ON DEVELOPMENTAL VARIATION IN HIERARCHICAL SYMBIOTIC POLICY SEARCH

by

Stephen Kelly

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2012

© Copyright by Stephen Kelly, 2012

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "ON DEVELOPMENTAL VARIATION IN HIERARCHICAL SYMBIOTIC POLICY SEARCH" by Stephen Kelly in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: August 16, 2012

Supervisor:

Dr. Malcolm I. Heywood

Readers:

Dr. Denis Riordan

Dr. Nur Zincir-Heywood

DALHOUSIE UNIVERSITY

DATE: August 16, 2012

AUTHOR: Stephen Kelly

TITLE: ON DEVELOPMENTAL VARIATION IN HIERARCHICAL SYMBIOTIC POLICY SEARCH

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc. CONVOCATION: November YEAR: 2012

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

A hierarchical symbiotic framework for policy search with genetic programming (GP) is evaluated in two control-style temporal sequence learning domains. The symbiotic formulation assumes each policy takes the form of a cooperative team between multiple symbiont programs. An initial cycle of evolution establishes a diverse range of host behaviours with limited capability. The second cycle uses these initial policies as meta actions for reuse by symbiont programs. The relationship between development and ecology is explored by explicitly altering the interaction between learning agent and environment at fixed points throughout evolution. In both task domains, this developmental diversity significantly improves performance. Specifically, ecologies designed to promote good specialists in the first developmental phase and then good generalists result in much stronger organisms from the perspective of generalization ability and efficiency. Conversely, when there is no diversity in the interaction between task environment and policy learner, the resulting hierarchy is not as robust or general.

The relative contribution from each cycle of evolution in the resulting hierarchical policies is measured from the perspective of multi-level selection. These multi-level policies are shown to be significantly better than the sum of contributing meta actions.

Glossary

Atomic Actions — The set of domain specific actions selected a priori with respect to each task. In this work atomic actions are discrete, as in a discrete set of legitimate actions for a robot.

Lexicographic — A lexicographic cost function considers multiple properties or objectives with a pre-determined hierarchical ordering.

Meta Action — Any policy learned in a previous cycle of evolution. These policies are no longer evolved and may now be adopted as the action component of symbiont programs in the current cycle of evolution.

Policy Search — An approach to temporal sequence learning in which a control policy is constructed directly from interactions with a task domain. Performance evaluation, and therefore credit assignment, is only performed once a task specific definitive outcome is encountered (e.g., success or failure) or some computational limit is enforced on the number of interactions between environment and task. Policy search is assumed to be the generic method of model discovery in GP [39].

Policy Tree — In this work, a policy tree refers to hierarchical policies developed over multiple cycles of evolution. These policies combine multiple previously learned behaviours, or meta actions, into a single decision making organism.

Sarsa — Sarsa (State-Action-Reward-State-Action) is a value function approximation algorithm for learning a Markov decision process policy in temporal sequence learning problem domains [56].

State Variables The set of variables that describe the current state of the task environment. In control-style temporal sequence learning domains, state variables may be thought of as the sensory information available to a robot. In this work state variables are real valued.

Tabula Rasa In this work, a tabula rasa approach to task diversity implies that minimal information is employed regarding the sampling of initial task configurations. This sampling remains a purely stochastic process throughout the course of evolution.

Temporal Sequence Learning Also known as reinforcement learning, commonly requires an agent, or decision maker, to take sequential actions within an environment in order to achieve a predetermined objective or to maximize a cumulative reward. [56, 39, 2]. Solutions are found through a series of trial-and-error interactions with the environment. Two common approaches to this type of problem are value function approximation and policy search, each defined herein.

Value Function Approximation An approach to temporal sequence learning in which the decision maker attempts to learn a value function describing the reward associated with each state-action pair. If this value function is known, it can be used as the basis for a complete policy. Unlike policy search methods, value function approaches apply incremental refinement with respect to each stateaction pair, or online learning [56, 39].

Chapter 1

Introduction

Cooperation between multiple specialized learners is increasingly recognized to be a key factor in scaling evolutionary robotics to complex control tasks. Symbiosis represents a biologically-inspired mechanism under which genetic programming (GP) approaches are able to support cooperation and problem decomposition with minimal a priori information or manual intervention [32]. This work explores the role of ecology [17] in symbiotic evolutionary policy search as applied to temporal sequence learning tasks.

Temporal sequence learning problems commonly require an agent, or decision maker, to take sequential actions within an environment in order to achieve a predetermined objective or to maximize some notion of reward relative to the task at hand. During an episode, each action taken by the agent results in a corresponding reward or penalty as defined by the environment, as well as an update to the current environmental state. The learning process generally takes the form of a series of trial-and-error episodes over which the agent attempts to develop an optimal strategy [56]. Thus, reward quantifies performance over the episode in proportion to the sequence of interactions between agent and environment. Evolutionary policy search (EPS) algorithms, unlike the more well known value function approaches to these tasks, do not attempt to adapt solutions during a training episode by adjusting their strategy based on the reward received from each action. Instead, a policy is evaluated only after the goal state is reached or the episode ends for another reason, such as a time constraint [39]. Consequently, policy search algorithms require training over a diverse set of problem scenarios as sampled from the task domain in order to produce solutions that generalize well and maintain engagement [29]. By presenting the agents with a diverse set of problem configurations, some easier than others, a gradient in the resulting solution quality can be observed and used to evaluate the agent's policy. More generally, the interaction between configurations of the environment and

capability of the policies can be explicitly varied and tuned throughout evolution or even coevolved. Likewise, assuming a combination of fitness functions which reward the satisfaction of a sequence of goals can also mitigate the significance of assuming an initial population of stochastically configured policies. Without such incremental fitness functions, it is very likely that outright disengagement between the capability of agents and the task will take place, thus reducing policy search to a random walk [19].

Hierarchical approaches are increasingly being used to scale decision making agents to more complex tasks [3]. Layered learning, which represents a promising approach in domains such as robot soccer, relies heavily on employing a priori information in decomposing the problem into appropriate subgoals [55, 60]. For example, the game of robot soccer may be manually decomposed into appropriate subgoals such as getting near the ball, getting open, kicking the ball, passing, etc. These behaviours are first learned separately and later deployed within an a priori decision tree. The agent in this scenario is thus able to reason at different levels of abstraction. In this research we take a different approach in that no a priori problem decomposition is necessary. Defining appropriate subgoals is treated as a secondary problem to be addressed by adopting a process of developmental variation within the symbiotic coevolutionary framework.

Evolutionary Computation potentially covers a wide range of model building paradigms appropriate for conducting policy search. To date a lot of emphasis has been placed on schemes for evolving neural networks [14, 63]. Indeed, several very successful approaches exist for evolving both topology, connectivity and weights, as in the NEAT family of algorithms [53, 60, 52]. However, in this research we instead target the Genetic Programming (GP) paradigm [27, 28]. This paradigm has not received the same interest as neural representations for policy search, thus has yet unknown or little understood capabilities. The specific form of GP assumed for this study is the hierarchical Symbiotic Bid-Based (SBB) framework for GP [32, 15]. SBB supports both lateral and hierarchical task decomposition. The former occurs as a result of cooperative symbiotic coevolution within GP-based organisms as well as competitive coevolution between multiple organisms, detailed in Chapter 3, while the later is achieved by dividing the evolutionary process into independent, hierarchical

cycles. An initial cycle results in diverse policies that lack the capability to solve the entire task. The next cycle reuses policies developed during the first cycle as 'meta actions', combining these previously learned behaviours into much stronger solutions that ultimately outperform the sum of their parts. The SBB framework also maintains distinct populations of diverse task domain configurations, hereafter referred to as points. The combination of task configuration and fitness function represent the environmental component of the ecology.

Our central focus in this thesis is to explore the relationship between ecology and development, while the main contribution is to measure the utility of developmental diversity in providing the basis for a learned, rather than a manually formulated hierarchy. To achieve this, we make two modifications to the SBB framework:

- Introduce goal diversity in the point population such that different evolutionary cycles (levels of the hierarchy) are evolved relative to different physical properties of the task.
- Vary the fitness function such that different evolutionary cycles are evolved under different objectives.

In summary, adopting multiple cycles of evolution to incrementally construct hierarchical policies allows for the interaction between point population and policy learners to be varied during each cycle. At least two sources of variation could appear. Either the (task specific) routine for initializing points and / or the reward function might differ as different levels of the hierarchy are evolved. The underlying design philosophy assumed here is that more specific, **specialized** behaviours evolve at the initial level (identifying meta actions) whereas level 1 (the second cycle of evolution) will establish how to combine meta actions into a hierarchical policy to solve some more general task. We will investigate this hypothesis empirically in Sections 5.2 and 5.3.

The remainder of this thesis is organized as follows. Chapter 2 discusses previous work directly related to the research reported on here. Chapter 3 outlines the SBB algorithm in detail. Chapter 4 describes the two task domains, Pinball and Truck Reversal, in which our experiments are conducted. In Chapter 5 we outline our methodology and discuss results from a variety of experiments in each domain.

Chapter 6 presents our conclusions and looks ahead to future work stemming from this research.

Chapter 2

Background

In this chapter we review previous work relating to the major components of our own study; symbiotic coevolution, hierarchical architectures, the role of diversity in GP, and the role of developmental diversity in the construction of meta actions.

2.1 Symbiotic Coevolution

The synthesis of new organisms from a collective of originally independent behaviours is increasingly being associated with the concept of major transitions in evolution [57, 38, 45, 8]. The resulting synthesis represents a new organism that successfully integrates properties from individuals existing in an earlier / current population. The advantages that this confers on the new organism amount to capabilities that exceed a mere sum of the composite parts. From an evolutionary stand point, such advantages might be reflected in an ability to reach new food sources or exist in environments different from the original organisms. Moreover, a key component in developing such higher level organisms is the ecological interaction that organisms face during their lifetime [17].

Naturally, there are many specific mechanisms that might provide the basis for such transitions. In this work we will adopt that of (endo)symbiosis [36]. Symbiosis represents an evolutionary process in which it is viable to inherit entire 'components' of previously evolved material across multiple species. As such, symbiosis has been credited as the principle mechanism for the transition in evolution from prokaryotes to eukaryotes or the Serial Endosymbiosis Theorem [36]. The two basic components of such a symbiotic model are the host (organism/ compartment) and the symbiont(s). The relationship between host and symbiont is explicitly multi-level. From a developmental perspective we assume that the higher-level host is constructed through the aggregation of different lower-level symbionts [37]. Implicit in this process is recognition that the ecology of the environment promotes useful diversity in symbionts to

warrant their utility as (complementary) building blocks.

In distinguishing explicitly between a higher-level (host) and lower-level (symbiont) entities we also recognize that selection is now a multi-level concept. [42] distinguishes between two forms for multi-level selection (MLS) that are applicable to symbiotic models of inheritance. MLS1 defines fitness of a host as the average of the symbiont membership. Conversely, MLS2 measures fitness as that defined by the host behaviour alone. Okasha goes on to make the case for assuming MLS1 during a developmental phase prior to the appearance of symbiotic (group) relationships, but adopts MLS2 once hosts (cf., groups) exist. In this work we explicitly adopt MLS2 from the outset as our interest lies in evolving hierarchies of programs for increasingly abstract decision making, where host policies at every stage of the hierarchy are defined by the group behaviour of contributing symbionts.

From the perspective of evolutionary computation symbionts define the minimum inheritable 'building block' and hosts identify subsets of symbionts for possible coexistence [20]. Specifically, with reference to the Symbiotic Bid-Based architecture for GP adopted in this work [32], symbionts explicitly separate *context* and *action*. Actions are discrete, with each symbiont associated with a single (scalar) action. As such, the set of candidate actions can be defined by the task domain or in terms of previously evolved hosts (meta actions). Such a property has potential utility in a wide range of temporal sequence learning tasks in which it is very difficult to construct policies efficiently when limited to (atomic) actions of the task alone.

The purview of this work is to relate the SBB framework for constructing modular organisms under GP to incremental evolution. Specifically, the hierarchical formulation of SBB enables meta actions learned in one ecology to be explicitly redeployed under the next, but developmental variation between cycles of evolution has not yet been explored. The developmental diversity employed here synchronizes variation of the ecology with complexification of the SBB organism.

2.2 Hierarchical Models

Various generic architectures have been suggested for composing explicitly hierarchical solutions. The subsumption architecture of Rodney Brooks is one of the most widely acknowledged [6]. Such an architecture begins with general policies at the

'lowest level' after which additional levels add corrections / exceptions to the lower level policies. In addition, Brooks made explicit the requirement for a policy to be developed through a process of direct 'interaction' with the target environment. Domain knowledge is typically utilized to provide the necessary decomposition of the task into an appropriate hierarchy of controllers. For example, a recent neuro-evolutionary approach for helicopter control assumed three controllers (guidance, pitch and roll), with each controller receiving specific combinations of state variables and the guidance controller feeding the pitch and roll controllers [13].

Layered learning frameworks, as mentioned in the introduction, take a similar approach in that prior knowledge is employed to decompose a task into a series of sub-tasks or independent training scenarios [55]. In our research we refer to any previously learned policy as a meta action. Providing the capability to describe a new policy in terms of a series of references to (previously identified) meta actions represents one avenue to scaling (value function) reinforcement algorithms to more challenging tasks [3] i.e., the decision maker only needs to determine the sequence of meta action deployment as opposed to atomic action deployment.

In the specific case of Genetic Programming the metaphor that results in solutions taking an explicitly hierarchical structure is that of run-time libraries. The concept of run-time libraries (RTL) recognizes that although a particular parameterization of a GP run might be unsuccessful, the content of the population may contain subroutines of utility to a new run of GP. Thus, as opposed to just ignoring material from the unsuccessful run and starting from scratch, it is retained as a 'library'. The new run is augmented with an instruction set that supports references to individuals from the library. No further adaptation takes place in the RTL. Moreover, various authors have recognized that heuristics are often necessary to force the code under evolution to reference that in the RTL [46, 22, 31]; where such heuristics might have a negative impact on the resulting solutions. Conversely, modularity / code reuse as introduced by Automatically Defined Functions (ADF) attempt to evolve sub- routines and calling code at the same time [28]. As a consequence the search space is potentially that much larger; whereas the RTL approach first evolves the library, then attempts to integrate the library into the calling code. A recent development assumes a coevolutionary methodology in which tagging – sets of scalar references – are used

to identify candidate code modules dynamically [51]. Unlike ADFs, tagging results in a behavioural approach to code reuse, the number of code modules is therefore a dynamic property of evolution. However, it is also clear that tagging itself is sensitive to the specific GP representation employed. Thus much like ADFs, depending on the task, tagging may or may not result in an improvement [50]. Conversely, the original study employed an advanced form of GP based on a linear representation [51]. This meant that it was much easier to deal with exceptions caused by, for example, calls to modules that do not exist or provide more elegant schemes for dealing with non-existent return variables.

In the case of the proposed approach we evolve hierarchical policies layer-by-layer. Only the highest layer undergoes adaptation, all previous layers remain fixed, thus a previous layer (of meta actions) represents a RTL. However, the number of meta actions utilized by each host policy is a function of evolution. The resulting framework therefore builds on the RTL metaphor rather than that of ADFs or subroutines [23].

2.3 On External Sources of Developmental Diversity

In the following we will assume that coevolutionary frameworks are capable of benefiting from at least three basic external – that is environmental – sources of developmental diversity: goal diversity, behavioural diversity, and task diversity. This is in addition to the diversity that is implicit in assuming a population based approach to machine learning [44]. Thus, diversity in evolutionary computation can be viewed from at least three perspectives: 1) as maintained from a genotypic perspective, 2) as occurs during the mapping from genotype to phenotype, or; 3) by external sources of variation acting on the phenotype. This research concentrates on the utility of the latter 'external' developmental sources of variation. Conversely, only very specific forms of GP support variation of the geno–phenotypic mapping (e.g., [62, 43])

In the following we recognize three forms of (external) diversity: goal diversity, behavioural diversity, and task diversity. **Goal diversity** might include variation in what fitness rewards during evolution as in incremental evolution (e.g., scaffolding [64], shaping [14] or chaining [4]). A suitable schedule (priori knowledge) for adapting (specifying) the fitness function would be typically necessary. This thesis will begin by assuming a single goal for each task and then introduce goal diversity for emphasizing

the contribution of policy specialists. **Behavioural diversity** characterizes to what extent a population is able to maintain a diverse set of measurably different behaviours for the same state of the task. [10] noted two mechanisms in support of behavioural diversity: direct or indirect. Direct schemes might include variation operators (e.g., higher rates of mutation), selection operators (e.g., changed fitness selection [49]) or novelty as an objective [30]. Indirect schemes might include fitness sharing [47, 53], tournaments [11] or Pareto archiving [12]. The SBB framework utilized for this research assumes fitness sharing as the default approach to maintaining behavioural diversity between candidate solutions. **Task diversity** recognizes that if performance is evaluated under a single initialization of the environment, then the resulting policies are not likely to see a sufficiently rich set of contexts from the task domain to provide 'robust' solutions [29]. In this work we assume a tabula rasa sampling of initial states from the task environment in order to concentrate on the contribution from hierarchical policies. However, several forms of competitive coevolution also lend themselves to the sampling of 'good' initial configurations of training scenarios in order to minimize the impact of factors such as disengagement [9]. We note, however, that when real-valued performance functions are employed (as in the work reported here), then competitive coevolutionary mechanisms might not be beneficial or may even be detrimental, e.g. Chapter 6 of [32].

2.4 Discussion

As task domains become more difficult, both value function and policy search paradigms make use of some form of a priori task decomposition. Thus, a priori sub-goals are frequently utilized in value function methods against which meta actions are first independently developed [3]. Assuming success in solving each sub-task, then an a priori scheme for deploying solutions to each sub-task, such as a decision tree, might be adopted for solving a more generic task [61] . The drawback is that a specific task decomposition has to be assumed, whereas the SBB framework has the potential to directly build a 'decision tree' through multiple cycles of evolution. In this work we demonstrate the advantages of developmental diversity when developing meta actions in hierarchical symbiotic policy search.

In order to make this case, we compare hierarchical and non-hierarchical SBB both

with and without developmental diversity. This represents a reasonable scope for four alternate configurations having similar complexity and computational overhead. Julian Togelius [58] makes a similar comparison with regard to an evolved neural network controller. In the process he outlines a naming scheme which we adopt for this study.

Monolithic evolution will refer to configurations where a single-cycle of evolution (no hierarchy) is performed with the same developmental conditions throughout evolution. Thus, there is no goal diversity, support for behavioural diversity remains unchanged (the fitness sharing assumption of SBB) and task diversity assumes a tabula rasa sampling (Section 2.3). This is also sometimes referred to as 'direct' evolution, and mirrors the classical formulation for genetic programming in which a population of individuals is evolved relative to a single, common objective. 'Monolithic' does not describe the structure of the SBB framework, which is inherently modular in every configuration, as multiple distinct components (symbionts and hosts) are always present.

Incremental evolution again refers to single-cycle experiments (no hierarchical transitions) but now goal diversity is introduced at a fixed point during the course of evolution. Incremental evolution is commonly used to slowly, or incrementally, scale learning agents towards increasingly complex tasks as in [59, 19, 4, 13]. This process usually begins by presenting agents with a simple learning task and then increasing complexity and/or difficulty over time as the learners become proficient enough at each stage.

Modularized evolution in this work refers to SBB with hierarchical model building enabled, but without goal diversity. Thus, the developmental conditions remain as in the case of the Monolithic experiments. This is similar in principal to work involving robot controllers comprised of multiple neural networks [41, 7] and, from a genetic programming perspective, to subtree encapsulation [46] and ADFs [28].

Finally, **Layered** evolution is the the term we use when each cycle of evolution in hierarchical SBB is associated with a different goal. Thus, the developmental conditions remain as in the case of the Incremental experiments. This has been explored widely from the perspective of neuroevolution in an attempt to scale temporal sequence learning to increasingly complex tasks. [58] used layered (neuro)evolution to

develop a controller for a simulated robot that learns which light source to approach in an environment with obstacles. It was shown that evolving layers one at a time while incrementally introducing new obstacles significantly improved learning as compared to other models of evolution. As mentioned in the introduction, layered evolution has also been used to scale neuroevolution towards more complex tasks such as keepaway soccer [61, 55], requiring a significant amount of human intervention.

This research takes inspiration from the works mentioned in this section and makes a comparison of various evolutionary models within the context of symbiotic policy search with GP. The symbiotic bid-based (SBB) framework for GP represents the starting point for policy search adopted in this work (Chapter 3 summarizes SBB). Hierarchical model building represents a central component of SBB. The process by which policies are constructed with an explicitly hierarchical structure is through conducting independent cycles of evolution. Thus, from the perspective of previous GP research this is closer to the reuse of previously evolved solutions than code modularity, as in Koza's ADFs or tagging. The general question asked by this research is to what extent developmental diversity plays a role in constructing better policies under temporal sequence learning tasks. However, of the three forms of developmental diversity, SBB already assumes a specific mechanism for enforcing behavioural diversity, care of competitive fitness sharing. Likewise, as reviewed in Section 2.3 a tabula rasa approach to task diversity will also be assumed. The specific focus of this thesis will therefore lie in the role of goal diversity in facilitating the construction of better SBB policies of an explicitly hierarchical structure. Unlike related research in Layered Learning, a specific prior task decomposition is not assumed, although there is no reason why such information could not be utilized. The natural trade off would be that constraints imposed by a prior task decomposition may preclude the identification of more optimal solutions; a result recently identified relative to constraints applied to A* under the Acrobot handstand task [15]. Conversely, not enforcing a prior task decomposition (constraints) may render it impossible to identify solutions solely through machine learning alone.

Chapter 3

Hierarchical Symbiotic Policy Search and Cycles of Evolution

In this section we review the generic architecture for SBB and highlight the key structural components that provide the basis for a simple ecology. SBB explicitly enforces symbiosis by separating host and symbiont into independent populations, Figure 3.1. Within a cycle of evolution (conducted over a fixed number of generations), each host represents a candidate solution in the form of a group of symbionts existing independently in the symbiont population. Performance is measured relative to the interaction between a subset of initializations from the task domain (points) and host. A breeder model of evolution is assumed, thus a fixed number of hosts and points are deleted/ introduced at each generation. Different cycles of point–host–symbiont development build new levels of complexity to the overall organism.

3.1 Development Within Cycles of Evolution

3.1.1 Symbiont Population

Members of the symbiont population assume a Bid-Based GP representation [34]. As such, each symbiont, sym, is represented as a tuple $\langle a, p \rangle$; where a is an action as selected from the set of atomic actions associated with the task domain and p is the corresponding symbiont's program. The program defines a context for deploying its action. Execution of a symbiont's program results in a corresponding real-valued outcome in the output register, $R[0]$, or the 'bid'. The linear representation [5] leads to programs being defined by a simple register addressing language of the form: 1) Two argument instructions, or $R[x] \leftarrow R[x] < op_2 > R[y]; op_2 \in \{+, -, \div, \times\}$; 2) Single argument instructions, or $R[x] \leftarrow < op_1 > (R[y]); op_1 \in \{\cos, \ln, \exp\}$; 3) A conditional statement of the form "IF $(R[x] < R[y])$ THEN $(R[x] \leftarrow -R[x])$. In addition, $R[y]$ can be either a register reference or index a state variable.

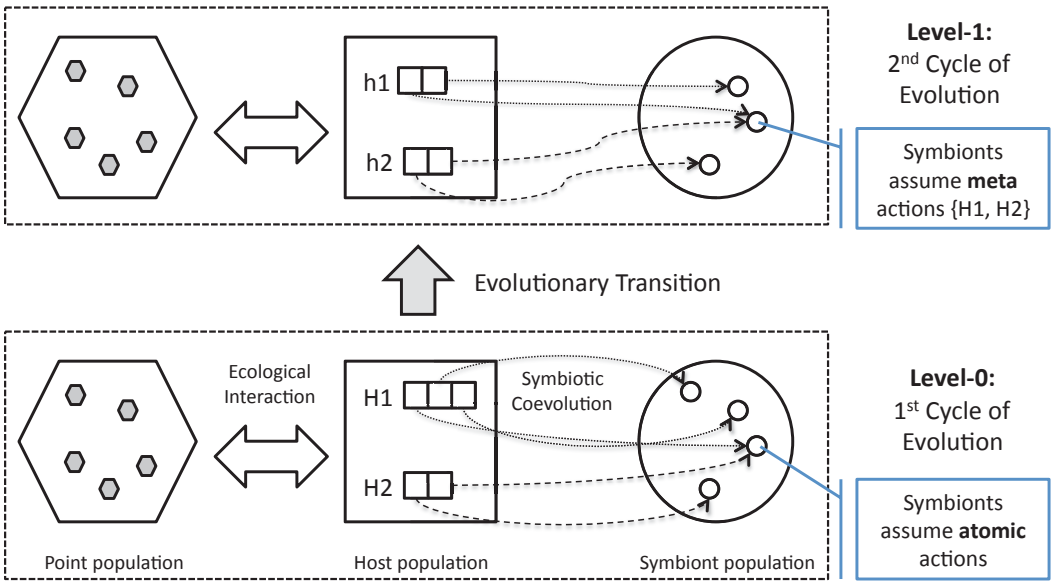


Figure 3.1: Hierarchical symbiosis through cycles of evolution. A cycle of evolution begins at *level-0* with the ecology defining an interaction between point population and host population, where fitness is evaluated. Hosts identify a group of symbionts defined in the symbiont population. Symbionts in level-0 are limited to the atomic actions of the task domain. During the next cycle of evolution (*level-1*) the process repeats under a new point–host–symbiont partnership. This time symbionts assume actions defined by previously evolved hosts. For simplicity, host–symbionts developed during previous cycles of evolution remain fixed.

3.1.2 Host Population

Symbionts are explicitly limited to deploying a single action. Thus a host needs to identify relevant subsets of symbionts that are capable of collaborating. To do so, each host indexes a subset $[2, ..., \omega]$ of the symbionts currently existing in the symbiont population. Relative to a single host, h_i, fitness evaluation is conducted against a set of initial configurations of the task domain, as defined by individuals from the point population, p_j. Such a process has the following form:

1. Present the state variables describing the current state of the task domain, or $\vec{s}(t_s)$;
2. $\forall sym \in h_i$, identify the corresponding symbiont bid, or $sym(bid(\vec{s}(t_s)))$;
3. Identify the 'winning' symbiont as that with the maximum bid from host h_i or $sym^* = \arg_{sym \in h_i} \max[sym(bid(\vec{s}(t_s)))]$;
4. Apply the action from the winning symbiont to the task ecology and update the state variables accordingly.

Symbionts therefore use bidding to establish the *context* for deploying their respective action. The number of symbionts per host and 'mix' of actions appearing in a host are both an artifact of the evolutionary cycle. We assume episodic style reinforcement learning tasks, thus at some point a terminal condition is encountered. The relative uniqueness of each hosts's performance across training cases will be discounted under competitive fitness sharing (Section 3.2.2). This diversity maintenance is a key requirement in supporting symbiotic complexification through resampling of previously evolved traits ([37, 45, 54]).

3.1.3 Point Population

The role of the point population is to sample initial conditions from the task with sufficient diversity to provide variation in the behaviours of hosts as measured through the reward function. A tabula rasa approach is assumed, where this is generally taken to imply minimal information regarding the sampling of an initial task configuration, $\vec{s}(t = 0)$. The combination of point population and reward function define the environmental component of the ecology.

3.2 Selection and Replacement

3.2.1 Point Population

At each generation P_{gap} points are removed with uniform probability and a corresponding number of new points introduced. The process for generating points is naturally a function of the task domains and will be detailed once these have been defined (Sections 4.1.4 and 4.2.4).

3.2.2 Host Population

As per the point population, a fixed number of hosts, H_{gap} are removed at each generation. Host removal is applied deterministically with the worst H_{gap} hosts targeted for removal at each generation. A competitive fitness sharing formulation [47] maintains diversity in the host population. Thus shared fitness, s_i of host h_i takes the form:

$$s_i = \sum_k \left(\frac{G(h_i, p_k)}{\sum_j G(h_j, p_k)} \right)^3 \tag{3.1}$$

where $G(h_i, p_k)$ is the task dependent reward defining the quality of policy h_i on test point p_k (see Equations 4.1, 4.2, and 5.1).

Naturally, deleting the worst H_{gap} hosts may result in some symbionts no longer appearing in a host. This is taken to imply that such symbionts are noncompetitive, thus they are also deleted. The size of the symbiont population will therefore vary while the host population size remains fixed.

3.3 Variation Operators

Symbiosis is an explicitly hierarchical coevolutionary process. From a exploration/ exploitation perspective it is important not to disrupt 'good' symbiont combinations while simultaneously continuing to search for better hosts. Moreover, variation needs to be maintained at the symbiont level without disrupting symbionts that are already effective. The process for maintaining exploration (diversity) without unduly disrupting the better host–symbiont relationships therefore follows an explicitly hierarchical formulation. Following the removal of H_{gap} hosts, the remaining $H_{size} - H_{gap}$ hosts are

sampled for cloning with uniform probability. The resulting clones have both their host and symbiont content modified. As such, it is ensured that new symbionts are only associated with new hosts and therefore no disruption of previous host behaviour takes place. For a detailed presentation of this process see [32, 33].

3.4 Cycles of Evolution and Hierarchical Development

The above description summarizes the process as applied to a single 'level' of symbiosis, or lower point–host–symbiont interaction (level 0 of Figure 3.1). The division of labour is therefore purely *lateral*, both with respect to host content (symbiont complement) and across the host population (cf, fitness sharing). However, after evolving for a fixed number of generations, the content of the host–symbiont population pair might not provide any outright solutions. Rather than begin evolution afresh from a completely new host–symbiont parameterization/ initialization, the current host population is considered to represent a set of candidate behaviours (meta actions) for constructing more complex organisms or a *hierarchical* division of labour. At this point the populations of meta actions are functionally diverse care of the competitive effect of fitness sharing, thus recombination of the current behaviours under new contexts has the potential to lead to new capabilities.

To do so, hosts previously evolved at level 'l' represent the spectrum of actions that a new symbiont population at level '$l+1$' may index. Thus, at level $l=0$ symbiont actions are always defined by the task domain, or 'atomic actions.' Thereafter, for symbionts at level $l>0$ the action set is the set of all hosts from level $l-1$ or $a \in \{H^l\}$. Naturally, the subset of actions utilized by each host is a function of policy search. For simplicity we assume that evolution at each level is an independent process or *cycle*. The evaluation of host i at level l or (h_i^l) now has the following hierarchical form:

1. Present the state variables describing the current state of the task domain, or $\vec{s}(t_s)$;

2. $\forall sym^l \in h_i^l$, identify the corresponding level l symbiont bid, or $sym^l(bid(\vec{s}(t_s))$;

3. Identify the 'winning' symbiont for host h_i^l, or $sym^* = \arg_{sym^l \in h_i^l} \max[sym^l(bid(\vec{s}(t_s)))]$;

4. IF $l == 0$ THEN Step (5) ELSE

   (a) Descend a host–symbiont level: $l = l - 1$

   (b) Look up the new current host as identified by the action of the 'winning' symbiont sym^* as returned by Step 3: $h_i^l \leftarrow sym^*(a)$;

   (c) RETURN to Step (2);

5. Apply the atomic action from the winning symbiont to the task domain and update any state variables accordingly: $\vec{s}(t_s + 1) \leftarrow \text{ecology}(t_s + 1) \leftarrow sym^*(a)$.

Evaluation of a policy is thus top-down while construction of hierarchical policies is a bottom-up process. Each level in the hierarchy is the product of a distinct cycle of evolution in which a diverse collection of behaviours is developed. Only the lowest level hosts retain symbionts with actions specified in terms of the task domain's atomic actions. After this cycle of evolution, the set of candidate actions for new symbionts is defined in terms of indexes to the host–symbionts evolved at the previous cycle of evolution (cf., meta actions). Variation operators only act on the point–host–symbiont populations in the new cycle of evolution, or level-1 in Figure 3. Given this capability, it is natural to ask whether, by varying the ecological reward at each level (specifically goal diversity, Section 2.3), can we improve the resulting generalization of the overall policy?

Chapter 4

Task Domains

Two control style task domains are considered for benchmarking purposes: pinball and truck reversal. In both cases it is necessary to develop policies for maneuvering in a non-linear state space using orientation and spatial control, establishing which to prioritize when. Agents are required to make multiple decisions, potentially in the hundreds, before the final outcome is known. In the following we summarize the properties of each task domain, introduce findings from previous research, as well as define reward function and the routines used for initializing the point population.

4.1 Pinball

4.1.1 Task Domain

The pinball domain was recently proposed as a more demanding benchmark than the Acrobot (height) or mountain car tasks frequently employed in the evaluation of RL algorithms [24]. Specifically, the pinball domain assumes continuous state variables and possesses multiple discontinuities and extended dynamic control characteristics. The goal is to provide a policy for navigating a maze. However, it is actually useful to collide with walls; in this case in order to change direction. The atomic actions either add or subtract energy to a ball (or make no change) whereas the ball is subject to a drag coefficient (0.995). Unlike the original formulation of the task we are interested in solving for a uniform distribution of start points – thus demonstrating generalization – relative to a difficult maze configuration from the original study, Figure 4.1. The task domain itself is available as a Java distribution.[1]

[1] http://www-all.cs.umass.edu/~gdk/pinball/

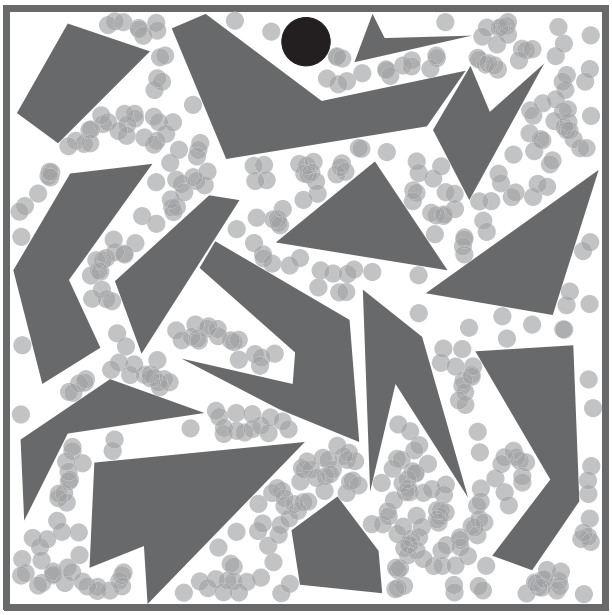


Figure 4.1: Pinball task domain. Large black circle (top centre) represents the single (global) exit location. Smaller grey circles represent the set of 500 initial start conditions over which post training generalization is evaluated.

4.1.2 Previous Results

Konidaris *et al.* has employed the pinball task in multiple demonstrations of skill-chaining (a form of task decomposition) for value function approximation [24, 25]. In short, the task is known to benefit from being able to apply different policies for different regions of the state–action space, hence a potentially informative benchmark for the modularized symbiotic framework explored in this work.

4.1.3 Atomic Action and State Variables

Cartesian co-ordinates provide the basis for a four dimensional state space describing both position and velocity or $\{x, y, \dot{x}, \dot{y}\}$, Table 4.1. Atomic actions add, decrement, or make no change to the velocity of the pinball or $a \in \{0, \pm\dot{x}, \pm\dot{y}\}$, Table 4.1.

Developmental Diversity

During the construction of host–symbiont policies it will be possible to vary the properties of the environment at different points of evolution (see the concluding paragraph

of Section 2.3) cf., goal diversity. Thus, rather than always evaluate against the single 'global' exit we let the point population define pairs of pinball locations during the first cycle of evolution i.e., both initial and exit locations. The insight guiding this is that strong **specialists** that learn how to transition between different local regions might develop during the initial evolutionary phase (first part of Incremental evolution, first cycle of Layered evolution). However, policies are ultimately built against the single exit location alone in the final phase of evolution. No knowledge is necessary to establish what the best locations are for (local) pinball start or finish locations during the first cycle; this is merely a function of the stochastic process for point initialization below, section 4.1.4.

4.1.4 Point Population Routines

The point population specifies the set of initial conditions for the pinball. Fitness evaluation is conducted against the entire content of the point population. Generating P_{gap} new points and creating the initial point population will assume the following routine:

1. Select x and y defining the initial location for the ball (and when applicable the exit location) with uniform probability over the range of the task domain $[0, 1]$;

2. Test the validity of the point(s) i.e., must not be within a wall object or within the global goal, Figure 4.1. Return to Step (1) if this test fails;

3. When applicable, test the candidate exit location to verify that it is not already touching the ball or corresponds to the global goal location.

4. Initial ball velocity is zero or $\dot{x} = \dot{y} = 0$.

4.1.5 Reward Function

Fitness evaluation applies a competitive fitness sharing function (Equation 3.1, Section 3.2.2) to characterize the overall utility of each host. Specifically, each host, h_i, is evaluated against each point, p_k. A domain specific characterization of the reward collected over an episode $G(h_i, p_k)$ is now required. The pinball domain defines an instantaneous reward, $r(t_s)$, of $-1(-5)$ for each use of the no change (any other)

action respectively and a 'goal' reward τ of $10,000$ ([24]). The latter is used here to represent the exploratory 'budget' for the evaluation of each host over each point in the pinball domain. A raw episodic reward, $g(h_i, p_k)$, is therefore defined in terms of the accumulated instantaneous reward: $\sum_{t_s=0} r(t_s)$; thus, $g(h_i, p_k) < 0$.

However, we also apply a dynamic limit on the amount of time a host spends on any one episode. Let us assume that after completing each episode a corresponding accumulated reward is estimated and normalized relative to τ. Thus, relative to point evaluation $k : \hat{\tau}(h_i, k) = \tau + \frac{1}{k} \sum_{j=0,\ldots,k-1} g(h_i, p_k)$ i.e., $\hat{\tau}(\cdot)$ is always positive. Thus as long as the difference between the remaining evaluation budget and episode specific reward is positive, the evaluation of host h_i against point p_k continues. Hosts that perform well on some subset of points will thus be allowed incrementally more time to explore in each episode. The reward collected over an episode can now be characterized as follows:

$$\begin{array}{cc} \text{IF} & (\hat{\tau}(h_i, k) + g(h_i, p_k)) > 0 \\ \text{THEN} & G(h_i, p_k) \leftarrow 1 + \frac{g(h_i, p_k)}{\tau} \\ \text{ELSE} & G(h_i, p_k) \leftarrow 0 \end{array} \tag{4.1}$$

where the assumption is made that the budget for the first episode of host h_i assumes a value of $\hat{\tau}(h_i, 0) = 500$, thereafter the estimated values from each completed episode(s) are employed. The motivation for this exploratory budget stems from the relatively long compute times required when, without the dynamic budget, each episode potentially consists of up to $10,000$ time steps.

4.1.6 Summary

The pinball environment follows that established by [24]. State variables and atomic actions are summarized in Table 4.1. [24] conducted training relative to two start locations for the pinball and measured performance relative to progress made on these two configurations during training only. They did not report performance w.r.t. independent test configurations. We conduct post training evaluation against 500 initial pinball locations and the single common goal location, Figure 4.1. Naturally, there is no guarantee that the test point initializations will be encountered during training.

Table 4.1: Task Domain Parameterizations.

Pinball domain	
Total host evaluation budget (τ)	$10,000$
State variables	$\{x, y, \dot{x}, \dot{y}\}$
Atomic actions (a)	$\{0, \pm\dot{x}, \pm\dot{y}\}$
Truck Reversal domain	
Distance traveled per time step	1.0m
Length of Cab	6.0m
Length of Semi	14.0m
Max episodic simulation steps (D_{max})	600
State variables	$\{x, y, \theta_c, \theta_s\}$
Atomic actions (a)	$\{0°, \pm 30°\}$

4.2 Truck Reversal

4.2.1 Task Domain

Truck Reversal is a complex, non-linear control task in which the agent must back a semi-cab up to a loading dock. The semi–cab is reversed at a fixed velocity thus only the policy for steering needs to be defined [1]. An additional wall obstacle is introduced in this work as per [32]. The introduction of the wall makes the task significantly more difficult than without as 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-left 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. In addition to the initial direction of the semi–cab, the wall therefore renders the problem deceptive;

2. Constraints are enforced on illegal behaviours such as jackknifing and colliding with obstacles and;

3. Training configurations are defined stochastically by the content of the point population (c.f., the tabula rasa assumption).

4.2.2 Previous Results

[32] demonstrated that SBB is capable of building effective policies for the 'difficult' configuration of truck reversal task under Monolithic and Modular evolution with the *standard* reward function derived from [27]. Modular evolution, specifically 2-level hierarchical SBB, proved beneficial for this task. Earlier results are limited to the original (obstacle free) formulation of the truck reversal problem [40]. An emphasis was placed on demonstrating the potential for problem decomposition [18, 21], albeit with hand designed task decompositions. Koza evolved solutions relative to 8 hand crafted semi–cab configurations and did not report performance under an independent set of test conditions [27]. We verify that the solution from Koza was sufficient for solving the task without the wall, but failed entirely when the object was included. Even ADFs, as reported on in (A) fail to provide solutions to anything but the simplest of test points under the 'difficult' configuration. Evolving neural networks has also been considered, although again this appears to have taken place relative to hand designed training scenarios [48]. In this work we introduce a new, specialized fitness function for the truck reversal domain and explore the role of developmental diversity within SBB.

4.2.3 Atomic Action and State Variables

During evolution the point population specifies starting configurations of the semi–cab; whereas the host–symbionts assume responsibility for providing the steering behaviour to return the semi–cab back to the origin i.e., a single goal state under a constant rate of reverse [1]. The state variables take the form of cartesian coordinates identifying the end of the semi, (x, y), the angle of the cab, θ_c, and the semi, θ_s (Figure 4.2). As such, this is similar to the information provided by a GPS; where the goal is to reverse the semi–cab back to within some tolerance of the origin, but *without* any local information. The single atomic action a that each symbiont at level 0 may assume were selected from the set $\{0°, \pm 30°\}$, and a training episode was terminated when: (1) the back of the semi crossed the y-axis, (2) the semi–cab jackknifed, (3) the semi–cab could not return to the origin within some maximum number of time

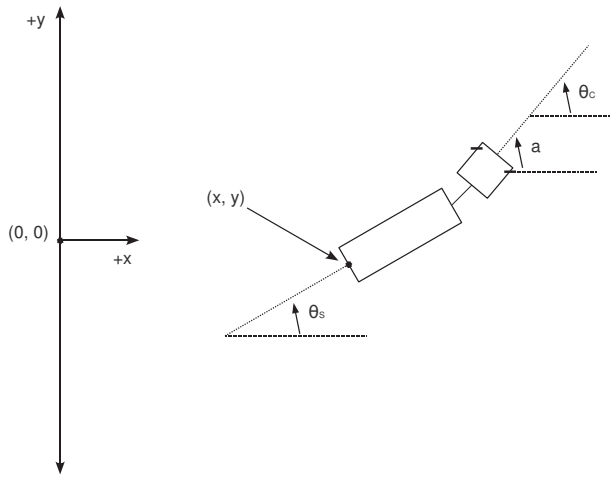


Figure 4.2: Truck reversal state variables. The controller is supplied with cartesian coordinates denoting the end of the semi, (x, y), the angle of the cab, θ_c, and the semi, θ_s. Not shown is the global 'field' in which the semi–cab may begin or the obstacle in the middle of the field (see Figure 4.3).

steps [2], or (4) the back of the semi collided with the wall.

4.2.4 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;
3. Select θ_s (in radians) with uniform p.d.f. under the interval $(-\pi, \pi]$; and,
4. Let $\theta_c = \theta_s$.

4.2.5 Reward Function

The SBB algorithm as described defines fitness under a generic model of fitness sharing (Equation 3.1, Section 3.2.2), which then requires a domain specific episodic reward

[2]Enforced by setting a limit of D_{max} where this is estimated as: steps used so far *plus* steps 'as the crow flies' from the current location to the origin.

function, $G(\cdot, \cdot)$. 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 4.2.3, the corresponding final values for x, y, and θ_s are used to provide the reward function:

$$G(h_i, p_k) = \frac{1}{\sqrt{x^2 + y^2 + \theta_s^2 + 1}} \tag{4.2}$$

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 jackknife near the origin.

4.2.6 Summary

This work adopts the original formulation of the truck reversal domain in terms of the dynamics of the semi–cab ([40]) and adds the wall obstacle. 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 4.3; there is naturally no guarantee that such configurations will be encountered during training. The task parameterization is summarized in Table 4.1.

4.3 Parameterization

In the following we summarize a common parameterization for SBB as well as introduce our approach to establishing a baseline for the relative difficulty of the task domains. In the latter case we employ the widely utilized SARSA value function approximation approach to reinforcement learning.

4.3.1 SBB Parameterization

Parameterization of SBB is summarized in Table 4.2. Needless to say, no claims are made regarding the optimality of these parameters. Indeed most of the parameterization is carried over from that previously established for supervised learning (no capacity for constructing policy trees) [35, 16]. Two parameters were varied between

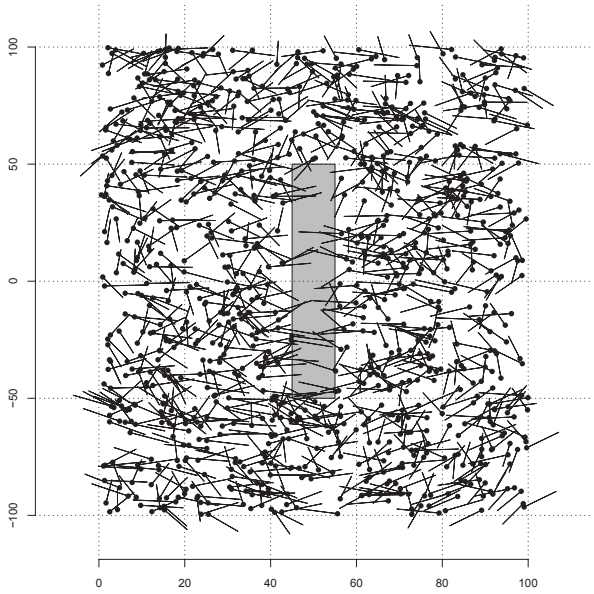


Figure 4.3: Truck Reversal Task: Summary of the 1 000 initial conditions for post training testing evaluation of solutions from the Truck Reversal domain. The goal is to return the truck to the origin $(0, 0)$ without triggering a fault condition. The rectangle at the centre of the world represents the wall obstacle. The 'pinhead' location denotes the end of the semi.

tasks. Under the pinball task a value of 10 was sufficient for the maximum number of symbionts per host (ω) to avoid the upper bound being reached whereas this was increased to 15 under truck reversal. A smaller number of generations (t_{max}) was sufficient in the pinball experiments in order to save time without compromising the goal of our experiments. In total each experiment is conducted over 60 independent runs.

4.3.2 SARSA Value Function Approximation

The focus of this research lies in policy search as opposed to value function approximation. However, we also include baseline results for each task domain using the SARSA value function method with an ϵ-greedy stochastic component [56] under a recently proposed Fourier basis [26] and the widely utilized approach of tile coding. In both cases the source code is available as part of the RL-glue project.[3] The principle motivation for including a value function approximation method here is to establish

[3]http://glue.rl-community.org

Table 4.2: Parameterization of Host and Symbiont populations. t_{max} reflects the generation limit per layer. 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 (105)	ω	10 (15)
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

Table 4.3: Parameterization for SARSA value function approximation.

Parameter	Pinball task	Truck reversal task
learning rate (α)	0.001	0.25
eligibility trace (γ)	1.0	1.0
discount factor (λ)	0.9	0.25
exploration rate (ϵ)	0.01	0.05

a baseline for generalization under each task. In the case of the pinball task we could make direct reference to a previously established parameterization for the Fourier basis of order 4 [24]. Under the truck reversal domain extensive benchmarking was necessary to identify the relevant learning parameters, eventually resulting in a preference for the tile coding under this task. The resulting SARSA learning parameters are summarized in Table 4.3.

In the case of the truck reversal task it is also necessary to define the reward as provided by the environment. A considerable amount of experimentation was necessary in order to provide acceptable results. Specifically, best results were achieved with a per time step reward of 0 (as opposed to -1 as in the pinball task) with a cost of -600 encountered should a jackknifing condition occur. The end of episode reward took the form of: $C - \sqrt{x^2 + y^2 + \theta_s^2 + 1}$ and is therefore similar to that employed for SBB (see Equation 4.2). The best performance was obtained when the constant C took the value of zero (as opposed to a large positive value).

Chapter 5

Empirical Evaluation

5.1 Standard Evaluation Practices

SBB evolves a population of policies 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. Such a set is established using the point initialization routine to create validation points. The individual with the most solutions across the validation set establishes the representative 'champion' behaviour from each run. An independent test set is created in the same way for assessing solution robustness/ generalization, complexity, and other post-training characteristics of interest. We use 500 validation points for Pinball and 1000 for the Truck reversal task. Given the number of degrees of freedom in the tasks, there is little likelihood in test cases being encountered during training or validation and there is no bias in sampling cases that are later encountered during test. Except where noted, all results are analyzed with respect to 60 independent trials in order to ensure statistical significance.

5.2 Pinball Domain

5.2.1 Evaluation Methodology

A total of four primary experiments will be considered:

Case 1 – Single level, common goal (Monolithic evolution): SBB only builds a single level, thus no capacity exists for defining hierarchical policies. Relative to the parameterization of Table 4.2, the symbiont per host and program length limits are doubled ($\omega = 30$, $maxProgSize = 96$). Members of the point population specify a unique start position relative to the common goal, Figure 4.1.

Case 2 – Single level, goal diversity (Incremental evolution): SBB again builds a single level with parameterization as in Case 1. This time developmental diversity is introduced by altering the task half way through evolution. For the first

105 generations, members of the point population define goal locations as well as start conditions for the pinball. Evolution is then conducted relative to the common 'global' goal for the last 105 generations, Figure 4.1.

Case 3 – Two level, common goal (Modularized evolution): This scenario introduces hierarchical policy discovery (two levels), in which all training episodes are conducted with respect to the single 'global' goal or maze exit, Figure 4.1.

Case 4 – Two level, independent goals (Layered evolution): Hierarchical policy discovery again appears (two levels), but this time developmental diversity is introduced as level 0 lets the point population define goal locations as well as start conditions for the pinball. Evolution at level 1 is again conducted relative to the common 'global' goal.

5.2.2 Generalization Tests

500 initial pinball locations (sampled with uniform probability), or the small grey balls in Figure 4.1, are used to assess post training performance of a single 'champion' host from each trial. Naturally, the single global goal remains the same under all test conditions i.e., find a path to the 'exit' hole at the top centre of the maze. A violin plot summarizes the resulting distribution (over 60 trials) for each of the above four SBB configurations and SARSA, Figure 5.1. When more than one cycle of evolution is present, such as in modular and layered evolution, only the champion at the highest level is considered. The wide variation in the distribution of SARSA solutions illustrates that the task is indeed nontrivial. All SBB solutions demonstrated better consistency.

The single level, monolithic experiment effectively establishes a median base line in which 50 test cases cannot be solved (Monolithic in Figure 5.1). Introducing developmental diversity within a single cycle of evolution (Incremental in Figure 5.1) results in statistically significant improvement in generalization performance over Monolithic evolution. By letting points specify arbitrary goal locations in the early stage of evolution we are forcing agents to first learn partial solutions, or *specialist* behaviours, prior to introducing the ultimate task goal and thus rewarding generalization. This ordering of specialization followed by generalization is one aspect of this work that distinguishes it from other approaches to incremental evolution [4]. Another major

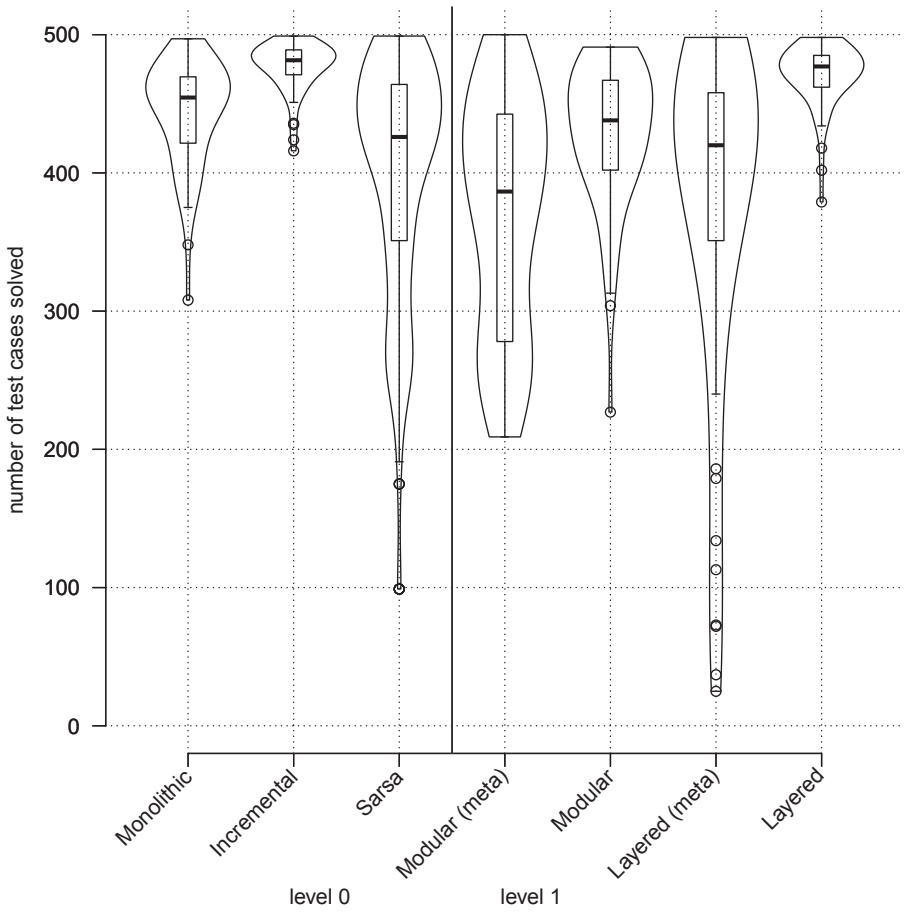


Figure 5.1: Pinball Task: Generalization performance or count of number of test points solved by the champion host (y-axis) at each level. 500 test points in total. x-axis labels correspond to 4 different styles of SBB evolution. Distributions labeled 'X(meta)' describe the cumulative number of test cases solved by all meta actions when evaluated individually as oppose collaboratively within hosts of distribution X, see Section 5.4. SARSA is the baseline Fourier Basis value function approximator. Vertical line in x-axis marks split between level 0 and 1.

difference is that no a priori information is necessary to define specialist learning tasks. Where other work [4, 61] required human intervention in designing appropriate subtasks, this work relied on the stochastic point initialization process alone. However, it could be argued that the simplicity of the task domain largely makes this possible rather than the SBB framework. Indeed, a similar approach to automating subtasks was infeasible under the truck reversal domain, see Section 5.3.1.

Adding an additional cycle of evolution (at a corresponding reduced host size, program limit and generation per level limit) is unable to provide any improvement when the same global goal condition is retained at both levels (Modular in Figure 5.1). Indeed, the use of a common goal for both levels of evolution results in a much lower solution consistency as compared to Monolithic or Incremental evolution. Conversely, letting the point population define arbitrary target locations during the evolution of meta actions (level 0) and reintroducing the global goal location at level 1 provides a statistically significant improvement relative to either baseline without developmental diversity (compare Layered to Monolithic and Modular in Figure 5.1). Table 5.1 summarizes p-values under the Mann-Whitney non-parametric hypothesis test. There is no significant difference in the generalization performance of cases employing developmental diversity (Incremental and Layered evolution), both significantly outperform their respective baselines (Monolithic and Modular). The natural implication of this is that encouraging diversity in the early stages of evolution (at level 0 in layered evolution) is significant in constructing 'good' generalist behaviours later on.

When a hierarchical structure is present, as is Layered evolution, goal diversity at level 0 helps to produce a population of hosts (meta actions) that specialize on solving some subset of training scenarios. When reused and re-contextualized by higher-level hosts, these (specialized) meta actions prove to be more useful building blocks than meta actions developed relative to the overall task (generalists). Moreover, these meta actions may be deployed relative to *any* valid generalization task of the same domain, reinforcing the value of archiving meta actions in the form of an RTL as oppose to developmental diversity without archiving (Incremental evolution). This versatility is explicitly tested in section 5.2.5

Table 5.1: Pinball Task: p-value for pairwise Mann-Whitney non-parametric hypothesis tests.

Pairwise comparison	p-value
Modular vs. Monolithic	0.1228
Modular vs. Incremental	$1.286e-10$
Modular vs. Modular (meta)	0.0001803
Layered vs. Monolithic	$2.903e-06$
Layered vs. Incremental	0.079
Layered vs. Modular	$7.199e-08$
Layered vs. Layered (meta)	$2.097e-08$
Incremental vs. SARSA	$2.2e-16$
Layered vs. SARSA	$1.56e-13$

5.2.3 Specialization and Generalization in Hierarchical SBB

Further insight into the role of population diversity and specialization during the initial cycle of hierarchical policy search can be obtained by looking more closely at the Layered evolution experiment. Figure 5.2 describes the test performance of the single best level 0 host (meta action) and level 1 host from Case 4 (Layered evolution) as well as the corresponding cumulative *population wide* performance. That is to say, the cumulative population wide performance counts the total number of unique test cases solved by all members of the host population as opposed to the single 'champion' host detailed in Figure 5.1. It is now apparent that meta actions (champ level 0) typically solve 70 percent of the test cases. However, when measuring the cumulative population wide performance of meta actions (pop level 0) all the test cases might have solutions. Thus, competitive fitness sharing has successfully cached a diverse set of specialist behaviours across the host population.

These behaviourally diverse level 0 hosts thus represent good candidate meta actions for host–symbiont development at level 1. The level 0 hosts, or meta actions remain fixed after the first cycle of evolution. This is how hierarchical SBB supports 'complexification' without falling into the over-learning trap. Meta actions themselves do not evolve, level 1 hosts evolve (learn) appropriate contexts in which to deploy meta actions. Thus, the champion performance at level 1 (champ level 1, Figure 5.2) generally matches the cumulative population wide performance at level 0 (pop level 0) with a much tighter consistency. Hosts at level 1 have thus been able to

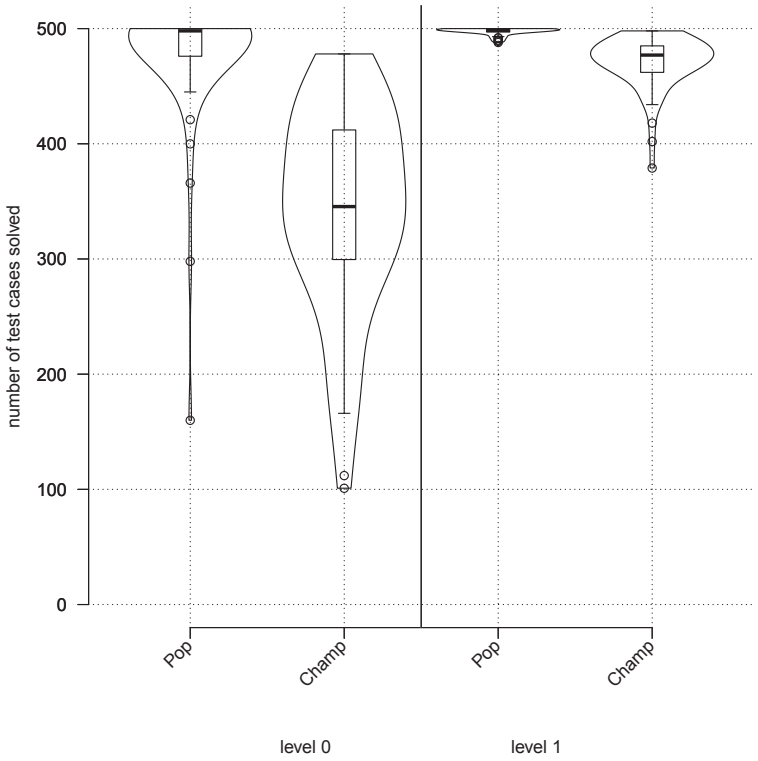


Figure 5.2: Pinball Task: Performance of Case 4 SBB (Layered evolution) hosts over 500 test points. 'Pop' identifies columns with cumulative solution count performance as estimated across the entire final population whereas the other columns denote performance of the single best individual from a run. champ+ is the collective performance of meta actions as identified by champion policy trees. Vertical line in x-axis marks split between level 0 and 1.

leverage the diversity of meta actions to form a single highly fit solution at level 1. Note, however, that the above analysis is all post evolution, whereas during evolution effective hierarchies need to be discovered without reference to such 'global' pictures of host capability.

5.2.4 Hierarchical Policy Deployment

In order to demonstrate the application of meta actions evolved at level 0 we need to establish to what degree their purposeful application and reuse appears. Thus, in order to provide some insight into this we consider both structural and behavioural aspects of a specific Case 4 solution solving 492 of the 500 test cases.

Figure 5.3 summarizes the architecture for this particular solution. The *large* circle – top centre – denotes the single level 1 host or root of the decision tree. This

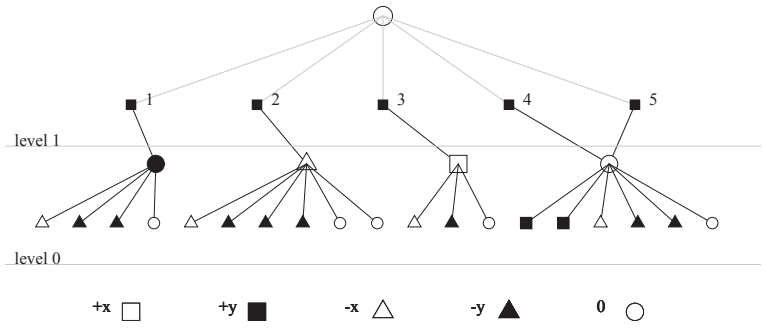


Figure 5.3: Pinball Task: Structure of a hierarchical SBB solution solving 492 of 500 test cases. Top-centre circle represents level 1 host. Black squares represent level 1 symbionts indexed by this host. Each level 1 symbiont assumes one level 0 host as its meta-action, represented here by the black/ white circle/ square. Each level 0 host indexes multiple level 0 symbionts. The shape of each level 0 symbiont denotes which atomic action is assumed (see legend at the bottom of the figure).

level 1 host is comprised of 5 unique symbionts, or *small* black squares labeled 1 through 5. Each of these level 1 symbionts assumes a meta-action as defined by a host evolved at level 0, or the *large* black circle, white triangle, white square, and white circle of Figure 5.3. The level 0 hosts index a subset of level 0 symbionts each with a corresponding (task specific) atomic action. Naturally, as each symbiont learns a unique context for deploying its action, the same atomic action may appear in multiple places at level 0. Likewise, multiple level 1 symbionts may assume the same level 0 host as their action, each providing a unique context in which to deploy the same meta-action. Figure 5.3 supplies a legend mapping level 0 symbiont symbols to corresponding atomic actions.

Some insight into the degree of reuse and therefore the potential for supporting temporal abstraction (deploying a meta-action under multiple contexts) is now apparent. At level 1 symbionts 4 and 5 both use the same host 0 policy, implying that the level 1 symbionts have learned to deploy it under different conditions. Likewise, a host at level 0 in three of the four scenarios includes multiple symbionts with the *same* atomic action (the three symbionts associated with the 3rd level 0 host being the only exception); implying that multiple contexts appear for the same symbiont action.

Having established the basic structure of a hierarchical policy, we can now map

the states at which level 0 hosts are deployed by symbionts from the level 1 host (root node of the decision tree) relative to a specific test condition, Figure 5.4. The start condition corresponds to the large grey circle (bottom left) and global goal is the large black circle (top centre). The overall trajectory is expressed in terms of shapes corresponding to the level 0 host deployed at each time step (defined by Figure 5.3).

At least three factors are now readily apparent: 1) the degree of interleaving or cooperation between different meta-actions (level 0 hosts); 2) the relative specialization of meta-actions; and, 3) the degree of reliance on wall bouncing/ exploration for reorientation of the pinball versus linear/ greedy exploitation of a given direction at different parts of the trajectory. Thus, for example, the level 0 host/ meta action represented by a white triangle seems to be deployed predominantly on the left side of the board while the white square, white ellipse, and black ellipse hosts are used predominantly on the right side. This decomposition is a direct result of the communication between level 1 symbionts at each time-step. The level 1 symbionts have learned appropriate contexts for their meta-actions and are able to express this information through bidding.

We can also summarize the consistency of bidding behaviours across all test points by plotting the (x, y) co-ordinate for which different level 1 symbionts (thus host 0 deployment) represent the winning bid, Figure 5.5. The resulting spacial organization clearly reflects a specific distribution of geographically-specialized host 0 behaviours in which only the case of (level 0) host 4 appears to be redundant. Moreover, it is clear that multiple hosts from level 0 are involved in providing solutions to all but the simplest of test cases.

Finally, we note that it is also possible to review the same trajectory in terms of which domain-specific atomic actions are deployed at each time-step, Figure 5.6. It is apparent here that level 0 hosts in general have learned to use their actions sparingly, most often applying no force to the ball (action 0, circle) and thus incurring minimal penalty and maximizing their reward. However, the application of other forces, such as de-acceleration, appears to follow a systematic pattern of application before entering regions where significant amounts of oscillation between walls are necessary (the regions of oscillation appearing as the principle mechanism by which the direction of travel is changed).

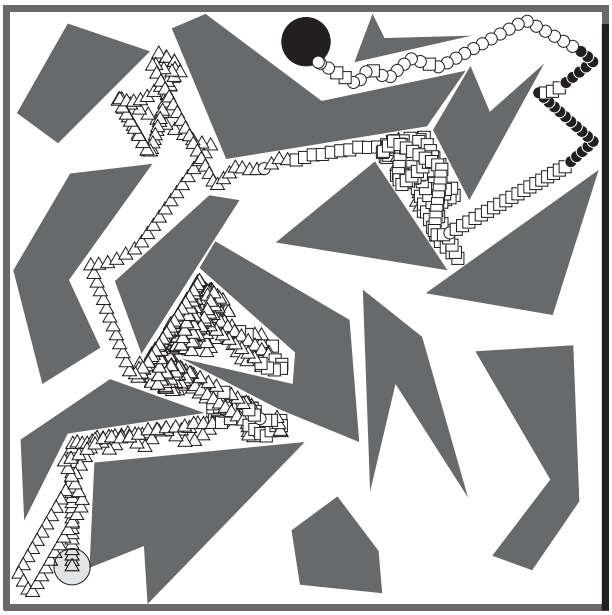


Figure 5.4: Pinball Task: Sample trajectory in terms of meta-action deployment w.r.t. hierarchical policy of Figure 5.3. Large grey circle represents start location for this specific test case. Large black circle represents the global target. Shapes correspond to *level 0* hosts (meta-actions) deployed at each location/time-step in trajectory, as mapped to specific hosts in Figure 5.3.

5.2.5 Learning and Organizing Meta Actions

In the context of evolving hierarchical SBB policies, the role of hosts at level 0 is to learn specialized behaviours that do not necessarily provide complete solutions to the problem but instead are able to solve some relevant subgoal (Figure 5.5). However, this should be an emergent property, with no requirement to define these subgoals manually using a priori information. As described in Section 3.1, this work uses training points in level 0 that define *both* an arbitrary start location for the ball and an arbitrary (sub)goal location. Fitness sharing between hosts encourages each level 0 host to specialize at solving a unique subset of these training points. This diversity is then exploited by level 1 hosts as they autonomously identify subsets of level 0 hosts to use as meta actions.

Figure 5.7 summarizes the set of level 0 *training points* that the host identified by the black circle in Figure 5.3 was able to solve. Goal locations are represented by translucent squares and ball start locations are represented by circles. As individual points define one target location and one ball starting location, each ball has a hairline

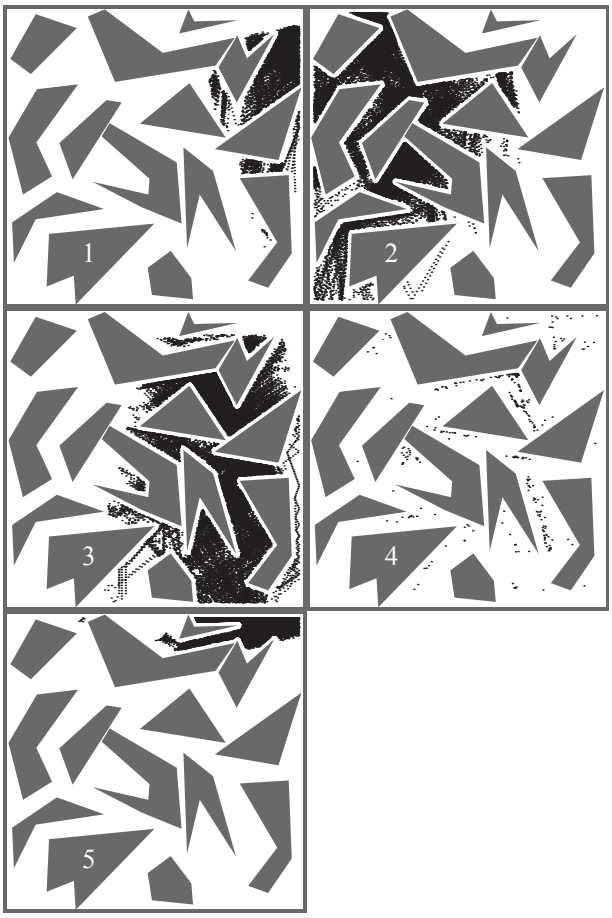


Figure 5.5: Pinball Task: Distribution of winning bids across all test conditions w.r.t. level 1 symbionts of Figure 5.3. Subsampling applied equally to each sub-plot limit total figure size to 600KB from 6MB.

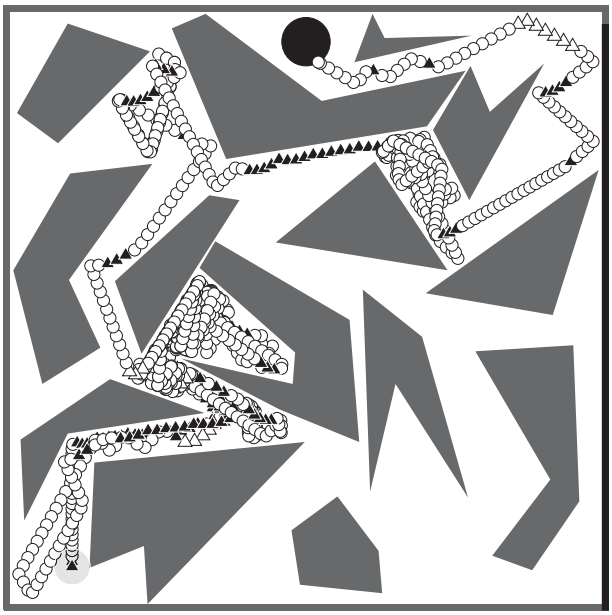


Figure 5.6: Pinball Task: Sample trajectory in terms of atomic actions deployment w.r.t. hierarchical policy of Figure 5.3. Large grey circle represents start location for this specific test case. Large black circle represents the global target. Shapes correspond to *atomic actions* deployed at each location/time-step in trajectory, as mapped to domain-specific actions in Figure 5.3.

protruding toward its corresponding target. The length of the hairline is relative to the length between the ball and target. Likewise, each target has a line of the same length protruding from it towards its associated ball. Clearly, this host focused on solving points with target locations in the upper-right and upper-left corners of the Pinball domain. Naturally, the target locations in these clusters can be seen to represent subgoals of the ultimate problem of reaching the goal location in Figure 4.1. Referring back to the final deployment under the specific test configuration / solution trajectory of Figure 5.4 we note that this particular meta action is deployed in the area corresponding to greatest point density during training. Likewise this also correlates with the region in which the same level 1 symbiont has the winning bids (Subplot 1, Figure 5.5). The level 1 host depicted in Figure 5.3 was thus able to autonomously organize a subset of symbionts indexing useful meta actions from a large set of symbionts with a diverse group of potential meta actions.

To further illustrate this we can consider one final experiment identical to Case 4 (Layered Evolution) above except that level 1 training points defined a target location

Figure 5.7: Pinball Task: Set of training points (pairs of start and goal locations) that the black circle level 0 host from Figure 5.3 was able to solve. Ball start locations are represented by circles while goal locations are represented by translucent squares. Each ball/goal has a hairline line protruding toward its corresponding goal/ball. The length of the hairline line is relative to the length between the ball and target.

at the opposite side of the Pinball world from that in Case 4, in the bottom-center of the world. Again, we select the best level 1 host and review the training points solved by one of the the meta actions it uses, Figure 5.8. Clearly, this meta action specializes at solving target locations near the bottom centre of the world. Significant here is that the top level host has selected a meta action from the library of available actions which is specifically appropriate for the alternate goal. This suggests that the library of meta actions (specialists) is diverse enough to be reused under a variety of global tasks. Selection of an appropriate subset of meta actions from the library relative to a particular task and learning the relevant context in which to deploy each action thus represents the goal of the highest level of Layered evolution. This represents a key advantage to Layered over Incremental evolution, where no such library is constructed.

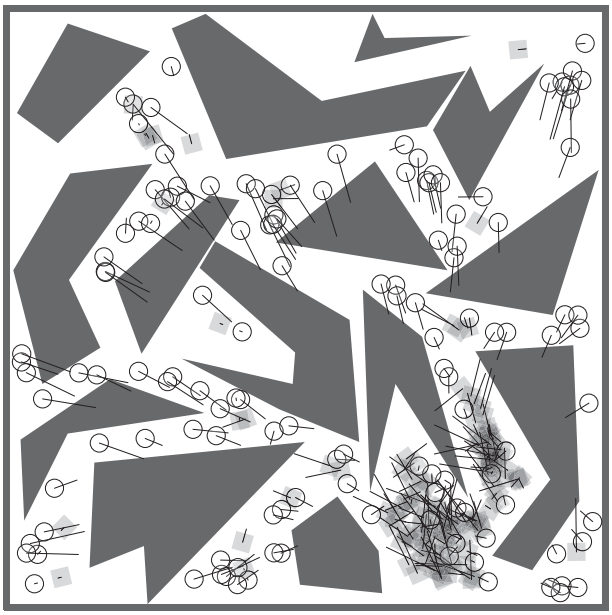


Figure 5.8: Pinball Task: Set of level-0 training points solved by one meta action later deployed w.r.t the alternate global goal. Ball start locations are represented by circles while goal locations are represented by translucent squares. Each ball/goal has a hairline line protruding toward its corresponding goal/ball. The length of the hairline line is relative to the length between the ball and target.

5.2.6 Solution Complexity

From an architectural perspective, the relative counts for the number of symbionts deployed per level and corresponding instruction counts (post intron removal), give some insight to the relative complexity of solutions. Figure 5.9 summarizes the symbiont counts per host as a function of experiment (Monolithic, Incremental, Modular, Layered) and layer (level 0 and level 1). The non-hierarchical scenarios (Monolithic and Incremental) naturally result in hosts with large variance and high single layer symbiont counts (column Monolithic, Figure 5.9). Recall that single-level experiments are parameterized for double the maximum number of symbionts. Also note that symbiont counts per host for Incremental and Monolithic evolution are nearly double the per-level symbiont counts in multi-level experiments. However, hierarchical solutions are still more complex overall, care of their capability to redeploy multiple meta actions. Whether it is easier to understand how 5 meta actions are redeployed versus 8 symbionts is very much in the eye of the beholder.

In the case of instruction count, a similar pattern is noted (Figure 5.10). However,

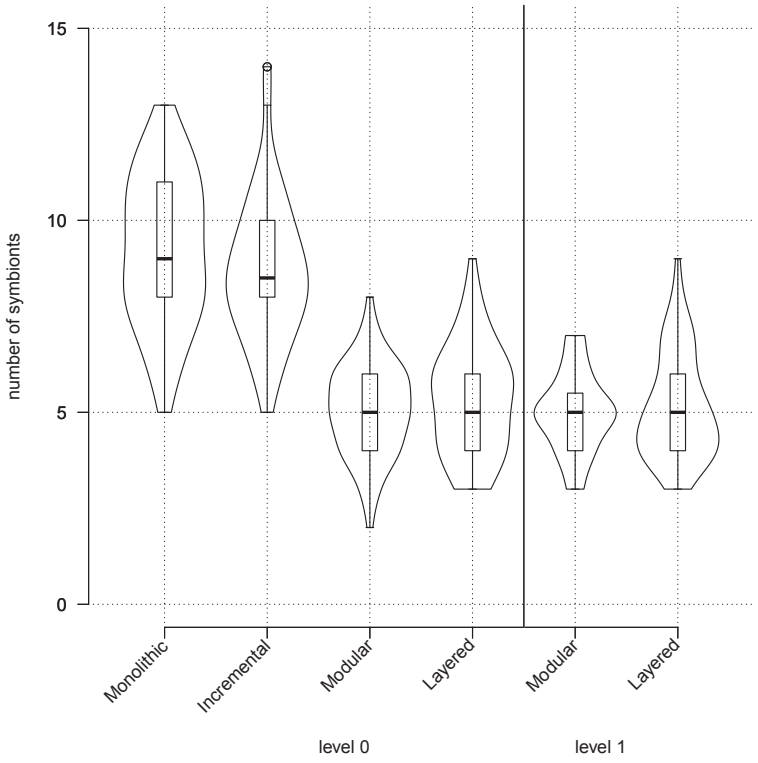


Figure 5.9: Pinball Task: Symbiont Counts per Champion Host. x-axis labels distinguish between experimental cases. Vertical line in x-axis marks split between level 0 and 1.

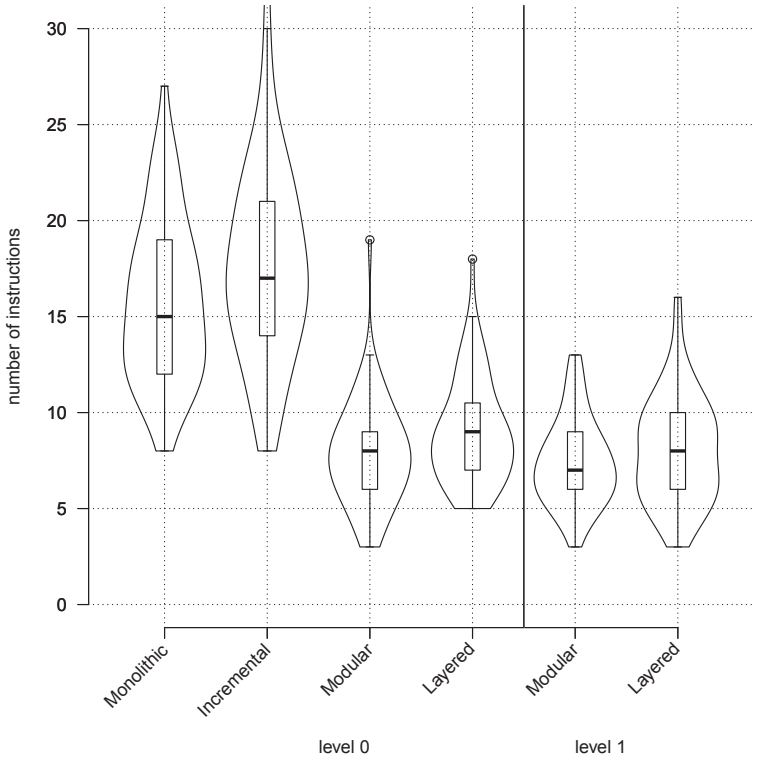


Figure 5.10: Pinball Task: Average Instruction Counts per Symbiont. x-axis labels distinguish between experimental cases. Vertical line in x-axis marks split between level 0 and 1.

a significant increase in the level 0 instruction count appears for Incremental evolution and hierarchical SBB with goal diversity (column Incremental, Layered level 0, Figure 5.10). This is most likely an artifact of the more diverse scenarios encountered during training. Again, when SBB is limited to a single level the instruction counts per symbiont also undergo a significant increase. By way of comparison, under the original study of the pinball domain [24], a value function representation assumed a 'Fourier basis' in which between 256 and 1296 basis functions were required *per policy*. Under SBB there are typically 5-10 symbiont programs per host and 7-17 instructions per symbiont program or between 35 to 170 instructions per policy.

5.2.7 Efficiency

Solution Efficiency

The relative efficiency of solutions can be measured by counting the number of time steps used in each successful episode. Each violin plot in Figure 5.11 outlines the

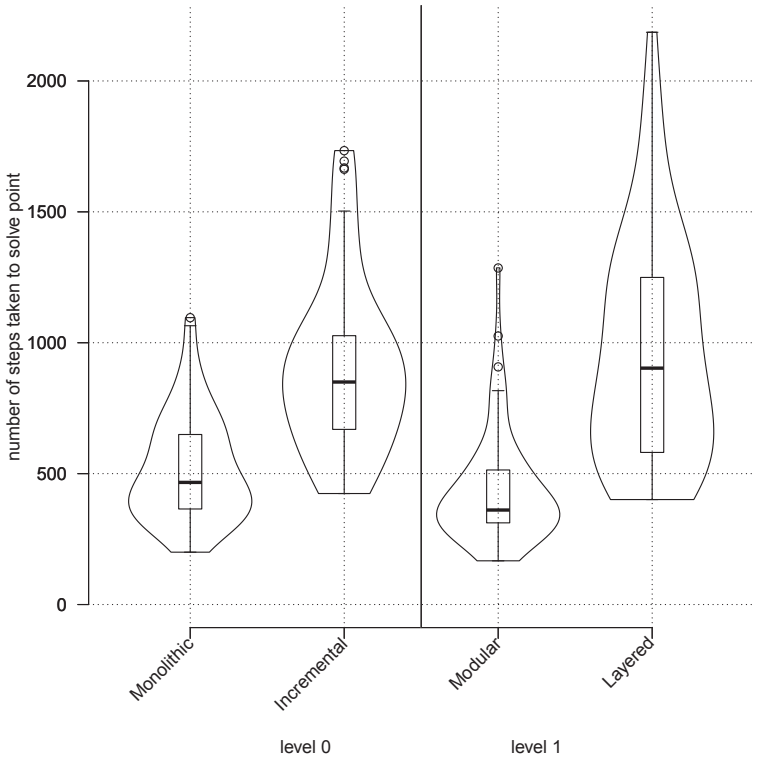


Figure 5.11: Pinball Task: Median number of time steps used by champion over all successful episodes. x-axis labels distinguish between experimental cases. Vertical line in x-axis marks split between level 0 and 1.

time steps used by the champion at the highest available level of evolution in each experiment. As these plots are relative to the test cases actually solved, generalizing to more solutions naturally results in an increase in the median number of time steps used.

Learning Curves

Most of the analysis up to this point has been relative to post-training, test performance. In this section we look at host fitness over the coarse of evolution to compare the learning efficiency of our Layered evolution experiment against the similar-performing Incremental evolution. Figure 5.12 (a) and (b) express the maximum host fitness over all training points at intervals of 20 generations for each experiment. These plots are similar, with each model having quick initial fitness increase in the first few generations followed by slow steady improvement until the developmental variation, or switch from arbitrary goals to common goal, takes place at generation

105. For Layered evolution, this developmental variation is mirrored by the hierarchical evolutionary transition from level 0 to 1. A transition in performance is seen here as a new symbiont population is initialized with random programs at level 1, while level-0 host–symbionts are now frozen to become meta actions. A new group of cooperating symbionts quickly forms within each level-1 host and fitness again rises as symbiont programs learn to deploy previously-learned meta actions. Incremental evolution undergoes a corresponding but much less dramatic dip in fitness at this point. No archiving of previously learned behaviours takes place, thus *all* host–symbionts continue to evolve relative to the new common goal. In either case, the learning rates of Incremental and Layered evolution seem to benefit equally when developmental variation appears in the form of goal (location) diversity.

5.2.8 Alternate Pinball Worlds

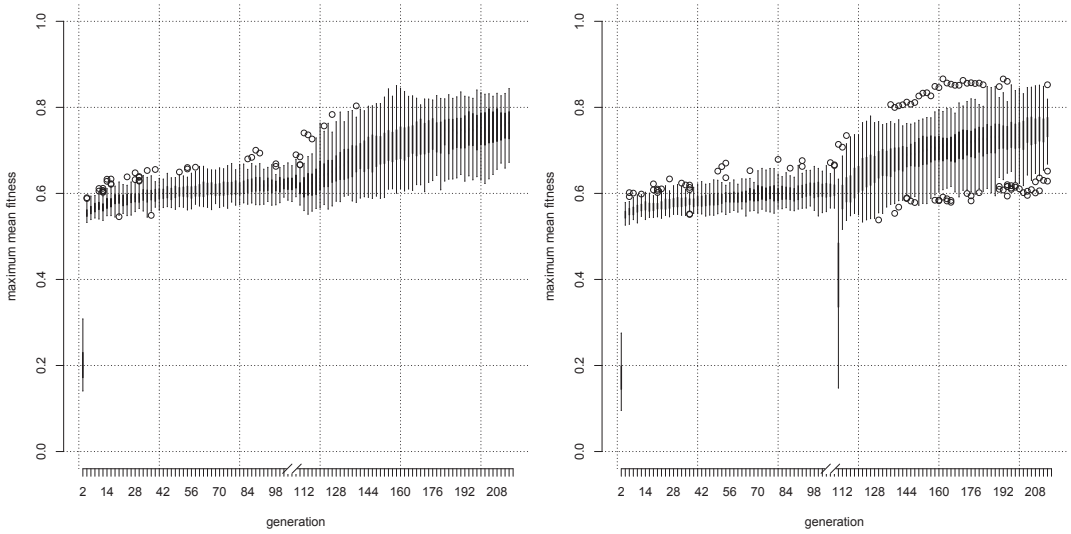
[24] defined two Pinball worlds in their original work with this domain. As it was unclear which configuration provided the more challenging task, we duplicated the above experiments and analysis for the alternate world. No significant difference was apparent in any of our analysis as compared to the above results. Relative performance between SBB configurations remained constant over both Pinball world configurations.

5.3 Truck Reversal Domain

5.3.1 Evaluation Methodology

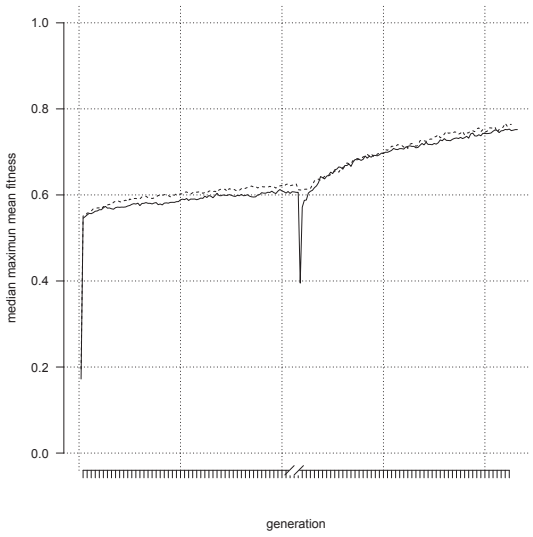
Champion Selection and Test for Generalization

A post training validation set is again utilized to identify a champion from each run (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 (4.2), 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



(a) Incremental

(b) Layered



(c) Combined Medians

Figure 5.12: Pinball Task: Training Curves for Incremental and Layered Experiments. (a) and (b) depict the maximum individual host fitness over 210 generations for each experiment where box plots summarize the distribution w.r.t. 60 independent trials. (c) plots the median from each of the above distributions where the dashed line represents Incremental and solid line is Layered. Gap in x axis marks the point at which developmental variation and hierarchical transition (Layered only) take place.

configurations is created in the same way for assessing solution robustness/ generalization, Figure 4.3. A *test case is considered solved* if the co-ordinates for the stopping state of a trajectory are within the following limits: $|x|, |y| < 1.0m, |\theta_s| < 45°$.

On Developmental Diversity

In the case of the pinball task it was possible to develop stronger specialist behaviours during the first phase of evolution by letting the point population define both the 'entry' and 'exit' locations for the pinball. Indeed, it was not possible to develop effective hierarchical policies without this.

However, the dynamics of the truck reversal task are rather different form Pinball, thus requiring us to take a rather different approach to introducing developmental diversity. Automating goal variation in truck reversal is difficult. For example, the standard goal location in truck reversal lies on the y-axis, which essentially acts as a wall because crossing the y-axis is not permitted. Thus changing the x coordinate of the goal essentially makes the world smaller and potentially places the wall in locations that make solutions impossible. Conversely, changing the y coordinate of the goal will encourage policies to approach the y-axis further from the origin, which does not correspond to a useful subtask relative to the ultimate goal location. Thus, the mechanism we will investigate under the truck reversal domain for promoting stronger meta actions will be to reward both distance minimization and, should the goal criteria be satisfied, the number of time steps taken. The motivation being that rather than promote the learning of subtasks, we wish to promote meta actions that *specialize*. With this in mind, the following **spatiotemporal** episodic reward function is defined:

$$\begin{array}{cc} \text{IF} & (|x| < 1) \wedge (|y| < 1) \wedge (|\theta_s| < \pi) \\ \text{THEN} & G(h_i, p_k) \leftarrow \text{Eqn. } (4.2) + \frac{D_{max}}{t_s} \\ \text{ELSE} & G(h_i, p_k) \leftarrow \text{Eqn. } (4.2) \end{array} \tag{5.1}$$

where the antecedent tests for a valid solution (Section 5.3.1); $D_{max}(= 600)$ is the (episodic) maximum number of interactions between policy and task and t_s is the count of interactions. When satisfied, fitness is expressed as a function of the original distance-based goal function and a normalized count of the number of steps taken

to provide the solution. When not satisfied, fitness is expressed as a function of the original distance-based goal function alone.

A total of four primary experiments will be considered:

Case 1 – Single level, spatiotemporal fitness (Monolithic Evolution): SBB only builds a single level, thus no capacity exists for defining hierarchical policies. Spatiotemporal fitness is used throughout. Relative to the parameterization of Table 4.2, the symbiont per host and program length limits are doubled ($\omega = 30$, $maxProgSize = 96$).

Case 2 – Single level, goal diversity (Incremental Evolution): SBB again builds a single level with parameterization as in Case 1. This time developmental diversity is introduced by altering the task half way through evolution. Spatiotemporal fitness (Equation 5.1) is employed for the first 1000 generations. Evolution is then conducted relative to the standard fitness measure (Equation 4.2) for the last 1000 generations.

Case 3 – Two level, spatiotemporal fitness (Modular Evolution): This scenario introduces hierarchical policy discovery (two levels) and applies the spatiotemporal fitness measure (Equation 5.1) at both levels.

Case 4 – Two level, independent fitness measures (Layered Evolution): Hierarchical policy discovery again appears (two levels), but this time additional developmental diversity appears by means of spatiotemporal fitness (Equation 5.1) at level 0, while level 1 employs the standard fitness measure (Equation 4.2).

5.3.2 Generalization Tests

Hosts are evaluated relative to 1000 unique initial domain configurations, Figure 4.3. Figure 5.13 summarizes the performance of each experiment in terms of the number of test cases solved per level. Again, each violin plot depicts the distribution of results over 60 independent trials. Each distribution represents the number of test cases solved by the single best / champion host at the highest level available in each experiment. After 2,000 generations, the champion individuals in our non-hierarchical, Monolithic experiment are able to solve roughly 340 test cases (Monolithic, Figure 5.13). Alternative combinations of standard and spatiotemporal fitness failed to improve on this under the non-hierarchical constraint. For example, incorporating

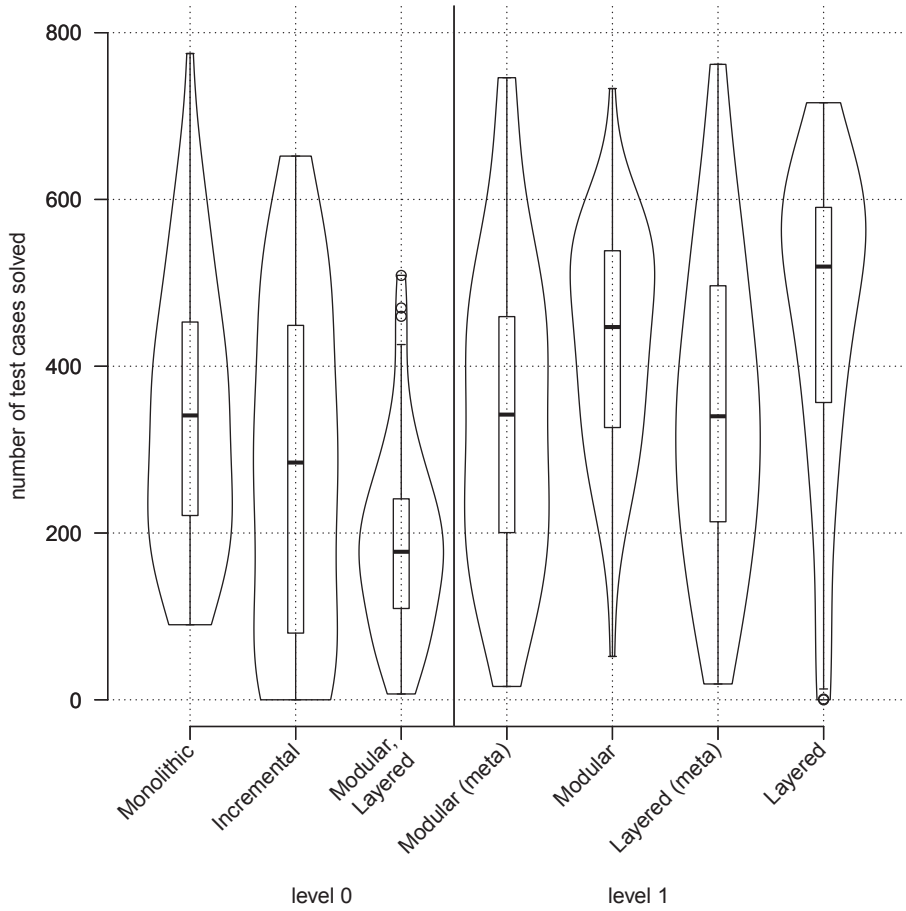


Figure 5.13: Truck Reversal Task: Generalization performance or count of number of test points solved by the champion host (y-axis) at each level. 1000 test points in total. X axis labels correspond to 4 different styles of SBB evolution. Distributions labeled 'X(meta)' describe the cumulative number of test cases solved by all meta actions when evaluated individually as oppose collaboratively within hosts of distribution X, see Section 5.4.Vertical line in x-axis marks split between level 0 and 1.

spatiotemporal fitness and then standard fitness within a single cycle, or Incremental evolution, results in significantly less solutions being discovered by champion hosts (Incremental, Figure 5.13). A trend in the opposite direction was observed with Incremental evolution in the Pinball domain. The difference in behaviour is most likely due to a combination of the difference in the form of ecological diversity employed and the less constrained nature of the task domain.

Modular evolution introduces hierarchical policy discovery with spatiotemporal fitness at both levels. Champion hosts are now typically only capable of solving 180 cases at level 0 (Modular level 0, Figure 5.13). However, the hosts making up the

Table 5.2: Truck Reversal Task: p-value for pairwise Mann-Whitney non-parametric hypothesis test.

Pairwise comparison	p-value
Modular vs. Monolithic	0.001404
Modular vs. Incremental	$3.225e-05$
Modular vs. Modular (meta)	0.003827
Layered vs. Monolithic	0.0001194
Layered vs. Incremental	$2.313e-06$
Layered vs. Modular	0.05277
Layered vs. Layered (meta)	0.0009439

final population from level 0 at generation 1000 now become the set of (meta) actions assumed by level-1 symbionts. These level-1 symbionts evolve contexts in which to deploy meta actions (level 0 hosts). The resulting champion individuals (Modular level 1, Figure 5.13) have succeeded in leveraging the meta actions provided from level 0 to significantly outperform Monolithic and Incremental champions (compare Modular level 1 to Monolithic and Incremental, Figure 5.13). Again, this is in contrast to findings under the Pinball domain where Modular evolution made no improvement over other experiments and in fact represented a decline in generalization performance.

Layered evolution again applies the hierarchical model but now spatiotemporal fitness is used at level 0 only while level 1 uses standard fitness. This further improves generalization performance of the champion at level 1 (Layered level 1, Figure 5.13) as compared to the baseline hierarchical model (Modular level 1, Figure 5.13). Developmental diversity essentially allows the model to produce stronger specialists at level 0 while reinforcing generalization at level 1.

Hypothesis testing confirms the statistical significance of the claims made w.r.t. generalization, Table 5.2.

5.3.3 Sarsa Base Case

The best SARSA results on this task were obtained under a tile coding as opposed to the Fourier basis. Again 60 initializations were conducted, which resulted in 1st, 2nd, 3rd quartile counts for the number of post training test cases solved of $\langle 9.0, 16.5, 30.0 \rangle$. In comparison, the NEAT framework for evolving neural networks under policy search [53] achieves a median of $\approx$ 150 test case solutions ([32], Chapter 7). Both results

emphasize the relative difficulty of the truck reversal (with obstacle) task. In order to provide more insight into what SARSA policies are capable of, the behaviour of a typical policy on the 1 000 test configurations, is reviewed. Figure 5.14 plots both the resulting semi–cab failure conditions and a snapshot of the semi–cab configurations after 100 interactions with the environment. In short, the general strategy identified by SARSA is to avoid jackknifing and the central wall obstacle. However, it fails to locate the goal condition with sufficient accuracy to provide successful solutions. Attempts to improve on this result by increasing the resolution of the tile coding failed; most likely on account of the resulting increase in state–space dimensionality. More complex approaches to value function approximation might well be able to address this short coming (e.g., adaptive tile codings, Chapter 7 in [60]), however, source code is not currently available to support this line of investigation. Review of the training reward indicates that SARSA had reached a performance plateau in all cases. Figure 5.15 represents the training curve for the SARSA policy of Figure 5.14. The initial steep curve indicates that the agent quickly learns to avoid jackknifing. This is followed by an extended period of improvement up to about 20000 episodes before reaching a plateau.

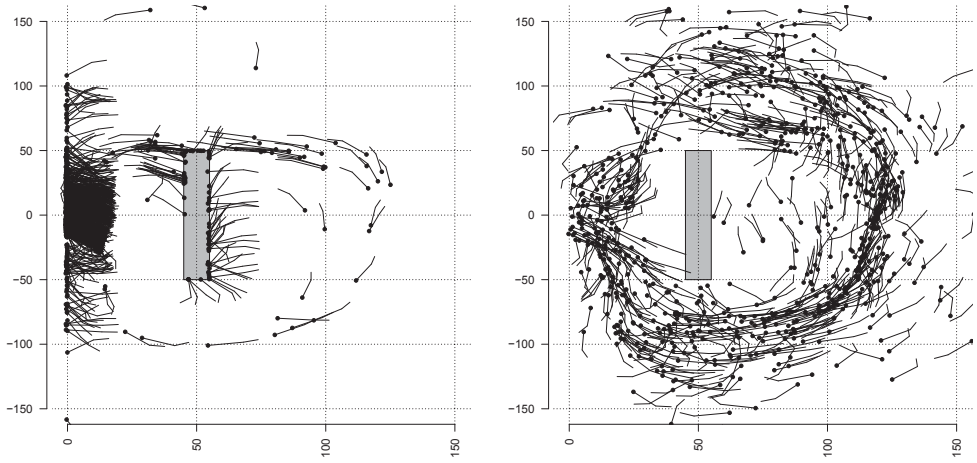
5.3.4 Solution Complexity

As with the Pinball domain, the effects of our unique paramteterizations for single-level and hierarchical experiments can be seen in the number of symbionts per champion hosts and average instruction counts per symbiont, Figures 5.16 and 5.17. However, there are no other significant differences in complexity between experiments. We include these plots for consistency only.

5.3.5 Efficiency

Solution Efficiency

We can again consider the relative efficiency of solutions by looking at the number of time steps used to solve test cases, Figure 5.18. Naturally, applying spatiotemporal fitness throughout evolution introduces a bias for faster solutions e.g., compare Layered and Modular, Figure 5.18. However, this is relative to the number of test cases



(a) Failure conditions (b) State at $t_s = 100$

Figure 5.14: Truck Reversal Task: Typical SARSA policy behaviour during evaluation of truck reversal task under 1 000 test cases. (a) Most failure configurations lie within the y-axis interval of $[-25, +50]$. Some reach the 600 step interaction limit, with the balance hitting the wall object or y-axis at more distant locations. (b) Location of semi–cab relative to all 1 000 test configurations for $t_s = 100$. Compare to Figure 5.21 for the SBB policy tree at the same time step. Note the increase to axis scale.

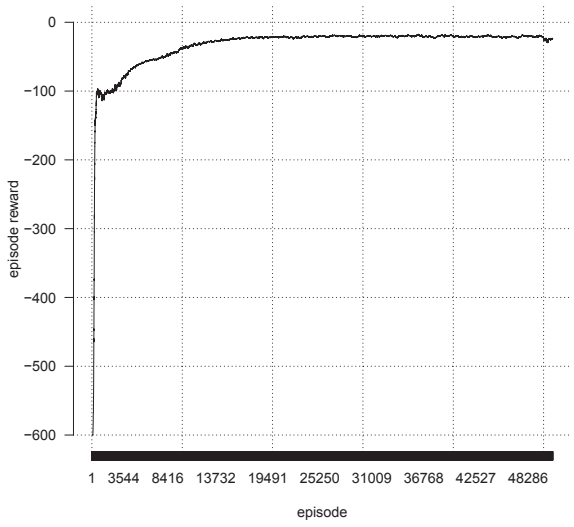


Figure 5.15: Truck Reversal Task: SARSA training reward over 50000 episodes. y-axis denotes median over 60 independent trials.

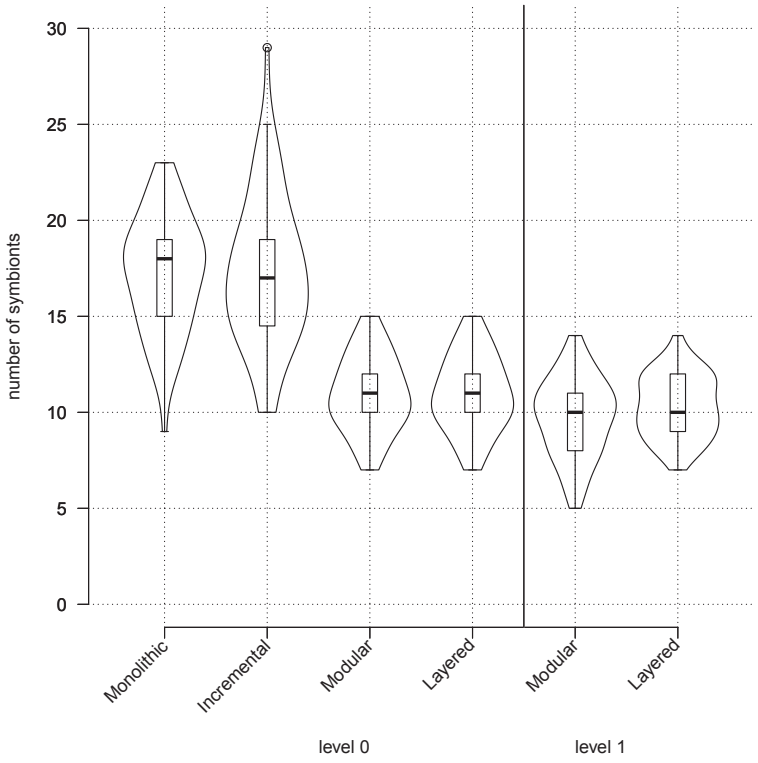


Figure 5.16: Truck Reversal Task: Symbiont Counts per Champion Host. x-axis labels distinguish between experimental cases. Vertical line in x-axis marks split between level 0 and 1.

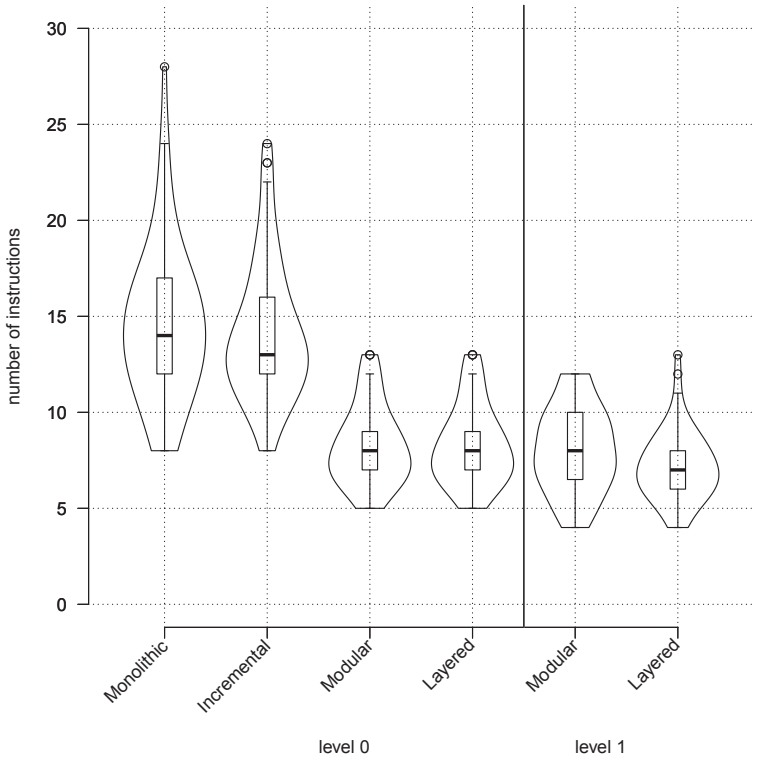


Figure 5.17: Truck Reversal Task: Average Instruction Counts per Symbiont. x-axis labels distinguish between experimental cases. Vertical line in x-axis marks split between level 0 and 1.

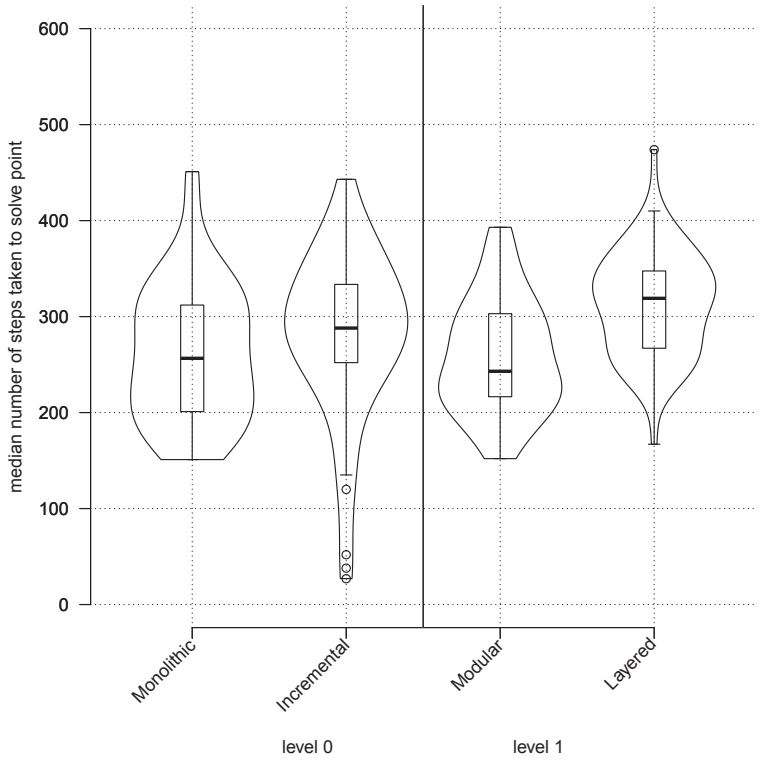


Figure 5.18: Truck Reversal Task: Median number of time steps used by champion host on solved test points. Vertical line in x-axis differentiates between level 0 and 1 distributions.

actually solved. Using more time steps is a natural consequence of generalizing to more test cases. Developmental diversity between levels, or Layered evolution, maintains an acceptable solution efficiency at level 1 while solving significantly more test cases (Figure 5.13). Conversely, the Incremental experiment results in the greatest variance w.r.t. time step efficiency, having some outlier hosts with median number of steps as low as, 27, 38, and 52. However, these outliers have total solution counts of 1, 10, and 8 respectively. Finally, the baseline hierarchical model (Modular) achieves similar efficiency to the single- cycle baseline (compare Modular and Monolithic, Firgure 5.18) while solving 40% more test cases (Figure 5.13).

Learning Curve

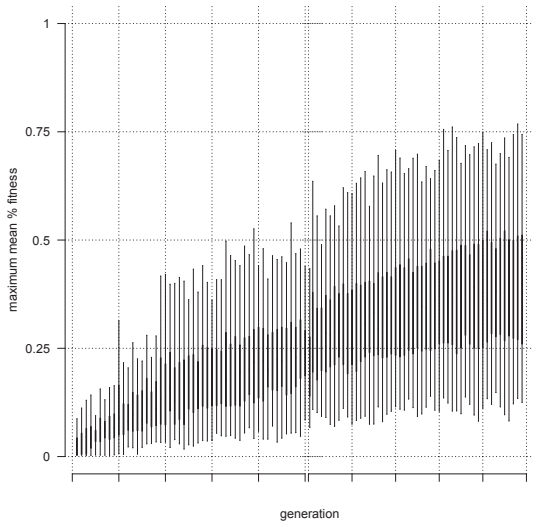
In the case of truck reversal, leverage obtained from the hierarchical transition between levels is clearly visible from the learning curve for Layered evolution as shown in Figure 5.19 (b) and (c). As in Figure 5.12 for the Pinball domain, these plot the

maximum host fitness over all *training* points at intervals of 20 generations for each experiment. The Layered evolution curve undergoes a pronounced jump as the next level of SBB is deployed. Such behaviour re-emphasizes the impact of meta actions as defined by the previously evolved hosts. Incremental evolution, with no archiving of previously learned material, fails to leverage in the same way. Also notable here is the absence of a significant dip in fitness corresponding with the developmental variation after generation 1000. This is because, unlike the Pinball experiments, the global goal remains the same. Therefore, even in the case of Layered evolution where initially random symbiont programs are being deployed to select meta actions, the entire library of meta actions is strong enough relative to the global task that moderate fitness is maintained. As the level 1 host–symbionts learn to deploy meta actions in more appropriate contexts, fitness experiences a further step increase and significantly improves on that of Incremental evolution, Figure 5.19 (c).

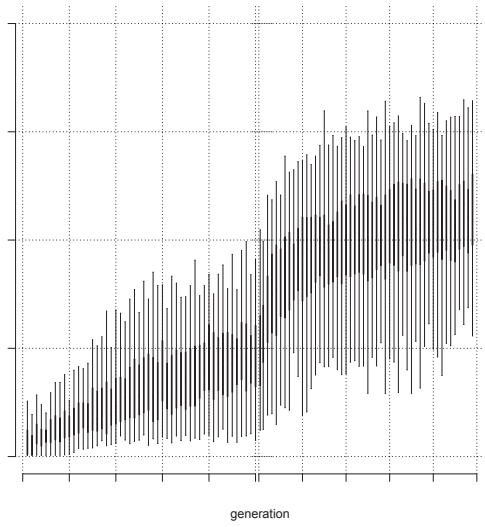
Hierarchical Policy Deployment and Symmetrical Behaviour

As in the Pinball domain, some insight into the capacity for a hierarchical policy to integrate the behaviour of previously evolved meta actions can be seen by plotting the use of meta actions during specific solution trajectories. Figure 5.20 provides two example trajectories for the same SBB champion policy. Meta actions are clearly being interleaved to create an integrated policy, as opposed to merely replaying meta actions for the duration relative to an initial condition. Moreover, there are clearly specific contexts / switching points under which different meta actions are deployed.

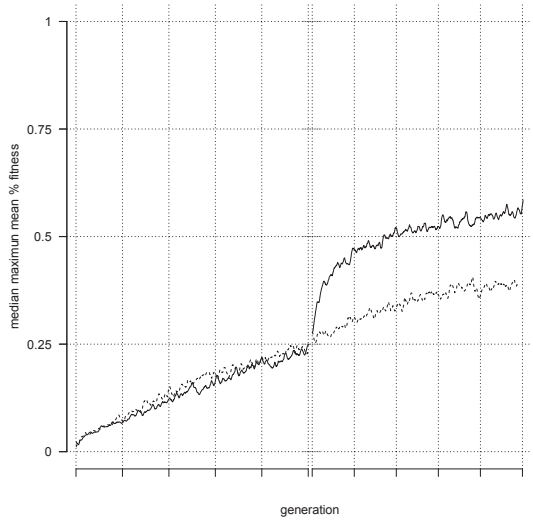
For the same SBB champion policy we can take a snapshot of progress against all test points at a specific time step of policy deployment. Figure 5.21 (a) details this for the same host at $t_s = 100$. A clear north–south partition is present, roughly relative to the corresponding distribution of initial states. However, this is not explicitly symmetrical (see also Figure 5.20.(b)). The north trajectory appears to be slightly more dominant, with trajectories from the south actually employing a re-alignment 'swirl' near the origin such that they explicitly join the northern trajectory before successfully reversing to the goal location. However, often the champion host behaviour will be explicitly *asymmetric* as in Figure 5.21 (b). In this case the asymmetric behaviour clearly results in less efficient solutions for start locations in the bottom half



(a) Incremental

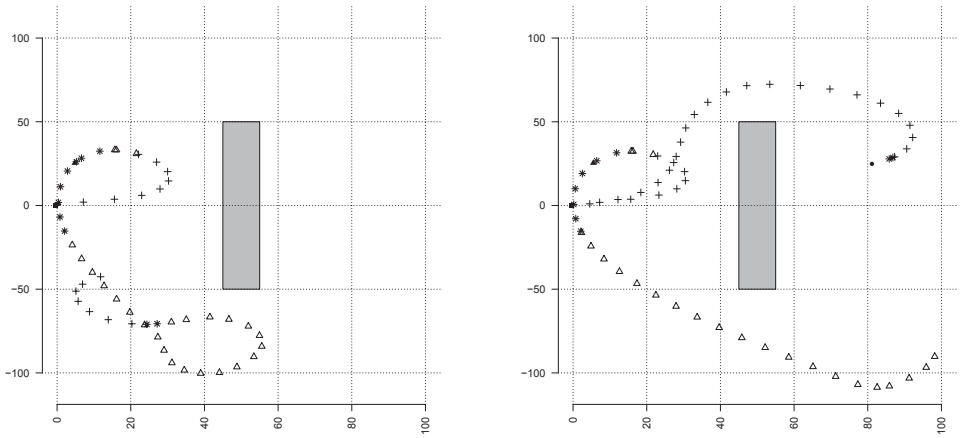


(b) Layered



(c) Combined Medians

Figure 5.19: Truck Reversal Task: Training Curves for Incremental and Layered Experiments. (a) and (b) depict the maximum individual host fitness over 2000 generations for each experiment where box plots summarize the distribution w.r.t. 60 independent trials. (c) plots the median from each of the above distributions where the dashed line represents Incremental and solid line is Layered. Gap in x axis marks the point at which developmental variation and hierarchical transition (Layered only) take place.



(a) Swirl (b) Symmetric

Figure 5.20: Truck Reversal Task: Example trajectories for solved test cases from the same SBB policy tree. Subplot (a) and (b) are relative to one and two specific initial semi–cab configurations respectively. Symbols distinguish between the selection of different (level 0) meta actions by the root (level 1) switching policy.

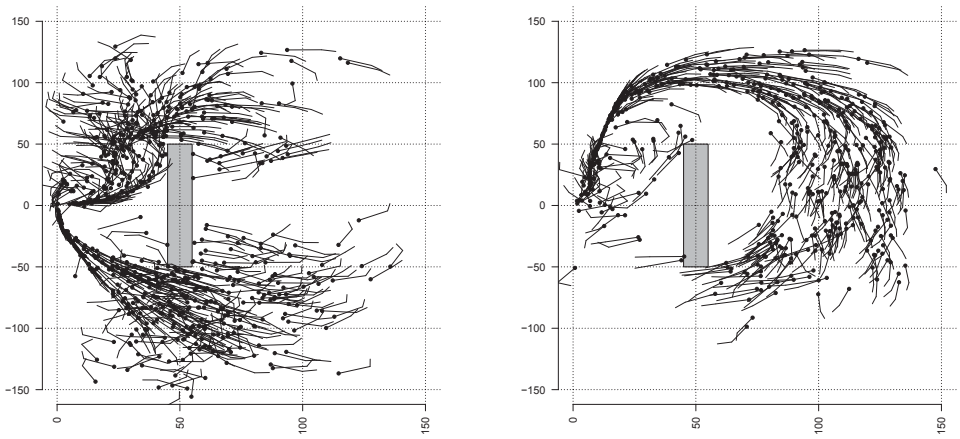
Table 5.3: Truck reversal task: Number of trials in which champion host assumed 'symmetric' behaviour.

Experiment	Symmetric Behaviours
Monolithic	2
Incremental	3
Modular	28
Layered	19

of the world. Explicitly rewarding time step efficiency, as in spatiotemporal fitness (Equation 5.1), potentially results in many more trials finding symmetric solutions. However, this benefit is only achieved under hierarchical SBB, see Table 5.3. This observation reinforces the importance of interleaving multiple (level 0) meta actions to achieve optimal behaviours.

5.4 Levels of Selection

In this section we add to the argument that hierarchical polices are doing more than just indexing a large number of meta actions and matching each to an initial configuration of the task. In order to provide an additional qualification on what the



(a) Symmetric (b) Asymmetric

Figure 5.21: Truck Reversal Task: Snapshot of the location of all 1,000 test initializations for $t_s = 100$. (a) is SBB policy tree from Figure 5.20. (b) is example asymmetric behaviour. The head of the pin denotes the end of the semi–cab.

hierarchy is contributing, we can explicitly identify the meta actions (SBB level 0 individuals) utilized by each champion. In each case, we count the corresponding accumulated number of test cases solved. The champion distribution in both task domains can now be compared directly to the distribution representing the sum of supporting meta actions (denoted '(meta)' in Figures 5.1 and 5.13). It is now apparent that a level-1 host does indeed explicitly generalize to more than the sum of the corresponding meta action contribution (compare Modular/Layered (meta) to corresponding Modular/Layered).

This observation is synonymous with that made in relation to multi-level selection and group formation [42], as discussed in Section 2.1. Recall that MLS1 defines fitness of a host as the average of the symbiont membership. Conversely, MLS2 measures fitness as that defined by the host behaviour alone. In the case of our experiments, MLS1 is synonymous with 'Modular/Layered (meta)' and MLS2 with 'Modular/Layered', and it follows that a significant improvement in fitness results following the transition to a hierarchical policy or $f(\text{MLS2}) >> f(\text{MLS1})$; where $f(\cdot)$ denotes fitness and Tables 5.2 and 5.1 confirm statistical significance.

Finally we note that, in the case of Layered (meta) of Figure 5.1, it appears that the meta actions represent a worst case starting point, yet the resulting policy trees

(Layered) provide among the strongest eventual solutions. This is because Figure 5.1 compares performance using the fixed target location (top centre, Figure 4.1), whereas meta actions of case Layered (meta) are trained against arbitrary start–exit locations, as defined by the point population.

Chapter 6

Conclusion and Future Work

The role of developmental diversity in the construction of policy search agents through symbiotic coevolution has been examined. Four configurations having equivalent complexity and computational overhead were tested in two control-style problem domains. Specifically, these four types of evolution (Monolithic, Incremental, Modularized, and Layered) allowed us to measure the impact of developmental diversity in general *and* the relative importance of hierarchical policy structures.

In the Pinball task, the less challenging of the two domains, both Incremental evolution (goal diversity without a hierarchical transition) and Layered evolution (goal diversity *with* a hierarchical transition) proved beneficial as compared to corresponding models with less environmental diversity. Developmental diversity was implemented in this domain by first evolving solutions relative to partial configurations of the task domain, i.e. arbitrary start *and* goal locations for the pinball, before switching to evolution against the common global goal. Thus, we conclude that building diverse specialist behaviours in the early stages of evolution prior to focusing on the global task is beneficial for the construction of single-level and hierarchical policies. However, the utility of archiving a library of meta actions was demonstrated in the context of model efficiency, where hierarchical models have the advantage of being able to redeploy meta actions relative to *any* valid task from the same domain without starting evolution from scratch.

In the truck reversal domain, a new lexicographic fitness function was introduced and shown to improve the performance of Monolithic and Modularized evolution relative to initial SBB experiments in this domain [32]. Moreover, developing solutions under two independent cycles of evolution with developmental diversity between cycles (Layered evolution) further improved both the efficiency and generalization performance of champion individuals. During the first cycle the ecology again rewards strong specialists i.e., reward for both spatial and temporal properties. In the second

cycle the ecology is more general in its reward, recognizing spatial properties alone, thus promoting the hierarchical composition of generalists from previously evolved specialists.

In all hierarchical models, the generalization performance of champion policies was shown to be greater than the cumulative performance from contributing meta actions. Thus, hierarchical policies are able to discover new solutions by grouping together multiple meta actions and developing the appropriate contexts in which each may be deployed.

Future research is likely to consider the case of automating the selection and ordering of domain-specific developmental variations during evolution. The research herein was conducted under a tabula rasa model of point generation. However, evaluating potential solutions against points that are themselves evolving (competitive coevolution) could be attempted. Recent research with the 'host-parasite' model [9] has shown that selecting for problem scenarios with reduced 'virulence' can be effective at combating the disengagement problem. Similar approaches could be applied to the problem of automated developmental variation.

Extending our findings to hierarchical structures with more than two cycles of evolution is another area of particular interest. Currently, the leverage obtained from the hierarchical transition between level 0 and level 1 (See Figure 5.19 (c)) does not repeat at subsequent transitions. Thus, some form of structural modification or further developmental diversity is likely necessary.

Appendix A

Truck Reversal with Automatically Defined Functions

Automatically Defined Functions (ADFs) were introduced by Koza in [28] as an approach to supporting the development of code modularity, thus problem decomposition. He noted that many problems contain regularities that would allow the repeated deployment of similar functions. Thus, each tree-structured GP individual in the ADF framework has a result producing branch (RPB) along with some number of local functions (ADFs). The ADFs evolve together with the RPB. The number of ADFs used, their arity, function set, and the degree of reference between ADFs (ADFs may be allowed to call each other) are all manually configured a priori.

Koza showed that, on problems that were decomposable, ADFs improved the performance of GP significantly. GP with ADFs was able to find solutions faster and produced simpler programs. While care was taken to configure ADFs appropriately for each problem, it was ultimately shown that *any* configuration of ADFs was beneficial in these problems. Finally, the number and structure of ADFs can be evolved instead of selected a priori, though this adds considerable computational overhead and source code is not currently available.

In this section we explore the utility of ADFs within the truck reversal task in order to establish a baseline hierarchical GP approach to this problem.

A.0.1 Parameterization

The truck reversal task implementation is derived from that outlined in Section 4.2 with specific modifications for the tree-structured GP framework as detailed in this section. The terminal set T used here consists of the truck state variables, or $T = (x, y, \theta_c, \theta_s)$, Figure 4.2. The function set F consists of four arithmetic operations, the two argument Arctangent function ATG, and the conditional comparative operator IFLTZ ('If Less than Zero'), or $F = (+, -, *, \%, ATG, IFLTZ)$. See [27] for a detailed discussion regarding this function set.

Naturally, the fitness function for these experiments is derived from that used for SBB. At initialization, 120 fitness cases are sampled with uniform probability following the point initialization process in Section 4.2.4. In each subsequent generation, P_{gap} fitness cases are removed with equal probability and P_{gap} new cases introduced as per the point initialization process. Fitness is the sum error, measured at the termination of each fitness case, over all fitness cases, where $error = x^2 + y^2 + \theta_s^2 + 1$. By way of comparison, [27] measured fitness for the truck reversal problem (without the additional wall object) relative to only 8 a priori defined fitness cases that remained constant over the course of evolution.

The search operators used in our standard tree-structured GP are crossover (applied to 90% of the population) and reproduction (applied to 10% of the population). With no mutation operator, a large population size is required in order to maintain a diverse spectrum of potential solutions. Our population size for all experiments was $2,000$ and we ran for 75 generations. This amounts to $18,000,000$ evaluations per experiment (calculated as $popSize * numFitnessCases * numGenerations$). For comparison, each SBB experiment in this domain had $16,800,000$ evaluations[1]. Tree-structured GP requires several other learning parameters, which we derived directly from the previous study within a simpler formulation of the truck reversal domain [27]. Finally, as in [27], GP individuals are not limited to 3 atomic actions. Instead, we convert the real- valued output of a GP individual to a value that can be directly applied as a steering angle for the truck tires. Thus, if the program evaluates to a number between -1.0 and 1.0, the truck turns its wheels to that particular angle (in radians). Outside that range the control variable saturates. All constraints on illegal behaviours outlined in Section 4.2, such as jackknifing and colliding with walls, are also enforced here.

Some analysis of the problem is necessary when selecting the appropriate configuration of ADFs. Since the truck reversal problem consists of two state variables that describe spatial coordinates (x, y) and two that describe orientation of the cab and semi (θ_c, θ_s), we decided that 2 is an appropriate choice for the number of arguments to a defined function for this task. For simplicity, no ADFs were permitted to reference each other. The terminal and function sets for an ADF are identical to T and

[1]The number of evaluations in each SBB experiment is calculated as $((H_{size} * P_{gap}) + (H_{gap} * (P_{size} - P_{gap})) * (t_{max} * 2)$

F above. Section A.0.2 details our findings for multiple ADF configurations in the truck reversal task as defined in Section 4.2 i.e., with the additional wall object.

A.0.2 Results

Post training test performance is measured relative to the same $1,000$ unique start positions for the truck as used under SBB, Figure 5.21. For simplicity we do not choose a 'champion' individual w.r.t. a separate validation set. Instead, the individual with the most solutions across the test set is selected as being the representative 'champion' of each trial.

Two initial experiments without ADFs are conducted in order to compare the difficulty of the truck reversal task as defined in Section 4.2 with that used in [27]. The main difference being the addition of the wall obstacle and the enforcement of tests to detect illegal behaviours such as jackknifing. Generalization performance for the champion individual from these two experiments is shown as '0ADF/no wall' and '0ADF' in Figure A.1. The introduction of the wall obstacle and additional constraints clearly increase the difficulty of the task.

Next, 4 separate experiments are conducted to evaluate an increasing number of defined functions available to each GP individual. Generalization performance for the champion in each experiments appears in Figure A.1. Clearly, the individual with the best generalization performance in our formulation of the task is that with no ADFs enabled, '0 ADF' in Figure A.1. Moreover, the champion performance decreases as more ADFs are added. We attribute this to the incremental increase in search space introduced as a result of each additional ADF. Unlike approaches to modularity in GP such as the RTL framework (Section 2.2) or SBB, where a library of meta actions is developed and 'frozen' prior to being redeployed as part of a larger solution, ADFs attempt to evolve defined functions and the result producing branch simultaneously. In the case of our truck reversal task, this seems to limit the ability of the framework to successfully decompose the problem into useful subtasks. The training curves for each experiment, Figure A.2, reinforce this conclusion.

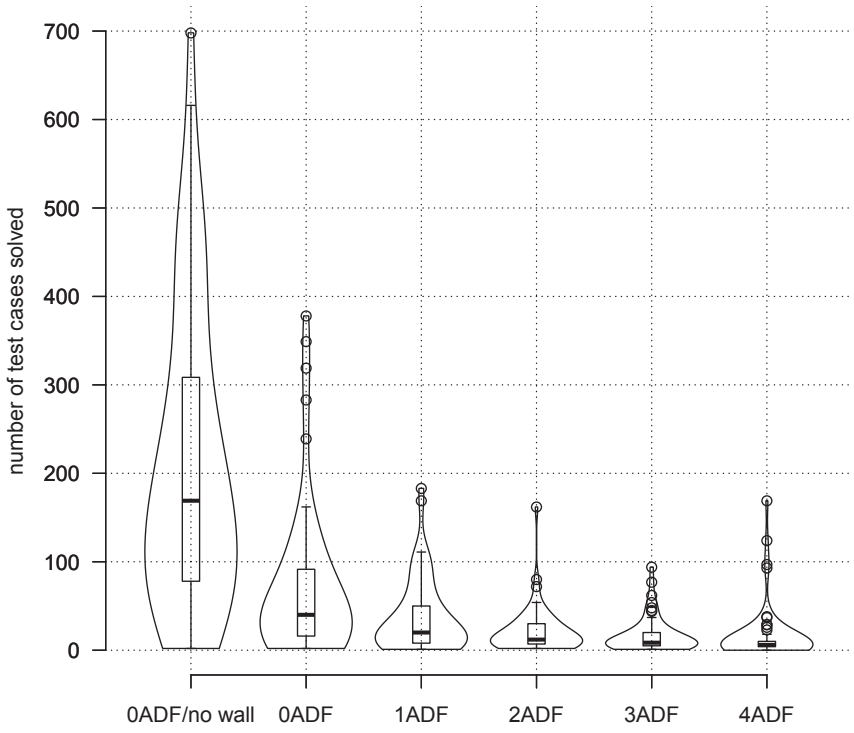


Figure A.1: Truck Reversal with ADFs: Generalization performance or count of number of test points solved by the champion individual (y-axis) from each experiment. 1000 test points in total. X axis labels correspond to 6 different experiments, each with a particular number of 2-argument ADFs. 'no wall' denotes the absence of the wall obstacle.

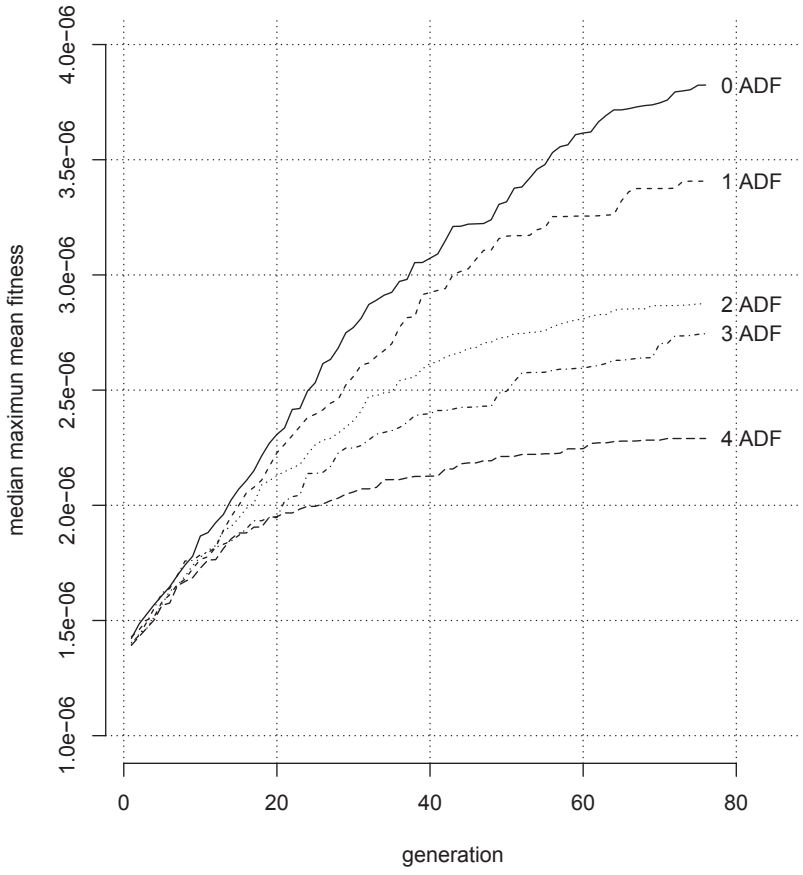


Figure A.2: Truck Reversal with ADFs: Median adjusted fitness (60 independent trials) over 75 generations. Each line represents a separate experiment with a particular number of 2-argument ADFs.

A.0.3 Summary

This exercise further establishes the difficulty of the truck reversal task with additional wall obstacle. Unlike [27], we evolved and tested solutions relative to a large diverse sampling of task domain configurations. Though we did not test against the 8 specific fitness cases hand-crafted by Koza [27], the median number of test cases solved was above 8 for all experiments except '4ADF'. Koza did not report on the use of ADFs in the truck reversal domain. Our findings illustrate that, at least for the configurations we chose to explore, ADFs do not improve the performance of GP for this task. Incrementally increasing the number of ADFs over multiple experiments continued to have a detrimental effect on learning rates and generalization performance. Given this result, it is unlikely that any form of developmental diversity could significantly improve on this performance.

Bibliography

[1] C. W. Anderson and W. Thomas Miller. A challenging set of control problems. In *Neural Networks for Control*, pages 475–508. MIT Press, Cambridge, MA, USA, 1990.

[2] A. M. S. Barreto, D. A. Augusto, and H. J. C. Barbosa. On the characteristics of sequential decision problems and their impact on evolutionary computation and reinforcement learning. In *Proceedings of the 9th international conference on Artificial evolution*, EA'09, pages 194–205, Berlin, Heidelberg, 2010. Springer-Verlag.

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

[4] J. Bongard. Behavior chaining: Incremental behavioral integration for evolutionary robotics. In *Proceedings of the International Conference on Artificial Life*, pages 64–71, 2008.

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

[6] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[7] R. Calabretta, S. Nolfi, D. Parisi, and G. P. Wagner. Duplication of modules facilitates the evolution of functional specialization. *ARTIFICIAL LIFE*, 6:69–84, 2000.

[8] B. Calcott and K. Sterelny, editors. *The major transitions in evolution revisited*. Vienna Series in Theoretical Biology. MIT Press, 2011.

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

[10] S. Y. Chong, P. Tiňo, and X. Yao. Relationship between generalization and diversity in coevolutionary learning. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):213–231, 2009.

[11] P. J. Darwen and X. Yao. Speciation as automatic categorical modularization. *IEEE Tractions on Evolutionary Computation*, 1(2):101–108, 1997.

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

[13] R. De Nardi, J. Togelius, O. E. Holland, and S. M. Lucas. Evolution of neural networks for helicopter control: Why modularity matters. In *IEEE Congress on Evolutionary Computation*, pages 1799–1806, 2006.

[14] M. Dorigo and M. Colombetti. Robot shaping: developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370, 1994.

[15] J. A. Doucette, P. Lichodzijewski, and M. I. Heywood. Hierarchical task decomposition through symbiosis in reinforcement learning. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, page to appear, 2012.

[16] J. A. Doucette, A. R. McIntryre, P. Lichodzijewski, and M. I. Heywood. Symbiotic coevolutionary genetic programming: A benchmarking study under large attribute spaces. *Genetic Programming and Evolvable Machines*, 13(1):71–101, 2012.

[17] N. Eldredge. The sloshing bucket: How the physical realm controls evolution. In J. P. Crutchfield and P. Schuster, editors, *Evolutionary Dynamics*, pages 3–32. Oxford University Press, 2003.

[18] S. Geva, J. Sitte, and G. Willshire. A one neuron truck backer-upper. In *IEEE-INNS International Joint Conference on Neural Networks*, pages 850–856, 1992.

[19] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, (5):317–342, 1997.

[20] M. I. Heywood and P. Lichodzijewski. Symbiogenesis as a mechanism for building complex adaptive systems: A review. In *EvoApplications: Part 1*, volume 6024 of *LNCS*, pages 51–60. Springer, 2010.

[21] R. E. Jenkins and B. P. Yuhas. 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.

[22] M. Keijzer, C. Ryan, and M. Cattolico. Run transferable libraries – learning functional bias in problem domains. In *Genetic and Evolutionary Computation Conference*, volume 3103 of *LNCS*, pages 531–542, 2004.

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

[24] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, pages 1015–1023, 2009.

[25] G. Konidaris, S. Kuindersma, A. Barto, and R. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trees. In *Advances in Neural Information Processing Systems 23*, pages 1162–1170, 2010.

[26] G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Conference on Artificial Intelligence*, pages 380–385, 2011.

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

[28] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

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

[30] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2010.

[31] X. Li, C. Zhou, W. Xiao, and P. C. Nelson. Direct evolution of hierarchical solutions with self-emergent substructures. In *International Conference on Machine Learning Applications*, pages 337–342, 2005.

[32] P. Lichodzijewski. *A symbiotic bid-based framework for problem decomposition using genetic programming*. PhD thesis, Faculty of Computer Science, 2011. http://web.cs.dal.ca/˜mheywood/Thesis/PhD.html.

[33] P. Lichodzijewski, J. Doucette, and M. I. Heywood. Symbiosis and hierarchical model building in temporal sequence learning. Technical Report CS-2011-06, Faculty of Computer Science, Dalhousie University, 2011.

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

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

[36] L. Margulis and R. Fester, editors. *Symbiosis as a Source of Evolutionary Innovation*. MIT Press, 1991.

[37] J. Maynard Smith. *A Darwinian View of Symbiosis*, chapter 3, pages 26–39. 1991. In ([36]).

[38] R. E. Michod. *Darwinian Dynamics: Evolutionary transitions in fitness and individuality*. Princeton University Press, 1999.

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

[40] D. H. Nguyem and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, pages 18–21, 1990.

[41] S. Nolfi. Using emergent modularity to develop control systems for mobile robots. *Adaptive Behavior*, 5:343–363, 1997.

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

[43] M. ONeill and A. Brabazon. mgga: The meta-grammar genetic algorithm. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano van Hemert, and Marco Tomassini, editors, *Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 143–143. Springer Berlin / Heidelberg, 2005.

[44] A. Prugel-Bennett. Benefits of a population: Five mechanisms that advantage population-based algorithms. *Evolutionary Computation, IEEE Transactions on*, 14(4):500–517, March 2010.

[45] D. C. Queller. Relatedness and the fraternal major transitions. *Philosophical Transactions of the Royal Society of London: Series B*, 355:1647–1655, 2000.

[46] S. C. Roberts, D. Howard, and J. R. Koza. Evolving modules in genetic programming by subtree encapsulation. In *European Conference on Genetic Programming*, pages 160–175, 2001.

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

[48] M. Schoenauer and E. Ronald. Neuro-genetic truck backer-upper controller. In *IEEE Congress on Evolutionary Computation*, pages 720–723, 1994.

[49] P. W. H. Smith and K. Harris. Code growth, explicitly defined introns, and alternative selection schemes. *Evolutionary Computation*, 6(4):339–360, 1999.

[50] L. Spector, K. H., and Thomas H. Tag-based modularity in tree-based genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 815–822, New York, NY, USA, 2012. ACM.

[51] L. Spector, B. Martin, K. Harrington, and T. Helmuth. Tag-based modules in genetic programming. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1419–1426, 2011.

[52] K. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8:131–162, 2007. 10.1007/s10710-007-9028-8.

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

[54] K. Sterelny. Symbiosis, evolvability and modularity. In G. Schlosser and G. P. Wagner, editors, *Modularity in Development and Evolution*, pages 490–516. Chicago University Press, 2004.

[55] P. Stone. *Layered learning in multiagent systems: A winning approach to robotic soccer*. MIT Press, 2000.

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

[57] E. Szathmary and J. Maynard Smith. The major evolutionary transitions. *Nature*, 374:227–232, 1995.

[58] J. Togelius. Evolution of a subsumption architecture neurocontroller. *J. Intell. Fuzzy Syst.*, 15(1):15–20, January 2004.

[59] J. Urzelai and D. Floreano. Incremental evolution with minimal resources. In *Proceedings of IKW99*, 1999.

[60] S. Whiteson. *Adaptive Representations for Reinforcement Learning*, volume 291 of *SCI*. Springer, 2010.

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

[62] G. Wilson and M. I. Heywood. Introducing probabilistic adaptive mapping developmental genetic programming with redundant mappings. *Genetic Programming and Evolvable Machines*, 8:187–220, 2007. 10.1007/s10710-007-9027-9.

[63] X. Yao. Evolving artificial neural networks. In *Proceedings of the IEEE*, pages 1423–1447, 1999.

[64] T. Ziemke, N. Bergfeldt, G. Buason, T. Susi, and S. Svensson. Evolving cognitive scaffolding and environment adaptation. *Connection Science*, 16(4):339–350, 2004.