# A Symbiotic framework for Hierarchical Policy Search

Peter Lichodzijewski* John A. Doucette† Malcolm I. Heywood‡

October 25, 2011

**Abstract**

Hierarchical reinforcement learning (HRL) traditionally represents a framework in which a machine learning algorithm is applied to build solutions to temporal sequence style problems under the guidance of a priori identified sub-tasks. Once learning relative to one set of subtasks is complete, these can then be reused to build more complex behaviours. The principal caveat is that appropriate subtasks can be identified, preferably without requiring a priori knowledge. This work proposes a generic architecture for evolving hierarchical policies through symbiosis. Specifically, symbionts define an action and an evolved context, whereas each host identifies a subset of symbionts. Symbionts effectively coevolve within a host. Natural selection operates on the hosts, with symbiont existence a function of host performance; or a form of group selection. It is now possible to support hierarchical policies as a symbiotic process by letting hosts evolved in an earlier population become the symbiont actions at the next. Two benchmarking studies are performed to illustrate the approach. An initial tutorial is conducted using a truck reversal domain in which the benefits of evolving a hierarchical solution over non-hierarchical solutions is clearly demonstrated. A second benchmarking study is then performed using the Acrobot handstand task. Solutions to date from reinforcement learning have not been able to approach those established 13 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 works, 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 domain 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 [26], [56], [39]. Moreover, task domains of any practical interest generally encounter the 'curse of dimensionality' with regards to the enumeration

*Faculty of Computer Science, Dalhousie University, NS. Canada. {piotr}@cs.dal.ca
†David R. Cheriton School of Computer Science, Waterloo University, ON, Canada. {jaedoucette}@gmail.com
‡Faculty of Computer Science, Dalhousie University, NS. Canada. {mheywood}@cs.dal.ca

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 i.e., any one decision potentially becomes a collection of state–action pairs in the task domain. 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., philosophical [49], cognition [43], [8] and complex systems [60]. Two popular approaches to developing such architectures include Layered Learning and hierarchical RL. Specifically, given sufficient a priori knowledge of the task domain, then appropriate hand crafted training scenarios may be designed, resulting in a hierarchy of increasingly goal orientated behaviours or Layered Learning e.g., [55]. Conversely, a priori knowledge can also be used to explicitly define sets of sub-goals/ task decomposition, with each sub-goal learnt using RL e.g., [3]. In both cases a priori knowledge is used to support the identification of suitable state and/ or temporal abstractions. That is, domain specific 'meta-actions' are learnt that enable the decision maker to formulate policy at a more abstract level than the original *atomic* actions of the task domain, thus potentially decoupling RL from the curse of dimensionality.

The goal of the research reported here is to suggest a framework for evolving a hierarchical sequential decision maker through a bottom up process with minimal recourse to a priori sources of information. To do so we assume that the most fundamental task of the learning algorithm is to identify *contexts* for actions. We make the assumption that a discrete enumeration of domain specific 'atomic' actions is known a priori. With this in mind, two distinct types of individual are to be coevolved: symbionts and hosts. Symbionts are defined in terms of a single (discrete) action and a program that identifies a corresponding context (for the action) through the evolution of a bidding strategy. A host population learns to build appropriate combinations of symbionts, thus policies, under a common reinforcement objective. Competitive fitness sharing ensures a diverse population of host behaviours with respect to interactions with the task domain. A bottom-up process for hierarchical policy identification is then facilitated by letting the population of currently evolved hosts represent candidate *meta-actions* for the next population of symbionts, thereafter the entire process of evolution is reapplied *hierarchically*. Thus, hosts represent behaviours as coevolved from symbionts described in terms of actions at increasing levels of abstraction. However, the level of abstraction is entirely a function of behaviours established by the previous host population – there is not an a priori definition of appropriate sub-tasks/ goals.

The resulting Symbiotic Bid-Based (SBB) framework for Genetic Programming (GP) will be illustrated under two examples of task domains taken from direct control. In the first case a tutorial is conducted using a modified version of the truck reversal domain in which the goal is to reverse a semi–cab from any position in a world to the 'loading bay' while avoiding a wall obstacle. Benchmarking relative to SBB without the capacity to build hierarchical policies and the NEAT algorithm for RL demonstrates that significant benefits result from supporting hierarchical policies, particularly with respect to generalization performance. The second task domain is that of the Acrobot handstand task. This task has previously resisted attempts to solve it using machine learning methods alone. Current best practice instead relies on the use of domain specific knowledge to simplify various aspects of the problem. Conversely, the SBB approach is able to solve both the 'swing-up' task as originally formulated (without hierarchical policies) and generalize to multiple Acrobot configurations outside the initial swing-up goal (with hierarchical policies). Indeed, this represents the first time that a learning algorithm has identified solutions that matched/ bettered the previous best case solution – a case that was itself only identified by making copious use of domain specific a priori information such that

the $A^*$ exhaustive search algorithm could be employed [7].

In the remainder of the paper we first provide a summary of related research (Section 2), where this covers a broad spectrum of developments in RL. The approach taken to designing a symbiotic GP framework for hierarchical policy identification is detailed in Section 3. Section 4 introduces the aforementioned reinforcement style benchmarking task domains, with the parameterization of the task and comparator models discussed in Section 5. Results for each task domain are presented for truck reversal (Section 6) and Acrobot handstand (Section 7). Concluding remarks and a discussion of future research opportunities then follows in Sections 8 and 9 respectively.

# 2 Background and Related Research

## 2.1 Temporal Sequence Learning

There are two basic machine learning approaches for addressing the temporal sequence learning problem: value (function) optimization [26], [56] and policy search/ optimization [39]. In the case of value optimization each state–action is assumed to result in a corresponding 'reward' from the task domain. Such a reward might merely indicate that the learner has not yet encountered a definitive 'failure' condition. A reward is generally indicative of the immediate cost of the action – as opposed to the ultimate quality of the policy e.g., as in the immediate cost of accelerating as opposed to not accelerating. In this case the goal of the temporal sequence learner is to learn the relative 'value' of state–action pairs such that the 'best' action can be chosen given the current state. Moreover, such a framework explicitly supports online adaptation [56]. Given that there are typically too many state–action pairs to exhaustively enumerate (especially when the state variables are continuous rather than discrete), some form of function approximation is necessary. Conversely, policy optimization does not make use of value function information [39]. Instead the performance of a candidate policy/ decision maker is assessed relative to other policies with the ensuing episode (sequence of state–action pairs) left to run until some predefined stop criterion is encountered. This represents a direct search over the space of policies that a representation can describe. An important implication of this is that training be conducted over a *sample* of training scenarios. Specifically, if training were performed against a single task scenario it would be likely to either: 1) fail to receive sufficient feedback to differentiate between policies (as in the disengagement pathology [11]) or, 2) policies would tend to be 'brittle' thus failing to generalize to anything beyond the single task scenario [29]. Conducting policy search over *multiple* training conditions represents an initial requirement for reducing this limitation in policy search and can be of benefit to value function methods as well [43], [59], [50].

Most examples of the value function approach are not evolutionary, as the interaction between states, actions and immediate reward facilitates a wide range of gradient based methods for learning; the class of Temporal Difference (TD) algorithms being particularly popular [26], [56], [3]. Likewise, examples of temporal sequence learning through policy search are almost exclusively based on some form of stochastic search, with evolutionary methods frequently appearing e.g., SANE [38] and NEAT [54]. Naturally, both value function and policy optimization present different advantages/ disadvantages. Indeed it has been suggested that there is a clear separation in the class of tasks for which value function and policy optimization are most appropriate [2].

## 2.2 State and Temporal abstraction in Hierarchical Reinforcement Learning

Temporal abstraction attempts to provide mechanism(s) by which sequences of state-action pairs can be referenced as one 'meta action'. This is generally addressed through the a priori definition of suitable sub-tasks/ goals. Once a sequence of state-action pairs are identified that satisfy a sub-task, the meta action can be referenced as per any other action as long as the conditions for calling the policy associated with the meta action are satisfied [45], [57], [17]. The related concept of state abstraction attempts to discover local regions in the state space for which the same action is optimal, possibly resulting in the identification of redundant state variables [25].

Assuming that mechanisms are available that support either temporal abstraction and/ or state abstraction, then hierarchical RL (HRL) is facilitated [55]. HRL attempts to *scale* RL to more complex task domains by avoiding the requirement to perform value optimization/ policy search entirely in terms of the lowest resolution state–action pairs of the task domain [3]. Instead, by first identifying policies that solve sub-tasks, policy reuse is facilitated.[1] Identifying a set of temporal/ state abstractions enables a change in the level of abstraction necessary to search for effective policies, thus decoupling the search process from the raw enumeration of (atomic) state-action pairs. Naturally, the resulting hierarchical policy may not be strictly optimal, but solutions to more challenging sequential learning tasks can now at least be identified [17].

Early examples of HRL assumed that solutions to the subtasks were learnt independently using standard approaches to value function approximation e.g. TD [3]. Once learnt, hierarchical formulations for reinforcement learning were adopted where these might also assume that a particular subtask hierarchy is already in place [17]. The work of Elfwing *et al*. demonstrated how such hierarchies can be evolved once the subtasks have been a priori defined and 'pre-learnt' using TD algorithms [20]. A second more recent development to the hierarchical reinforcement problem is through the cognitive science literature [43], [8]. Specifically, if one wanted to actually automate the task of subtask/ subgoal identification, the concept of 'intrinsic motivation' or 'curiosity' might provide the basis for developing a suitable framework for balancing the associated exploration versus exploitation tradeoff. Thus, with this in mind, various frameworks have been suggested for supporting 'internal critique' [43], [59], [50]. Such frameworks build on psychological/ neurological motivations – first encountered with the 'actor–critic' architecture of Barto *et al.* [4] – in which predicted reward and action were modelled independently. Moreover, these works also emphasize the significance of providing a suitably broad set of encounters to learn from, in effect a requirement to vary the sample of training scenarios encountered during policy development.

## 2.3 Novelty and Diversity

From an Evolutionary Computation stand point the aforementioned intrinsic motivation (Section 2.2) rewards 'phenotypic novelty' in policy search. There are at least two broad approaches: rewarding diversity as measured over an entire trajectory or rewarding diversity in the ultimate outcome associated with a policy. Specifically, let us consider phenotypic diversity relative to a maze solving task in which we know nothing of the ultimate exit location from the maze i.e., the exit needs to be discovered. In the case of diversity as measured over an entire trajectory, a novelty based fitness function would reward uncorrelated behaviours [32]. Building

[1] Needless to say, it is still necessary to describe all policies through the original domain specific 'atomic' actions.

such a metric, however, is not straightforward [22]. For example, metrics originally designed under the guise of genome analysis might be very effective – they are able to correlate offset subsequences within two candidate behaviours – but come a significant computational overhead. Naturally, such novelty style objectives could be deployed in combination with regular quantitive measures of performance [40], [18].

Conversely, rewarding diversity relative to a behaviour's outcome would imply that after executing an individual for a limited number of interactions with the maze task domain, the outcome is measured in terms of the location reached at the last step. Diversity is then measured relative to this final outcome, with more discounting being performed to solutions that resulted in similar outcomes, as in competitive fitness sharing [47]. We also note that, to date, novelty style performance functions tend to be applied relative to a common start point; thus rewarding the identification of a distribution of behaviours relative to a common frame of reference. Conversely, if the task domain – say configuration of a maze – remains static, but fitness is evaluated over a set of start states, then the goal state remains fixed with policies rewarded that are able to locate the (maze) exit independent of the start location. Both formulations potentially reward diversity in general. Moreover, the ability to generalize to a range of tasks 'post training' is known to be associated with both the diversity of training scenarios encountered [29] and phenotypic novelty [18].

## 2.4 Coevolution, Problem Decomposition and Symbiosis

In this work, a symbiotic model of inheritance will be assumed in which characteristics are potentially acquired from dissimilar species [37], [30]. From the perspective of building complex systems – and GP in particular – this means that should an effective metaphor for symbiosis be identified, then we have a suitable mechanism for incorporating solutions evolved from a previous population into that of the next [15], [23]. Symbiosis is therefore a coevolutionary process in which a hierarchical/ serial as opposed to a lateral/ parallel relationship exists between two independent populations: host and symbiont. The symbionts represent a population of building blocks indexed by individuals from the host population. It is therefore the host that defines a functioning organism. Moreover, neither host nor symbiont can exist without the other. Hierarchical SANE represents an early example of this generic architecture for symbiosis, defining a process for orchestrating the evolution of neural networks [38]. Neurones were symbionts and collections of neurones were defined by hosts to represent the ensuing candidate solution. A similar approach was taken by the Hierarchical ESP algorithm [61]. In this case a hierarchy of three populations exist, the first two representing host–symbiont interactions. Thus, the symbiont population defined neurones as per SANE, however, this time there were as many neurone populations as neurones in the target architecture (a constraint that implies that the number of hidden layer neurones needs a priori defining, also as per SANE). Hosts are represented at an independent population, each host declaring a complete neural network. Finally, there might be multiple neural networks cooperatively coevolved for problem decomposition; thus each pair of host–symbiont populations is repeated in line with (a priori) knowledge of a suitable task decomposition.

ESP also appears to be similar to team based GP frameworks for coevolving programs into team / group behaviour. The work of Bremier and Banzhaf assumed a representation in which a fixed number of programs were evolved concurrently [10]. This enforces gene alignment, but implies that knowledge of the number of (coevolving) programs is known a priori (as per ESP). Moreover, it was also clear that all programs tended to contribute something

to each action, making post training behavioural analysis difficult. More recently the OET framework was proposed where this also makes use of multiple populations from which an a priori specified number of GP individuals are sampled to build a team [58]. Specific programs could then evolve independent behaviours i.e., sub-tasks are evolved with out needing a priori identification.

## 2.5 Bidding metaphors

The proposed framework will make extensive use of bidding as a mechanism for cooperative coevolution. Naturally, this is not the first time that bidding metaphors have appeared in Evolutionary Computation in general. Learning Classifier Systems (LCS) have made extensive use of bidding algorithms. Indeed the earliest examples of our work – a single population of bidding programs [33], [34] – took direct motivation from the Hayek framework [5], [6]. Hayek as a system, however, emphasized the construction of an entire economic system, with taxes, wealth and auctions. This made replication of the Hayek framework difficult to achieve in practice [31].

More generally, research on LCS tends to fall into one of two generic frameworks: Michigan and Pittsburgh. The Michigan approach does not attempt to explicitly group individuals. Instead, the competition to present an action takes place across the entire population. Indeed the earlier bid-based GP framework was previously compared to such an LCS [34]. Conversely, the Pittsburgh approach does explicitly identify sub-sets of individuals. Thus, the bidding process we adopt later is most similar to that of the Michigan-style LCS; whereas we also explicitly identify teams of bidding individuals (the host–symbiont relation), thus simplifying the process of credit assignment, as motived by the Pittsburgh-style LCS.

## 2.6 Layered Learning

Layered Learning assumes a specific task decomposition from which the learnt solution to each subtask is potentially leveraged into building solutions to more complex tasks. The overall goal is to have the solution to simpler 'situation specific' behaviours managed by higher level behaviours. Key components of the generic process for layered learning were summarized by Stone as follows [55]: 1) Hierarchical decomposition of the task. As the overall task is too difficult to be solved directly through RL, simpler tasks need explicitly identifying (lowest level of the hierarchy). A significant amount of a priori knowledge is necessary to provide useful task decompositions. 2) Solutions are learnt independently, in a bottom up process. However, the nature of solutions from a previous layer may also inform the process for designing appropriate training scenarios/ tasks at the next; hence such solutions might impact on the state space features used at a next layer and/ or change the action set utilized at a new layer.

A recent study by Whiteson *et al.* was particularly effective at illustrating the relative significance of building RL policies using Layered Learning (LL) versus Cooperative Coevolution (CC) [61]. A total of three scenarios were considered. 1) Pure LL in which very specific training scenarios were designed to provide a specific skill. Once learnt, a skill is never revisited. 2) Pure CC in which independent populations represent each a priori identified sub-task and training is performed tabula rasa. Sub-tasks are assumed to match the skills from LL, however, one additional policy is introduced – as required by CC – representing a behaviour that indicates when to deploy which subtask. 3) Concurrent LL in which both LL and CC are combined. To do so, specific skills are learnt independently, as per LL. However the last task, that of the switching

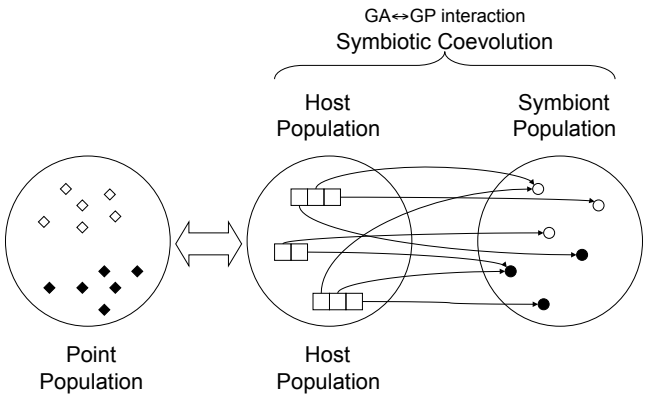



Figure 1: Generic architecture of Symbiotic Bid-Based GP (SBB). A *point population* represents the subset of training scenarios (sampled from the task domain) over which a training epoch is performed. The *host population* conducts a combinatorial search for the best symbiont partnerships; whereas the *symbiont population* contains the bid-based GP individuals who attempt to learn a good context for their corresponding actions.

policy – and specific to CC – is learnt under a tabula rasa training scenario with modification of the previously learnt skill specific policies also permitted. RL was conducted both with and without the aid of an a priori decomposition of sub-tasks into a hierarchal decision tree.

The hierarchical decomposition was fundamental to providing the strongest solutions; within this context LL was actually decremental. Conversely, removing the sub-task hierarchy resulted in very poor performance under CC. Conversely, folding the hierarchical sub-task decomposition back in through the use of LL again resulted in coevolutionary methods performing most effectively, albeit not matching the performance of the a priori defined sub-task hierarchy. From the perspective of state and temporal abstraction, both LL and CC required appropriate identification of state variables and sub-task specific actions. Our interest is on providing a process by which appropriate state/ temporal abstractions can be discovered and then deployed within a hierarchical policy.

## 2.7 Discussion

In this work we assume a policy optimization approach in keeping with the evolutionary basis of the framework. The principle contribution of this work is therefore associated with the formulation of a consistent approach for addressing scaling in the temporal sequence learning problem from the perspective of Evolutionary Computation. Any architecture applied to this problem should be capable of state and temporal abstraction. With regards to state abstraction, we make the case that this is a natural artifact of assuming a program based representation, there being a strong bias towards attribute sampling in GP [19], [36]. In order to support temporal abstraction we take this to imply a requirement for coevolution. However, in order for coevolution to be effective under temporal sequence learning, it is very important to be able to explicitly learn the context for applying an action. Given that the same action can have multiple contexts, we maintain that the action and the bidding activity used to identify its context should explicitly appear in the same symbiont. From the perspective of the symbiont population, multiple individuals can have the same action, but bidding policies should be unique.

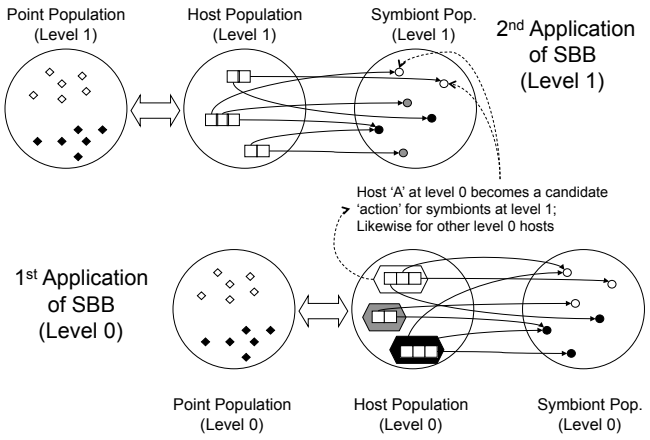



Figure 2: SBB architecture for evolving hierarchical policies. Level 0 represents the special case in which the set of actions that symbionts may assume are domain specific i.e., 'atomic' actions. Thereafter reapplying the SBB algorithm results in symbionts at level $i$ using the hosts evolved at level $i-1$ as the candidate actions. Thus host–symbiont policies at later layers are learning to deploy sub-tasks (meta actions) identified by hosts at earlier layers in potentially new contexts.

The above review of coevolution emphasized the use of bidding metaphors under various guises: switching policies in CC and Michigan style LCS. However, current approaches rely on either the optimal number of cooperating individuals to be defined a priori – as in CC – or assume that the relevant bidding behaviours will emerge across an entire population. Conversely, GP bidding behaviours evolved under a Michigan LCS architecture tend to be continuously disrupted by their own offspring [35]. Thus, separating the process of identifying who should be in a cooperative relationship (the host population) from the pool of available action/ bidding behaviours (the symbiont population) enables a much clearer path to credit assignment, Figure 1. Moreover, members of the host population should assume a variable length representation, thus letting the number of cooperating individuals 'float'. Finally, host content may overlap with applicable symbionts potentially appearing in multiple hosts.

Conversely, competitive fitness sharing applied to the host population provides a natural mechanism for novelty/ problem decomposition independent from the concept of cooperative coevolution (the latter only taking place within host individuals). This then avoids the requirement for a priori specifying the number of sub-tasks or, for that matter, the level of abstraction associated with policies defined by hosts. Each host therefore represents a candidate temporal abstraction. This analogy will be particularly effective when the host-symbiont pairing is applied hierarchically: symbionts at level 0 assume the domain specific 'atomic' actions, Figure 2. Thereafter reapplication of the host–symbiont evolutionary cycle results in symbionts assuming actions sampled from the previously evolved hosts. Increasing abstraction occurs because the bidding of symbionts interleaves the deployment of hosts evolved a earlier levels.

As indicated in Section 2.1, policy search has the potential disadvantage (relative to value function optimization) of not being able to explicitly conduct exploration *during* a training episode. Thus, if the initial condition used to evaluate a policy is too difficult for any of the current policies to solve, then policy search might face a pathological disadvantage [2]. Schemes for addressing this might include (competitively) co-evolving a set of training scenarios [11], [16], designing more informative fitness functions, and/ or crafting a sequence of training scenarios as in layered learning. In this work, we assume a tabula rasa basis for sampling a set of training scenarios (c.f., the point population) in conjunction with a real-valued fitness func-

tion. Such a combination is known to avoid generic policy learner pathologies, for example attempting to solve the pole-balance problem from a single cart pole configuration that is already in the solution configuration [2]. Moreover, by assuming similar processes for generating training (point population) and test scenarios we provide a consistent approach for assessing generalization [29].

Finally, we note that as per layered learning, the process for defining hierarchical policies with SBB is bottom up. However, each application is also coevolutionary, with hosts conducting a search for unique behaviours – care of competitive fitness sharing – while symbionts are engaged in a bid-based coevolutionary relationship to identify 'good' combinations of actions within a host, or candidate temporal/ state abstraction. Unlike layered learning a tabula rasa presentation of the task domain is assumed.

# 3 Symbiotic Bid-Based GP

Section 2.7 made the case for the generic host–symbiont architecture of Figure 1. This represents the first layer of symbiosis. The resulting hosts are either capable of solving the entire task or provide state/ temporal abstractions appropriate for seeding the next set of symbiont actions, Figure 2. In the following development of the proposed Symbiotic Bid-Based (SBB) framework we first provide a system view (Section 3.1). The specific processes for population initialization, selection and variation operators, and competitive fitness sharing are then developed (Sections 3.2 to 3.5). The process by which hierarchical policies are evolved through SBB is then established in Section 3.6.

## 3.1 Generic host–symbiont architecture

A single level of the SBB framework consists of three independent populations, Figure 1, in which candidate solutions – sets of cooperating programs – are identified by the symbiotic relation between host and symbiont populations. Fitness evaluation is only conducted at the host population, following evaluation of each host against the content of the point population. Points therefore constitute a sample of training cases. A breeder style model of evolution is assumed, Algorithm 1, in which a fixed number of points and hosts are added and replaced at each generation.

### 3.1.1 Symbiont Population

Each member of the symbiont population assumes a bid-based GP representation [33]. Thus, a symbiont, *sym*, is described by a tuple $\langle a, p \rangle$ in which '$a$' is an atomic action appropriate to the application domain selected from an a priori identified set of legal discrete actions, or $a \in \{a_0, ..., a_{A-1}\}$; where $A$ is the number of actions. The corresponding symbiont program '$p$' defines a bidding strategy for the symbiont. Thus, assuming a linear GP representation[2] [9], executing the symbiont program results in a real-valued numerical outcome at the output register, $R[0]$. This is mapped into the unit interval to provide the bid from each symbiont care of the sigmoid operator, or $(1 + \exp(-R[0]))^{-1}$. The instruction set takes the form of a simple register-register instructions defined in the form of: 1) two argument instructions

[2]No specific significance is attached to this choice of representation, although this does facilitate the efficient filtering of introns during fitness evaluation [9].

i.e., $R[x] \leftarrow R[x]\langle op_2 \rangle R[y]$ in which $op_2 \in \{+, -, \div, \times\}$; 2) single argument instructions i.e., $R[x] \leftarrow \langle op_1 \rangle (R[y])$ in which $op_1 \in \{\cos, \ln, \exp\}$; or 3) a conditional statement of the form "IF $(R[x] < R[y])$ THEN $(R[x] \leftarrow -R[x])$." In addition two instruction types are supported in which the $R[y]$ register reference can be an index to a state variable or a register reference ($R[x]$ is always a register reference). Appendix 10.1 provides an illustration for how a symbiont would be evaluated under such a representation.

### 3.1.2 Host population

Naturally, as any one symbiont can only represent a single action, the role of each host individual is to find combinations of symbionts that cooperate effectively. Each host therefore identifies a subset of $[2, ..., \omega]$ symbionts. Fitness evaluation for a specific host under an RL task domain consists of the following stepwise process: 1) w.r.t. current state of the task domain, $t_s$, present the state variables, $\vec{s}(t_s)$. 2) Execute all symbionts identified by the host under the current state of the environment, $\vec{s}(t_s)$, to identify the corresponding symbiont bids. 3) Select the host symbiont with maximum bid (Step 2.c.ii(A), Algorithm 1), thus identifying the 'winning' symbiont. 4) Present the action from the winning symbiont to the task domain (Step 2.c.ii(B), Algorithm 1) and update any task state variables accordingly, or $\vec{s}(t+1)$. Each host therefore represents a candidate policy. As a host interacts with the task domain different host symbionts will likely represent the winning bid and present their action. Each host will receive a reward (Step 2.c.iii, Algorithm 1) as per the generic process of policy search, and will then be discounted under competitive fitness sharing to encourage diversity/ novelty (Section 3.3).

---

**Algorithm 1** The SBB training algorithm under an RL context. $P^t$, $H^t$, and $S^t$ refer to the point, host, and symbiont populations at time $t$ and $t_{max}$ is the maximum number of generations; $t_s$ is the current step of the episodic interaction between policy and task; *world* denotes the task domain to which atomic actions $a$ are applied and corresponding state variables, $\vec{s}(\cdot)$, returned.

---

1. Initialize $< H^t, P^t, S^t : t = 0 >$
2. While $t < t_{max}$
   (a) Add $P_{gap}$ points;
   (b) Add $H_{gap}$ hosts; (and supporting symbionts)
   (c) $\forall h_i \in H^t; \forall p_k \in P^t$
      i. Initialize $t_s = 0; \vec{s}(t_s) \leftarrow p_k$;
      ii. While !*stop_criteria*
         A. $sym^* = [\arg_{sym \in h_i} \max(sym.(bid) \leftarrow \vec{s}(t_s))]$;
         B. $\vec{s}(t_s + 1) \leftarrow world \leftarrow sym^*.(a)$;
      iii. Return reward specific to point–host combination: $G(h_i, p_k)$
   (d) Delete $P_{gap}$ points;
   (e) Delete $H_{gap}$ hosts;
   (f) $t++$;

---

### 3.1.3 Point Population

A tabula rasa approach is assumed for sampling the task domain during training. Thus, it is implicitly assumed that a sufficient basis for maintaining engagement between point and host populations is provided by the combination of: 1) sampling a fixed number of different configurations of the world per generation, and; 2) provision of a suitably informative fitness function when a task stop condition is encountered. The first point is supported by having a point population sample initial conditions from the task domain against which evaluation of candidate policies takes place at each generation. The second point is assumed to be supported by utilizing a real-valued fitness function that describes quantitatively how close the current solution is to the task goal. This is not to say that actively decomposing training tasks into a sequence of scenarios of increasing difficulty has a negative influence on policy search; just that this entire domain of research lies outside the preview of the this contribution.

## 3.2 Initialization

A fixed number of points and host–symbionts are initially provided to 'seed' the breeder style of evolution (Step 1, Algorithm 1). Thereafter a 'gap size' number of points and hosts are added / deleted at each generation (Steps 2a, 2d and 2b, 2e for points and hosts respectively; Algorithm 1).

### 3.2.1 Point population

initialization follows a simple stochastic sampling algorithm in which state variables from the task domain are sampled to provide training instances at the initial generation, after which a fixed number of points are replaced at each generation. The process of sampling will naturally be limited by constraints from the task domain regarding legal values for state variables and, as such, will be presented within the context of the two test problems (Section 4). Other explicitly competitive coevolutionary interactions could naturally be adopted e.g., [11], [16]. However, as previously mentioned, we focus on the contribution of SBB rather than the relative importance of training case scenario selection.

### 3.2.2 Host and symbiont populations

observe a serial dependency, or group selection of symbionts by a host (Figure 1). Therefore, from the perspective of initialization, the content of both (independent) populations is established through the following stepwise process:

1. Initialize two new symbionts with *different* actions. Actions are chosen with uniform probability from the set of atomic actions associated with the task domain in question. Symbiont program initialization follows from the representation assumed for GP and also includes a test for 'diversity' in the bidding behaviour (Section 3.5);

2. Initialize a corresponding host with pointers to the two Symbionts established in Step 1;

3. Repeat Steps 1 and 2 until $H_{size} - H_{gap}$ hosts have been created. The remaining $H_{gap}$ hosts will be added at Step 2b of Algorithm 1 through the action of the Host–Symbiont variation operator;

4. For each host choose an initial host size over the interval $[2, ..., \omega]$ and proceed to add indexes from the host to symbionts *currently* existing in the symbiont population, until the initial host size limit is reached. Such symbiont indexes are chosen with uniform probability. Naturally, the content of the symbiont population does not change during this process;

Host population initialization is therefore a two stage process with Step 2 establishing the basis for minimal host content and Step 4 building additional diversity; whereas symbiont initialization is completed in Step 1. In particular, the first stage ensures that each host will satisfy a minimal requirement for describing a non-trivial behaviour i.e., hosts are not initialized with symbionts with the same action. Conversely, the second stage (represented by Step 4) does not add to the initial content of the symbiont population, but concentrates on providing more diversity to the symbiont combinations supported by members of the host population. Symbionts will now naturally appear in multiple hosts. Note that there is no attempt made to 'bias' the symbionts indexed by a host with regards to 'covering' the available set of actions. Indeed, it is this freedom that supports state abstraction, with the relative distribution of actions appearing as an artifact of evolution along with host size.

## 3.3 Selection and Replacement

### 3.3.1 Point population

At each generation $P_{gap}$ points are removed with uniform probability (Step 2d, Algorithm 1) and a corresponding number of new points are introduced (Step 2a, Algorithm 1). As a consequence initialization need only introduce $P_{size} - P_{gap}$ points (Step 1, Algorithm 1).

### 3.3.2 Host population

At each generation a fixed number of host individuals, $H_{gap}$, are deleted/ added (Steps 2e and 2b of Algorithm 1 respectively). However, in order to also promote diversity in the host population behaviours, we assume a competitive fitness sharing formulation [47]. Thus, shared fitness, $s_i$ of host $h_i$ has 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 an appropriate domain dependent reward function expressing the quality of solution $h_i$ on test point $p_k$.[3] Thus, for point $p_k$ the shared fitness score $s_i$ re-weights the reward that host $h_i$ receives on $p_k$ relative to the reward on the same point as received by all hosts.

Once the shared score for each host is calculated, the $H_{gap}$ lowest ranked hosts are removed (Step 2e of Algorithm 1). Any symbionts that are no longer indexed by hosts are considered ineffective and therefore also deleted. Thus, the symbiont population size may dynamically vary, with variation operators having the capacity to add additional symbionts, whereas the point and host populations are of a fixed size.

[3]Sections 5.1 and 5.2 detail $G(h_i, p_k)$ for the truck reversal and Acrobot domains respectively.

## 3.4 Hierarchical Host–Symbiont variation operator

Symbiosis as a coevolutionary process is explicitly hierarchical. The host population conducting a combinatorial search for appropriate symbiont partnerships under a variable length representation. Symbionts learn to cooperate – that is identify a specialization/ context – within a host. As such variation operators should also be applied under a similar hierarchical framework to avoid disrupting these linkages. Specifically, although crossover at the host population (sexual reproduction) is nominally possible, and was indeed used in earlier instances of SBB [35], it also tends to result in common symbiont combinations quickly dominating the host population or premature convergence. Moreover, there is no context for the exchange of genotypic material between pairs of symbionts. Thus, the following asexual hierarchical model of reproduction is assumed, in keeping with the underlying symbiotic motivation for the framework:

1. Select parent host for cloning from the $H_{size} - H_{gap}$ hosts available at generation $t$ with uniform probability (Step 2b, Algorithm 1);

2. Stochastically remove symbionts currently indexed by the host. The process supporting this assumes at least one symbiont will always be removed (providing that the host has a minimum of two symbionts), and thereafter applies a simple multiplicative weight to the probability of removing further symbionts, Algorithm 2;

3. Symmetrical to the process for removing symbionts, one symbiont is always added, Algorithm 3, with a multiplicative weight reducing the likelihood of adding further symbionts. Naturally, the process is limited to symbionts currently existing in the symbiont population at the outset of the current generation and subjected to a max host size limit ($\omega$);[4]

4. For each symbiont, test for introducing variation with probability ($p_{mm}$). On testing true, the symbiont is first cloned and an index unique to the cloned symbiont introduced i.e., the reference to the original symbiont is replaced. The bid component of the symbiont is then modified through the repeated application of a set of GP mutation operators (Section 3.5). The symbiont action can also be reinitialized at this point, again with a uniform p.d.f. ($p_{mn}$).[5] Note that operators testing for symbiont modification are applied until at least one symbiont is modified per host reproduction.

In summary, operators 2 and 3 introduce variation in the indexes that exist in the host population, whereas operator 4 extends variation down to the symbiont population. In the latter case, the modified symbiont receives a unique index, thus ensuring that other hosts using the original copy of the symbiont remain unaffected. It is this process which maintains the diversity of the symbiont population without disrupting host–symbiont contexts already established.

## 3.5 Symbiont specific variation operators

The set of mutation operators used to modify cloned symbionts – or the forth host–symbiont variation operator (Section 3.4) – have the following form: 1) delete an instruction with uniform

[4]Naturally, the symbiont population size may vary between $2 \times M_{size}$ and $\omega \times M_{size}$ individuals i.e., each host can contain between 2 and $\omega$ symbionts which, if all indexes are unique, represents the upper bound on symbiont population size.

[5]Symbiont initialization is the only point at which an action is defined, thereafter the symbiont action remains unchanged, thus placing most emphasis on symbiont context learning.

**Algorithm 2** Removing Symbiont indexes from a Host (Section 3.4): 'host.size' denotes the current count of symbionts referenced by the host in question; 'host.symbiont' is the set of symbiont indexes identified by the host; 'rand()' is a random number generator over the interval $[0, 1)$ with uniform p.d.f.; whereas 'rand(host.symbiont)' selects one symbiont index from the host with uniform probability.

1. b = 1;
2. While $b >$ rand() AND host.size $> 2$
   (a) delete(rand(host.symbiont));
   (b) $b = b \times p_{md}$;
   (c) host.size = host.size $-1$;

**Algorithm 3** Adding Symbiont indexes to a Host (Section 3.4): 'rand(symbiont_pop)' selects a symbiont from the content of the symbiont population (as available at the beginning of the generation) with uniform probability.

1. b = 1;
2. While $b >$ rand() AND host.size $< \omega$
   (a) host.symbiont $\leftarrow$ add(rand(symbiont_pop));
   (b) $b = b \times p_{ma}$;
   (c) host.size = host.size $+1$;

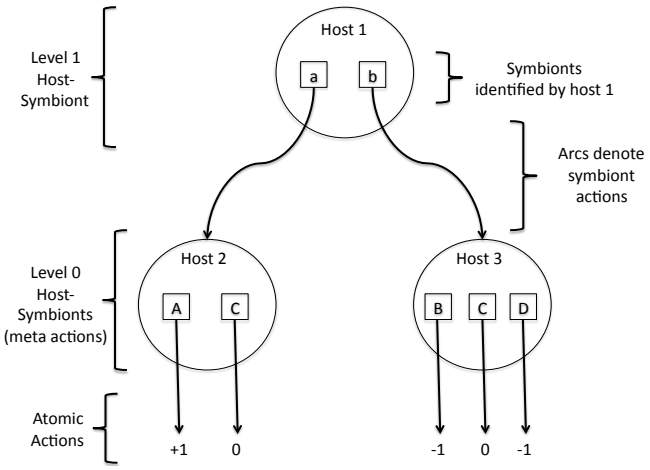



Figure 3: Visualizing a Hierarchical SBB as a tree. One host from a level 1 population is expressed. The level 1 host comprises of two symbionts 'a' and 'b'. These have as actions host '2' and host '3' from the previous application of SBB, or level 0. Symbionts of host 2 and 3 are identified in upper case and define actions in terms of task specific atomic actions. The tree only reflects a single level 1 host. Other level 1 hosts might also use host 2 and/ or 3.

probability ($p_{delete}$); 2) add an instruction with uniform probability ($p_{add}$); 3) change a current instruction/ register reference with uniform probability ($p_{mutate}$); and 4) interchange/ swap two existing instructions with uniform probability ($p_{swap}$). Following the introduction of a new symbiont a behavioural test is applied relative to a static sample of 50 parameterizations from the application domain. Specifically, if a candidate symbiont returns bid values over all of the static sample to within $10^{-4}$ of another symbiont, the mutation step is repeated. Naturally, the static sample against which this test is made are initial states sampled form the application domain and not complete policies. Moreover, given the static nature of the sample, outcomes from symbionts currently existing can be cached, again reducing the computational overhead of this process.

## 3.6 Evolving hierarchical policies under SBB

Symbiosis as defined by the above process is naturally hierarchical, mirroring the biological equivalent and providing the opportunity for incremental complexification [23]. Specifically, under the domain of temporal sequence learning, fitness sharing at the host population provides a mechanism for discovering *multiple* candidate temporal/ state abstractions for later use or 'meta actions'. In order to deploy these meta actions, a requirement exists for learning under what conditions to recommend their deployment. This is more general than merely dividing the state space into different regions and deploying a previously learnt policy. Instead we wish to establish new policies (meta actions) as a *mixture* of those previously identified.

The process can be visualized as that of building a form of decision tree. The population of hosts under evolution represent candidate 'root nodes' in the tree, Figure 3. Deploying the SBB algorithm as detailed in Sections 3.1 to 3.5 adds a new 'layer' to the tree, with policy trees being added level-by-level from the bottom up. Hosts represent nodes (c.f., meta actions) in the decision tree. The branching factor of each node is defined by the symbiont count of each host. The bidding process between symbionts of the same host establishes which arc is descended next. Moreover, the action of each symbiont defines the arc target in terms of the

available nodes at the next level down the tree. The special case of evolving the first level of the hierarchy implies that symbiont arcs (actions) are leaves and therefore take the form of the task domain atomic actions. Naturally, the set of hosts indexed at any one level of the hierarchy is left to evolve, but is limited by the diversity of the hosts as left by the previous cycle of evolution.

Evolving a new policy at level $l$ implies that a host at level $l$ is able to switch between previously discovered meta actions (hosts at level $l-1$) to describe a new policy, Figure 2. The only requirement necessary in order to support the evolution of a new host–symbiont level (and therefore hierarchical policies) is the definition of the set of actions symbionts may assume at the new level. Specifically, symbionts *after* level 0 assume an action set defined by the population of hosts most recently evolved, Figure 2; whereas symbionts *at* level 0 assume the atomic actions specific to the task domain in question. Thus, denoting the content of the previously evolved level $l-1$ host population by the set: $H(l-1)$, then the set of legal actions for the symbionts at level $l$ is merely $a \in H(l-1)$. Evaluating a host $i$ at level $l$ (or $h_i^l$) now has the form:

1. For the level $l$ host under policy evaluation, identify the corresponding symbiont bids: $\forall sym \in h_i^l : sym.(bid) \leftarrow \vec{s}(t_s)$;
   where $sym$ denotes symbiont and $sym.(bid)$ is the corresponding bid value w.r.t. state variable $\vec{s}(t_s)$ (see Appendix 10.1 for evaluation of a symbiont).
2. Select the level$(l)$ host symbiont with maximum bid: $sym^* = \arg_{sym \in h_i^l} \max[sym.(bid)]$ thus identifying the 'winning' symbiont of host $h_i^l$;
3. IF $l == 0$ THEN Step (4) ELSE
   (a) Lookup the level$(l-1)$ host indexed by the 'winning' symbiont $sym^*$ from Step (2);
   (b) Update level pointer: $l = l - 1$;
   (c) Update host pointer: $h_i^l \leftarrow sym^*.(a)$;
   (d) RETURN to Step (1);
4. Present the atomic action from the winning symbiont to the task domain and update any state variables accordingly i.e., $\vec{s}(t_s+1) \leftarrow world \leftarrow sym^*.(a)$;

Naturally, this process replaces Steps 2(c)iiA and 2(c)iiB of Algorithm 1. The hierarchical application of SBB will hereafter be referred to as 'hierarchical SBB'. For completeness, Appendix 10.2 presents a 'worked example' in which a hierarchical SBB solution is evaluated relative to the present state of a task domain.

# 4 Task Domains

Two 'control style' task domains are considered for benchmarking purposes in order to illustrate the utility of hierarchical SBB. The task domains were selected on the basis of their difficulty – without which there would be no need to resort to a hierarchical approach – and the ability to attach simple visual summaries of solutions post training. Specifically, the task domains share the property that although specific configurations might be 'solved' without recourse to a hierarchy, in order to truly demonstrate 'mastery' of the task the policy learner

needs to be able to generalize across multiple test configurations [29] (as discussed in Section 2).

## 4.1 Truck Reversal Domain

### 4.1.1 Task domain

takes the form of reversing a semi-cab [1], but with an additional wall obstacle. The introduction of the wall makes the task significantly more difficult than without as, unlike maze navigation tasks, there are no sensors for obstacle detection. Thus, relative to the original formulation of the problem the following additional properties exist:

1. A wall obstacle is present with upper-right corner at $(45, 50)$ and lower-right corner at $(55, -50)$. Thus, steering strategies that are effective on one side of the wall will not generalize to the other side of the wall. The wall, in addition to the initial direction of the cab–semi, therefore renders the problem deceptive;
2. Enforce constraints on legal behaviours such as jackknifing and colliding with obstacles and;
3. Let the training configurations be defined stochastically by the content of the point population (c.f., the tabula rasa assumption).

### 4.1.2 Atomic action and state variables

During evolution the point population specify starting configurations of the cab and semi; whereas the hosts assume responsibility for providing the steering behaviour to return the semi-cab back to the origin i.e., a single goal state. The **state variables** provided at each time step by the task domain takes the form of cartesian coordinates identifying the end of the semi, $(x, y)$, the angle of the cab, $\theta_c$, and the semi, $\theta_s$ (Figure 4). As such, this is similar to the information provided by a GPS; where the goal is to reverse the cab and semi back to within some tolerance of the origin, but *without* any local information. The single **atomic action** $a$ that each symbiont at level o may assume were selected from the set $\{0^\circ, \pm 30^\circ\}$, and a training episode was terminated when: (1) the back of the semi crossed the $y$-axis, (2) the cab–semi jackknifed, (3) the cab–semi could not return to the origin within some maximum number of time steps,[6] or (4) the back of the semi collided with the wall.

### 4.1.3 Point population routines

The process for point initialization and generation takes the form of a simple stochastic model, as follows:

1. Select $x$ and $y$-coordinates denoting the end of the semi with uniform p.d.f. under the corresponding range limits of $(0, 100)$ and $(-100, 100)$;
2. Should the $(x, y)$-coordinate fall within the boundary of the wall obstacle, repeat step 1;

[6] Enforced by setting a limit of *Max. steps* where this is estimated as: steps used so far *plus* steps 'as the crow flies' from the current location to the origin $<$ *Max. steps*.

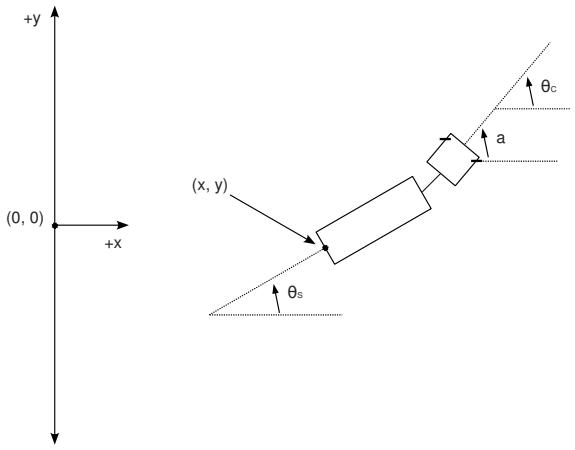



Figure 4: Truck Reversal state variables. The controller is supplied with cartesian coordinates denoting the end of the semi, $(x, y)$, the angle of the cab, $\theta_c$, and the semi, $\theta_s$. Not shown is the global 'field' in which the semi–cab may begin or the obstacle in the middle of the field. These global properties can be inferred from the summary of the test cases in Figure 5.

3. Selection $\theta_s$ with uniform p.d.f. under the interval $(-\pi, \pi]$; and,
4. Let $\theta_c = \theta_s$.

#### 4.1.4 Previous Results under RL

Previous results under the original (obstacle free) formulation of the truck reversal problem [42] emphasized the potential utility for problem decomposition [21], [24], albeit with hand designed task decompositions. The truck reversal problem has attracted widespread interest from Fuzzy System (FS) approaches to RL. The general goal of which is either to make use of a priori knowledge to provide seed control policies in the form of fuzzy antecedent–consequent rules or to learn these rules outright from interactions with the task domain i.e., knowledge driven RL versus purely data driven RL. In both cases, however, the resulting solution should be able to both generalize to a wide range of test cases while also providing a parsimonious solution. Recurring themes include the complexity of solutions based on expert rules [46] versus the sensitivity of data driven approaches to the training cases employed [44]; the latter under an Evolutionary Computation methodology for FS design. Specifically, the sensitivity to training cases of FS for RL has resulted in the utility of hand crafted subsets of training scenarios [44], as opposed to the purely stochastic sampling scheme adopted in this work. Specific results to date include solutions with 6 rules under 4 semi–cab test configurations [13], and 23 (versus 49) rules capable of solving 3636 test configurations [44].[7] From the perspective of previous results from non-FL methods, Koza evolved solutions relative to 8 hand crafted cab–semi configurations and did not report performance under an independent set of test conditions [28]. Evolving neural networks has also been considered, although again this appears to have taken place relative to hand designed subsets of training scenarios [48].

[7]The 49 rule solution was identified by the authors of [44] reapplying the framework described in [12].

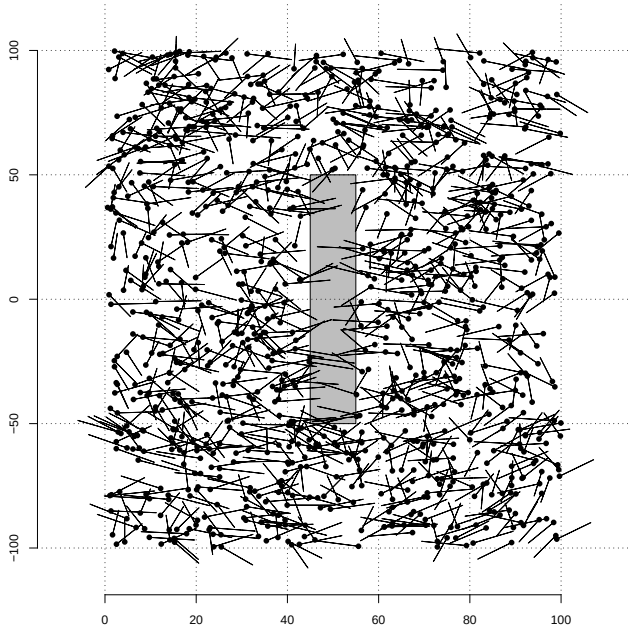



Figure 5: Summary of the 1 000 initial conditions for post training testing evaluation of solutions from the Truck Reversal domain. The goal is to return the truck from each of the initial configurations to the origin without triggering a fault condition. The rectangle at the centre of the world represents the wall obstacle. Constraints are enforced on initial configurations to require that the $(x, y)$ 'pinhead' location denoting the end of the semi lie outside the obstacle.

### 4.1.5 Concluding comments

This work adopts the original formulation of the track reversal domain in terms of the dynamics of the semi–cab [42] and adds the wall obstacle. This modification tends to result in rather long training interactions (up to 600 interactions per training point), hence both the Euler Method for updating the differential equations determining the non-linear plant and the controller assume updates at a rate of 1 meter. However, the relatively slow dynamics of the task domain appear to support such a modification. The composition of the training scenarios is completely determined by the content of the point population, whereas post training performance will be assessed in terms of 1 000 semi-cab configurations, Figure 5; there is naturally no guarantee that such configurations will be encountered during training.

## 4.2 Handstand Acrobot

### 4.2.1 Task domain

The Acrobot problem represents a domain in which a two link 'body' has a control force applied to the 'waist' or middle link (Figure 6) [51], [56]; whereas the hands remain fixed at an axel and the feet are not fixed to anything and are therefore 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. An alternate definition for the goal state requires that the Acrobot perform a *handstand*. 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

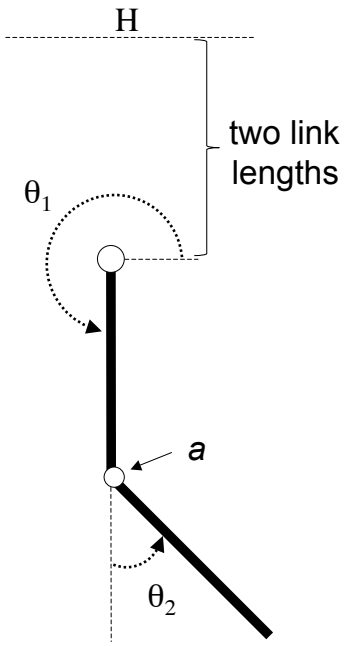



Figure 6: Acrobot 'handstand' domain. The controller is provided with state variable information in the form of the angular position $\theta_1, \theta_2$ and velocities $\dot{\theta}_1, \dot{\theta}_2$. The controller supplies the torque $a$ to the acrobot 'waist,' the feet are at the free end of the second link whereas the 'hands' are connected to the rotating bar of the first link. The height formulation of the task sets 'H' at one link above the axis of $\theta_1$ whereas this rises to two links in the case of the handstand task. The force the control policy can apply is limited to one of three values: $a \in \{0, \pm 1\}$ Nm.

we concentrate on the more difficult handstand task.

#### 4.2.2 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 [41]); or increased the control force beyond the $\pm 1$ Nm permitted (e.g., real-valued forces over the interval $\pm 30$ Nm [62], [27] or $\pm 2$ Nm [14]). The latter decreasing the need to identify suitable strategies for 'pumping' the waist to attain sufficient energy to achieve the swing-up trajectory, as well as making the removal of energy much easier as the vertical configuration is approached; thus significantly simplifying the nature of the task at hand.

#### 4.2.3 Previous result under exhaustive search

It is also possible to make extensive use of domain specific heuristics to simplify the problem to the extent that an $A^*$ search represents a feasible approach for discovering policies [7]. Specifically, Boone utilized information 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 exceptional 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 reinforcement learning algorithm (e.g., [56]), 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 (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.

#### 4.2.4 Atomic action and state variables

In this work we adopt the outlook assumed by Boone in which the basic goal is to identify control policies that solve the handstand task while enforcing the 1 Nm magnitude limitation on the control force [7] (Figure 6 and Table 2 identify the corresponding **atomic actions**). Also as per Boone, we are not interested in deriving a balancing controller as simple (first order) balancing controllers may be derived analytically when the Acrobot enters a 'capture region' (e.g., [51], [27]). The capture region is defined in terms of target values for the (unstable) vertical equilibrium position (Table 2). **State variables** have the form of angular position $\theta_1, \theta_2$ and velocities $\dot{\theta}_1, \dot{\theta}_2$ (Figure 6).

#### 4.2.5 Performance evaluation

Classically, previous research has concentrated on the 'swing-up' formulation of the handstand in which the ultimate goal is to provide a controller capable of swinging the acrobot from the at rest stable equilibrium position (zero velocity with body of the acrobot vertically aligned with feet under the hands) to the feet being within its capture region (see Section 4.2.4). Such an approach facilitates direct comparison with what still represents the best solution to the swing-up task (82 steps). A second experiment will then be conducted to explicitly measure generalization properties, where the generalization test takes the form of sampling the circumference of a circle established by setting the radii about the Acrobot 'hands' to twice the link length and sampling 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 4.2.6). Finally, the higher velocities that govern the dynamics of the Acrobot model are such that the simulation engine is more elaborate than required for the truck reversal domain (see Section 4.2.7).

#### 4.2.6 Point population routines

Acrobot configurations defined by the point population always assume alignment of the two limbs ($\theta_2 = 0$, Figure 6) and zero initial velocity. Points differ with respect to their $\theta_1$ 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 task. 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 $\theta_1$ with sign selected stochastically.

At initialization, $p_{pf}$ points are again set to the swing-up position, with the remainder initialized with $\theta_1$ 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.

#### 4.2.7 Concluding comments

A complete description of the system dynamics may be found in several previous works (e.g., [51], [56]), 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.

# 5 Parameterization

Following from the problem definitions provided above, domain specific decisions are necessary to define reward functions and a common set of learning parameters. Moreover, we will also establish a baseline for the relative difficulty of the task domains in terms of the widely used NEAT algorithm [54] for evolving neural networks.

## 5.1 Reward Function – Truck Reversal Domain

The SBB algorithm as described defines fitness under a generic model of fitness sharing (Equation (1), Section 3.3), which then requires a domain specific reward function. As per previous approaches to the truck reversal task, a real-valued reward function is assumed. Thus, on reaching one of the stop conditions as identified in Section 4.1.2, the corresponding final values for $x, y$, and $\theta_s$ are used to provide the reward function:

$$G(h_i, p_k) = \frac{1}{\sqrt{x^2 + y^2 + \theta_s^2 + 1}} \tag{2}$$

Naturally, host trajectories that result in final configurations closer to the overall goal state receive a larger reward. However, for the special case of a jackknife condition, the reward is always zero. This avoids preferring strategies that jack-knife near the origin.

## 5.2 Reward Function – Acrobot Handstand Domain

Two approaches will be considered for designing a reward function under the Acrobot domain. In the first case a lexicographic function will be utilized in which knowledge of the capture region and a bias to finding short swing-up trajectories will be assumed. In the second formulation no such information will be assumed.

#### 5.2.1 Lexicographic formulation

establishes a hierarchy of properties within a single scalar fitness function. Thus, constraints on each of the measured properties are sequentially satisfied in order. Three properties are

considered here – position, velocity and time – with position and velocity enforcing constraints associated with the capture region employed by Boone (Section 4.2) and provides a measure of solution quality; summarized as follows,

1. Position ($F_p$): expresses the need to first reach the capture region and was motivated by the approach taken by Munos and Moore [41] or,

$$\begin{array}{cl} \frac{\sin(\theta_1)+\sin(\theta_1+\theta_2)+2}{4} & \leftarrow \text{IF } (\theta_1+\theta_2 \notin \{\pi/2 \pm 0.3\} \\ & \text{OR } \theta_1 \notin \{\pi/2 \pm 0.3\}) \\ 1 & \leftarrow \text{otherwise} \end{array} \tag{3}$$

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

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

3. Duration ($F_d$): distinguishes between the quality of solutions based on the number of steps, $t_s$, necessary to reach the capture region, given a maximum duration to locate a solution, $D_{max}$ (Table 2) or,

$$\begin{array}{cl} \frac{t_s}{D_{max}} & \leftarrow \text{IF } (F_v = 1) \\ 0 & \leftarrow \text{otherwise} \end{array} \tag{5}$$

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,

$$G(h_i, p_k) = \frac{1}{3} \max_{t_s} (F_p(t_s) + F_v(t_s) + F_d(t_s)) \tag{6}$$

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., [27], [14], [7]).

### 5.2.2 Simple formulation

merely rewards policies that minimize the distance between desired height, $H$, and the Acrobot 'feet', $d$, as encountered over the duration of an interaction between host and point; as follows,

$$G(h_i, p_k) = \min_{t_s} (|H - d(t_s)|) \tag{7}$$

where $|\cdot|$ is the magnitude operator. Should the requirements for the capture region be satisfied then the current evaluation halts, otherwise the evaluation continues for the duration of the simulation ($0 \leq t_s \leq D_{max} - 1$) and returns the lowest cost encountered. Success with the Acrobot task under such a simplistic cost function have not to date been reported.

Table 1: Parameterization at Host (GA) and Symbiont (GP) 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 | $\omega$ | 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 |

## 5.3 SBB Parameterization

Parameterization of SBB follows exactly the same form as used in the earlier supervised learning (classification task domain) benchmark study [36], and is summarized in Table 1. Both task domains also require parameterization, as per Table 2. Needless to say, no claims are made regarding the optimality of these parameters. In addition, under the initial truck reversal domain a single layer SBB parameterization will also be considered in which all parameters remain the same other than the generation limit, which is set to reflect the total number of generations performed under the hierarchical SBB configuration. This latter scenario enables us to verify the relative contribution of recursion in SBB with all other factors kept constant.

## 5.4 Deploying the NEAT algorithm

The C++ distribution of NEAT [52], [53] is employed to provide a baseline for the relative difficulty of the task domains. The 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 as per the point population initialization and selection/ replacement algorithms of Sections 3.2 and 3.3. The reward function – that is before application of NEAT specific speciation/ fitness sharing mechanisms – is the sum of rewards over all points, or Equation (2) for Truck reversal and Equations (6) and (7) under the Acrobot problem. 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 [54]. As per recommendations from the NEAT users page [53], additional experimentation was then performed to tune the link weight mutation probabilities and assess the impact of permitting recurrent links. The resulting best case parameterization is summarized in Table 3. Naturally, no claims regarding the optimality of this selection is made, and we assume that the parameterizations for both NEAT and SBB are sufficient for both task domains.[8]

[8]SBB parameters have not been revised since benchmarking under its deployment in the domain of supervised learning for classification [36], whereas NEAT parameters do follow that established under RL tasks.

Table 2: Task Domain Parameterization. 'Max Steps' parameterizes the third fail condition for the Truck Reversal domain described in Section 4.1.

| Truck Reversal domain | |
|---|---|
| Distance traveled per time step | 1.0m |
| Length of Cab | 6.0m |
| Length of Semi | 14.0m |
| Max Steps | 600 |
| Atomic actions $(a)$ | $\{0°, \pm 30°\}$ |
| **Acrobot Handstand domain** | |
| Length of upper (lower) link | 1.0m (1.0m) |
| Length to Centre of Mass in upper (lower) link | 1.0m (1.0m) |
| Moment of inertia of upper (lower) link | 1.0 (1.0) |
| Gravitational constant $(g)$ | 9.8 |
| Control torque update freq. | 5 Hz |
| Simulation step size | 0.05 sec |
| Capture region position | $\frac{\pi}{2} \pm 0.3$ rad |
| Capture region velocity | $\pm 0.3$ rad/sec |
| Max simulation steps $(D_{max})$ | 200 |
| Atomic actions $(a)$ | $\{0, \pm 1Nm\}$ |

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.

# 6 Results: Truck Reversal Domain

The initial truck reversal domain provides a tutorial on properties that may result from the successful application of hierarchical SBB under minimal a priori information. Particular attention is place on the ability to scale to multiple post training test conditions and solution complexity. A behavioural analysis is then made of one particular solution in order to gain specific insight

Table 3: Parameterization at 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 | $\delta$ | 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 |

in to the nature of the resulting symbiont task decomposition.

## 6.1 Evaluation Methodology

Both NEAT and SBB make use of a point population during training. Thus, in order to decide which individual from a run to consider the 'champion' for the test evaluation, a post training validation set is utilized consisting of 1,000 semi–cab configurations. In order to reduce the chances of creating semi–cab configurations that are not solvable, configurations are subject to the constraint that if the semi–cab were to travel in a straight line from the initial condition, they will not collide with the obstacle. Assuming the reward function of Equation (2), the individual ranked with the highest mean reward across the validation set establishes the representative behaviour from each run. An independent test set of 1,000 semi–cab configurations is created in the same way for assessing solution robustness/ generalization, Figure 5. Test performance is then assessed from the perspective of individual-wise and population-wise behaviour per run, the latter reflecting the total cumulative number of test cases solved. A *test case is considered solved* if the co-ordinates for the stopping state of a trajectory are within the following limits: $|x|, |y| < 1.0m, |\theta_s| < 45^\circ$. In total each experiment is conducted over 60 runs.

In the following, three models are considered as follows:

1. SBB without recursion, hereafter referred to as 'SBB-2k', where this reflects the generation limit of 2,000 generations per run. This is twice the generation limit for each of the two SBB applications comprising a hierarchical SBB run;

2. Hierarchical SBB. Results from level 0 of the hierarchy will be denoted 'SBB 0' and those resulting from level 1 of the hierarchy will be denoted 'SBB 1'. 1,000 generations are used per level;

3. NEAT trained with an equivalent number of evaluations per run as the combined SBB 0/ SBB 1 hierarchy or equivalent to the SBB-2k parameterization (see Section 5.4).

## 6.2 Test Cases Solved

Performance post training for SBB and NEAT runs is summarized in Figure 7 in terms of a violin–box plot of the total number of test cases solved. It is readily apparent that SBB individual wise performance at level 0 (SBB 0) represents the lowest performance point. However, in the absence of a subset of a dominating individuals, the SBB framework concentrates on behaviour diversification, as emphasized by the corresponding population wide performance (SBB 0 pop). This is particularly evident when comparing 'NEAT' and 'NEAT pop' to 'SBB 0' and 'SBB 0 pop' respectively. Conversely, individual wise performance of SBB after adding level 1 to the hierarchy (SBB 1) is able to produce individuals that combine the population diversity provided by SBB 0 pop, into single individuals with performance matching this population wide behaviour. This establishes the best case individual wise performance. Likewise the resulting population wide behaviour (SBB 1 pop) represents the most consistently high performing test performance. A two-tailed Mann-Whitney U test supports these observations with the SBB 1 median exceeding that of each pairwise comparison at a significance level of 0.001 (Table 4).

The difference between solutions identified by NEAT and hierarchical SBB is particularly interesting. The context aware crossover operation of NEAT is particularly effective at building

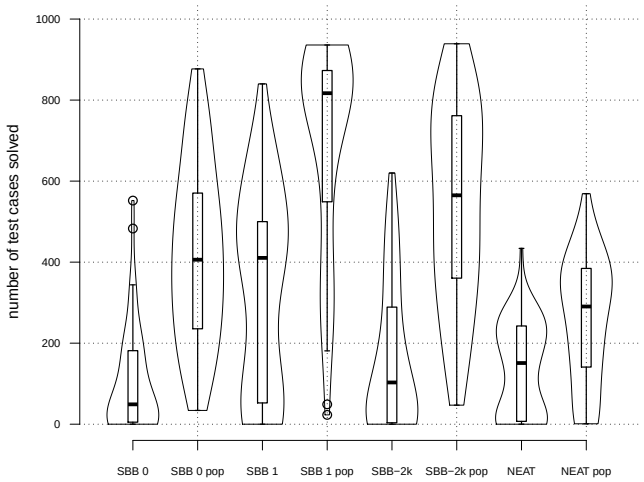



Figure 7: Number of test cases solved. SBB 0 and SBB 1 denote first and second levels of SBB; SBB-2k is the case of SBB without hierarchies enabled; NEAT denotes solutions from the NEAT algorithm. 'pop' identifies columns with cumulative solution count performance as estimated across the entire final population whereas the other columns denote performance of the single best individual from a run.

Table 4: Two-tailed Mann-Whitney U test. *p*-values w.r.t. a pairwise comparison of SBB 1 with corresponding NEAT, SBB-2k and SBB 0 test performance imply that SBB 1 distributions are significantly different. In all cases SBB 1 median was greater than the alternate algorithm.

| SBB 1 versus: | SBB 0 | SBB-2k | NEAT |
|---|---|---|---|
| Individual | 2.683e-7 | 3.774e-5 | 4.636e-6 |
| Population | 7.76e-9 | 1.386e-4 | 1.783e-13 |

strong individuals. However, this model of inheritance also appears to decrease the diversity of the NEAT population as a whole. In short, the speciation component of NEAT does not appear to be sufficient to resist this dynamic. Conversely, the explicitly modular approach to model building in SBB appears to maintain much more behavioural diversity. This is a fundamental requirement for facilitating temporal abstraction in hierarchical SBB. Behaviours as expressed by hosts at level 0 can then be redeployed as temporal/ state abstractions by symbionts at level 1.

SBB without recursion (i.e., the SBB-2k runs) incur the same computational overhead with regards to the number of fitness evaluations. However, hierarchical SBB has the capacity to build much more complexity than is available under SBB-2k. Another set of SBB-2k runs were therefore made with double the host capacity per individual ($\omega = 20$) to see if there is any correlation between improved performance and complexity *without* hierarchical policies. Figure 8 compares the ensuing test performance of the two SBB-2k configurations ('big' denoting the doubled host limit). No statistically significant difference was recorded. Thus, the improved performance of hierarchical SBB was purely due to the hierarchical effect as opposed to merely increasing the complexity of hosts.

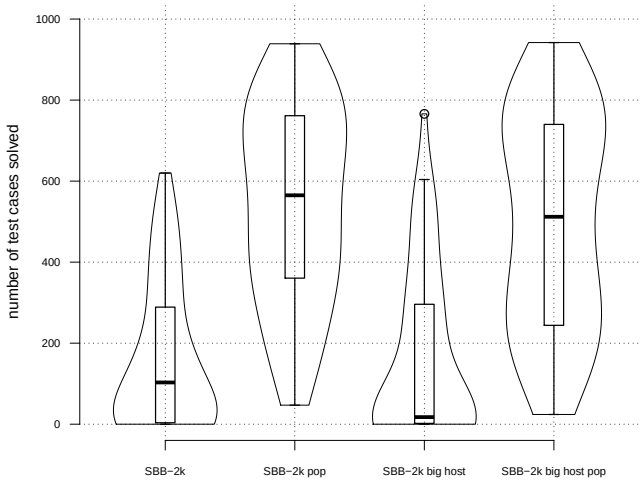



Figure 8: SBB-2k runs with original Max host size ($\omega = 10$) and double the original Max host size ($\omega = 20$) or 'big'.

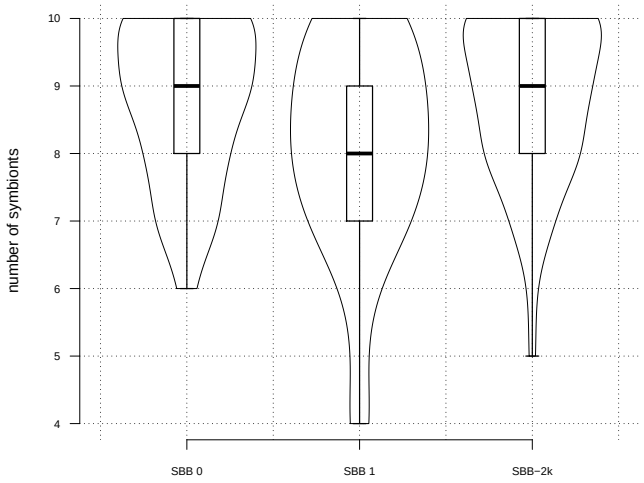



Figure 9: Number of symbionts per host. SBB $x$ denotes symbiont count per SBB layer, and SBB-2k is the single population double training epoch configuration.

## 6.3 Solution Complexity

Assuming independent host–symbiont populations also implies that solutions do not take the form of a single monolithic model [19], [36]. Under the RL domain, we can analyze solutions in terms of the broad count of the number of symbionts per host (Figure 9) and instruction count (Figure 10) where the latter is post intron removal. It is apparent that all symbionts have essentially the same complexity from an instruction count perspective, but level 1 hosts (SBB 1) frequently use less symbionts per host than in at level 0 (SBB 0) or SBB-2k. The relevance of this to temporal abstraction will be visited in Section 6.4.

It is also possible to review the structure of specific solutions from hierarchical SBB. Choosing for example, the SBB 1 host solving 840 of the test cases (Figure 7), we find a level 1 host comprised from eight unique symbionts (Figure 11). Each level 1 symbiont assumes an action defined by a host, as previously evolved, at level 0. The level 0 hosts naturally index a subset of level 0 symbionts taking pre-defined domain specific actions, in this case one of three cab

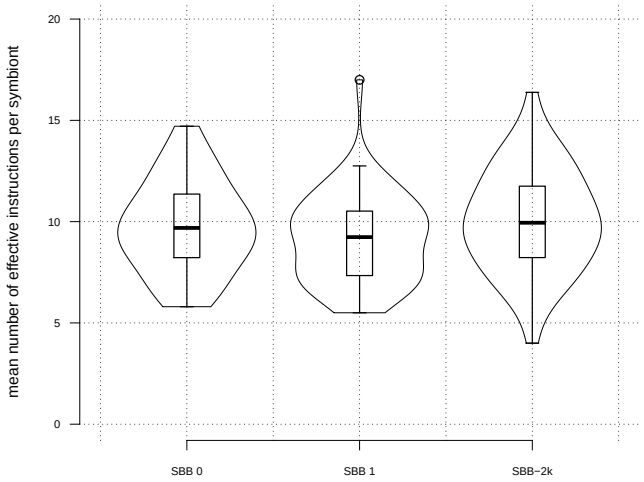



Figure 10: Instruction counts, post intron, per symbiont. SBB $x$ denotes symbiont count per SBB layer, and SBB-2k is the single population double training epoch configuration.

steering angles (Section 4.1). This association is illustrated in Figure 11 by the arcs between level 0 host (rectangle) and domain dependent action/ action subset (ellipse). Naturally, the same action may appear in multiple places as a symbiont learns a unique context for deploying an action. In total there are 30 unique level 0 symbionts (bottom row, Figure 11), grouped to indicate a common association with level 0 hosts. For each group of level 0 symbionts, the corresponding number of times that each action is represented is detailed. It is clear that in the case of this particular individual, the zero degree action is never employed, or state abstraction. This is naturally, an artifact of the particular control policy evolved, different individuals favouring different combinations of the domain specific actions. In total there are therefore 39 unique symbiont programs in this solution (30 at level 0 and 9 at level 1).

## 6.4 Example SBB behaviour

The basic behaviour identified by the hierarchical SBB solution of Figure 11 is to first align the semi–cab with the $y$-axis under a northerly trajectory, albeit without colliding with the wall. This trajectory continues until the leading diagonal between top right and origin is encountered. The semi–cab is then realigned with this diagonal path (again while avoiding the wall obstacle), finally coming to rest at the origin as required. Figure 12 summarizes this particular strategy in terms of a series of snapshots taken at intervals of 100 simulation steps relative to all the 840 solved test conditions particular to this solution. There is, however, a special case that represents an exception to this general behaviour. This takes the form of a swirling motion applied to some semi-cab configurations initialized in the vicinity of the leading diagonal.

Figure 13 describes one such instance of this strategy in terms of both the application of the domain specific steering actions (top) and which of the 5 unique meta actions (level 0 hosts) are deployed when (bottom). Specifically, the swirling trajectory appears as a result of the controller attempting to direct the semi-cab into the known leading diagonal route. However, the first two attempts have to be aborted in favour of jackknife avoidance. The third attempt finally succeeding in making a wide enough loop to join the leading diagonal policy without jacknifing. Figure 14 provides a similar illustration of control behaviour as adopted for a starting condition on the bottom right quadrant in which the semi-cab is initially pointing

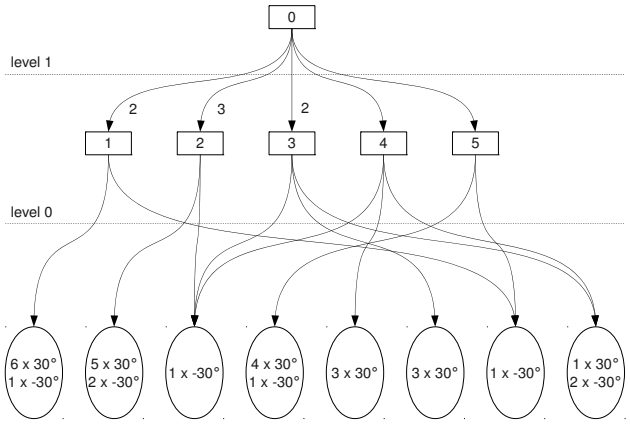



Figure 11: Structure of a solution from hierarchical SBB solving 840 out of 1000 test case solutions. Rectangles denote hosts, naturally there is a single host at level 1, that indexes five hosts previously evolved at level 0. For example, 'host 1' references 6 symbionts with the atomic action of $30^\circ$ and 2 symbionts with the atomic action of $-30^\circ$. One of the $-30^\circ$ symbionts is also employed by host 5, whereas the remaining symbionts are unique to host 1 relative to this particular solution. At level 1 five of the hosts from level 0 are employed as actions by level 1 symbionts of which level 0 host 4 and 5 are used as an action once, host 1 and 3 are used as an action twice and host 2 is used as an action three times. As per level 0 the multiple use of the same action by different (level 1) symbionts implies more than one context for an action. Appendix 10.2 illustrates how such an architecture is evaluated.

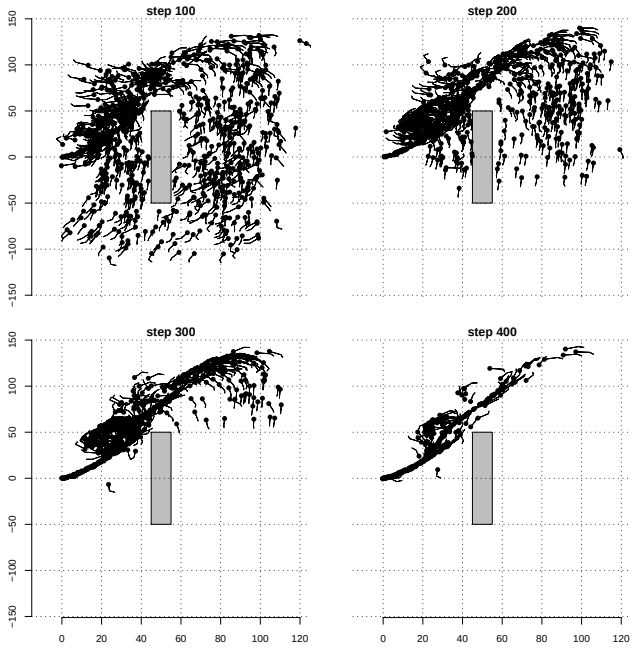



Figure 12: Sampled SBB level 1 behaviour over 840 test cases solved. A circle denotes the end of the semi. Two generic strategies apparent (1) shunt the sami-cab to the top right corner before assuming diagonal path to origin (2) if in top left sector cyclic path taken to orientate semi-cab with the latter diagonal path.

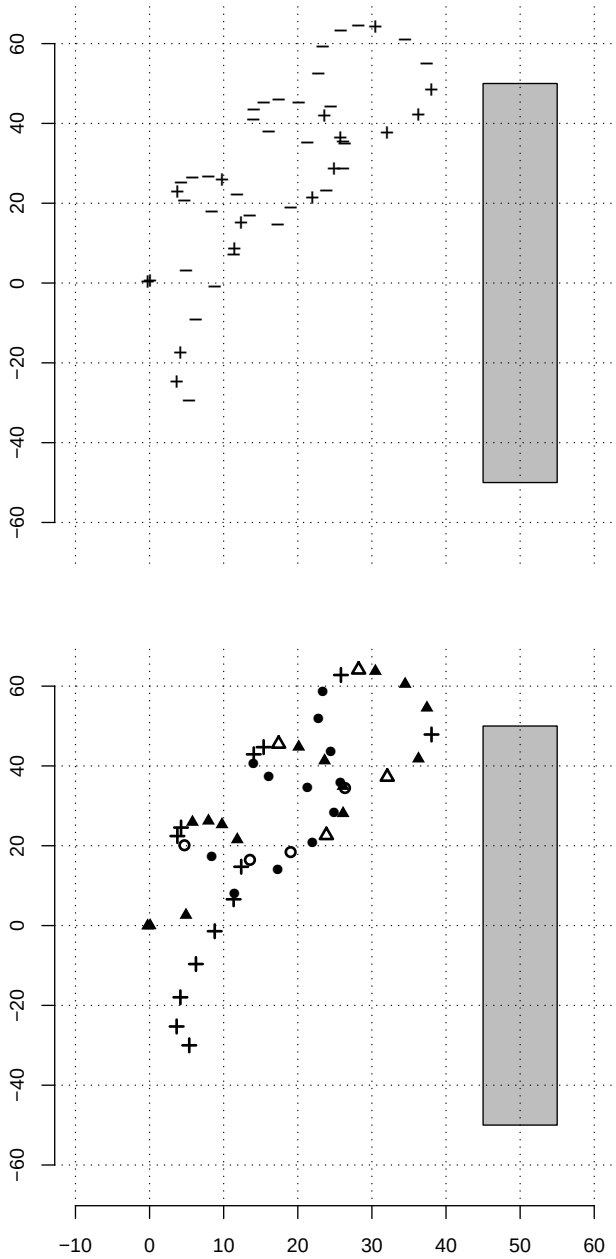



Figure 13: Example of a swirling style control policy. Upper plot represents changes to the steering angle (negative denotes $-30°$, positive denotes $30°$). Bottom plot represents level 0 host deployment (total of 5 symbols, one for each host). For clarity states are represented in terms of $x - y$ co-ordinate pairs at intervals of 10 interactions between policy and task domain.

southeast. Note that in both cases *multiple* level 1 symbionts are deployed during the same trajectory,[9] implying that level 1 behaviours are not solved by merely looking for the nearest previously learnt level 0 behaviour for a given semi–cab configuration and using this for the duration of a solution policy. Level 1 behaviours are clearly a composite of some subset of policies evolved from level 0 hosts. Figures 13 and 14 effectively provide an illustration of the discovery (at level 0) and then deployment of temporal abstractions (level 1) to provide the basis for scaling policies to more difficult test scenarios. This is particularly apparent in the swirling behaviour in which the same level 1 symbionts effectively identify a requirement independent of location, resulting in the repeated sequence of host 0 deployment over multiple 'swirls' (bottom plot, Figure 13).

One last point of interest is with regards to the frequency with which different state variables are utilized by the 60 hosts returned by SBB 0. If this is estimated relative to the atomic actions, we are now in a position to establish the relative weight of different state variables to each (atomic) action (Figure 15). Although no symbionts with a zero degree action (top row, Figure 15) were deployed in the 'champion' from this population, it is clear that symbionts exist using this action and (relative to this action) the $y$-axis and cab angle are rarely used. This probably implies that the cases in which such symbionts win represent a 'baseline' bid against which symbionts with angular actions bid more. Such scenarios being independent of the $y$-axis and jackknife conditions. Conversely, both angular actions (which did feature strongly in the champion from this population, Figure 11) place most emphasis on the angular attributes (middle and bottom row, Figure 15). This is probably a reflection of a desire to avoid jacknifing, where jackknife avoidance is given a greater weighting than wall avoidance. That said, both the $x$-$y$ co-ordinates also feature in between 45 to 65 percent of solutions, where this is most probably a reflection of the need to establish relative location with respect to initial state, goal state and obstacle. The ability to perform such state to variable abstraction is again a key enabling requirement for building solutions under hierarchical SBB.

# 7 Results: Acrobot Handstand

Section 4.2 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 7.1, and lexicographic cost function (Eqn. (6)). We then then go on to assess the "generalization properties" of the SBB approach by considering performance under an independent test set, as evolved under the second formulation of the point population, Section 7.2. We note that most research on the Acrobot handstand task is limited to the swing-up task alone, resorting to simplifications of the problem definition in order to make the problem feasible, and do not consider the wider generalization properties of the task. Finally, the configuration of the point population used under the generalization test will also be assessed under the non-lexicographic cost function of Eqn. (7), thus minimizing the impact of any domain bias from the cost function. In all cases trials are conducted over 100 runs for each test scenario.

In all cases Shapiro-Wilk tests are applied 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.

[9] Level 1 symbionts assuming the meta action (temporal abstraction) as established by hosts evolved at level 0.

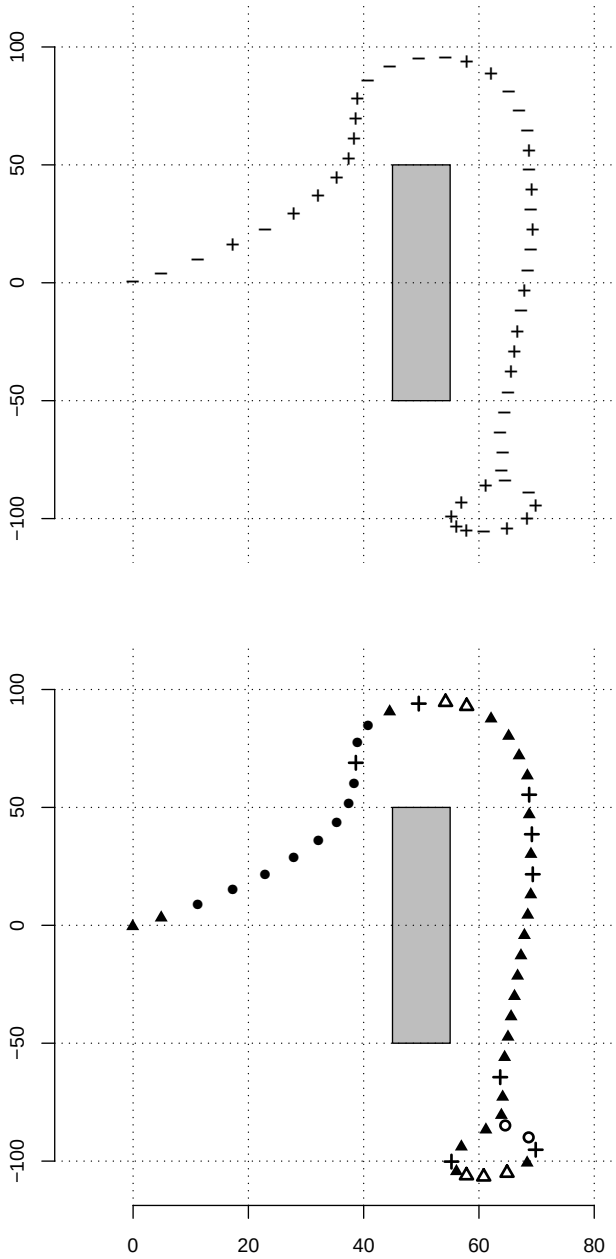



Figure 14: Example of a typical control policy (target the top right quadrant and then assume the leading diagonal to the origin). Upper plot represents changes to the steering angle (negative denotes $-30°$, positive denotes $30°$). Bottom plot represents level 0 host deployment (total of 5 symbols, one for each host). For clarity states are represented in terms of $x - y$ co-ordinate pairs at intervals of 10 interactions between policy and task domain.

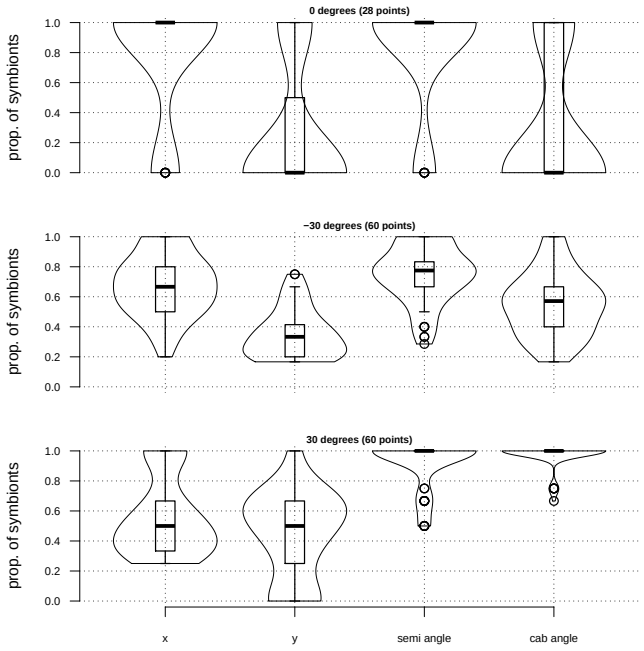



Figure 15: Attribute indexing. Top, middle and bottom row represent $0°, -30°, 30°$ atomic actions respectively.

Bonferroni corrections were used to adjust p-values based on the number of hypothesizes tested per data set.

## 7.1 Evolving solutions for the Swing-up task

The basic objective is to evolve solutions to the swing-up task using as minimal a 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 over the runs that succeeded in solving the swing-up task, as summarized by the violin plot in Figure 16. 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 [7] in 3 cases and in 2 cases identified a faster solution utilizing 80 and 81 steps respectively for both SBB 0 and 1. Figure 17 provides a profile of the action deployment, resulting energy 'pumped' into the Acrobot, and distinction between level 0 host deployment for the the 80 and 81 step solutions. The principle reason for the 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. Figure 17 also emphasizes that level 1 host deployment is not merely a question of selecting a level 0 host 'meta action' as a function of Acrobot initial condition. Instead, level 1 symbionts define contexts for the deployment of host 0 meta action. In Section 7.2 we emphasize how hierarchical SBB is able to leverage this capability into better generalization performance. Indeed, the 'swing-up' task as classically formulated in the literature does not require any generalization

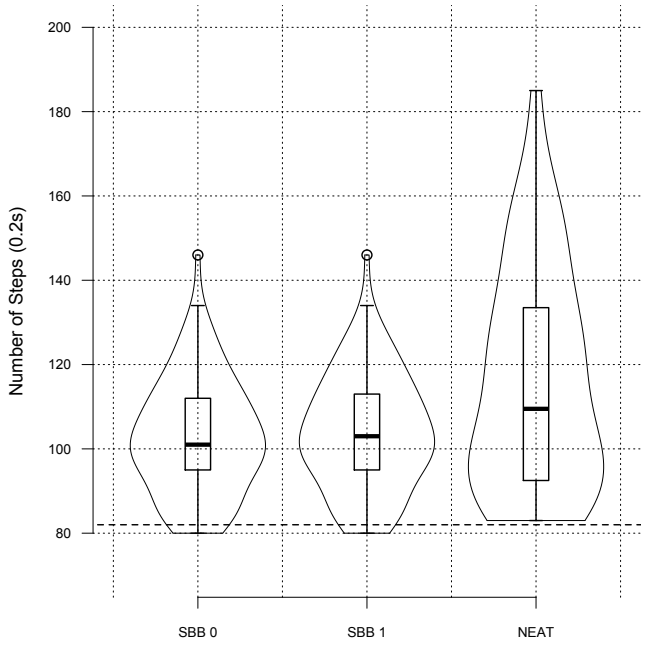


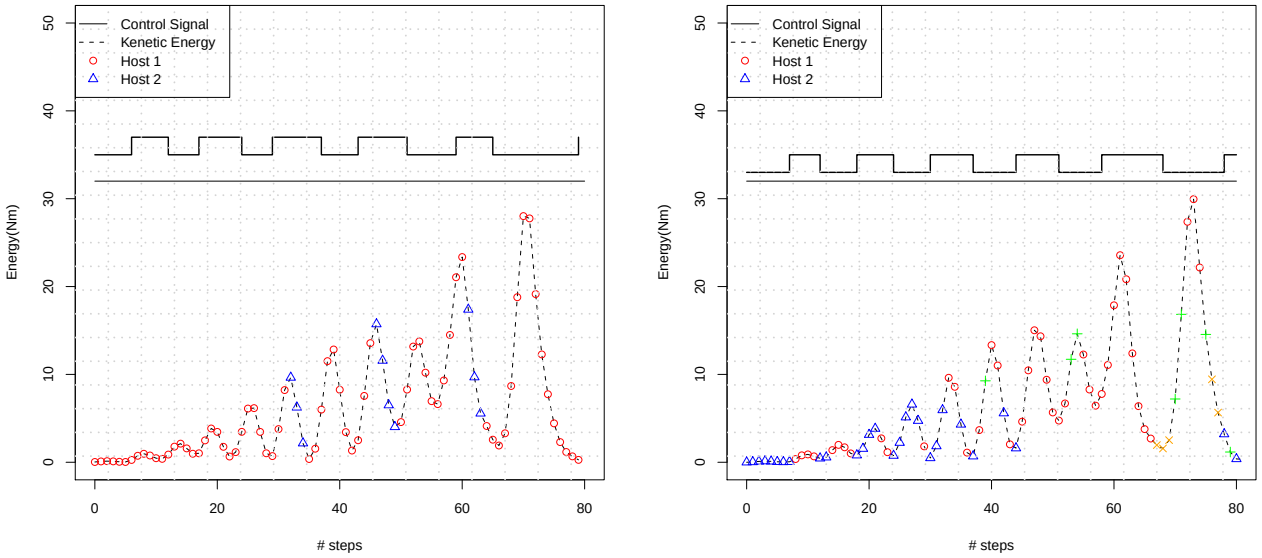

Figure 16: Number of steps necessary to solve the Acrobot swing-up task. The horizontal dashed line represents the case of the shortest previously known solution from Boone (82 steps) [7]. Only cases solving the swing-up task are included in the distribution. A maximum of 200 steps is given before a control policy is considered to have failed.

as such – the initial condition for this is always encountered during training; implying that the 'swing-up' task takes the form of solving a 'shortest path' problem [2] as opposed to rewarding policies that generalize to a wider number of initial states [29]. Finally, the interested reader is referred to Appendix 10.2 for a 'worked example' of how the three host hierarchical SBB solution corresponding to the 80 solution is evaluated.

## 7.2 Generalizing Swing-up solutions: Lexicographic Cost Function

The focus of the Acrobot swing-up task is to evolve a solution with as minimal a number of time steps relative to the single test configuration of the Acrobot feet lying at rest at the bottom of the domain. 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 with regards to the method for initializing/ replacing members of the point population, everything else, including the lexicographic fitness function (Eqn. (6)) remains unchanged. Indeed, the point population initialization/replacement algorithm is now simpler than under the swing-up task, as Arcobot $\theta_1$ configurations are now selected without bias (Section 4.2.6). Test cases are generated to provide 200 test cases sampled uniformly over the range of $\theta_1$ or $2\pi$. 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 goal state within 200 steps.

Naturally, test cases about the 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 18 summarizes the frequency with which a test condition of a




(a) 80 step solution (b) 81 step solution

Figure 17: Example SBB 1 solutions w.r.t. Acrobot 'swingup' task. Different coloured icons differentiate between different level 0 host (meta action) deployment by level 1 symbionts.

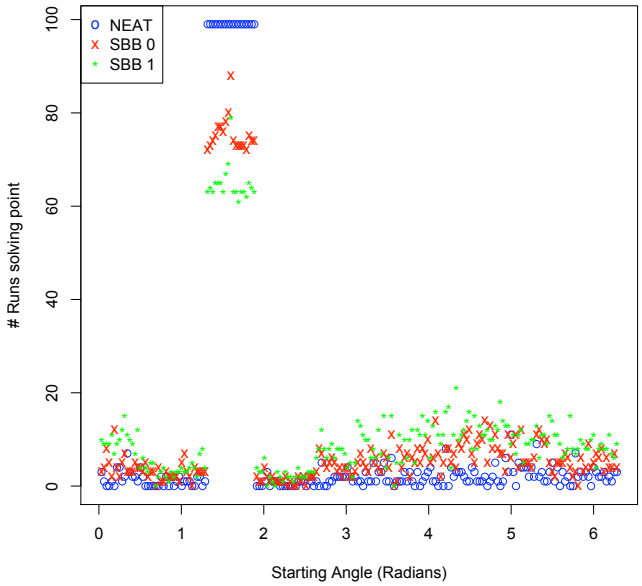
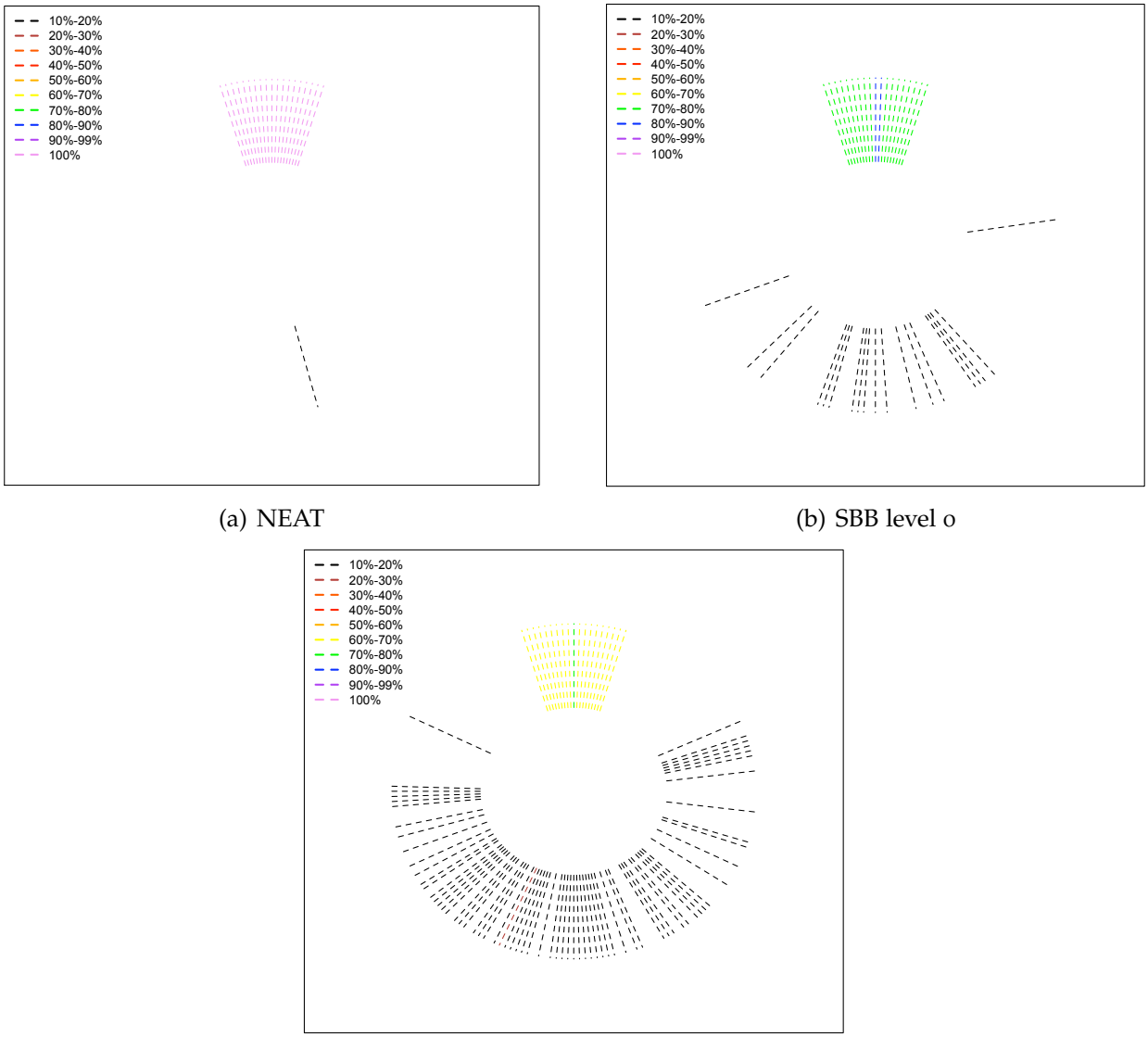



Figure 18: Scatter plot summarizes the percentage of runs for each method that solved an Acrobot initial state for a $\theta_1$ initial condition. Test cases in the vicinity of 1.571 radians correspond to the capture region and have a trivial solution. Cases in the vicinity of 4.712 radians correspond to the 'swing-up' task. The plot requires viewing in colour.




(a) NEAT

(b) SBB level 0

(c) SBB level 1

Figure 19: Frequency with which Acrobot test cases are solved by NEAT and SBB levels 0 and 1 under Lexicographic Cost Function. 100 runs in total. Each 'pin' illustrates the locality for the initial starting point of an Acrobot test case or the initial $\theta_1$, given $\theta_2 = 0$ e.g., locations around "6 o'clock" are synonymous with initial Acrobot positions corresponding to the swing-up task. The plot requires viewing in colour.

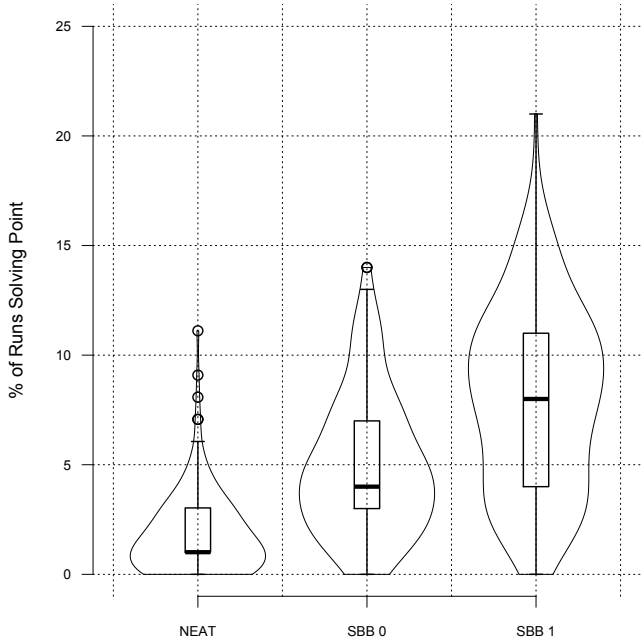



Figure 20: Percentage of individuals solving an Acrobot test configuration. The 19 test points within the vicinity of the capture region leading to trivial policies are excluded.

given initial $\theta_1$ are solved. Given the offset in defining the 'origin' for $\theta_1$ (Figure 6), then angles in the region of $\pi/2$ (approx. 1.571 radians) correspond to test cases in the capture region.

In summary NEAT solutions are dominated by those corresponding to test cases around the capture region, in effect a mediocre stable state dominates the resulting solutions. Note also that there is no bias in the generation of points in the point population to create points in this region, all point configurations are equally likely. The pin wheel plot of Figure 19 provides more insight to the development of SBB solutions between level 0 and 1 by applying histogram bins to the frequency distribution of Figure 18 and then plotting the bins as a function of initial Arcobot arm configuration or $\theta_1$. The addition of hierarchical SBB actually serves as a mechanism for redistributing test cases solved away from those within the capture region to focus on solving test cases elsewhere. 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 20 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 point or nearly twice as likely as a hosts at level 0.

## 7.3 Generalizing Swing-up solutions: Simple Cost Function

A final experiment is performed, in all ways the same as that conducted above, but without explicitly guiding solutions into the conditions of the capture region or expressing the temporal quality of a solution. Instead reward is merely a function of the best height encountered during

an interaction between point and host, as per Eqn. (7); thus biases towards minimizing the temporal component of a solution in particular are discounted. Note, however, that should the conditions of the capture region be satisfied, then a best case fitness for that interaction is returned. Naturally, the conditions for evaluating test performance remain unchanged relative to the two earlier Acrobot experiments.

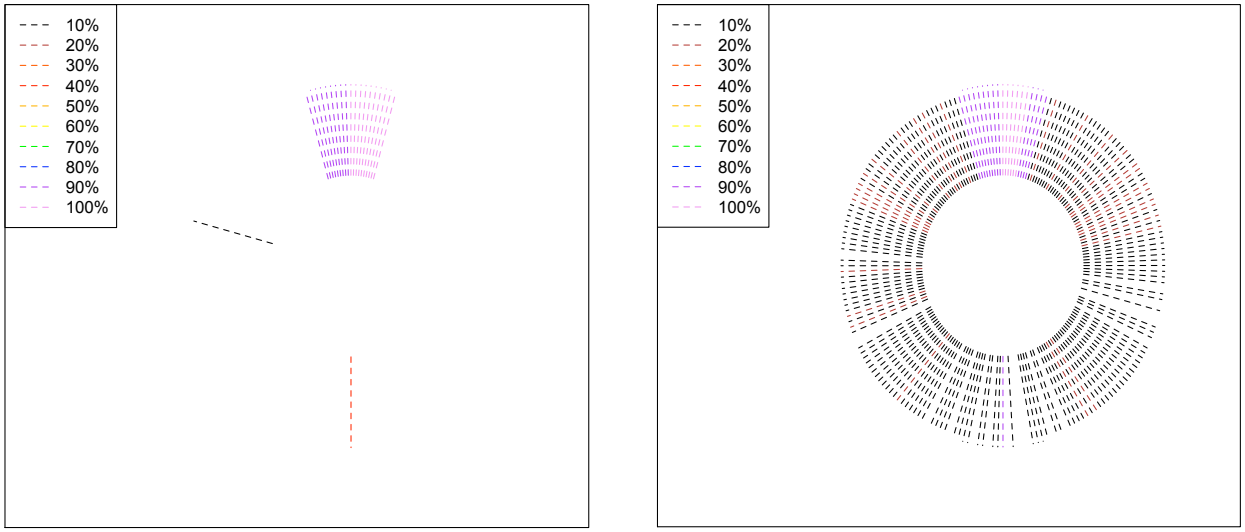
Figure 21 again illustrates the resulting outcome in terms of a pin wheel of the frequency with which different test conditions provide solutions. Little change appears in the results under NEAT, whereas both levels of SBB provide significant increases to the number of test conditions solved and the frequency with which such solutions are provided. Indeed, relative to the generalization performance under the reward function with all three terms included we note that SBB is now able to provide solutions for configurations relating to the top half of the pin wheel as well as the lower half. Conceptually, the lower half are synonymous with states that require much more energy to be 'pumped' into them for the capture region to be reached. Conversely, states in the upper half of the pin wheel need to be dealt with using a rather different style of policy. Needless to say, the resulting average number of solutions per individual under NEAT have not undergone any significant improvement (1.43%) whereas SBB level 0 and 1 now return average solution rates of 14.46% and 19.2% respectively (again excluding test cases initialized within the capture region).

# 8 Conclusion

A symbiotic approach for addressing the scaling problem under RL is presented for GP. Such a scheme explicitly avoids the requirement for a priori problem decomposition; instead the same overall problem goal is used throughout. Support for the effectiveness of the proposed SBB framework is first provided in the case of the truck backer-upper (with obstacle) domain in which solutions capable of solving 84% of the test cases were identified. Conversely, attempts to solve this task without support for hierarchies were limited to solving 45% of the test cases. In the Acrobot handstand task hierarchical SBB was able to generalize to multiple Acrobot configurations, a result that has not been demonstrated before under Boone's configuration of the task domain.

Both task domains illustrate how the hierarchical element of SBB is able to address the scaling problem of RL. This is demonstrated in two specific respects. Firstly, solutions for a test case require the organization of multiple previously evolved hosts, as opposed to merely associating a single previously evolved host with a specific region of the state space. Indeed, as individuals, the hosts from level 0 may not be particularly 'strong'; whereas the hierarchical redeployment of SBB provides a process appropriate for conducting a search for constructing more complex behaviours out of earlier policies c.f., the 'learned abstractions' of reinforcement learning [55] or 'modular interdependency' of complex systems [60]. Secondly, hierarchical SBB provides a significant advantage in terms of generalization. Both of these properties emphasize that any advantages conferred are from the hierarchical element of policy search performed by SBB.

Note, however, that the hierarchies – and therefore the temporal abstractions – constructed by (hierarchical) SBB do not conform to the classical model for HRL. We describe each application of SBB relative to the previous population of hosts. Thus, the goal is always to abstract previously learnt policies to extend the context under which such previously learnt policies were applicable. It is only at the lowest level that hosts result in the deployment of atomic




(a) NEAT

(b) SBB level 0

(c) SBB level 1

Figure 21: Frequency with which Acrobot test cases are solved by NEAT and SBB levels 0 and 1 under Simple Cost Function. 100 runs in total. Each 'pin' illustrates the locality for the initial starting point of an Acrobot test case or the initial $\theta_1$, given $\theta_2 = 0$. The plot requires viewing in colour.

actions. Moreover, in order to maintain multiple policies in parallel, the concept of diversity maintenance plays a central role. Hence, at the lowest level – or case of hosts describing policies in terms of atomic actions directly – it is most important to provide a set of abstractions that cover as wide a range of behaviours as possible. The next level of hosts evolved will define their behaviours through calls to behaviours identified at the last level evolved. Thus, to raise the granularity of the decisions made beyond that of the task domain's atomic actions we concentrate on reasoning about what contexts are most appropriate for applying the previously discovered policies: or the role of the symbionts.

Conversely, HRL under value function optimization emphasize learning to deploy a sequence of temporal abstractions that are always described in terms of the atomic actions. Part of this difference lies in the representation assumed. Value function optimization frequently assumes a representation based on a Markov Decision Process. As such defining temporal abstractions on the basis of start and end conditions is very natural. Conversely, SBB as described here assumes an instruction set limited to stateless execution, thus more similar to value optimization based on tile based representations. Detecting when an event occurs in the environment that constitutes a significant change in policy therefore implies that hosts are always evaluated from the 'root' downwards. Moreover, SBB develops multiple policies/ abstractions in parallel, whereas value function methods tend to incrementally add a single abstraction at a time under an incremental approach to model building.

# 9 Future Work

There are many future avenues for developing the hierarchical SBB framework. Consideration of different task domains representing the most immediate activity. More significantly the approach taken here for constructing solutions hierarchically is a 'proof-of-concept,' possible alternative formulations might include:

1. The set of actions at each new layer naturally assumed all hosts from the previous layer are equally appropriate. Pruning of the 'weaker' hosts before defining an action set might be useful for removing redundant behaviours, with the goal of reducing the size of the space of actions encountered after level 0;

2. The actions a symbiont may assume in higher layers are currently defined relative to the previously evolved hosts. Relaxing this constraint to permit the continued refinement of policy contexts over multiple levels might be appropriate.

3. The present work evolved a fixed number of levels for a fixed number of generations. Indeed, the same parameterization was employed across both task domains. Determining measures for characterizing when to stop adding levels or for how long to evolve a current level would be of interest.

4. Evolving level 0 relative to arbitrary stop and start locations rather than a single domain dependent goal state might provide the basis for recognizing temporal/ state abstractions as opposed to focusing on rewarding policies relative to the ultimate goal state. This essentially just amounts to choosing 2 pairs of conditions from the point population rather than one i.e., individuals from the point population represent start and stop conditions rather than just the start condition, as reported here.

Aside from these immediate considerations, the topics reviewed in the related work of Section 2 are likely to continue to inform the development of the SBB framework.

# 10 Appendix

## 10.1 Example evaluation of a symbiont

The following example illustrates the process by which an individual symbiont is evaluated. Each host consists of at least two symbionts. Thus, when evaluating a host the following process is assumed for establishing the outcome from each symbiont indexed by a host.

Each symbiont *sym* consists of an action and corresponding program i.e., $sym = \langle a, p \rangle$. Evaluation begins by first establishing the outcome of executing the symbiont program, $p$. Let our example symbiont express the following 'linear code':[10]

```
1. R[1] ← R[1] + s[0];
2. R[0] ← R[0] - s[1];
3. R[2] ← cos(R[1]);
4. R[0] ← R[0] / R[2];
```

Programs are initialized with a set of '*numRegisters*' registers. It is typically assumed that the register content is initialized to zero before commencing execution i.e., stateless behaviour.[11] Execution begins with the first instruction and processes sequentially through the individual.[12] Instruction '1' is a two argument instruction, thus the current value of state variable index '0' is added to the current content of register '1', the result also being stored in register '1'. Execution progresses instruction-wise in a similar vain until the last instruction (in this case instruction '4') which updates the content of the 'output register,' which by convention is assumed to be register '0' [10], [9]. Register '0' will contain some (scalar) real-valued number that we now map to the unit interval through the sigmoid operator, Section 3.1.1. This process is repeated for the set of symbionts that are a member of the same host $sym^* = \arg_{sym \in h_i} \max[sym.(bid)]$. The symbiont with largest bid value then winning the right to suggest its corresponding action, $sym^*.(a)$, where this either represents an atomic action (Section 3.1.2) from the task domain or a meta action as in hierarchical SBB, see Section 3.6.

## 10.2 Interpreting a Hierarchical SBB solution: 80 step Swing-up solution

Figure 22 summarizes the relationship between hosts and symbionts indexed at each level of the 80 step hierarchical SBB solution to the swing-up task of Section 7.1. 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 $-1Nm$ and the second with an atomic action of $+1Nm$. The arc connecting this pair

[10] For simplicity we have filtered any intron code from the example.

[11] Retaining register values between consecutive calls to the same symbiont's program would provide the basis for between execution memory (or 'stateful' behaviour), but is not considered in this work.

[12] The addition of 'control' style instructions would facilitate the alternation of program flow, however, this instruction type is not included here, hence we limit the discussion to the case of sequential execution alone.

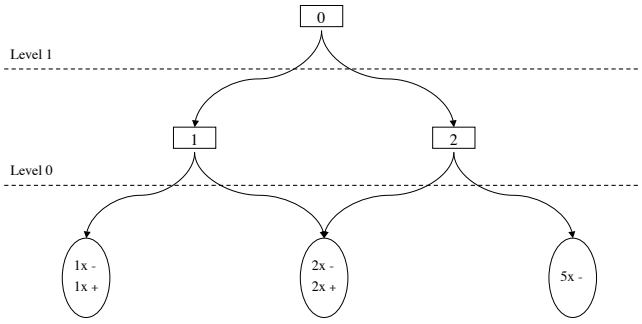



Figure 22: Breakdown of SBB architecture corresponding to the best Acrobot swing-up solution from level 1. A total of 169 instructions are distributed across 13 symbiont programs and 3 hosts. An equivalent solution (but with less generalization) from level 0 utilized 62 instructions over 4 symbionts (1 host).

of symbionts to the level 0 host with label '1' (left hand rectangle) implies that both symbionts are *unique* to this host. However, the four 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 $-1Nm$ 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/ temporal abstraction' associated with the behaviour of each level 0 host.[13]

Now, let us consider the case of the evaluating the policy as defined by this solution. Evaluation begins at the highest level, the single level 1 host labeled '0'. Presenting the current state variables for this time step, $\vec{s}(t_s)$, implies that the two symbionts of host '0' (corresponding to the two descending arcs) execute their respective bidding programs relative to $\vec{s}(t_s)$ (Step 1 of Section 3.6 or Appendix 10.1). The symbiont with maximum bid, say the arc pointing to the level 0 host labeled '1', wins the right to suggest its action (Step 2, Section 3.6). The action in this case is host '1' at level 0 i.e., a meta action. As the action is not an atomic action the process of evaluation repeats, this time relative to the symbionts at level 0 identified by host '1' (Step 3, Section 3.6). Hence, all six symbionts associated with host '1' execute their programs relative to the *same* task state variable(s) $\vec{s}(t_s)$ (Step 1 of Section 3.6 or Appendix 10.1). Again there is a single symbiont with a winning bid, say this time corresponding to one of the two symbionts with an atomic action of $-1Nm$ in the central ellipse (Step 2, Section 3.6). As this is an atomic action the action updates the state of the task domain, producing a new state, or $\vec{s}(t_s+1) \leftarrow world \leftarrow sym^*.(a)$ (Step 4, Section 3.6). Depending on whether the new state variable(s), $\vec{s}(t_s+1)$, represent a terminal/ goal condition or not, the entire process then repeats for the new state beginning at the (single) highest level host.

# Acknowledgements

Peter Lichodzijewski was supported in part by pre-Doctoral scholarships from Precarn, Killam and NSERC (Canada). John Doucette was supported in part by an NSERC USRA (Canada) while at Dalhousie University. Malcolm Heywood gratefully acknowledges support from the

[13] If two different symbionts were associated with deploying a level 0 host at level 1 a numerical index would appear next to the arc from the level 1 host, see Figure 11.

NSERC Discovery and MITACS grant programs (Canada) and SwissCom Innovations SA.

# References

[1] Anderson, C.W., Miller, W.T.: A challenging set of control problems. In: Neural Networks for Control, pp. 475–508. MIT Press, Cambridge, MA, USA (1990)

[2] Barreto, A.M.S., Anderson, C.W.: Restricted gradient-decent algorithm for value-function approximation in reinforcement learning. Artificial Intelligence **172**, 454–482 (2008)

[3] Barto, A., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems **13**(1–2), 41–77 (2003)

[4] Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics **13**(5), 834–846 (1983)

[5] Baum, E.B.: Manifesto for an evolutionary economics of intelligence. Neural Networks and Machine Learning **168**, 285–344 (1998)

[6] Baum, E.B., Durdanovic, I.: Evolution of cooperative problem-solving in an artiÞcial economy. Neural Computation **12**, 2743–2775 (2000)

[7] Boone, G.: Minimum-time control of the acrobot. In: International Conference on Robotics and Automation, pp. 3281–3287 (1997)

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

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

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

[11] Cartlidge, J., Bullock, S.: Combating coevolutionary disengagement by reducing parasite virulence. Evolutionary Computation **12**(2), 159–192 (2004)

[12] Chan, P.T., Xie, W., Rad, A.B.: Tuning of fuzzy controller for an open-loop unstable system: A genetic approach. Fuzzy Sets and Systems **111**, 137–152 (2000)

[13] Chen, S.M., Ko, Y.K., Chang, Y.C., Pan, J.S.: Weighted fuzzy interpolative reasoning based on weighted increment transformation and weighted ratio transformation techniques. IEEE Transactions on Fuzzy Systems **17**(6), 1412–1427 (2009)

[14] Coulom, R.: High-accuracy value-function approximation with neural networks applied to the Acrobot. In: European Symposium on Artificial Neural Networks, pp. 28–30 (2004)

[15] Daida, J.M., Grasso, C.S., Stanhope, S.A., Ross, S.J.: Symbionticism and complex adaptive systems I: Implications of having symbiosis occur in nature. In: Proceedings of the Annual Conference on Evolutionary Programming, pp. 177–186. MIT Press (1996)

[16] de Jong, E.D.: A monotonic archive for Pareto-coevolution. Evolutionary Computation **15**(1), 61–94 (2007)

[17] Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research **13**, 227–303 (2000)

[18] Doucette, J.A., Heywood, M.I.: Novelty-based fitness: An evaluation under the Santa Fe trail. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 6021, pp. 50–61 (2010)

[19] Doucette, J.A., Lichodzijewski, P., Heywood, M.I.: Evolving coevolutionary classifiers under large attribute spaces. In: R. Riolo, U.M. O'Reilly, T. McCounaghy (eds.) Genetic Programming Theory and Practice VII, pp. 37–54. Springer (2009)

[20] Elfwing, S., Uchibe, E., Doya, K., Christensen, H.I.: Evolutionary development of hiearchical learning structures. IEEE Transactions on Evolutionary Computation **11**(2), 249–264 (2007)

[21] Geva, S., Sitte, J., Willshire, G.: A one neuron truck backer-upper. In: IEEE-INNS International Joint Conference on Neural Networks, pp. 850–856 (1992)

[22] Gomez, F.J.: Sustaining diversity using behavioral information distance. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference, pp. 113–120 (2009)

[23] Heywood, M.I., Lichodzijewski, P.: Symbiogenesis as a mechanism for building complex adaptive systems: A review. In: EvoApplications: Part 1, *LNCS*, vol. 6024, pp. 51–60. Springer (2010)

[24] Jenkins, R.E., Yuhas, B.P.: A simplified neural network solution through problem decomposition: The case of the Truck Backer-upper. IEEE Transactions on Neural Networks **4**(4), 718–720 (1993)

[25] Jong, N.K., Stone, P.: State abstraction discovery from irrelevant state variables. In: Proceedings of the International Joint Conference on ArtiÞcial Intelligence, pp. 752–757 (2005)

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

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

[28] Koza, J.R.: A genetic approach to the truck backer upper problem and the inter-twined spiral problem. In: IEEE-INNS International Joint Conference on Neural Networks, pp. 310–318 (1992)

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

[30] Kutschera, U., Niklas, K.J.: Endosymbiosis, cell evolution, and speciation. Theory in Biosciences **124**, 1–24 (2005)

[31] Kwee, I., Hutter, M., Schmidhuber, J.: Market-based reinforcement learning in partially observable worlds. In: Proceedings of the International Conference on ArtiÞcial Neural Networks, pp. 865–873 (2001)

[32] Lehman, J., Stanley, K.: Exploiting open-endedness to solve problems through the search for novelty. In: Proceedings of the International Conference on ArtiÞcial Life XI (2008)

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

[34] Lichodzijewski, P., Heywood, M.I.: Coevolutionary bid-based genetic programming for problem decomposition in classification. Genetic Programming and Evolvable Machines **9**, 331–365 (2008)

[35] Lichodzijewski, P., Heywood, M.I.: Managing team-based problem solving with symbiotic bid-based Genetic Programming. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference, pp. 363–370 (2008)

[36] Lichodzijewski, P., Heywood, M.I.: Symbiosis, complexification and simplicity under GP. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference, pp. 853–860 (2010)

[37] Margulis, L., Fester, R. (eds.): Symbiosis as a Source of Evolutionary Innovation. MIT Press (1991)

[38] Moriarty, D., Miikkulainen, R.: Forming neural networks through efficient and adaptive coevolution. Evolutionary Computation **5**(4), 373–399 (1998)

[39] Moriarty, D., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. Journal of Machine Learning Research **11**, 241–276 (1999)

[40] Mouret, J.B., Doncieux, S.: Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In: IEEE Congress on Evolutionary Computation, pp. 1161–1168 (2009)

[41] Munos, R., Moore, A.: Variable resolution discretization for high-accuracy solutions of optimal control problems. In: International Joint Conference on Artificial Intelligence, pp. 1348–1355 (1999)

[42] Nguyem, D.H., Widrow, B.: Neural networks for self-learning control systems. IEEE Control Systems Magazine pp. 18–21 (1990)

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

[44] Pal, T., Pal, N.R.: SOGARG: A self-organized Genetic Algorithm-based rule generation scheme for fuzzy controllers. IEEE Transactions on Evolutionary Computation **7**(4), 397–415 (2003)

[45] Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems, vol. 10, pp. 1043–1049 (1998)

[46] Riid, A., Rustern, E.: Fuzzy logic in control: Truck backer-upper problem revisited. In: IEEE International Fuzzy Systems Conference, pp. 513–516 (2001)

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

[48] Schoenauer, M., Ronald, E.: Neuro-genetic truck backer-upper controller. In: IEEE Congress on Evolutionary Computation, pp. 720–723 (1994)

[49] Simon, H.A.: The architecture of complexity. In: The Sciences of the Artificial, chap. 4. MIT Press (1969)

[50] Singh, S., Lewis, R.L., Barto, A.G., Sorg, J.: Intrinsically motivated reinforcement learning: An evolutionary perspective. IEEE Transactions on Autonomous Mental Development **2**(2), 70–82 (2010)

[51] Spong, M.W.: The swing up control problem for the Acrobot. In: IEEE Control Systems Magazine, pp. 49–55 (1995)

[52] Stanley, K.O.: Neat c++ (2001). Http://nn.cs.texas.edu

[53] Stanley, K.O.: The neuroevolution of augmenting topologies (NEAT) users page (2010). Http://www.cs.ucf.edu/~kstanley/neat.html

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

[55] Stone, P.: Learning and multiagent reasoning for autonomous agents. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 13–30 (2007)

[56] Sutton, R.R., Barto, A.G.: Reinforcement Learning: An introduction. MIT Press (1998)

[57] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112**, 181–211 (1999)

[58] Thomason, R., Soule, T.: Novel ways of improving cooperation and performance in ensemble classifiers. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference, pp. 1708–1715 (2007)

[59] Vigorito, C.M., Barto, A.G.: Intrinsically motivated hierarchical skill learning in structured environments. IEEE Transactions on Autonomous Mental Development **2**(2), 132–143 (2010)

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

[61] Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P.: Evolving soccer keepaway players through task decomposition. Machine Learning **59**, 5–30 (2005)

[62] Yoshimoto, J., Ishii, S., Sato, M.: Application of reinforcement learning to balancing of acrobot. In: IEEE International Conference of Systems, Man and Cybernetics, pp. 516–521 (1999)