, pages 439–447. Springer, 2010.

[6] T. Jung, D. Polani, and P. Stone. Empowerment for continuous agent-environment systems. *Adaptive Behavior*, 19(1):16–39, 2011.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Machine Learning Research*, 4:237–285, 1996.

[8] K. Kawada, M. Obika, S. Fujisawa, and T. Yamamoto. Creating swing-up patterns of an Acrobot using Evolutionary Computation. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 261–266, 2005.

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

[10] P. Lichodzijewski. *A symbiotic bid-based framework for problem decomposition using Genetic Programming*. PhD thesis, Faculty of Computer Science, Dalhousie University, 2011. http://www.cs.dal.ca/~mheywood/Thesis/PhD.html.

[11] P. Lichodzijewski, J. A. Doucette, and M. I. Heywood. A symbiotic framework for hierarchical policy search. Technical Report CS-2011-06, Dalhousie University, Faculty of Computer Science, 2011. http://www.cs.dal.ca/research/techreports/.

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

[13] 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.

[14] D. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *J. of Machine Learning Res.*, 11:241–276, 1999.

[15] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *International Joint Conference on Artificial Intelligence*, pages 1348–1355, 1999.

[16] O. P. -Y, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

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

[18] H. A. Simon. The architecture of complexity. In *The Sciences of the Artificial*, chapter 4. MIT Press, 1969.

[19] M. W. Spong. The swing up control problem for the Acrobot. In *IEEE Control Systems Magazine*, pages 49–55, 1995.

[20] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

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

[22] R. A. Watson and J. B. Pollack. Modular interdependency in complex dynamical systems. *Artificial Life*, 11(4):445–457, 2005.

[23] J. Yoshimoto, S. Ishii, and M. Sato. Application of reinforcement learning to balancing of acrobot. In *IEEE Int. Conf. Systems, Man and Cybernetics*, pages 516–521, 1999.