A SYMBIOTIC BID-BASED FRAMEWORK FOR PROBLEM DECOMPOSITION USING GENETIC PROGRAMMING

by

Peter Lichodzijewski

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
February 2011

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 "A SYMBIOTIC BID-BASED FRAMEWORK FOR PROBLEM DECOMPOSITION USING GENETIC PROGRAMMING" by Peter Lichodzijewski in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dated: February 22, 2011

External Examiner: _____

Research Supervisor: _____

Examining Committee: _____

_____

_____

Departmental Representative: _____

DALHOUSIE UNIVERSITY

DATE: February 22, 2011

AUTHOR: Peter Lichodzijewski

TITLE: A SYMBIOTIC BID-BASED FRAMEWORK FOR PROBLEM DECOMPOSITION USING GENETIC PROGRAMMING

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: Ph.D. CONVOCATION: May YEAR: 2011

_____

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

This thesis investigates the use of symbiosis as an evolutionary metaphor for problem decomposition using Genetic Programming. It begins by drawing a connection between lateral problem decomposition, in which peers with similar capabilities coordinate their actions, and vertical problem decomposition, whereby solution subcomponents are organized into increasingly complex units of organization. Furthermore, the two types of problem decomposition are associated respectively with context learning and layered learning. The thesis then proposes the Symbiotic Bid-Based framework modeled after a three-staged process of symbiosis abstracted from biological evolution. As such, it is argued, the approach has the capacity for both types of problem decomposition.

Three principles capture the essence of the proposed framework. First, a bid-based approach to context learning is used to separate the issues of 'what to do' and 'when to do it'. Whereas the former issue refers to the problem-specific actions, e.g., class label predictions, the latter refers to a bidding behaviour that identifies a set of problem conditions. In this work, Genetic Programming is used to evolve the bids casting the method in a non-traditional role as programs no longer represent complete solutions. Second, the proposed framework relies on symbiosis as the primary mechanism of inheritance driving evolution, where this is in contrast to the crossover operator often encountered in Evolutionary Computation. Under this evolutionary metaphor, a set of symbionts, each representing a solution subcomponent in terms of a bid-action pair, is compartmentalized inside a host. Communication between symbionts is realized through their collective bidding behaviour, thus, their cooperation is directly supported by the bid-based approach to context learning. Third, assuming that challenging tasks where problem decomposition is likely to play a key role will often involve large state spaces, the proposed framework includes a dynamic evaluation function that explicitly models the interaction between candidate solutions and training cases. As such, the computational overhead incurred during training under the proposed framework does not depend on the size of the problem state space.

An approach to model building, the Symbiotic Bid-Based framework is first evaluated on a set of real-world classification problems which include problems with multi-class labels, unbalanced distributions, and large attribute counts. The evaluation includes a comparison against Support Vector Machines and AdaBoost. Under temporal sequence learning, the proposed framework is evaluated on the truck reversal and Rubik's Cube tasks, and in the former case, it is compared with the Neuroevolution of Augmenting Topologies algorithm. Under both problems, it is demonstrated that the increased capacity for problem decomposition under the proposed approach results in improved performance, with solutions employing vertical problem decomposition under temporal sequence learning proving to be especially effective.

List of Abbreviations Used

ADF	Automatically Defined Function
COIN	Collective Intelligence
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EMOO	Evolutionary Multi-Objective Optimization
EP	Evolutionary Programming
ES	Evolution Strategy
GA	Genetic Algorithm
GP	Genetic Programming
LCS	Learning Classifier System
ML	Machine Learning
NEAT	Neuroevolution of Augmenting Topologies
NIPS	Neural Information Processing Systems
OET	Orthogonal Evolution of Teams
RBF	Radial Basis Function
RSS	Random Subset Selection
SANE	Symbiotic Adaptive Neuroevolution
SBB	Symbiotic Bid-Based
SSS	Static Subset Selection
SVM	Support Vector Machine

Acknowledgements

I would like to thank my supervisor, Dr. Malcolm Heywood, for his guidance and support. I can still remember the first time we talked about potential research topics on a snowy Friday afternoon many years ago, and will miss the innumerable discussions we have had since then. A special thanks to my examining committee including Dr. Nur Zincir-Heywood, Dr. Dirk Arnold, and Dr. Terence Soule; I value your insights and appreciate you taking interest in my work. I would also like to recognize the contributions of my fellow lab mates, technical and otherwise, as well as those of the staff at the Faculty of Computer Science at Dalhousie.

Research takes time, sometimes bearing few immediately obvious benefits, so I must acknowledge the financial support of the Killam Trusts, Precarn, and NSERC which gave me time to focus on my work. I hope that these organizations will continue to foster Canadian innovation.

Finally, there is no doubt that I would not be here without the support of my family, and for this I am grateful.

Chapter 1

Introduction

The use of Evolutionary Computation (EC) as a problem solving tool is motivated by a number of factors [39]. The most basic motivation, and likely the initial inspiration for many in the field of EC, is perhaps that it draws heavily on metaphors from biological evolution. As such, if it can even partially match some of the extraordinary achievements of its biological archetype, it must be the case that EC has the potential to produce some extraordinary results.

A more practical and concrete motivation for the use of EC is the growing number and increasing complexity of problems that are appearing within a cross-section of domains such as security, medicine, engineering, business, and entertainment. It is therefore becoming increasingly difficult and expensive for experts to manually develop algorithms tailored to each individual task. In contrast, EC can be used to construct robust algorithms capable of automatically producing solutions to a wide range of problems.

The focus of this work is Genetic Programming (GP), a branch of EC where solutions take the form of computer programs. GP is part of the larger field of machine learning (ML) which concerns the design and analysis of computer algorithms that improve with experience [129]. ML has been found to be useful in applications where there is a lot of data (e.g., data mining), where expert knowledge is not available, and where there is a need for algorithms to adapt to changing situations. In all three cases, ML unburdens the practitioner from having to deal with what would otherwise be challenging tasks.

Given the growing number of increasingly complex applications, a key element of any proposed approach is viewed to be the ability to automatically decompose the problem. In particular, problem decomposition may be required if the problem is sufficiently difficult so as to make it unlikely that it can be solved 'in one go'. Furthermore, problem decomposition may lead to increased efficiency, e.g., through

module reuse, as well as solutions that can be interpreted more easily than those which are structured monolithically.

The goal of this work is therefore to investigate an evolutionary metaphor for problem decomposition, and to this end, symbiosis is considered; based on this metaphor, the Symbiotic Bid-Based (SBB) framework is presented. Perhaps because it was not originally included as a component of neo-Darwinism, symbiosis has received limited attention from the EC research community. In this thesis, it is argued that symbiosis can be interpreted as a biologically inspired metaphor both for *context learning* and for *layered learning*, where these represent two complementary approaches to problem decomposition. As an inheritance mechanism, symbiosis also provides an alternative to the sexual recombination operator commonly used in EC but requires fewer assumptions to be met, i.e., regarding the exchange of functionally equivalent units.

The primary aim of this chapter is therefore to establish the scope and context assumed in this thesis with respect to (1) problem decomposition, (2) a generic evolutionary framework for problem solving, and (3) a formal process of symbiosis including how it relates to common practices in EC. In the two sections following immediately, problem decomposition is discussed and a review of ML is provided, in both cases without reference to any single problem solving approach. Section 1.3 then provides an overview of EC, including GP, and highlights certain open issues. Having presented concepts from symbiosis in biology, Section 1.4, these are then considered in the context of an evolutionary framework in Section 1.5. In particular, a case is made for symbiosis as a metaphor for achieving problem decomposition through a combination of context learning and layered learning. The concluding sections of this chapter state the research objectives underlying this thesis and provide an outline of the remainder of this dissertation.

1.1 Problem Decomposition

In this work, problem decomposition is defined as the structural organization of a solution into explicit subcomponents that collectively exhibit non-overlapping behaviours. The degree of overlap can be viewed as relative with less overlap associated with increased problem decomposition. Two ways of structuring the decomposition

are considered. Under *vertical* problem decomposition, the subcomponents are structured into multiple levels with higher-up elements representing increasingly complex units of organization, i.e., the subcomponents are organized into a hierarchy where overlap is considered with respect to the units at a given level. In contrast, under *lateral* problem decomposition, the subcomponents represent peers with similar capabilities whose behaviours must be coordinated, e.g., using an *a priori* voting mechanism applied post training.

Intuitively, the key issue in vertical problem decomposition is learning to combine simple behaviours in order to produce increasingly complex behaviours, e.g., as in layered learning [180]. In contrast, the main motivation behind lateral problem decomposition is to effectively partition the problem space among a set of solution subcomponents of similar complexity, e.g., as in context learning. Lateral problem decomposition is viewed as more fundamental since it can be used as a basis for constructing the levels in a vertical organization. At the same time, the addition of higher-level behaviours which aggregate lower-level behaviours may provide a way to automatically coordinate the peers in a lateral organization. As such, it may be useful to combine both forms of problem decomposition, and this is the case in the proposed approach.

A requirement of problem decomposition stipulated in the above definition is that the behaviours be non-overlapping. This is illustrated in terms of the problem instances solved by a set of peers, Figure 1.1. On the left-hand side, the behaviours are correlated, with the three peers solving similar instances. On the right-hand side, the behaviours are much more distinct. Thus, relatively speaking, the former situation would not be considered as representative of a meaningful instance of problem decomposition[1].

The solutions to a given problem can be organized into three broad categories which exhibit increasing amounts of problem decomposition. The least amount of problem decomposition occurs when the solution is monolithic, i.e., it is not organized into distinct subcomponents. Next is the case of a solution which is organized into modules but those modules still behave similarly, e.g., as in the left-hand side in

[1]If the behaviours on the right-hand side of Figure 1.1 are interpreted in terms of the problem instances *not* solved by each peer, then optimal decisions would always be made by the group by taking a majority vote.

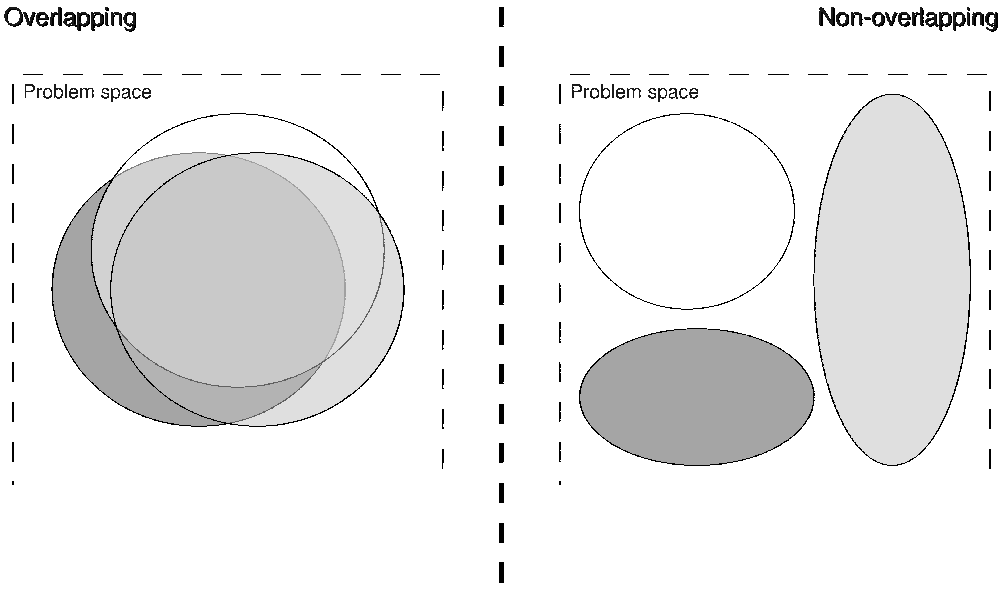


Figure 1.1: Degrees of problem decomposition. Each ellipse represents the set of instances that one of three peers is able to solve. On the left, the peers tend to solve similar instances, while on the right, a greater degree of problem decomposition is achieved as peers solve non-overlapping regions of the problem.

Figure 1.1. Finally, the greatest amount of problem decomposition occurs when a solution is organized into modules which behave independently.

This categorization suggests a number of benefits associated with problem decomposition:

1. **Solution transparency.** If the solution is expressed in terms of explicit subcomponents with different roles, it becomes easier to figure out how the problem is solved, e.g., the logic behind the resulting actions. In contrast, identifying potential modules may be the first step that one is required to take in understanding a monolithic solution – a step that is unnecessary under an approach that decomposes the problem. The issue of solution transparency is becoming increasingly more relevant as there is a growing interest in finding relationships in data as opposed to focusing on performance alone [41, 122, 131, 186]. At this

point, the association between solution transparency and solution complexity is also noted, in particular, a large solution is likely to be less transparent.

2. **Learnability.** Compared to building monolithic solutions, problem decomposition is viewed as a more effective method for learning difficult tasks as it reflects a divide-and-conquer mentality. Specifically, because modules are discovered in the process of problem decomposition, once the effort is applied in finding appropriate subcomponents these can be reused in 'factoring' the search [13]. Alternatively, modules can be viewed as building blocks, that, once discovered, are more likely to be retained compared to subcomponents that are not explicitly identified. Furthermore, through problem decomposition the search becomes more structured, e.g., under vertical problem decomposition simple behaviours may be discovered at lower levels to then be combined at higher levels. This is viewed as more manageable than trying to solve the problem in one go, and reflects the way in which learning occurs in nature.

3. **Efficiency.** In terms of training efficiency, the overhead of building solutions in a modular fashion may not be justifiable when the task is easy, i.e., if an equally effective monolithic solution can be constructed with less effort. Assuming that problem decomposition results in improved learnability, as the task becomes harder, the monolithic approach is likely to become less efficient to the point where it is not able to produce a successful solution given any amount of effort. Furthermore, if only a subset of the subcomponents need to be activated when the solution is applied to a problem instance, the runtime efficiency associated with a modular solution is likely to be higher. Here, it is also noted that the efficiency with which a solution is applied may be a function of its complexity, i.e., larger solutions may require more effort.

In this thesis, the solutions that are encountered are analyzed in terms of their transparency, learnability, and efficiency. The learnability associated with an algorithm is viewed in terms of its success on a given problem. Often, but not always, solution transparency and efficiency is interpreted in terms of the associated solution complexity.

Finally, it is noted that symbiosis represents a mechanism by which a problem

can be decomposed both vertically and laterally. This claim is further discussed below, Section 1.5.3, following the definition of symbiosis. Relevant ML approaches to achieving lateral and vertical problem decomposition are reviewed in Sections 2.2 and 2.3.

1.2 Machine Learning

1.2.1 Machine Learning Problem Domains

A number of ML problem categories are recognized. *Supervised learning* assumes that each input is associated with a known target output, whereas under *unsupervised learning* the task is to learn relationships between unlabeled instances. *Semi-supervised* learning [17] relies on both labeled and unlabeled data, typically small amounts of the former combined with large amounts of the latter, and is motivated by the possibly high cost of obtaining the target outputs. Finally, under what is referred to in this work as *temporal sequence learning*, the task is complicated by the introduction of a delayed reward as a problem instance is typically solved in a series of steps. This thesis concerns primarily classification, a form of supervised learning, and temporal sequence learning, though with reasonable modifications the proposed approach may be applicable to other types of problems as well. The two most relevant categories of problems are further discussed below.

Supervised Learning: Classification

Classification, in contrast to other forms of supervised learning such as, e.g., regression, is of particular interest in this thesis because the proposed approach assumes discrete actions. It involves paired data of the form $(\vec{x}_i = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_n} \rangle, y_i)$ where $\vec{x}_i$ is an input exemplar and y_i is its associated class label[2]. During training, the label y_i associated with each input $\vec{x}_i$ is known, and the goal is then to learn a mapping from the characteristics quantified by the exemplar to the associated label. Once the mapping is learned, the resulting model can be applied to predict labels for previously unseen exemplars. Classification is useful in a wide array of applications ranging from medical diagnosis [18] to intrusion detection [168].

[2]In contrast to *categorization*, under classification each exemplar is associated with exactly one label.

Under binary classification, the set of possible labels consists of two elements, whereas multi-class problems involve three or more possible labels. Naturally, multi-class problems tend to present a greater challenge. Other challenges related to classification that are addressed in this work are the high computational overhead associated with training on large sets of data and the difficulty of training on unbalanced datasets, i.e., when the classes in the training data are observed in different frequencies. In the latter case, part of the challenges lies in identifying an appropriate measure of model quality [83, 167]. Furthermore, classification algorithms may suffer from over-fitting whereby certain characteristics specific to the training data, but not the problem as a whole, are memorized.

To address the issue of high dimensionality in classification problems, three types of feature[3] selection methods have been proposed [59]. The simplest of these are *filter* methods that perform feature selection once before the main ML algorithm is invoked. *Wrapper* methods, on the other hand, use the main ML algorithm to iteratively evaluate and refine subsets of candidate features. As such, they represent an approach that is more sophisticated but also more computationally expensive than filter approaches. In both of these cases, the feature selection step is separated from the main learning task. In contrast, *embedded* methods integrate learning with feature selection, and as such, allow the feature set that is used to be adapted in a single invocation of the main ML algorithm.

Temporal Sequence Learning

The goal of temporal sequence learning is to find solutions to a *sequential decision task* through a trial-and-error interaction with an environment [6, 134]. Thus, in contrast to other types of problems, an episode of a sequential decision task involves multiple time steps. As such, allocating credit among individual actions in a sequence leading to a particular outcome is not straightforward – this is the issue of temporal credit assignment [6]. For example, decisions made early on may have ultimately led to the success of a solution, but if the reward is delayed, e.g., until the last step, this may not be easily recognizable.

[3]The terms 'feature' and 'attribute' are used interchangeably in this work. Strictly speaking, attributes are the input components as specified in the original problem, whereas features may represent intermediate abstractions of the original attributes.

Approaches to the temporal credit assignment problem have traditionally assumed one of two forms [6]. Under *phylogenetic* learning, the focus is on evaluating the solution as a whole and not on the individual actions that it takes, i.e., a solution is evaluated at the end of an episode. In this way, the issue of temporal credit assignment is effectively side-stepped. Under *ontogenetic* learning, on the other hand, a solution is gradually refined with respect to the individual decisions that it makes at each time step. The latter is synonymous with *reinforcement learning*, so in this work the term 'temporal sequence learning' is used to accommodate both approaches to the temporal credit assignment problem, i.e., it has been suggested that because phylogenetic approaches cannot take advantage of information until the end of the episode they are not methods of reinforcement learning [181].

Another way to characterize temporal sequence learning algorithms is with regards to the search space upon which they operate [134]. A temporal problem can be viewed in terms of a value function which returns the reward associated with each state-action pair, i.e., the outcome of taking the action while in the associated state. If this value function is known, it can be used as a basis for a policy, e.g., in any given state, choose the action associated with the highest reward. Two broad categories of algorithms can then be considered, those that attempt to learn this underlying value function, and those that search through the space of policies without explicitly evaluating state-action combinations. The former is more compatible with ontogenetic learning since it recognizes individual decisions, whereas the latter policy-based approach is more consistent with phylogenetic learning.

EC methods for temporal sequence learning are typically phylogenetic and policy-based, and this is true of the proposed approach. However, it is recognized that approaches that search both through the value function space and the policy space have their advantages and disadvantages, therefore, the aim of this work is not to suggest that one is strictly better than the other.

Finally, it is noted that in general temporal sequence learning is viewed as more difficult than classification. In particular, classification can be interpreted as temporal sequence learning restricted to a single time step [181]. Temporal problems may present additional challenges as they may be non-Markovian, stochastic, non-stationary, continual (versus episodic), and also because they may be associated with

very large (possibly infinite) state spaces [6]. In this work, only Markovian tasks are considered, i.e., the transition (reward and next state) depends only on the current state and action [129].

1.2.2 Modes of Problem Decomposition

Problem decomposition can occur in a number of different ways, and the exact nature of the decomposition that results may be problem-dependent. Under classification, problems may be decomposed with respect to the instance space and with respect to the attribute space. In both cases, different solution subcomponents are associated with different, non-overlapping, sets. In the former case, the sets contain the exemplars, while in the latter case, the sets contain attributes. Naturally, problem decomposition is likely to be more useful if the number of exemplars and the number of attributes is large. Similarly, under temporal sequence learning, decomposition may take place on the state space and on the attribute space. However, given that temporal problems often involve very large state spaces and relatively small numbers of attributes, the former is likely to play a more significant role.

1.2.3 A Systems Analysis Perspective

From a systems analysis perspective, a problem can be formulated in terms of an input connected to an output through an intermediate model [39]. Depending on which of these components is unknown, one of three problem types can be identified, Figure 1.2. Under *simulation*, Figure 1.2a, the input and the model are known, and the goal is to use these to generate an output so that it can be observed. Often, the model represents an abstraction of an object that is expensive or impossible to physically construct, and simulation is therefore used to predict that object's behaviour. Under *optimization*, Figure 1.2b, the model and the desired output (or a characterization of the desired output) is known, and the task is then to find an input that will produce the desired output. For example, in function maximization, the goal is to find a set of arguments that result in the largest function value. Finally, under *model building*, Figure 1.2c, the task is to find a model that adequately reflects the input and the output information; this is the case in classification, where the labels associated with each input exemplar are known, as well as temporal sequence learning, where the

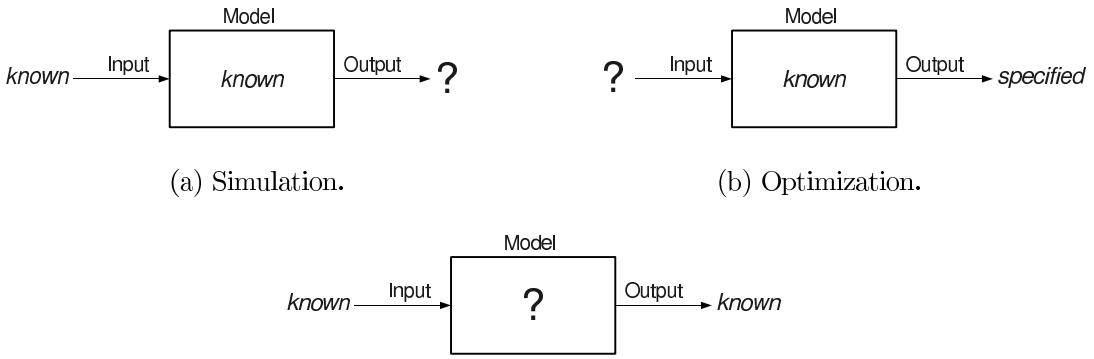


Figure 1.2: Three types of problems identified through a systems analysis perspective, adapted from [39]. The unknown component is denoted with a '?'.

output is characterized in terms of reward maximization. In this thesis, the focus is thus on model building.

Of the three problem types described above, only optimization and model building require that a search take place. Consequently, simulation is typically not considered under ML. In the case of optimization, the search space is the set of points (e.g., bit strings) representing valid inputs, whereas under model building, the search is executed on the space of possible models. Assuming that models are more structured than points, model building may be less likely to suffer from learning-and-forgetting, i.e., the model may act in part as a memory mechanism. Coevolution, Section 2.4, where cycling is identified as potential pathology, may therefore be more successfully applied under a model building context.

1.2.4 Representation, Cost Function, and Credit Assignment

Numerous decisions influence the performance of a ML algorithm, both in terms of the efficiency of the search process, as well as the quality and complexity of the resulting solutions. To complicate matters further, the choices may be interdependent and involve tradeoffs. From an abstract perspective, these design decisions can typically be categorized in one of three ways:

1. **Representation.** The choice of representation concerns the form of the candidate solutions. Nominally, the representation should define the basic elements

that can be used in their construction, as well as the rules governing the way in which these elements may be organized or how they may interact. As such, the representation specifies the set of all possible solutions to a problem, placing the onus on the search process to find the best of these [5]. Example representations include production systems, neural networks, and decision trees. Unless an appropriate representation is used, the algorithm may lack the capacity to adequately characterize the problem. For example, a system that uses boolean arithmetic will not be effective if applied to a real-valued regression task.

2. **Cost function.** The cost function refers to the process for assessing solution quality in terms of the original problem objectives. As such, it should provide a measure of how close a candidate solution is to the optimum. Unfortunately, the definition of an appropriate cost function may be complicated by a number of factors. First, an appropriate cost function may not be obvious, requiring the designer to explore heuristics that may not accurately reflect the underlying problem goals. Second, even if an appropriate cost function can be identified, it may be computationally infeasible, requiring instead the use of an approximation. Finally, even if an ideal and efficient cost function is available, it may not meet the restrictions imposed by the learning algorithm such as, e.g., smoothness constraints. Cost function design is often tied in with the definition of stopping criteria, i.e., the algorithm may be halted when a solution of sufficient quality has been found, or when a computational limit specified with respect to the cost function has been reached.

3. **Credit assignment.** The issue of credit assignment[4] refers to the manner in which information gleaned from the cost function is used the direct the search towards progressively better solutions, i.e., it refers to the way in which the ML algorithm uses information about the quality of solutions found so far to explore new solutions. In this regard, it often involves a tradeoff between the *exploitation* of current knowledge versus the *exploration* of novel but potentially useful solution configurations. Thus, the process of credit assignment provides a link between the cost function and the set of possible solutions under the

[4]Temporal credit assignment, a specific instance of credit assignment, has already been discussed in Section 1.2.1.

chosen representation, and can be viewed in terms of the policies and operators that form the basis for the search.

The above provides an overview of the key design issues in ML; they are further discussed in more concrete terms in the context of a class of algorithms that is particularly relevant to this work, Section 1.3.2.

1.2.5 Fitness Landscapes

Viewing ML as a search process, the interaction between the representation, cost function, and credit assignment components can be characterized in terms of a *fitness landscape* [106]. In particular, the credit assignment policy typically specifies *search operators* which leverage knowledge regarding candidate solutions found so far to produce new candidate solutions, i.e., these operators are applied in the representation space. Under this scheme, a single application of a search operator is referred to as a *search step*. The search operators thus define a topography over members of the representation space where the distance between two points depends on the number of search steps (and their magnitude) required to reach one point from the other, e.g., a point's *neighbourhood* can be defined as all other points reachable via a small number of search steps. A fitness landscape can then be constructed by considering, across the entire topography, the value of the cost function for points within an arbitrarily small neighbourhood.

For example, in a maximization problem involving two parameters, the fitness landscape can be visualized by representing the two parameters on the x- and y-axes, and for each of their combination, plotting the resulting value of the cost function on the z-axis. The resulting surface may be characterized in terms of a number of features such as local and global peaks, ridges, valleys, flat areas, and basins of attraction. The exact configuration of these features affects the difficulty of the search, e.g., the presence of many local peaks in a maximization problem may make it difficult to identify when a global peak is reached. These features can thus form a useful vocabulary for discussing the search process.

Unfortunately, the metaphor of a fitness landscape is limited as a visualization tool in applications where the surface is to be constructed over more than two dimensions or where the neighbourhood relationships are non-trivial, i.e., visualization

is difficult if not impossible in problems other than numerical optimization involving two parameters [106]. In particular, fitness landscapes where the representation involves complex structures, as in the case of GP, typically cannot be visualized. Another limitation of fitness landscapes is that they cannot be easily applied under cost functions with more than one objective. However, the metaphors suggested by the associated vocabulary can nonetheless be helpful, e.g., it is generally well understood what it means for the search process to be stuck in a 'local peak'. Furthermore, the dependence of the shape of the fitness landscape on the representation, cost function, and search operators (a form of credit assignment) emphasizes that these design choices are critical in the success of the search algorithm.

1.3 Evolutionary Computation

In this work, an *evolutionary system* is used to refer to a framework reflecting the theories of Alfred Russel Wallace [187] and Charles Darwin [32] regarding the way in which populations of biological life forms change and diversify over time (these are further discussed in Section 1.5.2). Here, it is noted that a key component in these theories is *natural selection*, an evolutionary mechanism by which the frequency of certain heritable traits, as observed in a population of individuals, may increase or decrease. In order for evolution through natural selection to take place, the following must apply to the system [5]:

1. Individuals in the population reproduce, i.e., give rise to new individuals.
2. Reproduction results in offspring that resemble, but vary slightly from, their parents.
3. The resulting variation in the population affects the chances of an individual's survival and subsequent reproduction, i.e., its *fitness*.
4. The reality of finite resources necessitates competition between individuals so that not everyone is able to survive.

Under these conditions, the principle of natural selection asserts that individuals that are better adapted to the environment, i.e., have higher fitness, will have a better chance of survival and procreation.

EC		**Problem solving**
Environment	$\longleftrightarrow$	Problem
Individual	$\longleftrightarrow$	Candidate solution
Fitness	$\longleftrightarrow$	Quality

Figure 1.3: EC as a metaphor for problem solving, adapted from [39].

Assuming this view of what constitutes an evolutionary system, EC is then defined as the design, simulation, and study of such systems through computational means, e.g., by implementing them on computers. In turn, the term Evolutionary Algorithm (EA) is used to refer to a particular instance of a framework considered under the domain of EC.

EC is therefore viewed as a computational tool for problem solving which uses metaphors from natural evolution to perform a trial-and-error search [39], Figure 1.3. On the surface, such a search strategy may not seem to be particularly effective. However, the variety, robustness, and complexity of the life forms observed in nature would suggest otherwise – this is a direct result of natural selection. Instead of trial-and-error, the term generate-and-test may be more suitable in this context since natural selection effectively tests individuals in their environment to determine if they represent a suitable solution.

Given the pivotal role that evaluation through natural selection plays in evolution, a distinction is made in this thesis between an ecosystem and an environment:

- An **environment** refers to the fixed set of conditions under which an organism is evaluated at any given time.
- An **ecosystem** specifies the environmental conditions and how they change over time.

As such, the concept of an ecosystem is more general than the concept of an environment, i.e., the environment can be determined by considering the conditions specified by the ecosystem at one point in time. Though not universal, these definitions highlight the fact that evaluation can be static or dynamic, where this may have an impact on the rate of convergence observed in the populations that are evolved. In this thesis, evaluation is often discussed in reference to an environment – whether or not the environment is changing is assumed to be clear from the context.

A number of forms of EC are recognized including Evolutionary Programming (EP) [49], Genetic Algorithms (GA) [70], Evolution Strategies (ES) [152], and GP [95]. Despite their differences, these various forms of EC share a set of common characteristics which can be used to define what it means for an algorithm to be an EA [39]. These characteristics are reviewed in the following section.

1.3.1 Elements of an Evolutionary Framework

A typical EA, Algorithm 1.1, begins by initializing and then evaluating a population of arbitrary individuals representing solutions to the problem at hand, lines 2 and 3 respectively. The main loop of the algorithm, line 4, is then executed until some termination condition is met, e.g., a solution of sufficient quality is found or an upper limit on the number of iterations is reached. The core steps of the algorithm are to select the parents, line 5, to apply search operators to generate offspring from the selected parents, line 6, to evaluate the offspring, line 7, and to select a subset of individuals that are allowed to survive into the next iteration of the algorithm, line 8. The last step reduces the size of the population to its initial value, applying a sort of selection pressure, and in combination with the other core steps, simulates natural selection. In this way, the goal behind the main loop is to produce solutions of increasing quality as quantified by the evaluation function used in lines 3 and 7.

Algorithm 1.1 A generic EA.

```
 1: procedure EVOLVE
 2:     INITIALIZE population of random individuals
 3:     EVALUATE initial individuals
 4:     while TERMINATION CRITERION not met do
 5:         SELECT parents
 6:         GENERATE offspring from parents
 7:         EVALUATE offspring
 8:         SELECT surviving individuals
 9:     end while
10: end procedure
```

Variations on the scheme described in Algorithm 1.1 are common [5, 87], but in most cases, an EA requires that the following be specified [39]:

- **Representation.** An individual is often viewed in terms of its *genotype* and its *phenotype*. The phenotype reflects an individual's observable characteristics and behaviour in the context of the original problem; its genotype, or representation,

is the low-level encoding of the phenotype upon which the EA directly performs the search. This dual view is adopted because in biology heritable characteristics are encoded primarily in an organism's genotype, whereas natural selection applies to the traits that are expressed in the associated phenotype.

- **Fitness function.** The fitness function measures the quality of an individual in terms of the original problem specification, i.e., with respect to the individual's phenotype. Once determined, the fitness values are used during parent selection and subsequent individual replacement, reflecting the individual's chances of survival and procreation.

- **Population.** The population is a collection of genotypes representing the candidate solutions. It is the population that evolves since, once they created, the individuals within the population remain fixed. A key factor in maintaining the plasticity of a population, and thus the effectiveness of the EA, is the preservation of genotypic diversity.

- **Parent selection policy.** The parent selection policy defines which individuals in the population are chosen to reproduce. Though typically stochastic, this selection is normally biased in favour of higher-fitness individuals. As such, genetic material found to be useful is exploited with the goal of improving overall solution quality.

- **Variation (search) operators.** Variation operators are components of reproduction, the latter referring to the more general process by which genomes are transferred from parents to offspring, and as such are applied in the genotype space. If only one parent is involved, i.e., the case of asexual reproduction, the variation operator is typically limited to *mutation* in the form of random and unbiased perturbations to the target genome. Sexual reproduction, involving two parents from the same species, additionally supports the combination of material from the two parent genomes through *crossover* (or *recombination*) modeled after the biological process of meiosis. Thus, whereas crossover exploits genetic material already present in the population, mutation tends be a background process which explores new material altogether; crossover therefore has the potential to rapidly combine multiple beneficial traits discovered across the population into a single individual. It is also noted that the use of both mutation and crossover under sexual reproduction reflects a neo-Darwinian view of evolution, Section 1.5.2.

- **Survivor selection policy.** Following the creation of offspring through the variation operators, more individuals are present than at the start of the main loop – if the EA continued in this way, the population would grow indefinitely with any substantial progress unlikely. The role of the survival selection policy is to maintain the size of the population by discarding individuals that are less fit, thus modeling competition for limited resources.

Furthermore, an important underlying feature of EAs is that the decisions they involve tend to be stochastic/probabilistic. For example, choices based on fitness, e.g., the parent selection policy, may favour fitter individuals but will not altogether exclude the least fit. This allows the algorithm to escape local optima, but may also result in *genetic drift* whereby changes in the gene pool occur due to chance and do not reflect the observed fitness values. An exception where decisions are not stochastic are *elitist* survivor selection strategies whereby the fittest individuals are always retained, and subsequently, the less fit are always discarded. In general, the strength of the bias towards fitter individuals is commonly referred to as *selection pressure*.

A more complete EA can be formulated by extending the set of common elements described above by including speciation and a changing environment. Speciation is viewed as particularly appropriate whenever sexual reproduction is present because it can be used to explicitly identify potential parent pairings so that the resulting offspring are more likely to be viable, Section 1.3.3. Furthermore, despite multiplication of species appearing as one of the central tenets of neo-Darwinism, Section 1.5.2, convergence to a single genotype is a pathology often observed in EC [57, 127]. This suggests that the canonical EA framework described above, without explicit speciation mechanisms, fails to model certain key aspects of biological evolution.

Use of a dynamic environment recognizes that natural selection can be either *stabilizing* or *directional*, though widespread awareness restricted to the former has often led to the perception that 'selection only eliminates but is not creative' [103]. Stabilizing natural selection takes effect when a population of individuals is already adapted to a fixed set of environmental conditions. In this case, any phenotypes that deviate from the norm will tend to be eliminated. Directional natural selection, on the other hand, may favour extreme phenotypes under dynamic environments when the norm no longer represents the optimal behaviour. Stabilizing natural selection can therefore be viewed as destructive since it suppresses innovations, whereas directional natural selection encourages novelty and can therefore be viewed as a creative force.

The primary benefit of formulating the problem in terms of a dynamic environment

is that it can lead to incremental learning. Specifically, it allows problem instances of increasing difficulty to be presented to the evolving population of candidate solutions in a way that matches their ability. This is consistent with the more general goal of problem decomposition because, assuming that easier problem instances necessitate simpler solutions, this allows basic behaviours to be learned first and then be subsequently incorporated into more complex behaviours. The challenge of incremental learning is to automate the process of identifying an appropriate set of instructive problem instances, and in this work this issue is addressed through the use of coevolution, Section 2.4.

An added benefit of using a dynamic environment is that it may lead to decreased convergence in the diversity of the population. In particular, if changes in the environment result in successful adaptations in the population of individuals, i.e., via directional natural selection, convergence to a single phenotype will not occur unless that phenotype is viable under all the conditions. It is also noted that *niching* is viewed as the equivalent of speciation, but whereas speciation applies to groups of organisms, niching applies to environmental conditions. In particular, a niche is viewed as a subset of related environmental conditions that may lead to the formation of a new species. Thus, if a dynamic environment can lead to the emergence of multiple niches, this can further support phenotypic diversity, which, from the point of view of this work, is a best-case scenario. It is acknowledged, however, that in other cases convergence to a single species may be the desired outcome.

1.3.2 Evolutionary Computation as Machine Learning

Having presented the components of a typical evolutionary framework, it is now possible to link these back to the key design issues in ML, Section 1.2.4. As suggested by the nomenclature, the representation and fitness function components in an EA reflect, respectively, the more general ML considerations related to the representation and cost function. Typical representations encountered when studying EAs include bit strings, real-valued vectors, and parse trees. Often, the terms 'cost function' and 'fitness function' can be used interchangeably, however, in the context of EAs the latter is typically interpreted in terms of the viability of an individual in the evolving population. In addition, the concept of an ecosystem provides a biological metaphor for a dynamic cost function in an evolutionary framework.

The other EA components – namely the population, the parent selection policy, the variation operators, and the survivor selection policy – detail the policies and

operators driving the search and as such reflect the design choices associated with credit assignment. In particular, the search is performed using a population of candidate solutions. With feedback from the cost function, this then allows decisions to be made regarding the suitability of individual population members, where these take the form of the parent selection and survivor selection policies. In the former case, more promising individuals can be further exploited through the application of the variation operators. In the latter case, less promising individuals are discarded and no longer considered in the search, recognizing an upper limit on the resources available in conducting the search, i.e., in terms of the population size. In both cases, the exploration-exploitation tradeoff can be characterized in terms of the amount of selection pressure, with greater pressure suggesting an increased focus on exploitation of genetic material found so far. In contrast, more aggressive variation operators, i.e., those that result in more variability in the offspring, reflect a more explorative search strategy. Finally, since crossover involves the exchange of genetic material already present in the population, it is often viewed as a means of exploitation. On the other hand, mutation, due to its unbiased randomness, is often viewed as a mechanism for exploration. The above discussion thus suggests that choices between exploration and exploitation appear throughout an EA, and that successful application of the algorithm relies on balance in the associated tradeoffs.

1.3.3 The Merits of Sexual Recombination

For natural selection to be effective, the transfer of genetic material from parent to offspring must be both stable and variable [5]. Lack of stable inheritance implies that offspring will not resemble their parents in which case selection of a fit parent will unlikely lead to a fit offspring – worse yet, the offspring may represent a degenerate phenotype. At the same time, diversity in phenotypes is necessary so selection can be based on a variety of options, therefore, inheritance should result in slight perturbations in the genome.

The variation operators used in EAs simulate the transfer of genetic information as observed in nature. In particular, crossover is meant to model biological sexual recombination[5] whereby offspring receive a mix of the parents' genetic material. However, crossover operators in EAs generally do not model the homology of sexual recombination in nature which is the primary reason why the latter is both stable

[5]Strictly speaking, crossover is modeled after the process of meiosis observed in diploid organisms, i.e., the creation of sperm and egg cells that happens before mating [39].

and variable [5]. As a result, with the exception of ES [16], crossover as commonly implemented in EAs tends to be unstable and even destructive in the sense that even if the parents are fit the offspring are likely to be degenerate, e.g., such destructive effects have been demonstrated under various forms of GP [140, 142, 183]. For this reason certain variants under the EC paradigm, notably EP algorithms, do not use crossover [39].

In contrast, *homologous recombination* refers to the exchange of genetic material in terms of functionally equivalent units, i.e., the exchange is made in such a way that the overall functionality expressed by the genome is preserved while introducing relatively slight variations. This implies that for homologous recombination to be feasible, the parents' genomes should be similar, otherwise, functionally equivalent units may not always exist. The process is therefore greatly facilitated if the parents belong to the same species because, arguably, it then becomes easier to identify matching functional units. Put another way, crossover at an arbitrary location, as is common practice in EC, is more likely to result in a viable offspring if the parents belong to the same species since speciation is one way to model homology. The probability of crossover resulting in viable offspring is therefore viewed as a spectrum, with the likelihood increasing as the operator more accurately models the homology observed in nature.

The process of homologous recombination is further complicated if the genetic material involves a high degree of *epistasis*, the latter referring to the amount of genetic interaction that exists among different components in the genome. For example, exchanging two units that appear to be functionally equivalent may result in degenerate functionality if certain links are broken, i.e., the units are placed in inappropriate contexts. A related concept is that of *genetic linkage* which refers to the tendency of certain combinations of genes to be inherited together. In particular, if a subset of genes exhibiting a high level of epistasis also exhibit strong genetic linkage, then it is likely that in the process of inheritance their aggregate functionality will be preserved. Thus, genetic linkage may be viewed as a solution to the complications introduced by the presence of epistasis.

The merits of using crossover as a variation operator are still open to debate as evidence suggests that in its current forms it may be no more than a macro-mutation [5, 39, 48]. Alternative interpretations regarding the role of recombination have also been proposed, e.g., as in the *genetic repair* hypothesis where it is asserted that recombination may dampen the effects of harmful mutations [16]. The view adopted here is that existing implementations of crossover are an oversimplification, and in particular, that they are not able to take advantage of contextual information (which

may not even be present) in the way that biological recombination does. Furthermore, symbiosis, Section 1.4, is viewed as an alternative to sexual recombination with an increased potential to identify, compartmentalize, and thus protect useful combinations of building blocks. Like sexual recombination, symbiosis therefore fulfills a similar role in that it involves the exchange of genetic material between individuals.

1.3.4 Genetic Programming

GP is a form of EC where the individuals are represented as computer programs, i.e., they accept input, apply operations to process the input, then produce an output. Most commonly, the genotype takes the form of parse trees [95] where a *terminal set* specifies the possible leaves and a *function set* defines the possible internal nodes. Together, the terminal and function sets define the *primitives*, or basic building blocks, used to build expressions, i.e., subtrees. The *closure property* in tree-based GP states that a function should be able to accept any argument it may be presented with, whether it is a terminal or a return value of another function [95]. Satisfying the closure property is trivial when terminals and return values are all of the same type, otherwise, strongly typed GP may be required [130]. Subject to the closure property, mutation and crossover in tree-based GP is then allowed to build arbitrary structures. Representing a superset of structures evolved using tree-based GP, graph-based representations have also been proposed [184].

There also exist variants of GP with linear genomes. Of principal relevance to this work is *linear GP* [20] where the genotype takes the form of assembly-level code operating on external inputs and a set of internal registers, i.e., code that runs on a register machine [29]. For example, a bit string may be interpreted as a sequence of words each consisting of a mode, opcode, and operands; this representation is used here, and is described in more detail in Section 3.6. Analogous to tree-based GP, the concepts of a terminal and function set as well as the closure property apply under linear GP.

All forms of GP fall under the model building paradigm, Figure 1.2c, since the programs that are induced represent models. It is stressed that the proposed approach does not rely on any specific form of GP[6], and linear GP was chosen partly for its efficiency [141] and partly for historical reasons. In general, advantages typically associated with GP are as follows:

[6]In fact, the proposed approach can use any evolutionary approach for model building to produce the bidding behaviour.

1. GP can be applied to specific problems without incorporating a lot of *a priori* knowledge. For example, a problem-specific function set is typically not necessary, placing the onus on evolution to find novel ways of combining operations.

2. If required, GP is flexible in terms of the function set that is used, e.g., the models can be based on a logical or arithmetic function set.

3. The models evolved using GP can represent arbitrarily complex and non-linear functions.

4. The lack of algorithm-specific constraints on the fitness function, e.g., smoothness, means that its design can focus on the problem domain requirements, potentially leading to improved search in terms of quality and efficiency.

5. As an embedded approach to feature selection [59], GP determines which features to use as part of the learning process. The feature values themselves require little preprocessing as they are scaled automatically.

6. Using GP, it is possible to evolve solutions that can be interpreted by people, especially if complexity is low, where this capability is becoming increasingly more desirable [41, 122, 131, 186].

7. GP can be deployed across entire categories of task domains with little modification. In contrast, ML approaches are often geared towards a single domain only, e.g., an algorithm designed for classification may not be applicable to temporal sequence learning.

The main drawback of GP, and many EAs in general, is that it is computationally expensive [5], with parallelization [15] and stochastic sampling of fitness cases [55] used as the main approaches to reduce overhead. In addition, canonical GP implementations are best suited for problems requiring binary decisions. For example, when the program output is a real-valued scalar, GP can be applied to binary classification problems by using a wrapper function [95]; in this case, output less than zero would be associated with one class, and all other output with the other. Otherwise, alternate approaches, e.g., binary decomposition [92], must be used.

A side-effect of evolution typically observed under GP is that program code may contain *introns*, that is, structures that do not affect an individual's chances of survival [5]. Alternatively, an intron can be viewed as any code segment that has no effect on the output for all possible inputs [18]. The latter definition is adopted here, in which

case, two forms of introns are recognized. *Structural* introns apply operations to variables which do not affect the final program output. For example, in linear GP, if the program output is extracted from register zero, then all instructions following the last instruction where register zero is the destination are structural introns. *Semantic* introns, on the other hand, can consist of one or more instructions whose net effect on a variable is nil, e.g., addition of zero, even if the variables accessed are relevant to the final program output. As such, semantic introns are much more difficult to detect.

Evidence suggests that introns emerge during the evolutionary process because they may help to protect useful code segments, i.e., those making a positive contribution towards an individual's fitness, from the destructive effects of crossover [5]. Introns also provide a mechanism by which genetic material can be retained in the population without being acted upon by selection pressure [18]. As such, diversity can be maintained, and novel structures can be explored without having to immediately reflect on an individual's performance. Unfortunately, introns are a drain on computational resources since their execution has no immediate consequence. If they can be identified, they should therefore be bypassed prior to program execution. Under tree-based GP, where all nodes are involved in the expression defined at the root, there are effectively no structural introns, so the focus has to be on the difficult task of locating semantic introns. On the other hand, the use of linear GP facilitates the run-time identification of structural introns during evolution, potentially leading to a considerable speedup [18]. The algorithm for removing structural introns that is used in this work is described in Section 3.6.

1.4 Symbiosis

It was suggested in Section 1.3.3 that the transfer of genetic material from parent to offspring should be both stable and variable. Sexual reproduction supports these requirements through a three-staged process involving species identification, gene alignment, and then variation through crossover. Conversely, asexual models of reproduction adopt a somewhat different approach in addressing the issues of stability and variability. In the following, the focus is on the role of symbiosis in the development of asexual organisms, where this model of evolution provides the motivation for the research undertaken in this thesis.

Characterization	Outcome	
	Partner A	Partner B
Mutualism	Beneficial	Beneficial
Parasitism	Beneficial	Harmful
Commensalism	Neutral	Beneficial
Competition	Harmful	Harmful
Amensalism	Harmful	Neutral

Table 1.1: Examples of cost-benefit characterizations associated with different symbiotic relations. Each is valid under de Bary's definition of symbiosis. In contrast, modern interpretations of symbiosis tend to be limited to mutualism [102, 120] in which case both parties benefit from the relationship. Only pairwise relationships are considered, though these could conceivably be used to obtain a characterization in situations involving more individuals.

1.4.1 What is Symbiosis?

In 1879, the mycologist Anton de Bary coined the term symbiosis in reference to the living together of different organisms [8]. His original definition suggested a tendency towards permanence and physical proximity, but did not specify the cost-benefit characterization of the relationship [31, 102]. That is to say, so long as the relationship involved sufficient intimacy, symbiosis could be viewed in terms of any one of a number of possible combinations of outcomes, Table 1.1. For example, under de Bary's definition, both partners could benefit, as in mutualism, or one of the partners could benefit at the expense of the other, as in parasitism. However, de Bary's requirement for intimacy excluded ecological interactions such predator-prey relationships between different species.

Subsequent to de Bary's definition of symbiosis, a spectrum of interpretations relating to what it is and what it is not have been proposed. In general, these can be categorized along two main lines of thought [102, 120]:

1. Symbiosis occurs whenever two different organisms live together in close association, regardless of the overall outcome on the parties involved. This interpretation follows de Bary's original definition, and suggests that symbiosis plays a significant part in evolution. It is considered to be the classical view of symbiosis.

2. Symbiosis is a mutually beneficial relationship between two dissimilar organisms, i.e., symbiosis is mutualism. This interpretation is more restricted than

de Bary's original definition and is more in line with the notion that the role of symbiosis in evolution is relatively small. This is considered to be a modern view of symbiosis often encountered in contemporary texts.

The modern interpretation of symbiosis can therefore be viewed as de Bary's original definition restricted to a single cost-benefit characterization. Given that a widely accepted definition of symbiosis is not recognized even among biologists [120], finding a definition that is consistent with the current state of the field of biology is beyond the scope of this work. Instead, as discussed below, an interpretation of symbiosis is adopted that is viewed as particularly useful for model building under an EC framework. It is noted, however, that the definition of symbiosis assumed here is more in line with the classical interpretation.

1.4.2 Terminology Used in this Work

In this work, the term *symbiont* is used to refer to an entity that exists in a symbiotic relationship with one or more other entities. Under certain forms of symbiosis, one of the symbionts may be a *host*, i.e., if it contains one or more of the associated symbionts. When this is the case, the terms host and symbiont are used to refer, respectively, to the containing entity and the entity being contained, though technically both are symbionts in the relationship. This distinction is further developed below in the context of the specific definition of symbiosis that is adopted; it is also noted that although the discussion often centers around species, the generic nature of that definition makes it compatible with other units of organization. Here, species is assumed to mean a set of organisms capable of 'natural reproduction resulting in viable and fertile offspring' [150], where this assumes inheritance through sexual recombination.

1.4.3 Some Biological Context

In nature, symbiosis can be viewed in terms of its spatial and temporal properties [120]. Whereas spatial properties refer to the physical proximity of the symbionts involved or the way in which the physical association is made, temporal properties characterize when in the symbionts' lifetimes the symbiotic association is made, e.g., the association could be temporary, permanent, or cyclic. If a symbiotic relationship exists it does so somewhere and at some point in time, and it can therefore always be described in terms of these two properties. The cost-benefit characterization of the relationship, Table 1.1, can be viewed as a third property applicable to every symbiotic association.

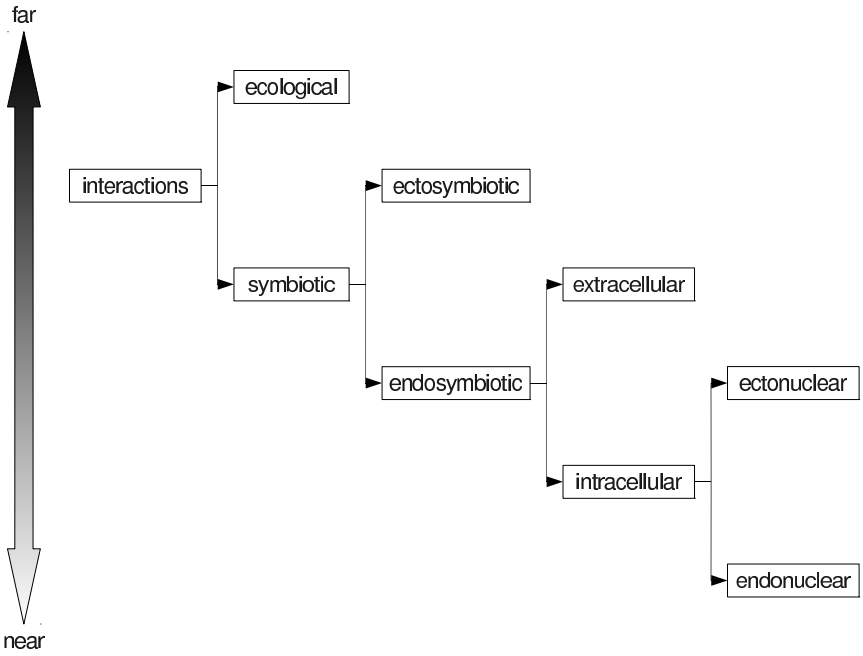


Figure 1.4: Spectrum of possible interactions between organisms of different species in terms of physical proximity. Under this interpretation, ecological interactions do not represent a form symbiosis.

As an example, spatial interactions can be viewed in terms of a continuum of relationships [31] ranging from the very distant to the very close, Figure 1.4. In this particular case, ecological interactions such as those between predator and prey or herbivore and plant are not considered to be forms of symbiosis, though this distinction is not made by all biologists. When ecological interactions are excluded, what remain are two main types of symbioses, *ectosymbiosis* and *endosymbiosis*. Ectosymbiotic interactions, e.g., behavioural or attachment relationships, do not involve one of the symbionts living inside the other. In contrast, under endosymbiosis, one of the symbionts lives inside the body of the other, the latter assuming the role of the host. Endosymbiotic symbioses can be *extracellular*, whereby the symbiont lives inside a cavity of the host or between the cells in the host's tissue, or *intracellular*, whereby the symbiont resides within the host's cell. Intracellular interactions can be further broken down into *ectonuclear* and *endonuclear* relationships depending on whether or not the symbiont lives outside or inside of the host's nucleus.

A wide range of interactions or processes supporting symbiosis in nature have also been observed which may appear in some symbioses but not others. These include

behavioural, allelochemical, anatomical, developmental, metabolic, and genetic relationships [120]. For example, metabolic relationships involve one organism consuming the metabolic byproduct of another organism, whereas genetic relationships may involve the exchange of genetic material between symbionts. These interactions are not exclusive in the sense that they are often observed in combination.

Horizontal gene transfer, an example of a genetic process supporting certain forms of symbiosis [31, 120], refers to the transmission of mobile genetic elements across species boundaries [165]. It has featured within endosymbiotic theory [126, 159, 188], the latter suggesting that certain organelles (mitochondria and plastids) in eukaryotic[7] cells originated from bacteria that were assimilated by the host cell. Another observed example where horizontal gene transfer has played a role is in the development of bacterial resistance to antibiotics through the transfer of 'prefabricated' resistance-conferring genes [165] – in contrast to mutation coupled with natural selection, this can result in much quicker adaptation. In general, horizontal gene transfer has typically been observed in simple life forms, e.g., prokaryotes, where asexual reproduction is the dominant mechanism of inheritance [36, 165].

Research into horizontal gene transfer has led to the reconsideration of two traditional insights related to the way that life evolved on Earth. First, it has been used in arguments against the notion that the evolution of species can be represented in terms of a 'tree of life' [32]. Specifically, it has been suggested that a single genealogical tree is an oversimplification, and that a more accurate model would be one where the branches merge together wherever genes are exchanged between 'lineages' [36], i.e., the tree of life should really be a network of life. Second, though neo-Darwinism asserts that with respect to biological evolution, sexual reproduction is the key mechanism for the transmission of traits, horizontal gene transfer has allowed asexual microorganism to develop most of the fundamental processes that are used by contemporary, higher-level, life forms [165]. As such, it appears to represent an alternate mechanism for achieving genetic innovation that has had a profound effect on the course of evolution. Though acknowledged to be a valid mechanism, as is the case with symbiosis itself, the significance of horizontal gene transfer remains a topic of debate [100].

The above discussion is included in order to highlight two points. First, many forms of symbiosis are observed in nature, making it difficult to specify a single definition that is applicable in all cases. Second, there appear to be many opinions

[7]In contrast to prokaryotes, eukaryotes have a member-enclosed nucleus. Furthermore, whereas eukaryotes are capable of genetic recombination, prokaryotes tend to rely on unilateral gene transfer.

and theories held by biologists regarding the way in which life evolved, sets of which range from inconsistent to contradictory.

1.5 From Biological Evolution to Evolutionary Computation

1.5.1 Symbiosis as an Operator

There is disagreement regarding the definition of symbiosis [31, 102, 120], and as such, it is not surprising to see a lack of consensus regarding its role in evolution and its impact in present-day life. Part of the reason for this may be that as a relationship, symbiosis can be regarded either as state or as an operator [31]. Defining symbiosis as a state means viewing it in terms of the characteristics that distinguish it from other types of relationships. For example, as discussed earlier, modern interpretations tend to view mutual benefit as a key property of symbiosis, thus excluding other characterizations such as, e.g., parasitism. One the other hand, as an operator, symbiosis is viewed in terms of a series of steps, i.e., symbiosis is viewed as an algorithm[8].

In this work, symbiosis is regarded as an operator. This interpretation is consistent with the classical definition of symbiosis, Section 1.4.1, as it does not associate a single cost-benefit characterization with the interaction. Whether or not the same underlying process is the basis for all instances of symbiosis is open to debate. It is therefore difficult, if not impossible, to capture all aspects of symbiosis in nature within a single computational model [31]. In this work, the focus is on the ability of symbiosis to support problem decomposition through complexification, i.e., the combination of relatively simple components into a single functional unit which may then serve as a component for successive, higher-level, units. This abstraction is seen as particularly useful for model building under an EC framework because it may allow increasingly complex behaviours to be evolved without an *a priori* specification of the subcomponents.

Specifically, in this thesis, symbiosis is viewed as a process consisting of three distinct stages [121], Figure 1.5:

1. The coexistence of two or more species[9].

2. The compartmentalization of individual members from the different species within a host entity.

[8]Though the end result of an algorithm may also be viewed in terms of properties, where this may be a source of further confusion.

[9]As in [121], the process is described in terms of species, though it is equally valid when applied at other levels of organization.

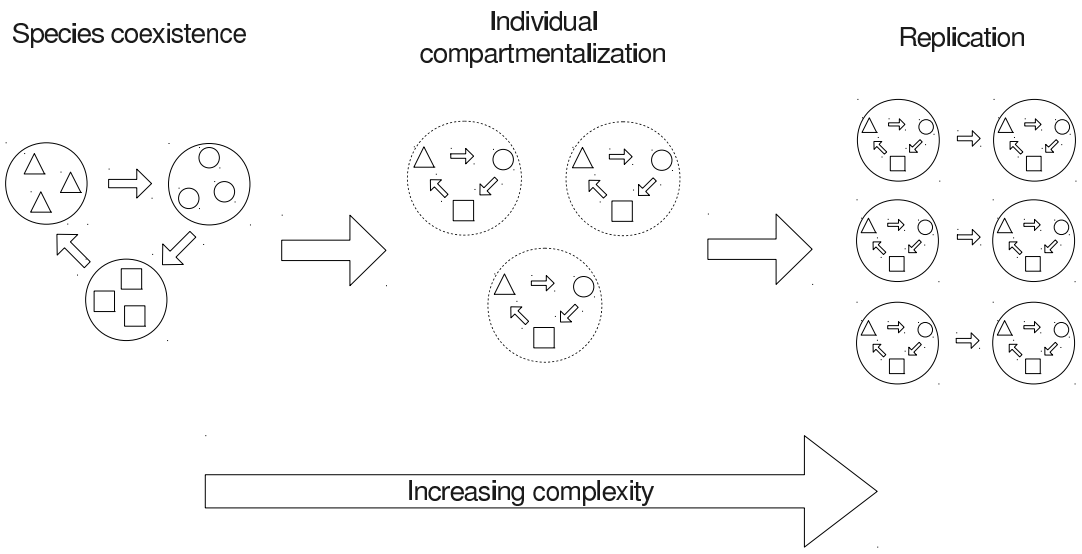


Figure 1.5: Symbiosis as a three-staged operator [121]. The illustration involves three species (circles, triangles, and squares), though it applies to any number of groups at different levels of organization. The first stage involves the coexistence of the three species, and may lead to the compartmentalization of individuals from different species in the second stage. In the third stage, a mechanism emerges for the replication of the coupled individuals which are then interpreted as an increasingly complex unit of evolution. The entire process may repeat using the latter as the building blocks in the next iteration.

3. The emergence of a mechanism for the joint replication of the compartments that are established.

The third stage in this definition effectively validates that the interaction between individuals from different species is beneficial under the pressures of natural selection.

The second stage, referring to the compartmentalization of species members, is taken to mean a closer association between specific individuals than is observed in the first stage. It represents an intermediate step when the compartment is tested under natural selection but before it is able to replicate. This tighter association is taken to be the sole condition defining compartmentalization, and the nature of the compartment itself remains otherwise unspecified. In some cases, it may mean that the interactions occur only between individuals in the same compartment, though this need not always be the case. Furthermore, the compartment need not be a physical entity. In terms of the terminology introduced earlier, the compartment could also be one of the symbionts, i.e., the host, so the terms 'compartment' and 'host' will be used interchangeably.

A consequence of natural selection acting on the compartment, as opposed to being applied to each symbiont individually, is that it now effectively represents a group selection (fitness) operator. As such, it may now be the case that the fitness of the group is greater than the sum of the symbionts' fitness values had they been evaluated individually. Should this happen, it would then support progression to the third stage of symbiosis.

Though abstract, the process of symbiosis described above is consistent with certain results in evolutionary biology, and in particular, it may reflect the transition from prokaryotes to eukaryotes [121]. Suggestions of a more speculative nature have also been made regarding the role of this process in a number of other evolutionary transitions in which the *units of evolution* have become increasingly complex [121]. A related fundamental question that was posed was why selection on the lower-level units did not disrupt the higher-level units. In this work, this issue was bypassed by limiting selection to the higher-level units.

Naturally, symbiosis as summarized in Figure 1.5 is not universal and does not reflect all cases in biology. For example, the model is likely to be invalid under ectosymbiotic relationships such as attachment and behavioural interactions, e.g., sea anemones clinging to the shells of hermit crabs. In these cases, the symbioses are not likely to include the third step involving the synchronized replication of the symbionts as a compartmentalized unit. However, the adopted definition of symbiosis is viewed as concise and generic, and thus suitably applicable in a ML context.

Finally, it is emphasized that any of the cost-benefit characterizations summarized in Table 1.1 are valid under this definition of symbiosis. For example, the trial-and-error nature of EC means that the symbiotic association made may be harmful to all involved, i.e., the interaction may be strictly competitive, so that the joint replication stage may not be realized since there is no associated benefit. Naturally, it is expected that mutually beneficial interactions, i.e., mutualism, will result in improved fitness in the higher-level units that are produced.

1.5.2 Symbiosis versus neo-Darwinism

The notion that life today has developed from a common set of ancestors through a series of transformations, i.e., the principle of evolution, is widely accepted as a fact [101]. What is not yet fully understood, and what remains a topic of debate, are the mechanisms driving evolutionary change. For example, even among advocates of symbiosis and its role in evolutionary history [102, 120], there are those that argue that

its significance has been overemphasized [102]. Horizontal gene transfer, discussed in Section 1.4.3, is another mechanism which is now providing new insights into how species are related [36]. In this section, symbiosis, Figure 1.5, is compared and contrasted in relation to neo-Darwinism, the latter representing the norm in EC. Before doing so, some historical context is provided to clarify exactly what is meant by neo-Darwinism.

Symbiosis can be viewed as a form of Lamarckism [135], the latter referring to the theory which asserts that traits acquired during an organism's lifetime can be passed on to its offspring [105]. Through symbiosis, such 'inheritance of acquired characteristics' [101] is in fact possible, e.g., the acquisition and subsequent transmission of organelles as suggested by endosymbiotic theory. This is also consistent with the definition of symbiosis adopted here, Figure 1.5, where the third stage can be interpreted as Lamarckian inheritance.

Notably, Lamarckism was part of Darwin's original theories as presented in his 1859 book [32]. Today it is recognized that Alfred Russel Wallace, Darwin's contemporary, made similar propositions regarding the role of natural selection in evolution which he published three decades later [187]. However, Wallace's account did not include the possibility of passing on traits acquired during one's lifetime. Nevertheless, Darwinism, as set of theories, is used to refer to work originally presented by Darwin and Wallace and includes Lamarck's principle [101]. In 1892 August Weismann augmented Darwinism through two significant contributions [192]. First, he provided empirical evidence against Lamarckian inheritance and rejected the principle altogether. Second, he argued for sexual reproduction, i.e., recombination, as a key source of variation in populations, with this latter assertion having a significant impact on EC. The result of augmenting Darwinism with Weismann's conclusions is referred to as neo-Darwinism [101].

Neo-Darwinism is thus founded on Charles Darwin's body of work, which in turn can be interpreted in terms of five theories [103]:

1. Evolution, not spontaneous acts of creation, is responsible for the state of life on Earth – this implies that life is constantly changing.

2. All organisms share a common ancestry.

3. Evolution is gradual, i.e, new species emerge as a result of a long-term accumulation of minute variations.

4. Species multiply, e.g., one species gives rise to two, and thus lead to diversity in life.

5. Evolutionary change is the result of genetic variation within populations coupled with a bias towards certain favourable traits.

Furthermore, taking into account Weismann's contributions [192], it is suggested that the genetic variation in populations is primarily a direct result of sexual reproduction (recombination) – the above theories thus characterize the definition of neo-Darwinism in the context of which symbiosis is analyzed. The last theory above, discussed in more detail in Section 1.3, refers to natural selection as a driving force behind evolution.

Symbiosis, as a mechanism driving evolutionary change, is compatible with the first two theories noted above. That is, if one believes that symbiosis is a valid and significant mechanism of inheritance, this belief does not contradict the notion that life is constantly changing and that all organisms share common ancestral roots.

On the other hand, symbiosis is inconsistent with the three latter theories, Table 1.2. In contrast to the gradual nature of evolution as suggested by neo-Darwinism, symbiosis can result in abrupt changes, e.g., as previously independent functional units are incorporated into a host – this can have troublesome effects such as the rapid proliferation of drug-resistant bacteria. Consistent with the notion of abrupt changes is the view that certain variation operators under symbiosis function on a larger scale. Though neo-Darwinian evolution can also result in significant changes, the process is likely to take much longer.

Symbiosis also provides an alternative to species multiplication as a way of generating diversity in life forms. The fourth theory above suggests that species in biological evolution can be represented in terms of a tree of life. As previously discussed, through symbiosis, the branches in this tree may in fact have merged, with each such event resulting in a new single species that is a combination of multiple existing species. Thus, in contrast to neo-Darwinism, under symbiosis distinct lineages may converge, e.g., as in the transition from prokaryotes to eukaryotes. The vast diversity achievable through symbiosis can be viewed in terms of the number possible combinations of symbionts. Under the restriction that the symbionts represent different species, each such combination may lead to the emergence of a new species.

The final theory refers to natural selection and to the mechanism for producing the genetic variation upon which it operates. Symbiosis is consistent with natural selection, i.e., it provide a mechanism for producing the units upon which natural

	neo-Darwinism	Symbiosis
Rate of change	gradual	abrupt
Possible lineages	single	multiple
Reproduction	sexual	asexual

Table 1.2: Comparison of symbiosis with neo-Darwinism. The differences reflect the last three theories used to summarize neo-Darwinism [103].

selection may then act [121]. However, it does provide an alternate to sexual recombination as a mechanism driving genetic diversity. For example, horizontal gene transfer, which falls under the definition of symbiosis adopted here, is believed to have allowed asexual organisms to develop many of the fundamental processes that continue to support life today [165]; in this case, however, mutation was also required as an underlying source of variation. As mechanisms of genetic variation, a fundamental difference between sexual reproduction and symbiosis is that the former assumes a certain degree of homology whereas the latter does not. Specifically, in contrast to sexual recombination, symbiosis does not assume that the partners belong to the same species.

1.5.3 Discussion

Symbiosis as Context Learning

A number of properties and interactions related to symbioses in nature were discussed in Section 1.4.3. Of these, five are viewed as potentially useful as metaphors applicable in an EC setting, Table 1.3, and thus may serve as a kind of biologically-inspired vocabulary for discussing symbiosis in EAs. Naturally, these are abstractions, and so it is acknowledged that alternate interpretations may be proposed. Furthermore, these abstractions are collectively referred to as relationships, though, e.g., 'temporal' may more accurately reflect a property of symbiosis whereas 'metabolic' may be more consistent with an underlying process. In the following, the relationships in Table 1.3 are discussed in the context of the definition of symbiosis that was adopted in this work, Figure 1.5.

The payoff relationships define the cost-benefit characterization of the interaction, i.e., whether each symbiont is better off or worse off by participating in the association, Table 1.1. They are relevant in the early stages of symbiosis-as-an-operator

Relationship	Description
Payoff	Defines the gain or loss incurred by each symbiont as a result of it participating in the symbiosis
Spatial	Defines the topographical organization of the symbionts, including the amount of separation
Temporal	Defines when in the symbionts' lifetimes they are intimately associated
Metabolic	Supports communication between symbionts, coordinates symbiont behaviours
Genetic	Supports the vertical or horizontal transfer of traits

Table 1.3: Symbiotic relationships adapted to EC.

as suggested in Figure 1.5 because by the third stage fitness is assigned on the compartment as a whole and not to its individual parts. In the earlier stages, before the compartment is finalized and when natural selection still applies to individuals, the payoff relationships may filter out potential symbionts. If the relationship is mutualistic, in which case all symbionts benefit from the association, then natural selection is likely to rule in favour of compartmentalization and joint replication[10].

Spatial and temporal relationships can be interpreted as quantifying the physical properties of the association, i.e., they answer the questions of where and when the association takes place. They are also most relevant in the early stages of the process of symbiosis where they help to identify who may be considered for host membership. In contrast, once the compartment is formed, the association is permanent and in many cases the physical location may be fixed.

Finally, metabolic and genetic relationships are viewed as more intimate in the sense that they represent the processes required to support symbiosis once the compartment is formed. As such, they are assumed to be more relevant in the latter stages of the process. Together, they establish the context for exchanging information between symbionts. Genetic relationships support the transfer of heritable information during replication, whereas metabolic relationships define the behaviour of the compartment as a whole as it interacts in the environment.

The relationships in Table 1.3 can therefore be interpreted according to their role. First are the spatial and temporal relationships under which potential symbiotic associations are identified. Second, the metabolic and genetic relationships define the mechanisms of communication. Finally, the payoff structure that emerges affects the viability of individual symbionts and may reflect the fitness of the host as a whole –

[10]The result of other cost-benefit characterizations, Table 1.1, is not as clear.

in the latter case one can be confident that mutualism will result in higher aggregate fitness.

This discussion suggests that symbiosis may represent an approach for problem solving through context learning, i.e., learning in what situation to apply which solution subcomponents. Under symbiosis, the spatial and temporal relationships may identify potentially useful combinations of subcomponents. The metabolic relationships may then serve to establish the means for collaboration among subcomponents, whereas genetic relationships may support further exploration of the space of possible subcomponents with respect to what has already been discovered. Finally, the payoff relationships characterize when a collaboration is deemed to be effective. Under this interpretation, one can view evolution through symbiosis as an instance of context learning.

With respect to the two views of problem decomposition discussed earlier, Section 1.1, this suggests that symbiosis has the capacity for decomposing the problem laterally. Indeed, context learning was suggested as a means of achieving lateral problem decomposition, Section 1.1. Furthermore, as the process of symbiosis-as-an-operator in Figure 1.5 is repeated iteratively, it can lead to vertical problem decomposition as the units of evolution transition from simple to complex [121]. Given that such incremental complexification results in layers of increasingly complex units, this can be interpreted as a form of layered learning. This thesis thus asserts that symbiosis has the potential to encompass both types of learning within the same framework.

An Alternative to Sexual Recombination

At the end of Section 1.5.2 it was suggested that a fundamental difference between sexual reproduction and symbiosis, as mechanisms supporting evolution, is that the former assumes a certain degree of homology whereas the latter does not. This is significant from an EC perspective where there is a lack of consensus regarding the validity of crossover as a variation operator, Section 1.3.3. Specifically, though the goal of crossover in EC is to simulate biological sexual recombination, typically the implementation is an oversimplification that fails to appropriately model *homologous* sexual recombination. As a result, with the exception of ES [16], the crossover operators used in EC tend to introduce variability but are generally unstable in that fit parents are likely to product unfit offspring.

The issue of an unstable crossover operator has been acknowledged to be a problem in EC. Some have excluded its use, notably, proponents of EP [39]. Others have

tried to address this issue, for example, by employing a more structured crossover that is less likely to produce degenerates [177], or by identifying species so that crossover could be applied locally as in certain classifier systems formulations [196]. Notwithstanding, when it is employed, the norm in EC appears to be to use sexual recombination indiscriminately. This may be justifiable once the population converges to what effectively can be viewed as a single species, because, as suggested earlier, this implies a tendency towards homologous genomes which in turn increase the likelihood of producing viable offspring. If, however, multiple species are present, such indiscriminate application of crossover is viewed as problematic, or in the very least, questionable.

Furthermore, assuming a single species may be reasonable when the fitness function is based on a fixed, relatively small, set of training exemplars. In this case, the entire training set may represent a single niche, and as such, it may have the capacity to support just a single species. In contrast, larger problems, e.g., classification involving massive datasets or real-world applications of temporal sequence learning, may involve sufficiently many environmental states so as to support multiple niches. In addition, if the state space is very large, a subset sampling strategy may be required, where this may also support the formation of new species as criteria under which individuals are evaluated change. Such dynamic conditions may prevent convergence to a single species, and as such, more care must be taken when applying the crossover operator.

Symbiosis provides an alternative to sexual recombination capable of both the variable and stable exchange of genetic material without the assumption of homology. Variability in genetic material is supported through the coexistence of distinct species which may eventually lead to the joint replication of different, possibly non-homologous, genomes. Stability is maintained because a given inter-species interaction does not progress to the joint replication stage unless the partnership is deemed to be mutualistic as measured through natural selection. That is, recombination is stable if parent fitness is correlated with the resulting offspring fitness, Section 1.3.3. Analogously, symbiosis is implicitly stable because the third stage is not likely to be reached unless the partnership between the symbionts increases the group fitness compared to the fitness of the individuals. In effect, once in the third stage of symbiosis-as-an-operator, genetic material from different, possibly non-homologous, sources has been successfully combined into a new organism that is fit and is itself capable of reproduction.

In terms of efficiency, the difference between symbiosis and sexual recombination

is not clear. Sexual recombination may be wasteful if it is likely to lead to unfit offspring since these normally do not advance the search towards solutions of high quality. However, symbiosis still requires that natural selection test the probationary partnerships in determining if joint replication is warranted, i.e., the partnerships may not be effective and thus also not advance the search. Thus, like the crossover operator, symbiosis may also be considered to be a macro-mutation. Nevertheless, symbiosis is viewed as a potentially useful alternative to sexual recombination in part because it has received limited attention from the EC research community. As such, if it receives a proportional amount of consideration, it may turn out to be an effective approach. Moreover, in contrast to recombination, the process of symbiosis assumed in this work clearly suggests a propensity towards problem decomposition through complexification. This, in turn, may lead to increased efficiency in conducting the search if the optimal behaviour of the symbionts is to some degree independent[11].

1.6 Research Objectives

The principal aim of this thesis is to propose symbiosis as a mechanism for context learning and layered learning, i.e., problem decomposition, under an EC framework for problem solving. To this end, the following explicit objectives are identified:

1. To present a novel EC algorithm that incorporates symbiosis as a primary mechanism driving evolution and which supports problem decomposition.

2. To demonstrate that the proposed approach is effective versus comparable approaches with a reduced capacity for problem decomposition.

3. To analyze the nature of the problem decomposition resulting from the application of the proposed approach.

4. To compare the proposed approach against current state-of-the-art algorithms on a set of challenging real-world problems.

5. To explore the design of a dynamic evaluation function and investigate the conditions under which specific design choices are appropriate.

In the following section, an overview of the thesis is provided including references to specific sections where the above objectives are pursued.

[11]This is related to the notion of modular interdependency which is discussed in Section 2.1.3.

1.7 Thesis Overview

In Section 1.1, lateral problem decomposition was described in general terms as the coordination of subcomponents which have similar capabilities, i.e., peers. This description implied that the subcomponents could be applied to tasks of similar scope, and that they were explicitly modular collectively exhibiting non-overlapping behaviours. Vertical problem decomposition, on the other hand, was viewed in terms of the hierarchical organization of increasingly complex units. Here, these definitions are extended, with a particular emphasis on the meaning assumed in this thesis:

- *Lateral problem decomposition* refers to the partitioning of the problem space among structurally distinct solution subcomponents. The partitions represent subproblems at the same level of granularity, while the solution subcomponents represent partial solutions to the overall task which must collectively learn the appropriate one-to-one association between subproblem and subcomponent. The overall goal is thus viewed as an instance of context learning – learning under what problem conditions (when) to apply which solution subcomponent (what).

- *Vertical problem decomposition* refers to the layering of previously learned behaviours with the goal of scaling the process of context learning to a much wider set of scenarios than would be possible under lateral problem decomposition alone. Such layering implies a hierarchical structure with more complex units of organization at higher levels. However, the learning process at each level is still viewed as an application of context learning but assumes a higher degree of granularity.

Under both types of problem decomposition, context learning is viewed as playing a pivotal role; under vertical problem decomposition, it provides a mechanism for choosing which lower-level behaviours to apply when. It is also recognized that learning the concepts of 'when' and 'what' may be facilitated if they are decoupled, i.e., learning a set of related situations is separated from the task of specifying what action is appropriate for each such set.

At the highest level, this thesis can then be described as a presentation of an evolutionary framework based on the biological process of symbiosis that is capable of both lateral and vertical problem decomposition. The content of specific thesis chapters is organized as follows:

- The primary goal of **Chapter 2** is to familiarize the reader with the most relevant ML approaches to lateral and vertical problem decomposition. The chapter begins by making a case for problem decomposition as a problem solving tool essential in many complex systems, and also includes an introduction to some early notions of modularity. This is followed by an overview of approaches to lateral problem decomposition which include ensemble methods, teaming using GP, and bid-based approaches, and which, with the exception of the bid-based approaches, are often observed to result in subcomponents whose behaviours tend to overlap. The chapter then discusses a number of approaches to vertical problem decomposition including a symbiotic approach to neuroevolution whose core architecture is incorporated into the proposed approach. The focus then shifts as the chapter concludes with an overview of coevolution; whereas symbiosis can be viewed as a form of cooperative coevolution, the proposed approach relies on what is traditionally referred to as competitive coevolution in order to support efficient training.

- The proposed framework is described in detail in **Chapter 3**. The key algorithm components presented include a bid-based approach to context learning, an evolutionary cycle based on symbiosis as the primary mechanism of inheritance, and the dynamic evaluation of candidate solutions through coevolution with a population of test cases. The manner in which the proposed algorithm reflects the formulation of symbiosis-as-an-operator assumed in this thesis, Figure 1.5, is also discussed, as is the hierarchical model construction process. At the end of the chapter, the specific form of GP used to identify context is detailed, though it is stressed that the proposed approach is compatible with other representations.

- The first chapter where experimental results are presented is **Chapter 4**. Here, the goal is to compare the proposed algorithm, which is explicitly modular, with a comparable implementation of monolithic GP with the goal of isolating the benefits that an increased capacity for lateral problem decomposition may bring. The two approaches are applied to four binary classification problems and evaluated in terms of classification performance, solution complexity, and training overhead, with the expectation that modularity will help in all three areas. Following the comparison, the solutions evolved under the proposed approach are further characterized in order to gain additional insight into the nature of any problem decomposition that may emerge. The chapter concludes with an analysis of population-wide classification performance under the proposed approach

motivating the construction of additional hierarchy levels.

- Continuing under the classification domain, **Chapter 5** evaluates the proposed algorithm against two well-established algorithms outside of the EC domain that are tailored towards classification. Specifically, the two approaches used in the comparison are a Support Vector Machine (SVM) implementation and boosting; whereas the former is viewed as the current state-of-the-art, the latter is nonetheless still considered to be robust and capable. In addition to the four binary problems used in the previous chapter, two additional multi-class problems are included in the evaluation with the goal of demonstrating the ability of the proposed approach to support multi-class learning. An argument made throughout the chapter is that the tradeoff between complexity and performance should be considered when identifying the most appropriate solution.

- In **Chapter 6**, the problem domain shifts from classification to temporal sequence learning, and specifically, to the task of learning to reverse a cab and semi from an arbitrary location onto a loading dock. The main goal of the chapter is to investigate the design of the policy for managing test case selection, which, under problems with many training instances, is an approach to reducing computational overhead. To this end, a number of policies ranging from the random selection of test cases to explicit coevolution are considered under both real-valued and binary reward schemes. The results of this chapter guide experiment design in subsequent chapters, and in particular, the choice of a real-valued reward scheme in Chapter 7 and the requirement for a coevolutionary function that explicitly models the interaction between test cases and candidate solutions in Chapter 8.

- The primary goal of **Chapter 7** is twofold. First, the chapter aims to compare the proposed approach against a neuroevolutionary algorithm which currently represents the state-of-the-art under temporal sequence learning. Second, for the first time in the thesis, the chapter attempts to evaluate the benefits of the hierarchical construction process, i.e., vertical problem decomposition. In the latter case, a two-level formulation of the proposed approach is compared with a single-level formulation under the same fitness evaluation limit. As in the previous chapter, the problem domain considered is the truck reversal task facilitating the visualization of the behaviour that is learned.

- In **Chapter 8**, the proposed approach is used in an attempt to evolve policies

for solving the Rubik's Cube. In contrast to the truck reversal task, where the inputs and output are continuous, the Rubik's Cube represents a discrete problem domain. Though hand-crafted algorithms for solving arbitrary cube configurations have been developed, the task represents a considerable challenge for ML algorithms which to this point have been unable to produce generic strategies. The proposed approach therefore represents one of very few (if any) documented attempts at automatically learning policies for solving arbitrary cube configurations.

- The thesis concludes with **Chapter 9**. Here, the key ideas underlying the proposed approach are summarized and a retrospective on the research objectives as stated in Section 1.6 is presented. The chapter includes comments on what are viewed to be the main contributions made in this work, as well as suggestions for future work.

Chapter 2

Related Work

This chapter reviews previous work that is relevant to this thesis, either because the proposed approach builds on that work, or because that work is viewed as useful in providing the necessary context in which the contribution of the proposed approach can be assessed. In some cases, the discussion involves only a high-level summary, where the decision to do this was made in order to be able to cover a wider range of topics without losing focus or comprehensibility. The overall intent is to provide sufficient backdrop for the proposed approach to make this thesis self-contained, while providing adequate references so as to allow the reader to investigate further details as required.

The chapter begins with a review of problem decomposition in complex systems, Section 2.1. Here, the modular nature of complex systems is noted, including the frequency with which hierarchies are observed, along with a hypothesis for why this may be so. Early notions of modularity are presented, followed by a discussion of more recent attempts to formalize what it means for a system to be modular along with the associated implications for the evolvability of such systems. Specifically, it is suggested that symbiosis is an appropriate mechanism for evolving modular systems even if they involve non-trivial module interdependencies. Sections 2.2 and 2.3 respectively review relevant approaches to lateral and vertical problem decomposition. With regards to the former, special attention is paid to the 'ensemble methodology' that is common in many lateral approaches but which is not represented under the proposed approach. Section 2.2 concludes with a discussion of bid-based approaches and, in particular, the manner in which they may decompose the space of problem instances. A number of approaches to vertical problem decomposition are then presented in Section 2.3 including a symbiotic neuroevolutionary approach that formalized the architecture subsequently adapted in this work. Both Sections 2.2 and 2.3 are peppered with comments that compare and contrast the proposed approach with those that are therein reviewed.

The proposed approach involves a significant coevolutionary component, and to

this end, Section 2.4 reviews concepts related to both competitive coevolution and cooperative coevolution. The difference between these two general forms of coevolution is discussed, and suggestions are made where each may be suitable. In particular, it is noted that many of the approaches reviewed in this chapter can be considered as a form of cooperative coevolution, though often labeling an approach as strictly one or the other may not be appropriate. Competitive coevolution, on the other hand, is viewed as an approach for constructing a dynamic ecosystem.

Much like the proposed approach, some of the specific frameworks that are discussed in this chapter do not fall under a single section heading, e.g., depending on the exact formulation, some approaches may be used to decompose the problem laterally or vertically. In such cases, alternative variations are often noted, though it is acknowledged that it is unreasonable to account for all possible interpretations.

Finally, it is acknowledged that numerous evolutionary approaches involving symbiosis have been proposed. What are viewed to be the most relevant of these are discussed in this chapter, however, the reader is referred to a recent review [67] in which a comprehensive discussion is presented in the context of the definition of symbiosis as adopted in this thesis, Figure 1.5.

2.1 Problem Decomposition in Complex Systems

This section begins by introducing complex systems, then proceeds to discuss common patterns in their structure, i.e., modularity and hierarchy. The effect of system structure on evolvability is then considered from an EC perspective and related to the evolutionary mechanism of symbiosis.

2.1.1 Complex Systems

Despite having roots as far back as the mid-twentieth century, the interdisciplinary study of complexity has not yet resulted in a universally accepted definition for what it means for a system to be a *complex system* [128]. One early definition views a complex system in broad terms as 'one made up of a large number of parts that interact in a nonsimple way' so that 'the whole is more than the sum of the parts' [163]. This definition implies that a reductionist approach to understanding complex systems is not suitable, i.e., given an intimate knowledge of the constituent parts and their interactions it is difficult to elaborate on the behaviour of the system as a whole. For the purpose of this thesis, this definition of complex systems is viewed as

appropriate as it conveys the core meaning of the term without introducing details that to some may be objectionable.

Examples of complex systems include economies, insect colonies, the immune system, and the World Wide Web; though vastly different, these systems share a number of common properties including complex collective behaviour resulting from relatively simple local behaviour, information processing and communication between system components, and the capacity to adapt in changing environments [128]. Viewed as a metaphor for problem solving, another property of complex systems is that they represent approaches based on problem decomposition. In particular, it tends to be the case that specific components or modules, or subsets thereof, within these complex systems assume specialized local roles that result in the emergence of elaborate global behaviours. In the following sections, notions regarding the structure of complex systems that support the process of problem decomposition are outlined. In doing so, the suggestion is made that whereas the potential benefits of problem decomposition have been recognized for a long time, frameworks which have fulfilled this potential in the context of real-world problems have been lacking.

2.1.2 Modularity, Hierarchy, and Near-Decomposability

One of the earliest to contemplate the modularity present in complex systems was Herbert Simon who specifically recognized that in many cases the modules are organized hierarchically [163], e.g., he cites a wide range of complex hierarchic systems including social, biological, physical, and symbolic systems. Furthermore, the suggestion is made that complex systems tend to be hierarchical because this allows them to be evolved more quickly. That is, the claim is made that the evolutionary process will be more efficient if there are certain *stable intermediate forms* that can act as a scaffolding, which in turn will lead to the formation of hierarchies as simple subsystems are combined to form more complex ones.

Simon's often-cited parable of the two watchmakers illustrates the potential benefits associated with a hierarchical approach to problem solving. The parable describes two watchmakers who both assemble watches consisting of about 1000 base elements. The approach of the first watchmaker is to assemble the watch all in one go. The second watchmaker, on the other hand, first assembles the 1000 elements into 100 subcomponents of 10 elements each, which he then uses to form 10 larger subcomponents, which he then finally assembles together into the final product. In both cases, if the process of constructing a single assembly is interrupted, the whole assembly

falls apart and has to be put together from the beginning.

The advantage of the approach taken by the second watchmaker then becomes apparent. In particular, if the first watchmaker is interrupted, he will tend lose a lot more work, e.g., in the worst case he may lose a partially-completed assembly consisting of 999 parts compared to at most 9 parts for the second watchmaker. This, in turn, means that on average it will take the first watchmaker a lot longer to assemble each watch. Perhaps speculatively, Simon suggests that processes similar to those in the watchmaker parable may be at play in biological evolution as well as in the emergence of other complex systems.

Another contribution of Simon's work is his distinction between systems that are *decomposable* and those that are *nearly decomposable*. The former is used to refer to systems where the components can be viewed as independent, i.e., no inter-component interactions are assumed. The latter, on the other hand, is used to refer to systems where the inter-component interactions are weak but still have an effect on the overall behaviour of the system. Specifically, a nearly decomposable system is viewed as one where the 'short-run' behaviour of the components is independent, whereas their 'long-run' behaviour is related but only in an aggregate way – this definition of near-decomposability therefore refers to a dynamic, as opposed to a structural, property of the system in question[1]. Since Simon's specification of a nearly decomposable system is less restrictive than his specification of a decomposable system, i.e., the former recognizes that a system can be viewed as modular even if inter-component interactions are present, it is likely to be more representative of complex systems in general. However, though insightful, Simon's definition of nearly decomposable was not precise, e.g., it was not clear how it would apply to the structural properties of a system.

Herbert Simon's treatise [163] is thus viewed as making two primary contributions, the recognition of the hierarchical, modular, structure present in many complex systems and the specification of a nearly decomposable system. Through the watchmaker parable, a secondary contribution is viewed to be the suggestion regarding why a hierarchical organization may be so common in complex systems. However, many of

[1]The example of a nearly decomposable system given [163] is that of a building subdivided into rooms which in turn are subdivided into cubicles, and where the air temperature in each cubicle is different. In this case, the key components are the rooms, and the dynamic behaviour refers to the heat exchange in the system. In particular, the short-run behaviour refers to the relatively rapid convergence of the temperatures in each room to a value that is independent of the temperature in the other rooms (since the walls are well insulated). The long-run behaviour, on the other hand, refers to the eventual convergence of the temperature to the same value across all rooms, the latter depending only on the average (aggregate) temperature reached in each room.

the points made in the discussion appear to be of a philosophical nature; a number of the arguments are vague and the conclusions reached are often speculative, a fact acknowledged by the author himself. As such, the ultimate goal behind the work may not have been accuracy and thoroughness in presentation but rather the provocation of thought and discussion on the topic of complex systems and their structure.

2.1.3 Modular Interdependency

Subsequent to Herbert Simon's discussion on the architecture of complexity, attempts have been made to formalize the notion of modularity and its effects on evolvability, i.e., the proportion of the search space that has to be explored before an acceptable solution is found [191]. Of particular relevance to this work, it was recognized that under the common definitions of modularity inter-module interactions are typically assumed to be negligible, an assumption that may be an oversimplification in complex dynamic systems. A possible reason for this, it was argued, was the tendency to interpret the functional behaviour of a system based on its structural interconnectedness. That is, the fact that one module does not share a strong structural connection to another should not imply that a change in the state of one of them is not sensitive to changes in the state of the other. This insight is also apparent in the definition of near-decomposability, though the latter does not explicitly make a distinction between structural and behavioural relationships[2].

Recognizing that a complex system may be structurally modular but that there may be non-trivial behavioural interdependencies between the modules, the notion of *modular interdependency* was proposed[3] [189]. Specifically, a module is identified in terms of the structural properties of the system, e.g., using connectivity to locate a clique in a graph. For a given module, the number of possible configurations of that module, denoted M, is then considered. It may then be the case that among these M possible configurations, a subset whose elements satisfy a particular property, e.g., optimality, can be identified irrespective of the state of the rest of the system. Denoting the cardinality of this subset by M', three scenarios are possible:

1. $M' = M$. Under some configuration of the rest of the system, each possible

[2]In fact, Simon recognizes that certain relationships are viewed in spatial terms, whereas others are viewed in terms of, e.g., social, interactions; he then redefines both in terms of the 'intensity' of the interaction.

[3]In the discussion, modularity is viewed in terms of structural and behavioural properties, though other characteristics of the system could also be considered.

configuration of the module will satisfy the chosen property. In this case, nothing can be concluded about the degree to which a configuration of the module satisfies the chosen property without simultaneously considering the state of the rest of the system. If this is true for all modules, the system is said to be *non-decomposable*.

2. $M' = 1$. Without regard to the state of the rest of the system, it is possible to narrow down the number of configurations that satisfy the chosen property to a singleton[4]. That is, a search can be performed on the module in isolation, and if a suitable configuration of the module is found, it will be equally suitable once the module is integrated with the rest of the system. If this scenario is true for all modules, the systems is said to be *separable*.

3. $1 < M' < M$. One or more configurations of the module can be eliminated from consideration because it is known that, regardless of the state of the rest of the system, those configurations will not satisfy the chosen property. At the same time, the suitability of the remaining M' configurations depends on the state of the other system components. Furthermore, the degree of interdependence can be interpreted by considering the magnitude of M relative to M'. If this scenario is true for at least one module, the system is said to exhibit modular interdependency.

Thus, a distinction is made between systems that are separable and systems that exhibit modular interdependency; if neither these conditions hold, as indicated above the system is said to be non-decomposable.

The above definition suggests that a system which exhibits modular interdependency may be more evolvable than a system that is non-decomposable since the former may result in a smaller search space, i.e., if only a subset of all the possible configurations of one or more modules needs to considered. Naturally, this implies that the search process must be capable of taking advantage of the inherent structural modularity. However, given that traditionally no distinction was made between modular interdependency and non-decomposability, this was not always recognized [191].

Intuitively, if a system is separable, i.e., it is both structurally and functionally modular, then the lack of interdependencies may make it trivial to evolve. On the

[4]Though the contribution of this single configuration towards the rest of the system may still depend on the context provided by the other modules, i.e., given the same configuration its contribution may vary, but it is not the case that under certain contexts another configuration is more suitable.

other hand, it has been shown that even if a system is structurally modular, if it involves functional dependencies between the modules then its evolvability can be significantly reduced since changes to one module may negatively affect other modules [189, 191]. However, the degree to which the system is evolvable depends on the adaptive mechanism driving evolution[5]. Specifically, two different types of mechanisms have been suggested, *accretive* and *compositional* [189]. Accretive mechanisms are consistent with the idea that the accumulation of small random variations under the pressures of natural selection will result in gradual evolution. Compositional mechanisms, on the other hand, refer to the combination of sets of genetic material that have been independently adapted, e.g., in different lineages, suggesting that evolutionary change may occur in bursts. Notably, symbiosis and homologous recombination are viewed as compositional mechanisms, whereas mutation and non-homologous recombination (where interdependencies in genetic material is not identified) are viewed as accretive mechanisms.

The difference in performance between accretive and compositional mechanisms was considered by applying both to the evolution of systems with modular interdependencies [189]. Results suggested that the time required for accretive mechanisms to evolve such systems is exponential in the size of the system, whereas the time required for compositional mechanisms can under certain circumstances be reduced to a polynomial growth rate. This was attributed in part to the fact that accretive mechanisms, in contrast to compositional mechanisms, do not explicitly recognize and therefore cannot manipulate structural modules, i.e., appropriate subsets of genetic material. As such, accretive mechanism cannot take advantage of the (interdependent) modularity in the system. Notably, these results were consistent with the suggestions regarding modularity and hierarchy and its effect on evolvability that were made decades earlier [163].

It has also been suggested that if a system is separable, then hierarchically structuring the modules is redundant [191]. Specifically, one way to view a hierarchy is as a system composed of related subsystems, each of which is itself composed of related subsystems, until some elementary atomic level is reached [163]. If the system is separable however, this implies that none of the elementary subsystems are related, hence, there is no measure upon which the hierarchical organization can be based.

[5]Where this mirrors discussions in biological evolution, Section 1.5.2.

2.1.4 Summary

Complex systems tend to exhibit modularity to the extent that this property has been used in their definition. Modularity in complex systems may be so common in part because, compared to monolithic structures, it may increase their evolvability. Furthermore, modularity enables the construction of hierarchies, i.e., groups and subgroups of related system subcomponents.

Traditionally, even if systems were structurally modular, if the modules shared behavioural interdependencies, then the systems were viewed as as non-decomposable and therefore difficult to evolve. However, under specific conditions, it has been shown that even if the modules are functionally related, then evolvability may still be high under compositional evolution, e.g., symbiosis, provided that the system exhibits modular interdependency. Furthermore, it has been suggested that it is these functional relationships that support hierarchical organization.

Previous work has considered modular interdependency on hand-crafted problems designed to demonstrate the potential of compositional mechanisms. In real-world applications that involve large numbers of intricate, non-symmetric, interactions it may be difficult to quantify the degree of modular interdependency. In these cases, it may be sufficient to know that modular interdependency is present so that an appropriate evolutionary mechanism may be applied.

Finally, the definition of problem decomposition adopted in this work, Section 1.7, refers to explicit subcomponents with non-overlapping behaviours. Thus, like the definition of modular interdependency, the adopted definition for problem decomposition assumes solutions with modular structure. With regards to the non-overlapping behaviours, it is noted that this requirement does not preclude functional interdependencies between the modules, i.e., two modules may have internally unique behaviours[6] yet they may still depend on each other's state.

2.2 Approaches to Lateral Problem Decomposition

In this section, relevant approaches to lateral problem decomposition are discussed. Some of these, such as ensemble methods, are generic and apply to a number of ML algorithms, while other are geared more towards GP. In many cases, the approaches reflect an ensemble methodology in that they result in a high-degree of overlap in the

[6]Modular decomposition was originally proposed in the context of optimization where modules were not associated with behaviours, only states/configurations [189].

behaviour of the solution subcomponents; an exception where this is not the case are the bid-based approaches that are discussed in Section 2.2.4.

2.2.1 Ensemble Methods

Ensemble methods are a broad class of learning algorithms that aim to improve performance by combining multiple base models under a supervised learning context. The underlying ensemble members are assumed to be *weak learners* implying individual error rates that are strictly less than, though which can be arbitrarily close to, one half. By combining a diverse set of such weak learners, the goal of ensemble learning is to produce a *strong learner* in which incorrect decisions by a few of the members can be filtered out, e.g., through a majority vote. Thus, in ensemble methods, the focus is on reaching a correct consensus despite the possibility of erroneous individual decisions, and so they can be viewed as schemes for achieving fault-tolerance. This makes them especially useful on noisy datasets where individual decisions may be unreliable. A certain degree of diversity in the ensemble members is key; for example, it makes little sense in combining multiple identical models as one is just as well off by relying on one of them.

The required variation in the ensemble members is introduced during training, where two common approaches for achieving this are bagging [21] and boosting [52]. In bagging, each ensemble member is trained on a set that is independently generated by stochastically sampling with replacement from the original training set, i.e., diversity is introduced by *arbitrarily* varying the frequency with which individual training instances are observed. As such, the ensemble members can be trained in parallel. Under boosting, on the other hand, the base models are trained in series, specifically, each training instance is associated with a weight[7] that is adjusted to reflect the error rate on that instance in previous iterations. Thus, the learning process in boosting is more directed as focus shifts to cases that are deemed to be more difficult, but can potentially result in over-fitting.

An alternative to modifying the set of training instances is to include diversity as one of the objectives that is used to evaluate the ensemble members. For example, in negative correlation learning [118], an explicit term in the evaluation function is used to encourage individuals to make negatively correlated decisions. By parameterizing

[7]Boosting can also be implemented by sampling instances in proportion to their associated weights, as opposed to sampling with uniform probability as is done in bagging, and can therefore be used with base learning algorithms that are incompatible with weighted instances [162].

the weight of the correlation term, the aim of the approach is to promote specialization in the ensemble members while at the same time producing correct aggregate decisions, e.g., through averaging. The approach is novel because it allows the ensemble members to be trained simultaneously while evaluating individuals in the context of other potential ensemble members. In contrast, other approaches such as bagging or boosting tend to train the base classifiers independently or in series, so, at best, only a single ensemble member is adapted with respect to the other members. As such, negative correlation learning is similar to the proposed approach in that both may be viewed as forms of context learning.

Provided that the algorithm used to construct the weak learners is unstable [22], i.e., sensitive to perturbations in the training data, ensemble learning places few other restrictions on the process used to generate the ensemble members. As such, ensembles have been constructed with models evolved using GP in the role of the weak learners [50, 53, 78, 144]. However, the high computational cost of GP training combined with the requirement to construct multiple base models is likely to make building GP-based ensembles very demanding. This has been identified as a major drawback of the approach, and to this end, a number of solutions have been proposed, e.g., the use of specialized hardware [50] or the leveraging of the GP population to obtain multiple models from a single run [53].

Given their capacity to reduce error [9], it is acknowledged that for certain ML tasks ensemble methods may be appropriate. However, ensemble learning is not likely to satisfy a number of the goals pursued in the proposed approach. First, following a weak learner methodology is likely to result in overlapping behaviours in the ensemble members, and as such, the resulting ensembles are unlikely to decompose the problem as per the definition adopted here, Section 1.7. That is, ensemble learning is likely to result in behaviour as observed in the left-hand side of Figure 1.1. Second, it has been acknowledged that ensemble learning is likely to lead to incomprehensible solutions [9], in part because ensembles may need to be constructed from hundreds and even thousands of members [53], but also because the overlap in behaviour makes it difficult to identify specific roles. Third, ensemble learning algorithms tend to be restricted to supervised learning tasks such as regression and classification. Finally, ensembles often aggregate the decisions of their individual members using simple mechanisms such as averaging or majority voting, often specified *a priori*, where this is viewed as an area open of research that is addressed in this work. Thus, it is stressed here that even though it is related, the proposed approach is not viewed as an ensemble learning algorithm.

2.2.2 Binary Decomposition

Binary decomposition [92] is an approach that has been used to scale GP to classification problems with more than two classes. As such, like ensemble methods, it has been traditionally limited to supervised learning. However, in contrast to ensemble methods where the onus is on reaching a correct consensus, binary decomposition aims to explicitly associate different solution subcomponents with different class labels present in the problem. Furthermore, if the subcomponents are applied in parallel, it is similar to the bid-based approaches to problem decomposition, Section 2.2.4, e.g., a strength value may be used to choose a subcomponent [92].

Traditionally, GP has been tailored to classification problems with two classes, e.g., through the use of a global wrapper function [95]. With this in mind, binary decomposition reformulates a classification problem with k classes as k binary classification problems. For each class, the approach constructs a classifier to distinguish between positive and negative examples of that class, i.e., to distinguish between instances as either in-class or out-of-class. In the original formulation [92], the classifiers were applied to unseen instances in parallel resulting in one of three outcomes. If exactly one of the classifiers identified the instance as in-class, the label associated with the classifier was assigned to that instance. If none of the classifiers identified that instance as in-class, the instance was associated with a 'reject' class suggesting that perhaps it is an outlier[8]. Finally, if more than one classifier identified the instance as in-class, a conflict resolution heuristic had to be applied, e.g., the decision was based on the classifier with the highest strength of association [92].

Given that a separate population is evolved for each class label in the problem, a significant drawback of binary decomposition is the required computational cost. To this end, approaches for extracting classifiers from the same population have been proposed, e.g., through niching [124]. The latter work also demonstrates how models evolved under the binary decomposition formulation can be combined hierarchically, e.g., by applying the classifiers in a predefined order and associating the exemplar with the first classifier prompting an in-class decision. Thus, variants of binary decomposition can also be viewed as a form of vertical problem decomposition.

2.2.3 Teaming using Genetic Programming

Teams of GP-based individuals have been applied to two types of problems [169, 170]. On the one hand, teaming has been used in the context of multi-agent learning where

[8]Viewed as a better outcome than forcing a decision.

a single agent is presumed incapable of efficiently solving the problem, e.g., predator-prey scenarios where more than one predator is assumed [65, 119] or problems involving coordinated exploration [158]. In this case, the evolved programs are typically used to control autonomous agents that interact in an environment. On the other, teaming has also been applied to problems such as classification that have traditionally been solved using single-individual solutions. As in ensemble learning, the goal of combining multiple individuals in this second case has been to improve performance. In this work, the focus is on the second problem type, i.e., problems that have traditionally been solved using single-individual solutions.

Island and Team Approaches

Three design decisions required for team building have been identified, namely, choosing team members, coordinating team member outputs, and allocating credit among team members[9] [170]. The first of these design decisions, regarding team membership, has been used to categorize GP teaming approaches as it relates to the units of evolution upon which natural selection operates. In particular, two categories of teaming approaches have been identified, those where natural selection acts on individual team members [44, 79, 145] and those where natural selection acts on the team as a unit [19, 146, 169, 171]. In the latter case, the team memberships are evolved, though typically the team size still needs to be specified *a priori*, and credit assignment becomes trivial since selection takes the form a group operator.

Teaming approaches where natural selection acts on individual team members have been referred to as *island* approaches because individuals comprising the team are often evolved in separate populations [173]. This term excludes membership strategies where individuals are drawn from the same population, which, having been found to result in poor performance in conventional single-population scenarios[10] [170], appear to have received less attention. In contrast, approaches where the team is evolved as a unit, and where natural selection does not recognize individual team members, have been referred to as *team* approaches[11].

Early studies have shown that while the teams in team approaches can perform significantly better than single-individual solutions, the performance of the team members in isolation was found to be poor [19, 169, 171]. These results also confirmed

[9]These are similar to the issues identified under cooperative coevolution, Section 2.4.2.

[10]Without explicit consideration for different species.

[11]This terminology may seem counter-intuitive since both island and team approaches are used to construct teams of GP-based individuals.

that cooperation between team members was taking place, i.e., individual errors were found to be uncorrelated. Island approaches, on the other hand, were found to produce individuals that were relatively fit but whose errors were likely to be correlated [79]. As such, the benefit gained by combining arbitrary individuals from different populations was found to be limited, and it typically took many random trials before effective collaborations were discovered.

Similar conclusions were drawn in later studies [173], where explanations for these trends were also proposed. Specifically, it was suggested that team approaches resulted in uncorrelated errors because, if selection pressure acted on the team as a whole, cooperation between team members was encouraged. In contrast, under island approaches, it was suggested that if there is a 'path of least resistance' evolution would follow that path even in independent runs. That is, problem decomposition is not explicitly encouraged under the island approach. This insight led to the development of a new class of algorithms, Orthogonal Evolution of Teams (OET) [173], which view the evolving population in terms of individuals or teams depending on the stage of the algorithm. As such, these algorithms aim to apply pressure at the team level to encourage cooperation but also at the individual level to encourage strong individual performance. For example, in one instance of the OET algorithm, parent selection was done at the individual level whereas the survivor selection policy was applied at the team level. Empirical results showed that indeed this approach resulted in both strong individuals and uncorrelated errors, improving overall team performance compared to having just one or the other.

Under the OET approach, it is suggested that applying selection pressure both at the team level and at the individual level will result in improved team performance. On the one hand, this assumption was validated empirically with respect to the OET algorithms [173]. On the other hand, when team and team member performance were implemented as two separate goals in a multi-objective fitness formulation, improved individual fitness was achieved only at the expense of a significant reduction in team fitness [19]. Naturally, if one is interested in the use of teams in the first place, the performance of the team takes precedence, and individual performance is significant only if it can improve team performance.

In summary, early work established that team approaches resulted in strong teams but poor individual performance. At the same time, the team approaches effectively resolved the credit allocation problem, and provided the means to evolve team memberships automatically. Island approaches, on the other hand, were found to produce strong individuals, but required an *a priori* specification of how to compose a team

and distribute credit among team members. To generate both strong teams and strong individuals, the OET class of algorithms was proposed. With few exceptions, e.g., [146], these approaches required an *a priori* specification of a team size, which remained constant across all teams, as well as the team composition [158].

Alternate Approaches to Cooperation in GP Teams

More recently, alternate mechanisms for encouraging cooperation in teams that do not apply pressure at the team level have been proposed; these approaches are also capable of building teams from a single population of individuals. The approach behind the Pareto-Coevolutionary GP Classifier [109] borrows heavily from Pareto-coevolution, Section 2.4.1, in that each problem instance is viewed as an objective to be optimized. As such, the algorithm searches for a set of non-dominated individuals with respect to the objectives that are identified, recognizing that different individuals may specialize on different problem instances. However, given the brittle nature of the Pareto-dominance relation, in the sense that a solution may be non-dominated by outperforming others on just a single objective, this approach was found to result in individuals whose behaviours overlapped considerably [125].

To produce solutions consisting of individuals with non-overlapping behaviours, i.e., to encourage problem decomposition, the Competitive Multi-Objective Grammatical Evolution framework combines a Pareto-coevolutionary archiving strategy with a multi-objective fitness function [123]. Specifically, while a set of archives is used to store promising individuals, the fitness function encourages individuals that exhibit a tendency towards strong classification performance, low parsimony, and low overlap in behaviour with respect to other individuals. In contrast to the discriminator approach traditionally adopted in GP, i.e., a switching function, key to the framework is a reformulation of the classification problem in terms of novelty detection. This, in turn, allows different individuals to be associated with different parts of the instance space, and provides a degree of confidence in the final decision output by the model.

Discussion

In all but a few cases, e.g., [123], GP teaming approaches, both island and team, tend to follow what can be considered to be an ensemble learning methodology. Specifically, to improve performance, they form collaborations consisting of *homogeneous*

individuals in the sense that each individual has similar capabilities[12]. In particular, in a homogeneous set, each individual typically has the capacity to represent a complete solution. As such, it is then possible to compare the performance of the team versus the performance of the individual, and to ask such questions as whether a strong team can be composed of not-so-strong team members. This approach can be viewed as an ensemble learning methodology, and the individuals in the role of weak learners, because the focus is not on evolving non-correlated behaviours but on evolving non-correlated errors, i.e., overlap is desired in situations when the individuals make correct decisions. As a result, as in ensemble methods, the team members' behaviour will naturally tend to overlap.

In contrast, in the proposed approach, an individual is not capable of representing a complete solution, or at least a non-degenerate solution. For example, in a classification problem, a single individual would always predict the same class label – clearly a degenerate solution. In this way, the individuals comprising a solution in the proposed approach are *heterogeneous* in the sense that they often play orthogonal roles. As such, analysis analogous to the individual versus team comparison discussed above does not apply. However, this is not viewed as a drawback, but rather a way of encouraging problem decomposition since cooperation is required for a solution to be effective.

It is therefore suggested that a key factor in determining whether the behaviour in a set of individuals, whether in a team or symbiosis, will overlap is the mechanism for aggregating their contributions. Specifically, ensemble methods tend to rely on voting or averaging, e.g., a majority vote or some weighted combination of votes, and the final decision is therefore based on the option that garners the greatest support. This places the onus on 'the majority to be right', and if the majority is to be right in many situations, then the individual behaviours will need overlap in many situations. In contrast, under the decision resolution mechanism used in the proposed approach, the individual needs to be correct only in situations where it is chosen to act, where this is viewed as a simplification of the learning task.

Given that the two terms are often used in similar contexts, a question that arises from the above discussion is whether or not there is a difference between what it means for a solution to be a team or an ensemble. In this respect, two differences come to light. First, GP teaming has its roots in multi-agent learning [65, 119, 169],

[12]The terms 'homogeneous' and 'heterogeneous' have been used in GP teaming literature [119, 170] with a slightly different meaning. Specifically, individuals were referred to as heterogeneous even if they had similar capabilities but provided that they were not identical, i.e., not cloned.

and as such represents an approach that is applicable to both supervised and temporal sequence learning. Ensemble methods, on the other hand, tend to be restricted to supervised learning. Second, whereas ensembles can consist of thousands of individual members, team member counts tend to be at least an order of magnitude smaller [19, 119, 170]. This, in turn, suggests a lesser degree in overlap between the behaviours of team members compared to the behaviours of ensemble members, where the evolution of non-overlapping behaviour is of particular relevance to this thesis.

A potential drawback of building teams versus single-individual solutions is the amount of computational overhead that may be required to execute multiple programs, where this is exacerbated by the high computational cost already associated with GP, Section 1.3.4. Fortunately, under the team approach, average effective instruction counts per team member were found to be small compared to the effective instruction counts in single-individual solutions [19]. Naturally, the same is not expected in island approaches which effectively evolve independent single-individual solutions.

As a final remark, the link between teaming and the field of cooperative coevolution, Section 2.4.2, is emphasized. In particular, the evolution of teams may be viewed as a form of cooperative coevolution. With few exceptions, e.g., [123], the GP teaming literature makes few explicit references to this association.

2.2.4 Bid-Based Approaches

Two bid-based approaches are most relevant to this work, Learning Classifier Systems (LCS) [72] and the economy-based Hayek model [10, 11, 12, 13, 14]. As is done in the proposed approach, they both define individuals in terms of separate 'bid' and 'action' components – it is because of this decomposition that in this work they are referred to as 'bid-based'. The bid is used to determine when an individual in chosen to act, whereas the action defines what the individual does. Such a formulation allows cooperation among heterogeneous individuals, Section 2.2.3, since, as long as all necessary actions are present in the final solution, individuals can focus on fulfilling specific roles, i.e., they can be associated with specific actions. Furthermore, as noted earlier, Section 1.7, the separation of the when from the what is consistent with the definition of context learning that is adopted in this thesis. Context learning, in turn, is viewed as central in the definitions of lateral and vertical problem decomposition.

Bid-based approaches of this form appear to be uncommon in GP-based teaming where voting and averaging strategies are more prevalent. One exception found

was the *winner-takes-all* combination method investigated in the context of team approaches [19]. Under this policy, the individual providing the clearest class distinction defined the output for the entire team. Compared to the other combination methods that were studied, the winner-takes-all approach was found to result in high specialization as well as small effective program sizes.

In the following, both classifier systems and Hayek are reviewed. The latter, aiming to correct a number of deficiencies observed in the former, is reviewed second. Afterwards, the concept of *default hierarchies* [71] is briefly introduced. Though traditionally associated with classifier systems, it is asserted that default hierarchies are relevant under bid-based frameworks in general.

Learning Classifier Systems

Classifier systems were originally proposed in the context of reinforcement learning as systems that learned to accumulate reward through their interaction with the environment [72]. In the original LCS formulation, a population of condition-action-strength rules (classifiers) was evolved; assuming that a rule's condition was satisfied, the strength value predicted the reward that could be expected as a result of applying the rule's associated action. Two key components of this framework were a credit assignment mechanism that used a bucket-brigade algorithm to update strength values of classifiers involved at different time steps, as well as a GA-based rule-discovery component that used strength as a fitness value. A classifier system of this form, in which a population of interacting rules representing partial solutions is evolved, came to be known as the *Michigan* approach.

Given an input state, one way to select an action under the Michigan approach may be as follows[13]. First, *a match set* is calculated as the set of classifiers in the population whose conditions are consistent with the input state. From this match set, in which different actions may be present, an *action set* is then chosen consisting of all classifiers that are associated with the same action. The choice of the action set can range from the purely exploratory, in which an action is selected from those present in the match set with uniform probability, to the purely exploitative, in which the action associated with the highest-strength rules is chosen. The chosen action is then applied in the environment, prompting a possible reward or penalty, at which point the learning parameters associated with the classifiers in the current or previous[14]

[13]It is acknowledged that many variants of Michigan-style classifiers systems exist [107] so the aim of this section is to provide a review of the relevant concepts underlying the approach.

[14]Depending on whether or not the last time step was reached.

action set are updated. Naturally, post training, action selection is typically biased towards higher-strength rules with the goal of maximizing reward.

The strength parameter in classifiers systems therefore plays the role of a 'bid' in that, depending on the implementation, it determines the likelihood that the associated rules are activated. However, in traditional classifier systems, strength is also used as the fitness measure. This is problematic because a rule that matches states leading to low rewards will have a low strength value, yet it may still be the most appropriate for those situations. As such, if strength also defines fitness then the rule will not likely be preserved in the population. Furthermore, over-general rules, having high strength because they are triggered in many states but which yield high reward in only a few, may be encouraged under this policy.

With this in mind, the XCS classifier system formulation replaces the strength parameter with separate prediction, prediction error, and fitness parameters [196]. The prediction is used in determining what action to take thus filling the role of the strength parameter in the original formulation. However, the fitness of an individual, used in the GA, is now a function of the prediction error, i.e., it reflects prediction accuracy, where both of these parameters are adapted along with the prediction in the credit assignment component. Furthermore, instead of applying the GA on the entire population, as was done previously, the GA in the XCS formulation operates on the match sets. This, in turn, eliminates competition between classifiers in different match sets, recognizing that the latter may need to cooperate in forming an effective chain of actions. Due to these two modifications, accuracy-based fitness and a local GA, it was suggested that the approach produces rules that cover the entire state-action space and whose associated prediction values are accurate [196, 197]. Furthermore, it was shown that the system has a tendency towards producing maximally general rules, i.e., matching as many states as possible while maintaining accuracy.

Thus, from the point of view of this work, Michigan-style classifiers systems are significant because they represent a similar form of problem decomposition, namely, using bids to decide what solution component to apply when. However, one difference between the proposed approach and classifier systems is that (given the same input) in the former the bid values remain constant while in the latter they are adjusted during an individual's lifetime. Furthermore, through the XCS formulation, and in particular, the local GA that it uses, it is recognized that the indiscriminate application of crossover may hinder evolution. That is, if classifiers forming different match sets are to cooperate, e.g., in forming a long chain of actions leading to a high reward, then some form of speciation must be used to allow specializations to emerge. In the

proposed approach, different species are directly supported through the process of symbiosis.

In contrast to Michigan-style classifier systems where the entire population interacts and is involved in a solution, the *Pittsburgh* approach [88, 166] evolves a population of variable-length individuals where each individual is a rule set representing a complete solution, e.g., a production system. That is, at the end of training under the Pittsburgh LCS approach, a single individual can be culled from the population and applied post training. As in team approaches to GP teaming, Section 2.2.3, this greatly simplifies credit assignment, though it requires a mechanism for resolving the contributions of the various rules present in a rule set. Furthermore, whereas Pittsburgh approaches can easily be used with Pareto-coevolution, Section 2.4.1, the same is not true of Michigan approaches where population members represent only partial solutions.

Here, it is noted that whereas the bid-based interaction used in the proposed approach is similar to that of Michigan classifier systems, the level at which fitness is evaluated is more in line with the Pittsburgh-style approach. However, in contrast to the proposed approach, Pittsburgh-style classifier systems tend to follow what are viewed to be purely neo-Darwinian principles, Section 1.5.2.

Hayek: Evolution of an Artificial Economy

The market-based Hayek model [10, 11, 12, 13, 14] evolves an artificial economy of agents, each with an associated wealth, that interact as follows. Given an input, e.g., world state, each agent computes a bid reflecting the price that it is 'willing' to pay for the property rights to the world – the world is therefore viewed as a resource. The agent making the highest bid then buys the world, and must pay an amount equal to its bid value to the previous owner. If an agent currently in possession of the world 'chooses' not to sell, it can bid arbitrarily high since it will make the ensuing payment to itself thus maintaining its level of wealth. Once in possession, the agent acts in the world and receives any external reward that its actions may yield. In this way, the goal of the framework is to motivate the agent who owns to world to increase the world's value, and in particular, the amount that future owners would be willing to pay in order to increase their own wealth. Put another way, the agent whose future actions will increase the world's value the most can afford to outbid the other agents [11].

At least four versions of the Hayek economic framework have been proposed

though they all share the incentive structure described above. The versions differ with respect to the representation used, e.g., assembly-level code [14] or production systems [13], as well as the fundamental mechanisms defining the economic framework itself including the process for initializing new agents, bid definition, and taxation. Likely due to the variation in and the complexity of the Hayek systems, reproducing the levels of success observed in initial studies, e.g., [13], was found to be difficult [104]. As a result, recently, attempts have been made to simplify the original Hayek formulation [110].

The Hayek framework, like classifier systems, recognizes separate bid and action components and uses a form of bucket-brigade for credit assignment across multiple time steps. However, when viewing classifiers systems as economic models, i.e., interpreting strength as wealth, the Hayek literature asserts that there are certain flaws with the approach. First, since classifiers enter the economy with strength values that are set in some *a priori* fashion, money is not conserved but rather artificially injected into the economy. Second, property rights are not enforced in the sense that multiple rules act at any given time step potentially leading to a 'tragedy of the commons'. Specifically, rules are encouraged to be active even when it is not appropriate for them to do so since they may then receive a portion of the *shared* reward (which may be better than receiving no reward at all). In both cases, it is suggested that the end result is an improper allocation of credit leading to a system that fails to meet the designer's goals [11, 13]. As a result of these observed deficiencies, property rights and conservation of money is strictly enforced in the Hayek systems. It is also noted that the above criticisms appear to be directed towards classifier systems where fitness is based on strength.

The Hayek population is similar to the population of classifiers in the Michigan approach in that each rule is normally associated with a single action and thus represents a partial solution[15]. As in the case of Michigan-style classifier systems, it is therefore not obvious how to integrate Hayek with Pareto-coevolution, Section 2.4.1, where this is viewed as a way of scaling the learning process to large state spaces. Thus, the principal influence of Hayek on the proposed approach is the auction-based mechanism for action selection which may lead to a decomposition of the input space among a set of cooperating individuals.

[15]Multiple actions could also be associated with an individual, though this would naturally increase the complexity of the system.

Condition		Action
00	$\longrightarrow$	0
01	$\longrightarrow$	1
10	$\longrightarrow$	1
11	$\longrightarrow$	1

(a) Non-hierarchical set.

Condition		Action
00	$\longrightarrow$	0
**	$\longrightarrow$	1

(b) Hierarchical set.

Figure 2.1: Example of a default hierarchy, adapted from [37], in the context of classifier systems. The rules implement the logical OR function which is applied to the two fields in the condition, specifying the result in the action. An asterisk (∗) matches all inputs.

Default Hierarchies

The concept of default hierarchies[16], originally proposed in the context of Michigan-style classifier systems, refers to the manner in which rules are applied to the state space [71]. In particular, a default hierarchy is a set of rules[17] where there is one general 'default' rule whose condition matches many situations, and in most of these situations, the default rule is appropriate. For cases where it is not, the population also includes 'exception' rules. The exception rules match a subset of the states matched by the default, and usurp the default whenever it is appropriate for them to do so, i.e., without these exception rules the default would be triggered leading to an incorrect action on the part of the system. As an alternative to default hierarchies, the behaviour of the rules in the system could be such that they match non-overlapping sets of situations.

To illustrate, two ways to partition the state space among a set of rules in a classifier system implementing the logical OR function are shown, Figure 2.1. The non-hierarchical set, Figure 2.1a, will produce the correct result no matter what the strength/prediction of each individual rule is since the conditions match non-overlapping states. The hierarchical set, Figure 2.1b, will produce the correct result only if the more specific (top) rule will take precedence over the more general rule whenever both of the rules' conditions match – the key to the effectiveness of default hierarchies is therefore to trigger the exceptions whenever the more general rules are inappropriate.

As shown in the above example, default hierarchies can result in more compact rule sets, and indeed, it has been suggested that this is true in general [71, 154]. That

[16]Despite the name, default hierarchies are viewed as a form of lateral problem decomposition.
[17]Without loss of generality, LCS terminology is used.

is, it may be easier to specify a prevalent but sub-optimal default behaviour along with a number of exceptions than to specify a set of ideal behaviours. The capability to succinctly model the problem is thus viewed as one of the main advantages of default hierarchies. Furthermore, through default hierarchies, the system can be gradually refined as additional exceptions are added to cover situations in which the more general rules fail [37].

The concept of default hierarchies can be generalized to frameworks other than classifier systems by focusing not on the mechanism for triggering rules, e.g., using condition and strength to obtain an action set, but rather on the resulting behaviour of the system. That is, the characteristic behaviour of a default hierarchy is that there are general rules which apply in most situations augmented with a set of exception rules that cover the extraordinary cases. Any framework where components bid for control can result in this characteristic behaviour, e.g., associating a constant bid with a general rule and requiring exception rules to bid above and below the constant bid as appropriate. Naturally, this type of behaviour may be more befitting when there are certain imbalances in the state space, e.g., in classification problems with large class imbalances.

2.3 Approaches to Vertical Problem Decomposition

The aim of this section is to provide a review of relevant approaches to vertical problem decomposition. Some of these, such as cascade architectures, have originally been proposed in the context of training neural networks but have subsequently been adapted for use with GP. Others, such as layered learning, are independent of any specific ML algorithm. In this regard, it is noted that the traditional approach to problem decomposition in GP has been code reuse, Section 2.3.2. Though related to coevolution, the neuroevolutionary approach in Section 2.3.5 is presented here since it explicitly separates the task of identifying appropriate solutions subcomponents from the task of effectively combining them.

2.3.1 Layered Learning

Layered learning refers to the decomposition of a complex task into a hierarchy of subtasks which can then be learned in a bottom-up fashion. An early formalization emphasizes four principles underlying the approach [180]:

1. As originally specified, the root problem is too difficult to be solved directly.

2. A bottom-up decomposition of the main task is available whereby more complex (higher-level) tasks are viewed in terms of simpler (lower-level) subtasks.

3. ML is used to address each subtask independently.

4. The output of lower-level models serves as input to higher levels, i.e., only the lowest levels receive the original problem inputs.

These principles suggest, or rather do not preclude, that layered learning can be used with different ML algorithms, and indeed, this is possible even within the same application [180]. Furthermore, a decomposition into different subtasks implies different goals and objectives, not only between layers, but possibly within a single layer. Finally, the first principle reveals that the primary motivation behind the paradigm is to allow ML to scale to complex problems. Here, the link between layered learning and *transfer learning* is noted, the latter referring to the use of knowledge learned in one task in solving a different, potentially related, task [179].

The above formalization was initially proposed in the context of multi-agent systems where its effectiveness was demonstrated in a robotic soccer domain. Example subtasks included learning to intercept a ball, and evaluating the likelihood of completing a pass to a teammate. Since then, a number of variations of the approach have been proposed, e.g., learning subtasks at different levels concurrently [194], though the key insights behind the approach remain. In particular, in this work, the term 'layered learning' is used to refer to any hierarchical approach in which the lower layers are associated with simpler subtasks whose solution forms the basis for the higher layers.

Early applications of layered learning using GP have also been in the robotic soccer domain [58]. In this work, two layers were evolved with the overall goal of learning effective keep-away strategies – the number of turnovers therefore served as a basis for the objective at the second layer. The goal of the first layer, on the other hand, was to learn to pass accurately with no opponent present, where this was viewed as a natural intermediate subtask. Furthermore, in contrast to previous layered learning formulations [180], the GP-based framework did not use output from lower-level models as a basis for higher levels, but rather the lower-level models themselves[18]. Specifically, the individuals from the final population at the lower layer were used to seed the initial population at the higher layer. Results suggested the approach was able to generate better solutions with less computation effort compared to standard

[18]This mode of operation may be more consistent with transfer learning.

GP. In addition, it was found that the computational effort applied at each layer, i.e., the number of generations, was a significant factor on performance, where this result was subsequently corroborated [75].

An acknowledged drawback associated with the layered learning approach is that the task decomposition has to be specified by a domain expert *a priori* [180]. In practice, identifying the relevant subtasks may be difficult if not impossible, and even if a decomposition can be specified by a domain expert, it may be suboptimal. To this end, two approaches for automatically inducing an appropriate task decomposition have been proposed, namely, defining simpler subtasks in terms of lower-order forms of the same problem or associating them with subsets of the original instance space [82]. In both cases, even though an *a priori* specification is still necessary, the approach is viewed as easier than having to identify task-specific behaviours and how they may interact. For example, in the former case, genetic material evolved on the 2x2 Rubik's Cube could be used in evolving solutions to the 3x3 Rubik's Cube. In the latter, higher levels of the hierarchy could be trained using cases that include Rubik's Cube configurations progressively farther away from the solved state. Empirical evaluation of both approaches suggested that whereas the former outperformed standard GP and GP with code reuse, the latter approach did not yield similar improvements [82].

To improve the performance of the subset-based approach to automating task decomposition, a selection architecture was proposed in which non-overlapping subsets of training cases are used to evolve independent program branches [81]. Once the branches are trained, a selection function, e.g., a switch statement, can then be used to pass control on to the appropriate branch based on the input case, much like in *gating* approaches that are common in temporal sequence learning [89]. It was found that highly fit branches could be evolved, thus, given an appropriate selection function, the performance of the overall system could also be strong – this is not surprising given that each branch was responsible for only a part of the problem specified *a priori*. Naturally, a fundamental issue in the selection architecture is how one arrives at an appropriate selection function.

The approach proposed in this thesis is viewed as an instance of layered learning that reflects in many ways the principles behind the original formalization [180]. In particular, a goal behind the proposed framework is to facilitate solutions to complex problems. To do so, the algorithm is designed to produce a bottom-up decomposition of the problem, though the aim is for this decomposition to be an emergent property of the solution. In terms of the way it is applied to individual problem instances, the proposed approach is similar to the selection architecture [81] in that internal

nodes in the hierarchy play a role analogous to that of the selection function. This suggests that the nodes in the lower levels of the hierarchy should be diverse. As in the selection architecture, this is achieved by associating different problem instances with different nodes, however, in the proposed approach this process is automated through the use of coevolution, Section 2.4. Furthermore, through the evolution of bidding strategies, the proposed approach automatically determines the selection policy.

A key difference between layered learning in general and the proposed approach is that in the former the hierarchy is activated in a bottom up-fashion, and in variants that follow the original formulation [180], all units in the structure are activated. In the proposed approach, the higher-level units are engaged first, which then triggers the activation of only a subset of the lower-level units.

2.3.2 Code Reuse

A number of different mechanisms for representing, identifying, and reusing code modules within GP individuals have been proposed [5]. Specifically, parts of code designated for reuse can be viewed as subroutines that may be called by higher-level code segments; such an organization is consistent with vertical problem decomposition. In the following sections, a number of such approaches for code reuse in GP are summarized. Loops and recursion represent alternative forms of code reuse though their review is beyond the scope of this work.

A key difference between the approaches for code reuse reviewed in this section and the proposed approach is that the former search for modules *within* a program, whereas the latter evolves modules that are structurally separated from the outset, i.e., are evolved in different GP individuals. Here, the second approach is viewed to simplify the learning task since candidate modules are explicitly identified so that heuristics for identifying module boundaries are not required. Alternatively, the added complexity of having to identify boundaries (necessary to preserve context) is avoided in the proposed approach.

Automatically Defined Functions

Automatically Defined Functions (ADFs) [97] are an approach for evolving subroutines together with the higher-level calling code within the same tree-based GP individual. Specifically, each individual consists of at least one *function-defining* branch and at least one *result-producing* branch. Each function-defining branch specifies the name, arguments, and body of a subroutine. A result-producing branch, on the other

hand, contains an expression for the final program output, and may call the subroutines in its associated function-defining branches. Thus, the main program code and the subroutines available to it are evolved in the same individual, though in separate, well-defined, substructures. Furthermore, the subroutines themselves may be able to call other subroutines, though care must be taken to prevent loops.

Experiments showed that the ADF approach could produce less complex solutions using fewer fitness evaluations provided that the problem was sufficiently hard [97], i.e., on simple problems, one was better off by using traditional GP. It was suggested that one reason why ADFs, and encapsulation in general, may improve GP performance is that it may protect the discovered code modules from the destructive effects of variation operators such as crossover [5]. Unfortunately, it does not appear that the threshold for determining when a problem is sufficiently hard to warrant the use of ADFs can be determined in a straightforward fashion.

Under the original ADF approach, the overall form of the individuals had to be specified including the number of function-defining branches, the number of arguments in each function-defining branch, as well as the restrictions on what subroutines could be called by what other subroutines. Furthermore, the specified structure was the same for all individuals in the population. Thus, a drawback associated with the original approach was that these were not evolved properties but rather that they had to be specified *a priori*. In response, a number of operations, e.g., branch and argument duplication, were proposed as a means of automatically exploring different program structures in the course of evolution [96].

From the point of view of this thesis, where the focus is problem decomposition, ADFs represent a step forward compared to traditional GP. However, the ADF approach is limited in that any useful code segments that are identified can only be accessed within the containing individual. In contrast, more widespread code reuse could be achieved by sharing subroutines among individuals, e.g., in a subroutine library, though this could potentially result in reduced diversity.

Module Acquisition

The technique of Module Acquisition [2] is viewed as improving on the ADF approach in two ways. First, it does not define subroutines in predefined branches separate from the main calling code resulting in a more generic representation, i.e., with fewer syntactic restrictions and parameters that have to be evolved or specified *a priori*. Second, in contrast to ADFs, Module Acquisition allows subroutines to be shared

between individuals by maintaining a 'genetic library' that is accessible to the general population.

Specifically, the approach evolves trees that, apart from the conventional restrictions, e.g., that they be syntactically correct, are not required to conform to any specific format. A *compression* operator is applied to individuals selected in proportion to their fitness. Given a depth and an arbitrarily selected node in the chosen tree, the operator creates a subroutine rooted at the specified node and extending down to the selected depth. The newly created subroutine is then added to the genetic library, and the corresponding subtree in the individual where the compression operator was applied is replaced by a call to the subroutine. The subroutine can be accessed by other individuals in the population, and if it proves to be useful as measured by natural selection, then it is likely to appear in many programs. Once defined, a subroutine is no longer modified, so to help maintain genotypic diversity an *expansion* operator is also included that replaces a reference to a subroutine in a single individual with the corresponding code.

The Module Acquisition approach was compared with canonical GP and GP with ADFs on the even-parity problem [91]. Though limited in scope, the study confirmed that ADFs could result in improved performance compared to canonical GP, but no such benefit was observed in using Module Acquisition.

As indicated in the original description of the approach [2], the compression operator is asexual. This suggests that Model Acquisition may be a form of symbiosis, and indeed, it appears to fit well with the definition adopted in this thesis, Figure 1.5. In particular, the compression operator is consistent with the compartmentalization stage. Furthermore, the way in which module fitness is measured in the approach is similar to the way in which symbiont fitness is measured in the proposed approach, that is, fitness is not assigned explicitly but rather the survival of a module (symbiont) is contingent on it appearing in at least one program (host).

Adaptive Representations

Adaptive Representations [155, 156] is an approach similar to Module Acquisition in that it aims to automatically identify useful subroutines that can be accessed by all individuals in the population. However, it does not publish the discovered modules in a genetic library but rather makes them available by dynamically extending the function set. Thus, the process can be more easily integrated with the evolutionary learning cycle as it only affects the primitives that are used to represent solutions.

Furthermore, the Adaptive Representations approach emphasizes the use of heuristics for subroutine identification, e.g., in favour of modules that are fit and frequently used, where previous approaches have relied more on arbitrary selection of candidate building blocks. Naturally, to make a more informed decision when selecting potential modules typically requires increased computational overhead, and this added cost is viewed as a drawback of the approach.

Compared to canonical GP as well as GP with ADFs on the even-parity problem, the Adaptive Representations approach was shown to find simpler solutions and with less computational effort [155], and in the game of Pac-Man, the approach found better solutions in less time [156]. Despite initial findings to the contrary, however, subsequent studies have questioned the ability of the approach to hierarchically decompose the problem [35]. Furthermore, the random heuristic used to select candidate subroutines in Module Acquisition was shown to be the most versatile across multiple domains suggesting that previous studies on the approach indicating the contrary [91] may indeed have been too limited.

Compared to the other approaches to code reuse in GP, the Adaptive Representations approach is significant because it explicitly recognizes the benefit of building solutions hierarchically in a bottom-up fashion [155]. That is, the function set is initialized to contain only the most primitive operators but may subsequently include subroutines of increasing complexity. Furthermore, levels in the hierarchy are associated with epochs, periods when the function set is not adapted, emphasizing an iterative construction process. These features of the Adaptive Representations approach are therefore analogous to the proposed approach where levels are constructed based on previously-evolved, progressively more complex, behaviours. Lastly, as with Module Acquisition, the process of code encapsulation can be viewed as a form of symbiosis.

2.3.3 Cascade Architectures

An early application of learning under a cascade architecture was the cascade correlation algorithm used to incrementally train neural networks [42]. This approach begins by training a network with no hidden nodes but in which every input is directly connected to every output node. Whenever a plateau in performance is reached, e.g., after training the initial minimalist network, a single hidden unit may be added to the network in two steps. In the first step, the original inputs, as well as the outputs of any existing hidden units, are connected as inputs to the new unit. The new unit is

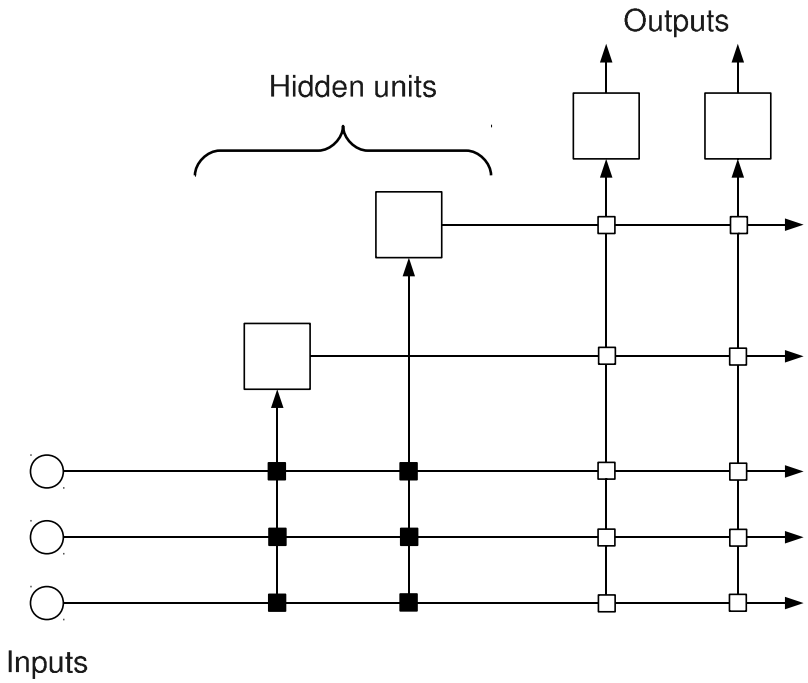


Figure 2.2: Example of a cascade architecture with three inputs, two hidden nodes, and two output nodes. The connections into the hidden nodes (marked in black) would be fixed if another hidden node was to be added.

then trained to maximize the correlation between its output and the magnitude of the error observed at each output node. In the second step, the new unit's input weights are frozen, its output is connected as an additional input to the output nodes, and the weights associated with the original output nodes are retrained. Since the newly added hidden unit is trained to correlate its output with the error at each output node, its contribution may be used by the system to cancel out this error. A sample structure with two hidden nodes generated under the cascade architecture is shown in Figure 2.2.

Variations of the cascade architecture have been proposed, e.g., using error minimization to approximate the target function at *each* cascaded unit [117] or using different learning algorithms at each layer in the cascade [54]. Despite its various forms, the following common characteristics of cascade architectures are identified:

1. Training begins with a minimalist structure, e.g., a fully connected network with no hidden nodes.

2. Once training of the current structure reaches a plateau, the structure may be augmented through the addition of a single unit.

3. Whenever a new unit is added, the parameters associated with the previously added units are frozen.

4. The output of any units that are added may form virtual inputs [117], i.e., the output of each additional unit may act as an input to all higher-level units.

Cascade architectures therefore represent an approach to incremental complexification and as such hold several advantages [42]. The approach allows new units to focus on learning specific behaviours that may compensate for the deficiencies in existing units. Because learning occurs only with respect to a single unit, it has been suggested that cascading has the potential to address the *moving target* problem whereby training a component is made difficult by concurrent changes to other components in the system. This has been demonstrated in the realm of neural networks where, despite adapting a single unit, the speed of learning was actually found to increase. Finally, under the cascading approach, it is not necessary to specify the complexity of the solution *a priori* as training, and the addition of new nodes, can be terminated when an acceptable level of performance is reached. A disadvantage of the approach is that fixing the parameters of the lower-level represents a greedy strategy that may result in sub-optimal structures, where this is also true in the version of the proposed approach presented in this thesis.

Using GP to evolve the cascaded units, the approach has been used for classification where it was found that higher levels of the cascade resulted in reduced error rates [114, 115]. In this example, an important part of the system was a fitness sharing component that penalized units that duplicated the behaviour of lower-level units. Cascading has also been suggested for applications with large feature counts and unbalanced datasets [98] where the cascade units played the dual role of feature detectors as well as filters of instances of the majority class.

Cascade architectures are related to vertical problem decomposition because lower-level units represent fixed behaviours that can be used in training additional higher-level units. For example, a new unit can choose to replicate the behaviour of lower-level units in some situations but not others. Due to restrictions imposed on the construction process, cascading architectures result in hierarchies that are very tall. Furthermore, the philosophy behind cascading appears to be that higher-level units should represent refinements over lower-level units. Finally, it is noted that cascade architectures can be viewed as an automated approach to layered learning, Section 2.3.1, and as such differ from the proposed approach in that layers are activated bottom-up and all units are activated at each layer.

2.3.4 Hierarchical Teaming using Genetic Programming

Hierarchical cooperative mechanisms have also been suggested as a way of achieving vertical problem decomposition in GP-based teams [172]. Specifically, in each team, one of the members is *a priori* designated to be a leader whose role it is to route input cases to one of the other team members. Once selected by the leader, the other team member then generates the final team output, e.g., predicted label under classification. Thus, each input case is processed by exactly two individuals, with the leader assuming responsibility for partitioning the input space among its associated team members. Naturally, such an architecture suggests that whereas the other team members must appropriately identify specific, low-level, behaviours, the leader must learn how to apply these behaviours in different situations.

The approach was evaluated on several classification problems [172]. Though results were mixed, it was found to perform relatively well on problems with high levels of epistasis and noise – to confirm these finding, further analysis was suggested.

The leader-based cooperation mechanism appears to be best suited for team (as opposed to island) approaches to GP teaming because without a persistent association between the individuals comprising a team the necessary context is lost. For example, replacing a team leader with another may lead to unpredictable results, even if the rest of the team remains the same. The approach was demonstrated using team sizes of three, i.e., one leader and two subordinates. Extending it to teams containing four or more members requires that the leader have to capacity to choose among three or more alternatives much like in the case of multi-class classification. Alternatively, requiring only binary decisions to be made by each individual, more than two levels of individuals can be defined within the same team.

2.3.5 Symbiotic Adaptive Neuroevolution

Symbiotic Adaptive Neuroevolution (SANE) [132] refers to a hierarchical approach for evolving neural networks that is of particular relevance to this work. In the next two sections, the key features of the algorithm are reviewed and its significance to the proposed approach is discussed.

Overview of the Two-Population Approach

The SANE framework [132] is a cooperative coevolutionary algorithm that aims to evolve a diverse set of complementary solution subcomponents in a single population,

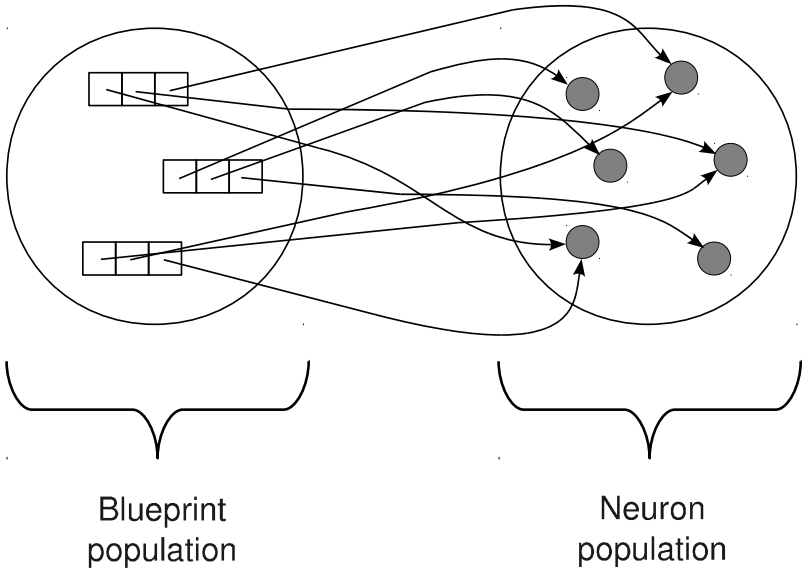


Figure 2.3: Populations under the SANE approach. Each fixed-size blueprint specifies the neurons (connections and associated weights) that are to be combined into a neural network.

though the architecture itself involves two populations. Specifically, a population of *neurons* and a population of *blueprints* is evolved[19], Figure 2.3. Each member of the neuron population specifies a set of connections and their associated weights. To form a solution, i.e., a neural network, several neurons are combined to collectively define the overall network structure. The neuron combinations are not arbitrary, however, but are specified by the members of the blueprint population. That is, each individual in the blueprint population contains a set of references to members of the neuron population which are to be used in constructing a single network. Thus, whereas the neuron population searches through the space of solution subcomponents, the role of the blueprint population is to identify and store the most effective neuron combinations so that they can be exploited by the search process. Fitness is evaluated both at the blueprint level, by considering the performance of the network specified by each blueprint, as well as the neuron level, by considering the fitness of the networks that the neuron appears in.

Two advantages associated with the SANE approach are suggested [132]. First,

[19]Two main versions of the SANE algorithm appear in literature, one that makes use of a blueprint population [132] and one that does not [133]. In this review, the focus is on the former.

compared to approaches where individuals represent complete solutions, e.g., traditional neuroevolution, operating at the level of neurons takes advantage of knowledge regarding the solution representation and allows individual building blocks to be identified and evaluated more effectively. That is, recognizing that neurons represent building blocks upon which selection and variation operators are explicitly applied is viewed as a simplification of the learning task. Second, since a single neuron is unlikely to perform well in isolation, the emergence of different roles or specializations is expected. This is viewed as particularly relevant since it may lead to sustained diversity in the population, or at least it may help to prevent premature convergence, the latter being an issue that has and continues to be of interest to the greater EC community [57, 74, 76, 127]. Furthermore, the way in which diversity is encouraged under the SANE approach is viewed as a form of *implicit fitness sharing* [73], that is, diversity is encouraged by restricting the ability of individual subcomponents thus requiring them to assume specific roles through cooperation. In contrast to explicit diversity maintenance schemes, e.g., fitness sharing [57], this is viewed as more efficient since pairwise comparisons between population members are not required. Implicit fitness sharing has been demonstrated in the context of a LCS [73], and here, the assertion is made that it applies to any bid-based system where individuals represent partial solutions.

The SANE algorithm was evaluated with regards to the quality of solutions found, the rate of population convergence, as well as the ability of the population to adjust to changes in the problem definition [132]. Compared to an adaptive neuron search in which combinations of neurons were selected arbitrarily [133], it was found that SANE discovered solutions of similar quality in early stages of evolution but in the latter stages it did significantly better – this emphasized the contribution of the blueprint population in leveraging knowledge regarding effective combinations of neurons. Despite relying on an aggressive, elitist, breeding strategy, it was found that the rate of convergence in the SANE neuron population was much slower than the rate of convergence in populations where complete networks were evolved, even if selection pressure in the latter was lower. In fact, as predicted, analysis indicated the emergence of diverse specializations in the SANE neuron population. Finally, compared to these other approaches, results showed that the increased diversity in the SANE neuron population allowed the evolving solutions to more quickly adapt to a dynamic environment.

Discussion

SANE is viewed as a contribution that is of particular relevance to this work because of the hierarchical architecture that it explicitly recognizes. That is, two populations are coevolved, Figure 2.3, a population of solution subcomponents and a population representing combinations of these partial solutions, i.e., complete solutions. Thus, in terms of a hierarchy, the blueprint population represents more complex behaviour than the neuron population, which is why SANE is viewed as a form of vertical problem decomposition. What the SANE literature fails to recognize is that this decomposition can be applied iteratively to generate increasingly more complex units of organization, e.g., using the networks evolved in an initial invocation of the algorithm as building blocks whose combinations are explored at subsequent levels of the hierarchy. A beneficial side-effect of evolving partial solutions in a single population is that it may enable implicit fitness sharing to take place.

The proposed approach uses an architecture that is similar to that of the SANE framework. There are two features of the SANE algorithm, however, that are viewed as deficiencies and as such are addressed in this work. First, the SANE blueprints always include the same number of references, and no clear reason is given as to why variable-length blueprints are not used. Implementing variable-length structures would require a corresponding upper limit to be set and the search operators to be modified so as to vary blueprint sizes. However, it would enable evolution to determine the necessary model complexity potentially leading to simpler or more effective solutions. Second, SANE assigns fitness to the neurons, and removes the neurons with the lowest fitness in each generation. However, the algorithm description does not specify what happens to the references in blueprints that index the neurons that are removed, e.g., are they reassigned or dropped altogether? In the proposed approach, fitness is only assigned at the level of complete solutions, i.e., at a level analogous to the blueprint population in the SANE approach, while solution subcomponents are culled only when their reference counts drops to zero. This reduces the number of design decisions that have to be made, and eliminates the potential 'dangling reference' problem.

The SANE literature defines symbiotic evolution as 'coevolution where individuals explicitly cooperate with each other and rely on the presence of other individuals for survival' [132]. Whereas this definition is more consistent with the view that symbiosis is strictly a mutually beneficial relationship, i.e., the interpretation of symbiosis as a state, Section 1.5, the process of compartmentalization that occurs in the blueprint

population suggests that the symbiosis in SANE may be more accurately described as an operator. That is, a particular blueprint may specify a network in which certain neurons are in fact parasites, so that cooperation cannot be an assumed property of the collaboration but rather emerges as measured by natural selection. In terms of the vocabulary introduced earlier, the blueprint and neuron populations can thus be viewed as host and symbiont populations. However, one feature that makes SANE inconsistent with symbiosis as viewed in this work is that the algorithm relies on recombination which is a mechanism associated with the traditional neo-Darwinistic approach to EC, Section 1.5.2.

Finally, it is noted that the idea of a two-level architecture, solution subcomponents at one level that are combined into complete solution at another, is not new or limited to the SANE framework. For example, the review in this chapter includes the Adaptive Representations approach [155, 156] which evolves programs (complete solutions) at one level which in turn reference subroutines (solution subcomponents) at another. A similar architecture also appears in one of the Hayek systems [13] where rules and rule sets respectively can be viewed as symbionts and hosts. However, the SANE framework does appear to be the first to explicitly recognize, separate, and evolve the two roles through cooperative coevolution.

2.4 Coevolution

Traditionally, two types of coevolutionary algorithms have been recognized, *cooperative coevolution* [132, 149, 195] and *competitive coevolution*[20] [4, 68]. Intuitively, cooperative coevolution refers to situations where partial solutions collaborate to decompose the problem, whereas competitive coevolution involves antagonistic interaction between complete solutions [23, 195]. Recently, efforts have been made to relate the two types of coevolution based on the underlying domain on which the respective algorithms operate [147]. Specifically, competitive is typically associated with domains where there are two interacting players and where the payoff matrix resembles a zero-sum game. In contrast, cooperative is typically associated with domains with two or more players, where the outcomes can be arbitrary, and often each player receives the same outcome. Though more rigorous, this classification is not complete, with many domains falling beyond its scope [147], e.g., *non-competitive* [195] forms of coevolution and the non-zero sum Iterated Prisoner's Dilemma [4].

[20]Not to be confused with the competitive cost-benefit characterization considered under symbiotic relationships, Table 1.1.

In EC, individuals are typically evaluated using a fixed fitness function. In contrast, the idea behind competitive coevolution is to base evaluation on a set of coevolving individuals. These coevolving individuals can be in the same population, as in certain games [4, 139], or in different populations, as is typical in optimization problems [46, 68]. From the perspective of this thesis, competitive coevolution can therefore be viewed as an approach to constructing a dynamic set of environmental conditions, i.e., an ecosystem. Under cooperative coevolution, individuals are also evaluated through interaction with other coevolving individuals, however, the other individuals do not provide a baseline against which performance is measured and a fitness function must therefore be specified *a priori*. Thus, whereas the traditional goal behind competitive coevolution has been to provide an evolving fitness function, cooperative coevolution has represented a means of problem decomposition.

In the context of coevolutionary search, a *solution concept* [45, 47] identifies the subset of the search space that contains acceptable solutions versus the space of non-solutions. Examples of solution concepts include the set of strategies in a Nash Equilibrium [47], the set of individuals that maximize the expected score against an arbitrary opponent [85], and the Pareto-optimal set [46, 139]. It is argued that a key reason for failure in coevolutionary algorithms is often the lack of an appropriate definition for the relevant solution concept [45]. In loose terms, it is difficult to find something if one does not know what exactly it is they are looking for. Formally specifying a solution concept also allows progress in the search to be measured, and in some cases, monotonic improvement can be assured [85, 86, 47]. Unfortunately, solution concepts have been primarily considered in the context of competitive coevolution.

The adaptive nature of coevolutionary algorithms potentially makes them very useful, e.g., in focusing limited resources on a small, relevant, region of the search space. Unfortunately, it appears as if initial assumptions about the behaviour of such algorithms were incorrect, and despite a number of successes coevolutionary algorithms often fail to fulfill their anticipated potential [34, 45, 195]. This motivated a number of investigations using uncomplicated algorithms applied to simple domains so that coevolutionary dynamics could be more easily observed and understood [149, 190, 195]. In contrast, in this work coevolution is applied in more complex contexts making similar insights more elusive, with the associated tradeoff being that coevolution could now be demonstrated in more practical situations.

In the proposed approach, elements of both competitive and cooperative are present, therefore, both forms of evolution are reviewed in the following sections.

2.4.1 Competitive Coevolution

As suggested earlier, Section 2.4, the goal behind competitive coevolution is to provide an evolved fitness function so that one does not have to be specified *a priori*. As such, an individual's fitness can be viewed in two ways [23, 86]. First, an individual's *absolute* fitness is a global measure of solution quality analogous to the static fitness function traditionally used in EC, e.g., it can be determined by calculating the solution's proximity to the solution concept [86]. In particular, an individual's absolute fitness remains fixed regardless of the makeup of the coevolving population. In contrast, the *relative* fitness of an individual corresponds to its performance as measured through its interactions with other coevolved individuals. Naturally, since coevolution implies change, the relative fitness is dynamic. The terms objective and subjective fitness have also been used in place of absolute and relative respectively [86, 190, 195].

The dynamic nature of the relative fitness function under competitive coevolution has a direct effect on the learning gradient, the aspects of the problem that are emphasized, and the degree to which relative fitness reflects absolute fitness [190]. In each case, the overall effect may be positive or negative. For example, because fitness is evaluated in relation to other coevolved individuals and not some fixed baseline, open-ended learning may be possible. At the same time, if relative fitness does not reflect absolute fitness, nothing useful may be learned, or worse, individual quality may degrade. Without careful design, competitive coevolution can thus lead to pathologies such as disengagement (loss of gradient), focusing (over-specializing), cycling, and mediocre stable states [23, 190, 195].

Formulated appropriately, competitive coevolution has a number of advantages, with the principal benefit being that it can be used in situations where no suitable fitness function is known [147, 190]. Since it allows evaluation on a small subset of training cases, competitive coevolution can also be used to improve training efficiency [34, 46]. The latter is especially relevant to this work where large datasets and problems with large state spaces are considered. Finally, under difficult problems, coevolution has the potential to be more efficient by focusing the search on, e.g., test cases that match the ability of the solutions that are evolved [46, 190].

In the following sections, Pareto-coevolution and the host-parasite model are reviewed. The former represents a solution concept that is relevant to the proposed approach to the extent that it has been considered in previous versions [111, 112]. The host-parasite model, on the other hand, refers to a dynamic that may develop

between the populations evolved under competitive coevolution.

Pareto-Coevolution

The defining characteristic of Pareto-coevolution is that, with respect to any given individual, it views each other competing individual as an objective to be optimized [46, 139]. Given this interpretation, techniques from Evolutionary Multi-Objective Optimization (EMOO) can be applied to the search problem, and in particular, the Pareto-dominance relation is often used. Specifically, given a set of objectives O individual x_i dominates individual x_j if and only if

$$\forall o \in O : G(x_i, o) \geq G(x_j, o) \wedge \exists o \in O : G(x_i, o) > G(x_j, o) \tag{2.1}$$

where $G(x_i, o)$ is the interaction function that defines the outcome of individual x_i with respect to objective o. Without loss of generality, higher outcomes are considered to be better. According to the above relation, individual x_i dominates x_j if and only if it performs at least as well on every objective, and does strictly better than x_j on at least one objective. Typically, x_i and x_j are assumed to be in the same population with the individuals representing objectives evolved in a separate population. Under Pareto-coevolution, the solution concept is then the set of solutions that are mutually non-dominated, i.e., choosing one individual over another from this non-dominated set involves a tradeoff in objective values.

Pareto-coevolution often involves the interaction between two populations, a population of *teachers* and a population of *learners* [86, 34, 46, 111, 112, 147]. The learners represent the potential solutions, while the teachers are tests which collectively define the relative fitness function. The learners are evaluated by applying the Pareto-dominance relation to the set of objectives defined by the teacher population, i.e., for each learner there is one objective per teacher. Given n learners, the teachers are evaluated by considering the $n^2 - n$ pairwise comparisons between the learners (or n^2 comparisons if a learner is also compared with itself). Specifically, for a teacher t_k, the value of objective $i \cdot n + j$ is calculated as

$$\begin{cases} 1 & \text{if } G(l_i, t_k) > G(l_j, t_k) \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where l_i and l_j are the ith and jth learners assuming a sorted list. Thus, if t_k distinguishes between l_i and l_j the objective value is set to 1, i.e., a distinction is

made, otherwise it is set to 0. As such, the learners are evaluated based on how well they perform on the test cases represented by the teachers, and the teachers are evaluated based on how informative/useful they are in distinguishing between the learners.

A feature of distinction-based Pareto-coevolution is that, despite falling under the label of competitive coevolution, it exhibits both competitive and cooperative characteristics in calculating teachers' fitness. For example, a teacher receives a minimum relative fitness if it defeats all learners to the same extent, or when it is defeated by all learners to the same extent – in both situations, no distinctions are made. In contrast, a fit teacher is one that defeats some learners but is defeated by others, and a fitness measure that captures this property is also used in the proposed approach, Section 3.2.6. Though many coevolutionary algorithms may be like this in that they may not be strictly competitive or cooperative, this has traditionally been ignored as practitioners tend to assign one label or the other [45, 195].

Given unlimited resources, monotonic progress can be guaranteed under distinction-based Pareto-coevolution [86]. In practice, the limits imposed on the size of the archives that are maintained makes this impossible. Another fundamental problem with Pareto-coevolution is that as the number of objectives grows, the non-dominated set tends to include most if not all of the individuals under consideration [46, 139] – and the individuals' behaviours tend to overlap significantly [125]. Given the high number of mutually non-dominated individuals, differentiating between them becomes a problem. Indeed, this is a well known issue in EMOO where algorithms perform well on two objectives but where performance tends to degrade sharply as additional objectives are considered [51, 80].

There are therefore two recognized problems with Pareto-coevolution resulting from the large number of objectives, the growth of the non-dominated set and the difficulty in performing optimization involving many objectives. To cope with the former, heuristics for ranking mutually non-dominated solutions such as eliminating 'nearly dominated' strategies [139], counting instances where learners out-score one another [46], and competitive fitness sharing [157] have been proposed. Unfortunately, these heuristics break the assumptions that form the basis for certain theoretical results, e.g., on monotonic progress [86].

Two less recognized issues with distinction-based Pareto-coevolution are the efficiency of performing the $n^2 - n$ pairwise comparisons on the learners and the resulting meaningfulness of the distinction that are or are not made. The number of

comparisons is proportional to the square of the number of learners, therefore, typical parameter settings may result in tens of thousands of comparisons (objectives to be optimized). At that point, it becomes unclear if all of the distinctions that are made are significant. For example, a distinction may be more significant if it is made between two strong individuals compared to one that is made between two weak individuals [46], or if it is made by only a single teacher [157]. Other studies have shown that even if test cases are deemed fit and diverse in the phenotype space, they may be trivially different in the genotype space and thus unrepresentative of the entire problem domain [113].

Attempts are made in the proposed algorithm to address these two less recognized issues. Specifically, a test fitness function is proposed that does not rely on making pairwise distinctions yet still rewards tests if they distinguish between the learners, Section 3.2.6. Depending on the reward scheme, the resulting algorithm is found to match or outperform distinction-based coevolution, Section 6.4.2.

Host-Parasite Model

The host-parasite (or predator-prey) model refers to a dynamic that may occur under competitive coevolution in general, i.e., it is not specific to any solution concept. Under the host-parasite model, coevolution is viewed as an analogy to a 'biological arms race' [68]. Two populations are typically assumed, a population of hosts and a population of parasites, though the analogy could likely be generalized to any number of populations. Through coevolution, a series of adaptations and counter-adaptations [33] may result. As such, though the relative fitness of the two populations may stay the same, a phenomenon often referred to as the Red Queen Effect [195], the absolute fitness may improve – the development of this dynamic is therefore viewed as an important factor in the success of the coevolutionary algorithm. Key to the emergence of the host-parasite dynamic is the avoidance of disengagement and cycling. Methods to address the former include reducing parasite virulence [23] and using monotonic solution concepts [86, 47], whereas the latter is typically addressed through the use of test case archives [86, 47, 125, 157].

2.4.2 Cooperative Coevolution

Cooperative coevolution refers to the evolution of solutions consisting of 'interacting coadapted subcomponents' and is motivated by the notion that increasingly complex problems will require increasing levels of modularity [149]. In contrast, traditional

EAs do not model interactions between individuals in the population and as such return what are viewed as monolithic solutions, e.g., a single program, bit string, or real-valued vector. At best, heuristics such as fitness sharing [57] may be applied to subsets of individuals in the same population, though such interactions are not explicit and as such fall beyond the scope of what is considered cooperative coevolution [195].

Successful cooperative coevolution typically requires four common issues to be addressed [149]:

1. The number of subcomponents and their roles must be defined. In many real-world applications, the number of subcomponents may not be known beforehand, let alone their roles. Is therefore important for the algorithm to have the ability to determine this decomposition automatically.

2. The decomposition of complex problems often requires complex interactions between the resulting subcomponents. These interactions must be modeled during evolution otherwise subcomponents, in their respective roles, will not emerge.

3. An appropriate mechanism for credit assignment must be defined. This is trivial if credit is assigned at the level of complete solutions, but not if it is to be shared among solution subcomponents in proportion to the contribution that each makes.

4. The solution subcomponents must be sufficiently diverse for there to be a benefit in combining them. For example, there is little use in applying a voting heuristic to a set of subcomponents where each makes the same decision.

In the proposed approach, the first issue is addressed through a symbiotic relationship that results in the exploration of different combinations of subcomponents in different roles, i.e., following the metaphor established in the SANE algorithm. Interactions between solution subcomponents are modeled using a bid-based metaphor – this is a generic approach applicable across a number of problem domains. The problem of credit assignment is bypassed by not explicitly assigning credit to solution subcomponents. Instead, the same subcomponent participates in multiple solutions, and its survival is contingent on the fitness of those complete solutions. Finally, the proposed approach uses mechanisms resembling fitness sharing [62, 157] at multiple levels to ensure diversity, in addition to the pressure for evolving diverse subcomponents that results from the symbiotic relationship that is central to the algorithm [132].

A widely accepted framework for cooperative coevolution assigns a separate species to each solution subcomponent and selects one individual from each species to form a complete solution [149]. Typically, each species is evolved in a separate population. The number of species can be specified *a priori*, or it can be varied at run-time, e.g., by adding species when stagnation is observed and removing species that make insufficient contributions. The approach[21] has been used in a number of problem domains such as ensemble learning for classification [90], multi-agent learning [138], scheduling [77], and function optimization [148]. It was found to exhibit a tendency towards complete solutions that are resilient to change in their constituent subcomponents but which otherwise may be suboptimal [195]. Thus, it appears to be more suitable for domains where robustness is important, e.g., multi-agent learning, as opposed to situation where the global optimum is required.

Some of the behaviours observed under competitive coevolution are also possible under cooperative coevolution, though perhaps in different forms [195]. For example, the learning gradient may be lost if one species is so poor that all collaborations that its members take part in receive minimum fitness. If, on the other hand, gradient is maintained, an *adaptive evolutionary balance*, a cooperative analogy to the arms race discussed in Section 2.4.1, may result in continued improvement in the absolute fitness.

Though not traditionally associated with cooperative coevolution, the study of collectives may provide insight into evolving cooperative behaviours. An example is the collective intelligence (COIN) framework [185] where the goal is to define a local utility function that is aligned with a global utility function. That is, each solution subcomponent aims to maximize its own utility, and in the process, the utility of the system as a whole is maximized – in this sense, cooperation between the agents is achieved. COIN defines a family of *difference utility functions* that attempt to calculate the contribution of a single agent towards the performance of the collective, and may thus address the issue of credit distribution among solution subcomponents.

Finally, it is noted that although the multi-population approach [149] appears to be a representative framework for what in EC is widely accepted as 'cooperative coevolution', a number of other related algorithms have also been proposed and many have been reviewed earlier in this chapter. The foremost of these are likely classifier systems [72, 196] where the action sets can be viewed as the collaborating components. In similar spirit, the Hayek model [10] employs an economic metaphor to evolve a

[21]Though formally presented as a generic approach to cooperative coevolution in [149], this framework appears in literature before then.

population of bidders that learn to coordinate their actions. The SANE framework [132] is yet another model where the solution subcomponents are evolved in a single population. Finally, many of the teaming approaches reviewed in Section 2.2.3 can be viewed as forms of cooperative coevolution. The requirement of cooperative coevolution for solutions in the form of 'interacting coadapted subcomponents' [149] is rather general and as such may apply to any evolutionary approach which explicitly encourages modularity.

Chapter 3

Symbiotic Bid-Based Model

This chapter provides a detailed description of the SBB model which is the focus of this thesis. It begins with an overview of the key principles underlying the framework in Section 3.1. The core training algorithm is presented in Section 3.2, where it is also interpreted in relation to the process of symbiosis as an operator, Figure 1.5. A number of additional details, viewed as useful in improving performance but not central to the framework, are then discussed in Section 3.3, while Section 3.4 suggests a way to extend the core SBB algorithm by incrementally constructing solutions hierarchically. The chapter concludes with Sections 3.5 and 3.6 which discuss, respectively, the nature of the cost function and the details of the GP implementation.

The model overview and the description of the core training algorithm, Sections 3.1 and 3.2 respectively, present the SBB framework as an approach to lateral problem decomposition through context learning; key to this assertion is the bid-based approach detailed in Section 3.1. The ability of the SBB approach to achieve vertical problem decomposition is suggested in Section 3.4, where the core training algorithm is applied iteratively as a process for layered learning.

3.1 Model Overview

The proposed approach can be characterized in terms of three underlying principles:

1. **A bid-based approach to context learning.** The issues of 'what to do' and 'when to do it', viewed as central to the type of context learning considered in this thesis, are separated through the use of a bid-based metaphor [110, 111]. Specifically, each bid-based individual (or bidder) consists of a bid component and an action component, Figure 3.1. The action component is a scalar chosen from a finite set that defines what to do, while the bid component represents the bidding behaviour that is used to determine when to apply the associated action. Given an input, an *auction* or *bidding round* is triggered whereby each individual submits a real-valued bid based on its bidding strategy, and the action to be applied is then deterministically chosen as the one associated with the highest bidder, Figure 3.2.

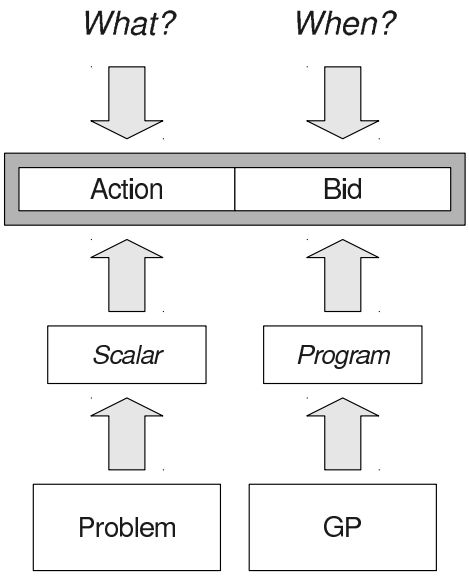


Figure 3.1: Components in a bid-based individual as implemented in this work. A single action (the 'what') is chosen from a finite set of scalars, while the bid component (the 'when') takes the form of a program; whereas the action is problem-dependent, the program is evolved using GP. Once created, these components remain fixed in a given individual.

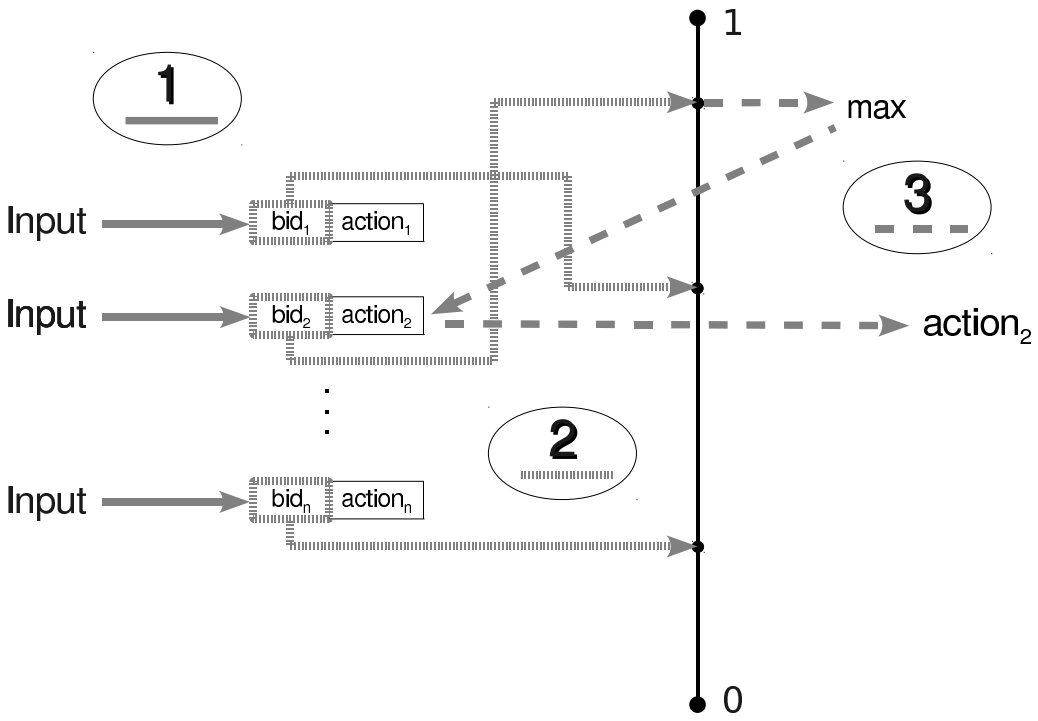


Figure 3.2: The bid-based action selection procedure, or auction, involves three main steps. In the first step, the input is forwarded to the bid component in each individual. In the second step, each individual submits a real-valued bid based on the input which, in this case, is mapped to the range $[0, 1]$. In the final step, the maximum bid is identified, and the action of the individual associated with that bid is selected as the final output. In the above example, the action associated with the second individual (consisting of bid_2 and $action_2$) is output.

In this work, the bidding behaviour is evolved using GP while the possible action values depend on the problem domain. In classification problems, the actions correspond to the possible class labels. In control-style problems, the set of possible actions could be discrete, e.g., rotations of a Rubik's Cube, or continuous, e.g., force applied in balancing a pole. In the latter case, discretization can be performed to produce a finite set of actions. Given their restricted form, the bidders are typically not capable of representing non-trivial solutions if they are considered in isolation, e.g., under classification an individual would always predict the same class label. Thus, they are encouraged to cooperate through what can be viewed as implicit fitness sharing, Section 2.3.5.

The bid-based approach casts GP in a non-traditional role. Whereas typically GP is used to map input directly to output, e.g., exemplar to class label, in the bid-based approach GP is used to map input to a bid value which is then used to select an output via the auction mechanism, i.e., selection of the highest bidder. Intuitively, GP has traditionally been used to identify both the 'what to do' and 'when to do it', possibly confounding the decision as a whole. In contrast, here, the role of GP is restricted with the goal of facilitating learning; the bidding behaviour identified by a program corresponds to a fixed set of problem conditions, for example, those inputs where the program results in a high bid. This then supports the exploration of combinations of these fixed behaviours through which complexity may emerge in terms of the cooperative interactions that may be identified.

2. **Evolution of solutions by means of symbiosis.** The introduction of a symbiotic component into the bid-based framework was originally motivated by issues stemming from the fact that in the early implementations [110, 111] the bid auction was held across all bidders in the population, i.e., the entire population was considered to be a solution. As such, changes to the population through the addition, removal, or modification of bidding individuals could potentially have very disruptive effects, e.g., the introduction of an individual that outbid all others could result in degenerate behaviour. In addition, unless the target problem domain was supervised learning, it was not obvious how to incorporate competitive coevolution into the approach because outcomes were determined with respect to the population as a whole and not the individual bidders, i.e., since an individual represented a partial solution, its performance

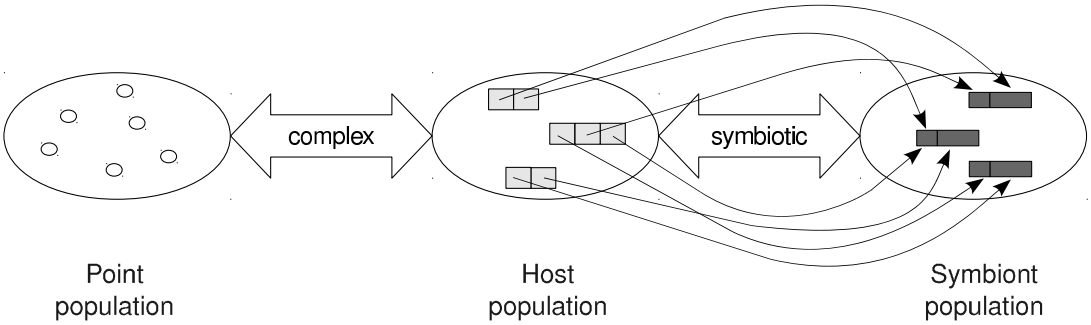


Figure 3.3: Three populations evolved in the SBB framework. Under the symbiont-host interaction, the same symbiont can appear in multiple hosts, and multiple symbionts associated with the same action can appear in a single host. In the proposed implementation, the host-point relationship involves both cooperative and competitive interactions as discussed in Section 3.5.

could not be evaluated in isolation. A related issue was that of credit assignment, specifically, the assessment of the contribution of each individual bidder towards the aggregate performance of the population.

To this end, the population of bid-based individuals was augmented with a second population whose role it was to enumerate useful combinations of bidders, center-right of Figure 3.3. The relationship between the two populations could thus be viewed as a symbiosis with the original bidders as symbionts and members of the second population as hosts, much like as in the SANE algorithm [132]. As such, the process of symbiosis as an operator, Figure 1.5, could now be effected. Whereas the first stage of this process, species coexistence, could already be realized under the single-population model, the second and third stages, i.e., compartmentalization and joint replication, were now supported as a direct result of including the host population.

By including the second population, the concept of a 'complete solution' was redefined. Previously, the entire population of bidders formed the complete solution, whereas now, a single host identified a complete solution in terms of a subset of individuals from the symbiont population. Thus, multiple candidate solutions could now be extracted from a single population of symbionts, and the bid auctions were restricted to symbionts within each candidate solution. Since a host, once created, remained fixed, modifications to the symbiont population could therefore no longer disrupt previously discovered symbiont combinations.

Furthermore, competitive coevolution could now be employed using established techniques by considering the performance of the hosts; the fitness of a symbiont in isolation was no longer explicitly evaluated alleviating the need to measure each symbiont's contribution directly.

A number of other benefits resulting from the addition of the host population can be suggested. First, the host size and composition is determined through evolution, whereas previously, the bidder population size and hence the solution size was not an evolved property but had to be specified *a priori*. Second, given that a symbiont may appear in multiple hosts, this may lead to a more accurate evaluation of its usefulness. Third, since a single symbiont is unlikely to perform well in isolation, population diversity may be increased if different specializations emerge, i.e., through implicit fitness sharing, Section 2.3.5. Finally, the processes behind the bid-based metaphor and the compartmentalization of symbionts are generic in the sense that they can be used to construct hierarchical solutions in which hosts evolved at lower levels form a basis for symbionts at subsequent levels.

3. **Dynamic evaluation through coevolution.** Since a host has the capacity to represent a complete, non-trivial solution, the host population is coevolved against a population of points representing training scenarios, center-left of Figure 3.3. Under classification, the point population represents a subset of the training instances, while under temporal sequence learning, a point corresponds to an initial environmental state[1]. As a result of this interaction, each host is assigned an explicit fitness value, but this information is not used directly to evaluate the symbionts, e.g., by assigning to each symbiont the average fitness of the hosts that it appears in. Instead, symbionts are considered to be viable as long as they appear in at least one host. This reflects the view of natural selection as a group fitness operator, and also unburdens the designer from having to specify a symbiont fitness function.

   The coevolutionary component between hosts and points was included to allow the approach to scale to problems with many training scenarios, e.g., classification problems with many exemplars or the numerous real-world temporal sequence learning problems that may be of interest. In particular, the desired

[1]The term 'point' is used to refer to an instance or initial state whether or not it actually appears in the current point population. For example, the training set in classification is viewed as a set of points from which the current point population is drawn.

scalability can be achieved if a learning gradient can be maintained using a (relatively) small sample of all possible training cases, thus incurring a reduced training overhead all the while advancing the search towards progressively better solutions. Furthermore, the interface between hosts and points in the proposed approach remains generic and as such is compatible with established approaches to competitive coevolution.

Given that the contents of the point population are subject to change, the fitness function in the proposed approach is dynamic. As suggested earlier, Section 1.3.1, a dynamic environment is likely to decrease the rate of convergence in population diversity, since, nominally, directional natural selection will then lead to the emergence of novel phenotypes; in contrast, stabilizing natural selection is likely to suppress innovations in favour of the norm. If niches can be established to support the viability of new species, this then complements the symbiotic metaphor which can take advantage of the increased diversity by combining individuals evolved under different lineages (as in the case of biological evolution). Under similar conditions, the recombination operators typically employed in EC are unlikely to succeed as they fail to consider the context under which genetic material is exchanged.

From a more pragmatic perspective, if the space of possible points is large, a smaller training sample may need to be selected and used for evaluation. If this sample is static, however, it is unlikely to support speciation in the host population unless it identifies the relevant niches from the outset. Here, it is suggested that this likelihood is low because of the large space from which the sample is drawn, e.g., the number of niches may be greater than the sample size. Furthermore, different niches may be relevant at different periods during evolution, e.g., niches corresponding to easier scenarios may be more helpful during early stages but may hamper the development of more sophisticated solutions. By employing a dynamic point population, these problems can be mitigated if training can be dynamically focused on the relevant niches.

Context learning in the SBB approach, as summarized above, can be viewed as a two-staged process. In the first stage, context learning is observed in the symbiont population where the conditions for applying specific actions are identified, i.e., in the form of the bid-based individuals that separate the what from the when. In the second stage, the search through combinations of symbionts carried out in the host population aims to establish the appropriate context under which individuals

Entity	Role
Symbiont (bidder)	Identifies context (when) for applying an individual action (what)
Host (compartment)	Identifies combinations of symbionts that are effective under the bid-based metaphor
Point	Training case that provides a direct measure of host fitness, and an indirect measure of symbiont fitness

Figure 3.4: Entities coevolved under the SBB model.

forge effective collaborations; context is therefore viewed with respect to the other symbionts participating in the same compartment.

As a final remark in this model overview, it is acknowledged that the algorithm details presented in the following sections represent one of possibly many implementations that reflect the three principles behind the proposed approach. Where possible, attempts are made to justify the design choices that are made, though it is by no means implied that the implementation is optimal and that refinements would not lead to improved performance.

3.2 Core Training Algorithm

Table 3.4 summarizes the entities relevant to the SBB model, while Algorithm 3.1 outlines the core training process which begins by first initializing the points, hosts, and symbionts in lines 3 and 4. The algorithm is generational, where the main loop, line 5, represents a generation. It iteratively creates new points and hosts, lines 6 and 7, evaluates the points on the hosts, line 10, and selects a subset of the points, hosts, and symbionts discarding the rest, lines 13 and 14. These steps are described in detail in the following sections, and the relevant algorithm parameters are summarized in Table 3.1.

Both the point and the host populations in Algorithm 3.1 follow what is considered to be an elitist generational model, though alternate models are by no means precluded. Specifically, a fixed number of new individuals is added to the population at the beginning of each generation. Subsequently, the same number of individuals representing the worst performers is deterministically removed at the end of the generation – whether or not the deleted individuals are parents or offspring in the current generation is not considered. This is consistent with a $(\mu + \lambda)$ selection strategy popularized in ES approaches where μ is the number of parent individuals, λ is

the number of offspring individuals, and the addition sign implies that both parents and offspring may be removed in any one generation [39]. By adopting an elitist policy, the algorithm always preserves what are identified to be the best individuals regardless of the amount of selection pressure applied.

A significant feature of the proposed algorithm is that the only variation operator applied to the symbiont population is mutation where this is reflective of the role played by symbiosis. Specifically, since crossover – the norm in GP – is not used, this places the onus on the process of symbiosis to combine genetic material from different individuals. It is therefore possible to draw analogies between the three stages of the process of symbiosis as adopted in this work, Figure 1.5, and the proposed framework as summarized in Algorithm 3.1:

1. Species coexistence is equated with the presence of individuals from different lineages in the symbiont population S^t. It could be argued that it is not possible to objectively ascertain whether or not different species do in fact emerge in the symbiont population. However, if symbiotic associations are successfully formed, this would suggest that speciation is taking place since little benefit would be gained by combining symbionts that behave in the same manner. As such, if a symbiont is observed to make distinct, non-trivial, contributions towards the behaviour of the host, this is viewed as evidence in support of species coexistence.

2. The compartmentalization stage is realized whenever multiple symbionts are assembled into a single host, i.e., host initialization, line 4, as well as host and symbiont generation, line 7. Little bias is introduced when choosing which symbionts to compartmentalize, therefore, any of the cost-benefit characterizations from Table 1.1 may result. If the interaction between the associated symbionts is mutualistic, however, it is expected that the likelihood of the host surviving (and reaching the joint replication stage) will be increased.

3. The final stage, joint replication of compartments, is triggered in two situations. First, it occurs whenever a new host survives into the next generation, i.e., it is not removed at line 14, at which point the host becomes part of the parent set used to create offspring in that subsequent generation. Joint replication also occurs when Algorithm 3.1 is applied iteratively to produce multiple levels of hosts, at which point, hosts at lower levels become fixed and reproduce as a unit. This second case is discussed in more detail in Section 3.4.

Algorithm 3.1 The core SBB training algorithm. P^t, H^t, and S^t refer to the point, host, and symbiont populations at time t.

```
1: procedure TRAIN
2:     t = 0                                                    ▷ Initialization
3:     initialize point population P^t
4:     initialize host population H^t (and symbiont population S^t)
5:     while t ≤ t_max do                                       ▷ Main loop
6:         create new points and add to P^t
7:         create new hosts and add to H^t (add new symbionts to S^t)
8:         for all h_i ∈ H^t do
9:             for all p_k ∈ P^t do
10:                evaluate h_i on p_k
11:            end for
12:        end for
13:        remove points from P^t
14:        remove hosts from H^t (remove symbionts from S^t)
15:        t = t + 1
16:    end while
17: end procedure
```

3.2.1 Point Population Initialization

The point population is initialized, line 3 of Algorithm 3.1, to contain $P_{size} - P_{gap}$ points. Here, P_{gap} corresponds to the number of points created and removed every generation, lines 6 and 13 respectively. Thus, the ratio of P_{gap} to P_{size} quantifies the selection pressure on the point population, allowing the amount of resources allocated to the exploitation of existing points versus the exploration of new points to be adjusted. That is, an increase in P_{gap} relative to P_{size} implies an increased bias towards exploration of new points and an increase in selection pressure.

Under classification, a balanced sampling heuristic is used to initialize the points. First, a class is selected with uniform probability. If all points belonging to that class have already been selected into the initial population, the class selection is repeated. Once a class is chosen, a point belonging to that class is sampled assuming a uniform probability. Throughout the process, duplicate training instances are not allowed. The goal of this two-step initialization process is to make the initial point population insensitive to the class distributions in the training dataset since many of the large, real-world, datasets are often highly skewed [193]. Naturally, adopting such a heuristic will bias the search in favour of solutions that recognize all classes but risk achieving

Parameter	Description
t_{max}	Number of training generations
P_{size}	Point population size
P_{gap}	Number of points generated/removed each generation
H_{size}	Host population size
H_{gap}	Number of hosts generated/removed each generation
ω	Maximum number of symbionts in a host
p_{genp}	Probability of generating an offspring point from a parent
p_{sd}	Probability of symbiont deletion
p_{sa}	Probability of symbiont addition
p_{sm}	Probability of symbiont mutation
p_{am}	Probability of action mutation
K	Number of nearest neighbours used in point fitness calculation
ϕ	Number of points used in calculating bid profiles
τ_{init}	Number of symbionts considered during mixing

Table 3.1: SBB model parameters.

a lower overall error rate.

Under temporal sequence learning, the point population is initialized to contain a set of random points. This notion of randomness is problem specific, and, in this work, an attempt is made to produce a physically diverse set of initial conditions. The specific procedures for generating points under each environment will be described when the environments themselves are presented, Chapters 6 and 8.

3.2.2 Host and Symbiont Population Initialization

The host initialization process, line 4 of Algorithm 3.1, generates $H_{size} - H_{gap}$ hosts where H_{size} and H_{gap} are defined analogously to P_{size} and P_{gap}. Each host is initialized in two steps, Figure 3.5. In the first step, two symbionts are added to an initially empty host by uniformly selecting two different actions and associating those actions with two new bidding behaviours; by requiring different actions, the likelihood of trivial behaviours is reduced with the goal of improving search efficiency. The programs defining the bidding behaviours are randomly generated in an implementation specific way, Section 3.6. Since each symbiont appears in exactly one host, at the end of this step there are exactly twice as many symbionts as there are hosts.

The goal of the second step is to produce hosts of varying sizes and to include the same symbiont in multiple hosts. A symbiont that is included in multiple hosts is evaluated in multiple contexts, where this allows the contribution of the symbiont to

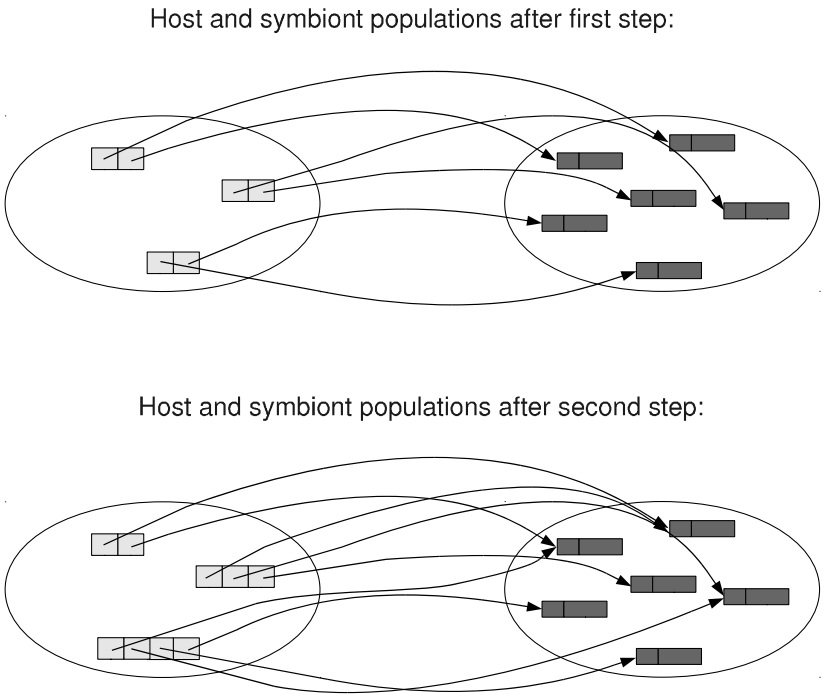


Figure 3.5: Example of the two steps in host initialization. The host populations, left, indexes symbionts in the symbionts population, right. All the symbionts are created in the first step, and mixing of symbionts occurs in the second step.

be more accurately evaluated, e.g., a symbiont whose inclusion always results in poor host behaviour will likely be eliminated. Each host from the first step is considered in turn and its final size selected with uniform probability from $\{2, \ldots, \omega\}$. Existing symbionts are then selected and added to the host until its size reaches the chosen limit (the heuristic used to select learners is described in Section 3.3.3). After the second step, there are still twice as many symbionts as there are hosts, i.e., no new symbionts are added, but each symbiont may appear in more than one host.

3.2.3 Point Generation

At the beginning of the main algorithm loop, line 5 of Algorithm 3.1, the point population contains exactly $P_{size} - P_{gap}$ points. This is because either the loop is entered for the first time following point initialization, line 3, or because P_{gap} points were removed near the end of the last iteration of the loop, line 13. The point generation step, line 6 of Algorithm 3.1, is therefore included to fill the point population to capacity through the addition of P_{gap} new points. Following this step, the point population will contain exactly P_{size} points.

Here, a fundamental difference typically observed between points under classification problems and points under temporal sequence learning problems is noted:

- Under classification, there is no obvious structure in the set of training exemplars that can be used by the variation operators to generate an offspring point from a parent point, i.e., the variation operator would need to relate points that have been fixed *a priori*. For example, it is not clear how to define a meaningful mutation operator on an index to generate another index in the training dataset. Alternatively, one could apply mutation to the feature values of an exemplar, though this is likely to produce a vector whose values do not match any other training case in the dataset. Finally, structural information could be extracted through an exploratory preprocessing step, i.e., unsupervised learning, though this is considered to be beyond the scope of this work.

- Unlike classification, under temporal sequence learning, the space of training points is typically not enumerated in part because explicitly accounting for the possible training states may not be practical. Instead, structural information is often provided in terms of a set of constraints specifying feasible states, and this information can be used by variation operators to generate offspring points. For example, an action can be applied to a state to generate a new configuration

of the world (assuming that a single action has a relatively small effect on the world). Alternatively, if the features are quantities, e.g., angles, distances, temperatures, then nearby states can be reached by mutating each of the state vector components[2].

Acknowledging this difference between points under classification and temporal sequence learning, the point generation step is implemented as follows. Under classification, where no structural information is available for use in designing the variation operators, P_{gap} new points are generated using the same class-balanced uniform sampling heuristic that is used during point initialization, Section 3.2.1.

The point generation procedure under temporal sequence learning is summarized in Algorithm 3.2. The algorithm assumes a set P of *parent points* defined as the contents of the point population P^t at the beginning of the current generation t. With probability p_{genp}, a parent is selected with replacement from this set using fitness-proportionate roulette wheel selection[3] using the fitness of the point, Eq. 3.6, as the wheel slice. A copy of the selected parent is then mutated to produce the new offspring point in a problem-specific way, i.e., by taking advantage of the available structural information, and added to P^t. With probability $1 - p_{genp}$, the new point is not generated from a parent but rather through the same procedure as is used in point initialization, Section 3.2.1. This occasional insertion of a random point is a heuristic meant to prevent convergence in the point population. Generation $t = 0$ is also a special case when all new points are generated as in point initialization since none of the points have an associated fitness value.

3.2.4 Host and Symbiont Generation

Analogous to the case of the point population, Section 3.2.3, at the beginning of the main loop, line 5 of Algorithm 3.1, there are exactly $H_{size} - H_{gap}$ hosts in the host population. Host and symbiont generation, line 7 of Algorithm 3.1, therefore creates H_{gap} new hosts and adds them to the host population H^t increasing its size to H_{size}. New symbionts that are created are added to the symbiont population S^t. Since each host may contain between two and ω symbiont references, the size of the symbiont population may vary between $2 \times H_{size}$ and $\omega \times H_{size}$.

[2]In this work, the former approach is used in the Rubik's Cube problem, while the latter is used in the truck reversal problem.

[3]In which the probability that an individual is selected is defined to be the fitness of that individual divided by the fitness values summed across all individuals under consideration.

Algorithm 3.2 Point generation algorithm under temporal sequence learning. The set P represents the contents of P^t at the start of the current generation t, while $rand(0,1)$ returns a uniformly selected scalar in the unit interval. With probability p_{genp}, a new point is created as an offspring of an existing point, otherwise, a random point is added. A random point is also added if it is the first generation.

```
1: procedure GeneratePoint(t, P^t, P)
2:     if t ≠ 0 and rand(0,1) < p_genp then
3:         select p_k ∈ P using roulette wheel selection
4:         copy p_k into p'_k
5:         mutate p'_k
6:         insert p'_k into P^t
7:     else
8:         insert random point into P^t
9:     end if
10: end procedure
```

Host generation first defines the set of *parent hosts* as the host population H^t at the start of the current generation t. The set referred to as the *available symbionts* is analogously defined as the symbiont population S^t at the start of the current generation. Thus the parent hosts and the available symbionts represent the contents of the host and symbiont populations before the host and symbiont generation step, i.e., they represent the available genetic material. Each of the H_{gap} offspring hosts is then generated in four steps, Figure 3.6:

1. A parent is selected from the set of parent hosts with uniform probability, and an initial offspring host is created as a copy of the parent, i.e., referencing the same subset of available symbionts.

2. Uniformly selected references are removed from the offspring host, Algorithm 3.3. The number of references that are removed depends on the probability of symbiont deletion p_{sd}.

3. References uniformly selected from the set of available symbionts are added to the offspring host, Algorithm 3.4. The number of references added depends on the probability of symbiont addition p_{sa}.

4. Host mutation, Algorithm 3.5, changes at least one symbiont in the host, though changes are applied only to copies so that the existing symbionts are preserved

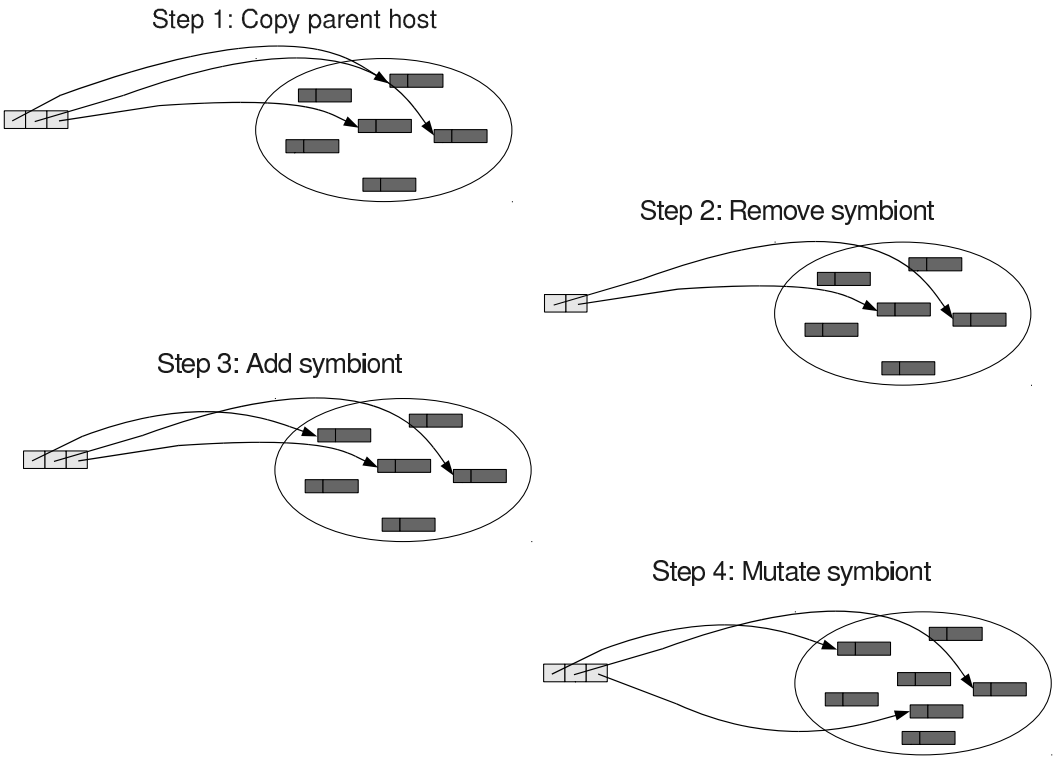


Figure 3.6: Example of the four steps in host generation. The offspring host is shown along with the symbiont population. After the first step, the offspring is a copy of the parent. In the second step, a symbiont is removed from the host, and in the third, an existing symbiont is added. The final step mutates a symbiont resulting in a new individual in the symbiont population, and leaving the symbiont originally selected for mutation unchanged.

Algorithm 3.3 The symbiont removal algorithm removes at least one individual if the host h_i contains 3 or more symbionts, thereafter the likelihood of removal decreases by a factor of $p_{sd} \in (0, 1)$. The function $rand(0, 1)$ returns a uniformly selected scalar in the unit interval.

1: **procedure** RemoveSymbiont(h_i)
2: $b = 1$
3: **while** $b > rand(0, 1)$ **and** h_i contains at least 3 symbionts **do**
4: remove uniformly selected symbiont from h_i
5: $b = b \times p_{sd}$
6: **end while**
7: **end procedure**

Algorithm 3.4 The symbiont addition algorithm adds at least one individual if the size of host h_i is less than ω, thereafter the likelihood of addition decreases by a factor of $p_{sa} \in (0, 1)$. The set S represents the contents of S^t at the start of the current generation t, while function $rand(0, 1)$ returns a uniformly selected scalar in the unit interval. If the selected symbiont already exists in the host, it is not added and no attempt is made at selecting an alternate.

```
1: procedure AddSymbiont(h_i, S)
2:     b = 1
3:     while b > rand(0, 1) and h_i contains fewer than ω symbionts do
4:         insert uniformly selected symbiont from S into h_i
5:         b = b × p_sa
6:     end while
7: end procedure
```

(since they may also appear in other hosts). A symbiont is mutated with probability $p_{sm} \in (0, 1)$. If selected for mutation, the bid component in the symbiont copy is mutated in an implementation-specific manner, Section 3.6, applying the operators repeatedly until a change in genotype is observed to occur. With probability $p_{am} \in (0, 1)$, the action of the symbiont copy is changed to a uniformly selected value, ensuring that all actions may appear in the host.

Algorithm 3.5 The host mutation algorithm. Each symbiont s_i in the host h_i is considered in turn and mutated with probability $p_{sm} \in (0, 1)$. The mutation process first copies the symbiont and updates the reference in the host to point to the copy. The bid and action components in the symbiont copy are then mutated in an implementation-specific manner. As before, the function $rand(0, 1)$ returns a uniformly selected scalar in the unit interval.

```
1: procedure MutateHost(h_i, S^t)
2:     repeat
3:         for all s_i ∈ h_i do
4:             if rand(0, 1) < p_sm then
5:                 copy s_i into s'_i
6:                 remove from h_i reference to s_i
7:                 add to h_i reference to s'_i
8:                 mutate bid and possibly action of s'_i
9:                 insert s'_i into S^t
10:            end if
11:        end for
12:    until at least one symbiont is mutated
13: end procedure
```

This process ensures that at least one symbiont in each offspring host is mutated resulting in the addition of at least H_{gap} new symbionts to the population S^t every generation. In addition, each new symbiont appears in just one host, though it is expected that if it is useful then it will spread to other hosts.

Since offspring hosts are initially copies of parent hosts, and symbiont mutation is applied to symbiont copies, host generation does not affect any of the parent hosts or available symbionts. Thus, any solutions carried over from the previous generation are not disrupted. Together, these steps represent the mechanism by which different symbiotic associations, i.e., compartmentalizations, are explored in the offspring hosts.

3.2.5 Evaluation

Evaluation, line 10 of Algorithm 3.1, determines the outcome $G(h_i, p_k)$ of applying each host $h_i \in H^t$ to each point $p_k \in P^t$. The outcomes are limited to the unit interval, and it is assumed without loss of generality that higher outcomes are better. Two cases are recognized, depending on whether the problem domain is classification or temporal sequence learning:

- Under classification, the outcome is simply defined as

$$G(h_i, p_k) = \begin{cases} 1 & \text{if host } h_i \text{ classifies point } p_k \text{ correctly,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

  The process by which a host classifies a point involves invoking the bid-based metaphor, Section 3.1, to predict the action associated with the highest bidder.

- Under temporal sequence learning, the outcome $G(h_i, p_k)$ represents the reward accumulated in applying host h_i to the initial world configuration defined by point p_k. In this case, the bid-based auction is invoked at each time step, allowing different symbionts to assume control at different stages in a given episode. The outcome (or reward) structure is problem-specific, and is therefore defined in the task descriptions in Chapters 6 and 8.

Here, it is also noted that whereas the outcome on a given point under classification will typically be binary, i.e., either a hit or a miss, under temporal sequence learning real-value outcomes are also likely, e.g., the distance to the goal location. Naturally, more information about the quality of solutions is expected under real-valued outcomes.

3.2.6 Removal of Points

The removal of points frees up resources so that offspring can be added to the point population in the subsequent generation, and as such, forms a basis for dynamic evaluation which is viewed as an integral part of the proposed system, Section 3.1. It can also be viewed as the step in which the influence of natural selection is most prominently effected in the point evolution. The goal of the point removal procedure is to favour points that support both unique and useful host behaviours while maintaining a diverse set of environmental conditions. With no explicit mechanism to encourage the latter, the point population can collapse to a small region of the state space, resulting in host behaviours that are nominally distinct but deviate only slightly from the same underlying behaviour [113, 125], i.e., analogous to the ensemble methodology discussed in Section 2.2.3. If this happens, the exploration of unseen regions of the state space may be severely limited as the search becomes trapped in a local optimum.

The point fitness used to determine which points to remove, line 13 of Algorithm 3.1, is calculated in two steps:

1. A *raw fitness* is calculated that measures the degree to which the point identifies useful and unique host behaviours.

2. The raw fitness is transformed into a *normalized fitness* by rewarding individual points that are genotypically unique.

The goal of the second step is thus to produce a diverse set of environmental conditions. The two factors are combined into a single scalar fitness which is then used to deterministically remove the least fit P_{gap} points.

Given that point fitness is based on two factors, the degree to which a point identifies useful and unique host behaviour as well as genotypic diversity, application of techniques from EMOO may be appropriate. However, this work recognizes that approaches based on the Pareto-dominance relation, Eq. 2.1, are not suitable, because it is unreasonable to maintain points in the population that provide no learning gradient simply because they are genotypically unique. Under the Pareto-dominance relation, these points would linger in the non-dominated set, wasting evaluation cycles without providing any useful information about host fitness.

Having outlined the process, the details of the calculation of the raw fitness and the normalized fitness are discussed in the following two sections.

Raw Fitness Calculation

The raw fitness calculation considers the case of binary and real-valued outcomes, Section 3.2.5, separately. The core steps of the calculation are directly applicable under the binary case, and without loss of generality, binary outcomes are assumed in the set $\{0, 1\}$. If the outcomes are real-valued, they are first discretized by considering the mean performance of all hosts on the point in question. The two cases are described below:

- When the outcomes are binary the raw fitness of a point p_k is based on the number c_k of hosts which achieve a positive outcome on the point, i.e.,

$$c_k = \sum_{h_i} G(h_i, p_k) \tag{3.2}$$

  where the sum iterates over all hosts h_i. The raw fitness F_k of point p_k is then defined as

$$F_k = \begin{cases} 1 + \frac{1-c_k}{H_{size}} & \text{if } c_k > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

  The term $\frac{1-c_k}{H_{size}}$, restricted to values of zero or less, becomes smaller as c_k increases, to a minimum value of $\frac{1-H_{size}}{H_{size}}$. This yields a minimum raw fitness of $\frac{1}{H_{size}}$ for the case of $c_k > 0$. The shape of this fitness function is illustrated in Figure 3.7.

- Under problem domains where the outcomes are real-valued, the count c_k is obtained by first calculating the arithmetic mean outcome μ_k on point p_k as

$$\mu_k = \frac{\sum_{h_i} G(h_i, p_k)}{H_{size}} \tag{3.4}$$

  where the summation iterates over all hosts h_i. If μ_k is effectively zero[4], then all hosts are viewed as having failed on p_k, and c_k is set to zero. Otherwise, c_k is determined as the number of hosts h_i such that $G(h_i, p_k) \geq \mu_k$, i.e., the number of hosts with outcome on p_k greater than or equal to the mean. Having established c_k, the raw fitness of the point is calculated as in Eq. 3.3.

This fitness function embodies several properties that are viewed as desirable:

[4]This was implemented using a threshold below which an outcome was considered to be zero.

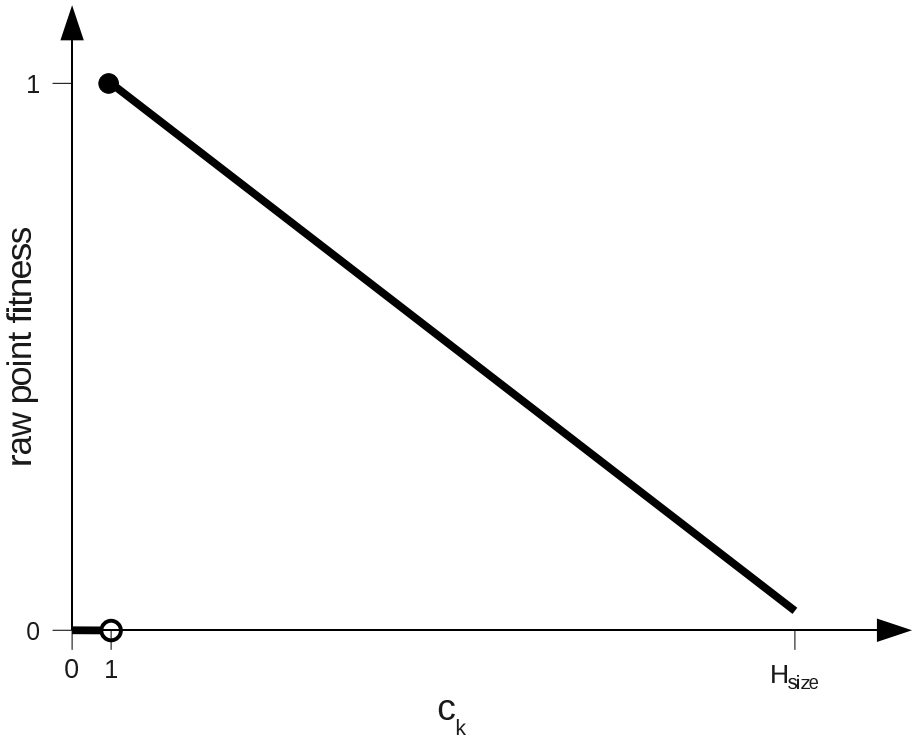


Figure 3.7: Shape of the raw point fitness function. Since c_k may assume values over $\{0, 1, \ldots, H_{size}\}$, the function is defined over this set only, but is drawn here over the entire range $[0, H_{size}]$. The function is maximized when c_k is one, and minimized when c_k is zero.

1. A point will receive a higher raw fitness if it identifies regions of the problem space where only a few hosts achieve high outcomes, i.e., c_k is low but not zero. If the point population is evolved to contain points with a high raw fitness, this corresponds to a host population where individual behaviours do not overlap, i.e., decompose the problem space, right side of Figure 1.1.

2. A point will receive a low fitness if many hosts do well on it and a few hosts do poorly on it. If the point is removed, the fitness of the hosts that do well on it will be adversely affected. However, since there are a lot of these hosts, it is less likely that all of them will be removed (compared to having just a few of these hosts), so the knowledge related to doing well on the point is likely to remain in the host population.

3. A point will receive a high fitness if most hosts do poorly on it and few hosts do well on it. The point is therefore more likely to persist in the population, supporting the few hosts that do well on it, and helping to sustain the associated behaviour that is learned.

4. Compared to distinction-based Pareto-coevolution, Section 2.4.1, which calculates distinction vectors whose dimension is squared in the number of individuals (hosts in this case), this raw fitness function is simpler and more efficient. In addition, the raw fitness function avoids difficulties associated with large non-dominated sets that require tie-breaking heuristics.

In summary, the raw fitness function used in point selection is meant to provide a simple way of supporting non-overlapping behaviours in the host population, right side of Figure 1.1. It is related to the idea of reducing parasite virulence in host-parasite[5] systems [23], however, the motivation behind the two approaches is different. Whereas virulence is meant to prevent disengagement by not driving the hosts to extinction, the proposed fitness function is meant to reward points that identify unique and useful host behaviours. However, as was previously observed under temporal sequence learning [113], encouraging non-overlapping behaviours without requiring genotypic diversity in the supporting points may not be sufficient, motivating the normalization step that follows. In particular, it is asserted that the normalization step is required under both the proposed as well as distinction-based point fitness.

[5]The 'host' in host-parasite has a different meaning from the term used throughout this thesis.

Normalization of Raw Fitness

Once the raw fitness is calculated, a measure of genotypic distance is used to reward points that represent unique locations in the problem space. The reward for each point is calculated based on the distance from the point to a subset of its nearest neighbours and is based on ideas from outlier detection [62]. First, the genotypic distance $d(p_k, p_l)$ between every pair of points p_k and p_l is calculated. This distance is then scaled to the unit interval by dividing the original value by the maximum distance over all pairs of points in the current population, yielding a normalized distance $d'(p_k, p_l)$ for each pair p_k and p_l. The normalized fitness for point p_k is then calculated as follows:

1. The set of K points nearest to p_k is calculated.

2. A reward factor r_k is calculated as

$$r_k = \left(\frac{\sum_{p_l} (d'(p_k, p_l))^2}{K} \right)^2 \tag{3.5}$$

   where the summation is taken over the set of K points p_l nearest to p_k. Each term in the summation in Eq. 3.5 is limited to the unit interval, and since there are K such terms, r_k is limited to the unit interval. The greater the distance between p_k and the points in K, the greater the reward factor.

3. The reward for point p_k is then used to normalize its raw fitness value as

$$F'_k = F_k \cdot r_k \tag{3.6}$$

   where F'_k is the final normalized fitness. Since r_k is limited to the unit interval, so is the normalized fitness F'_k.

Squaring distances results in a non-linear reward scheme as larger distances are weighted more, though alternate parameters could also be used to fine-tune the impact of the genotypic distances.

Once the normalized fitness F'_k for each point p_k is determined, the P_{gap} points with the lowest normalized fitness values are deterministically removed reflecting the elitist policy that is adopted.

One item omitted from the above description is the calculation of the distance $d(p_k, p_l)$ between two points p_k and p_l. In this work, the distance under classification problems was always assumed to be one. Since all points then receive the same

reward, this effectively causes the fitness normalization step to be bypassed. This was done because under classification a diversity-promoting balanced sampling heuristic is already in place and also because a meaningful distance measure is not immediately obvious. Under temporal sequence learning, the distance $d(p_k, p_l)$ is problem-specific and is defined in Chapters 6 and 8 for the truck reversal and Rubik's Cube tasks respectively.

3.2.7 Removal of Hosts and Symbionts

Analogous to the removal of points, Section 3.2.6, the goal of host removal step is to reduce the size of the host population by H_{gap}. This vacates room for H_{gap} new individuals to be added in the next generation, and represents what can be viewed as the culminating phase of natural selection. As in the removal of points, the host fitness calculation considers two factors, namely, host performance and the degree to which similar performance is observed in other hosts. The latter is included to encourage diversity in the host behaviours by discounting similar outcomes, but whereas in the case of the point population it was calculated with respect to the genotype, here it is calculated on the phenotype; the reason for this is that it is not obvious how to measure genotypic distance in GP-based individuals. However, given that pressure to establish niches based on genotypes is included in the point population, the goal of including the phenotypic diversity component in the host fitness is to encourage the emergence of a complementary set of species.

Host and symbiont removal, line 14 of Algorithm 3.1, takes into account host performance on the P_{size} points before point selection since using more points is viewed as more informative. The shared fitness S_i of host h_i, based on *competitive fitness sharing* [157], is calculated as

$$S_i = \sum_{p_k} \left(\frac{G(h_i, p_k)}{\sum_{h_j} G(h_j, p_k)} \right)^3 \tag{3.7}$$

where h_j iterates over all hosts in the current population H^t, and where p_k iterates over all points in P^t on which at least one host in the current population received a non-zero reward. For each such point p_k, the shared score S_i considers the reward that h_i receives on p_k, weighting it by the aggregate reward on p_k across all hosts. If more hosts receive a high reward on p_k, then the weight of the associated term tends to be lower. In this way, hosts are rewarded more for doing well on points where few other hosts perform well.

Once the shared score S_i for each host h_i is calculated, the H_{gap} hosts with the lowest S_i values are deterministically removed reflecting the elitist policy that is adopted.

Following host removal, individuals may exist in the symbiont population that are not referenced by any hosts; symbiont removal involves removing these individuals. This is consistent with the top-down nature of the variation operators in the proposed approach, i.e., care is taken not to disrupt existing hosts, which, once introduced into the population, persist unchanged until they are removed or the algorithm terminates. This process is also consistent with the idea that natural selection is a group operator applied to hosts and not to the symbionts inside a host. In contrast to the SANE framework [132], which requires an explicit symbiont fitness, this also results in an algorithm with fewer design decisions.

Viewing each point as an objective, the Pareto-dominance relation, Section 2.4.1, could have been used to select hosts instead of the competitive fitness sharing score. This was not done for two reasons. First, the number of points in the point population results in too many objectives for the Pareto-dominance relation to be useful. In particular, with objectives typically numbering in the hundreds, the non-dominated set would likely contain most if not all of the hosts [51, 80]. Second, the Pareto-dominance relation has been found to result in a high degree of overlap in the solution behaviour [125], left side of Figure 1.1, whereas the goal of this work is to evolve diverse inter-host behaviour.

3.3 Additional Algorithm Details

For clarity, a number of algorithm details were omitted from the description of the core SBB algorithm in Section 3.2. These are not related to the key principles underlying the proposed approach, but are viewed as nonetheless useful for improving performance, i.e., they can be viewed as 'tweaks'. For completeness, these details are reviewed in the following sections.

3.3.1 Pruning Symbionts from Hosts

At the end of each generation, a pruning procedure is triggered which may remove from hosts references to symbionts deemed as inactive. An inactive reference is defined as a reference to a symbiont which has never won a bidding auction in a given host. When a host is created, all references to symbionts in the host are marked as inactive. If, during the bid-based action selection process, a winning symbiont is found in a host

and the reference to that winning symbiont is still marked as inactive, that reference is marked as active. The change from inactive to active is local to the host, so it is possible that a reference in one host is marked as active but a reference to the same symbiont in another host is marked as inactive. Once marked as active, a reference remains active for the lifetime of the host.

If the generation is the last generation, the pruning procedure is applied to all hosts in the host population. If the generation is not the last generation, then the decision on whether or not to prune each host depends on its size, i.e., the number of symbionts that the host contains. In this case, a host is only pruned if its size equals the maximum host size ω. This represents a compromise between pruning all hosts each generation (regardless of size), and pruning hosts only at the end of the last generation, allowing genetic material in the form of inactive symbionts to develop over multiple generations while occasionally reducing the overhead of program execution. It effectively allows symbionts to mature, allocating resources to enable individuals to develop to the point where they can provide a meaningful contribution to the host. Given that there is a limit to the number of symbionts that may appear in a host, this also frees up space to allow external symbionts to enter the compartment. As an alternative approach, an additional age parameter could be introduced and used to determine when to truncate non-effective members.

The pruning procedure itself first calculates the set of inactive symbionts in the host. The symbionts in this set are then selected with uniform probability and removed from the host as long as the host size remains above the threshold of two.

3.3.2 Generating Unique Bidding Behaviours

When a new symbiont is created during host initialization and host generation, lines 4 and 7 of Algorithm 3.1, its bid program is mutated as long as it duplicates the bidding behaviour of a program in an existing symbiont in S^t. Two programs are considered duplicates if their bid profiles are the same, where a bid profile is a vector of bid values across a fixed set of ϕ points. This set of points is initialized at the beginning of each training run using the same process as in the point initialization step, line 3 of Algorithm 3.1, e.g., under classification this involves class-balanced uniform selection. As long as *all* the bids of the new symbiont are within 10^{-4} of the corresponding bids of an existing symbiont, the bid program of the new symbiont is mutated as in the host generation step, line 7 of Algorithm 3.1, by applying implementation-specific variation operators.

3.3.3 Mixing Symbionts during Initialization

The host initialization procedure, Section 3.2.2, consists of two steps. After the first step, all hosts contain two symbionts and each symbiont appears in exactly one host. The goal of the second step is to mix the hosts up, exploring hosts of varying size and symbionts in multiple contexts. To achieve this, symbionts are added to each host until its size reaches a limit uniformly chosen from $\{2, \ldots, \omega\}$, where each symbiont to be added is chosen as follows. First, τ_{init} possibly duplicate symbionts that do not already appear in the host are selected with uniform probability. From these, the symbiont with the fewest references, i.e., appearing in the fewest hosts, is selected and added to the host. In this way, the heuristic is meant to place each symbiont in roughly the same number of hosts, where this is significant given the policy for removing symbionts based on their reference count.

3.4 Hierarchical Construction through Symbiosis

The bid-based action selection metaphor can be applied iteratively to produce hierarchies of hosts. Naturally, hierarchies are useful if lower-level components represent different behaviours, therefore, the proposed construction process is complemented by the inclusion of a diversity component in the host fitness function, Section 3.2.7. Under this approach, hosts at the first level consist of symbionts whose actions are defined by the problem, e.g., class labels in classification. Hosts at successive levels consist of symbionts whose actions reference hosts at the level directly below, Figure 3.8. Action selection is then performed as follows. If the host is at the first level $h = 0$, the action of the highest bidder is returned as before. If the host is at some higher level $h > 0$, then the highest bidder in the host is determined, and the action selection procedure is applied to the host at level $h - 1$ referenced by this highest bidder. The action of the highest bidder at level $h > 0$ cannot be returned directly since it has no meaning in the context of the problem.

The host levels are trained as follows. First, level 0 is trained. Once training of level 0 is complete, the final host population at that level is fixed and is used to form the action set for training level 1; thus, the level 0 symbiont population is also recorded. Level 1 is then trained, and the final level 1 host population is used to form the action set for level 2. This continues until the desired number of levels are trained. Algorithm 3.1 is applied at all levels without modification – the only property that changes is the definition of the action set. At the first level, the action set is defined a

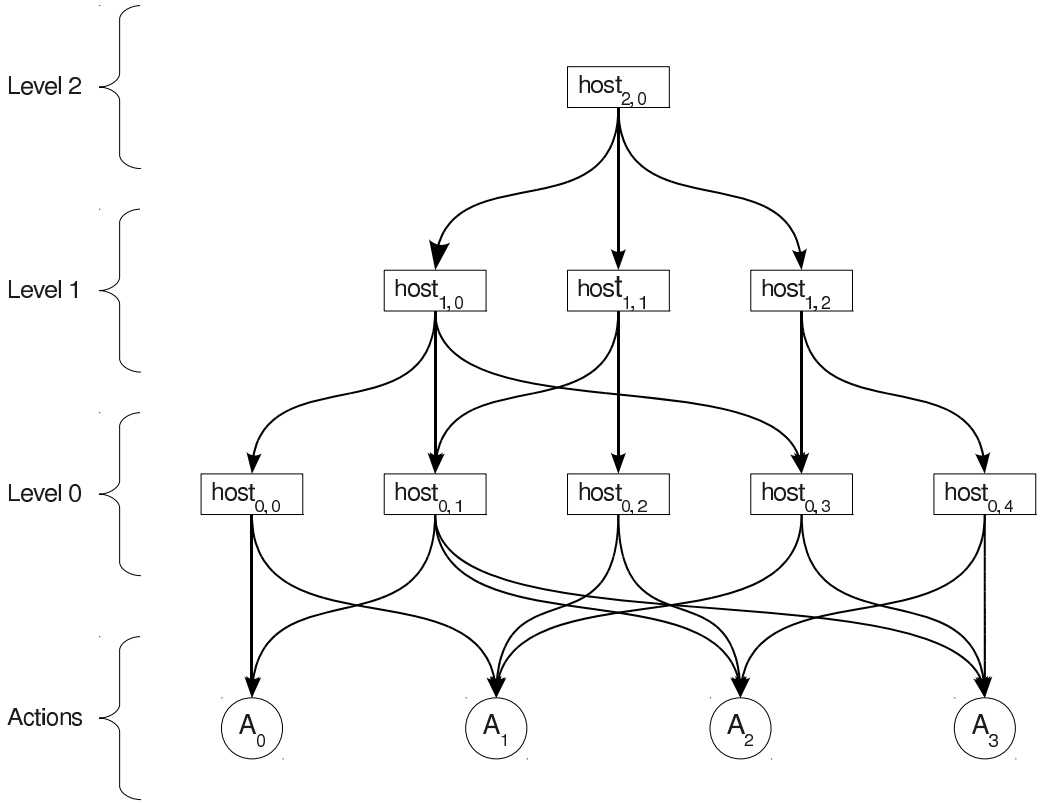


Figure 3.8: Example of a three-level host-of-hosts. The level 0 hosts access actions A_0 through A_3 defined by the problem. The level 1 hosts reference level 0 hosts, and level 2 hosts reference level 1 hosts. Just as multiple hosts at level 0 can reference the same problem-specific action, different hosts at level h can reference the same host at level $h-1$ multiple times. For example, hosts $host_{0,1}$ and $host_{0,3}$ are reference twice by level 1 hosts.

priori by the problem domain. Thereafter, actions become increasingly more abstract as defined by the behaviours identified during the evolution of each level. It is also emphasized that through this process the bidding behaviours, i.e., the programs, are evolved at each level independently.

The evolution of abstract actions as described above has several implications. First, the description of the proposed approach thus far has always suggested that the action associated with the bid, Figure 3.1, is always problem-specific. Under the hierarchical construction process, strictly speaking, this is no longer the case except at the bottom level. Second, whereas the symbionts at the lowest level were previously described as heterogeneous, Section 2.2.3, symbionts at the higher levels may now be homogeneous as they have the capacity to represent complete solutions. Finally, this also means that at higher levels the degree to which implicit fitness sharing occurs, Section 2.3.5, may be limited.

As the host-of-hosts is applied to a problem instance, the action selection process traverses all levels of the hierarchy at each time step. Under problems requiring one time step, such as classification, a host at level $h > 0$ can be viewed as handing control over to the host at level $h - 1$, since once the lower-level host is selected the host at level h is not considered again in the episode. If there are two levels in total, then the level 1 host can be viewed as choosing which level 0 host to apply when. This can be useful in combining multiple level 0 hosts into a single solution, e.g., the three models in the right side of Figure 1.1.

Under temporal sequence learning, where each point population instance represents an episode likely involving multiple time steps, the behaviour of the host-of-hosts is more complex. At different time steps, different paths through the hierarchy may be taken, and different level 0 hosts may be used. This may result in situations where a single level 0 host is applied for one part of the episode, then control switches to another level 0 one host, and so on.

The hierarchical host construction in this work was meant as a proof-of-concept, and different design options could be implemented as described below:

1. The action set at level h consists of all hosts at level $h - 1$. Pruning could be applied in order to eliminate some of the lower-level hosts from consideration at higher levels, though deciding which hosts to prune may not always be clear.

2. The same fitness function is used at all levels, whereas altering the fitness function between levels may be more effective. For example, lower levels could focus more on diversity, with the goal of establishing a set of building blocks,

and higher levels could focus more on overall performance with the goal of combining the building blocks into a single solution.

3. It may be possible to train all levels of the hierarchy simultaneously, however, fixing all but the topmost level is viewed as a simpler first step. For example, the optimal number of levels to be constructed may not be known *a priori*; by adding levels incrementally a suitable stopping condition may be more easily identified, e.g., when the topmost level does not improve on the previous level.

3.5 Nature of the Host-Point Relationship

Under classification[6], three observations can be made regarding the host-point relationship that results from the respective fitness functions used, Sections 3.2.6 and 3.2.7:

1. If one host receives a reward of one on all points and all other hosts receive a reward of zero on all points, then the shared score value for the host will be the maximum possible. At the same time, the raw fitness for all points will also be the maximum possible.

2. If, for each point, only a single host receives a reward of one and all other hosts receive a reward of zero on that point, then the raw fitness for each point will still be the maximum possible. The fitness of a host will be equal to the count of the number of points that it classifies correctly. This refers to the situation where every point is classified correctly by exactly one host, though that host does not have to be the same across all points.

3. Assuming that the point is classified correctly by at least one host, the raw fitness of a point decreases as more hosts classify that point correctly. Compared to the situation in observation 2 where each point is classified correctly by exactly one host, if more hosts classify the same point correctly, the host fitness values will also tend to decrease.

The first observation illustrates that maximum outcomes may occur in the host and point populations simultaneously, i.e., a higher outcome in one population does not imply a lower outcome in another. The second observation illustrates the case where the point fitness remains fixed, but the fitness of a given host may vary depending

[6]This discussion is restricted to classification where the outcomes are binary for simplicity.

on the manner in which the hosts partition the points into sets solvable by each host. The final observation suggests that the key factor in the fitness of members of both populations is the overlap in host behaviour.

Characterizing the point-host interaction as cooperative, competitive, or symbiotic is therefore not a straightforward task. The reason for this is that point and host fitness is not determined by having a single point interact with a single host in isolation. Instead, the fitness of points depends on the aggregate performance of the host population, while fitness of a host depends on its performance and how that performance overlaps with other hosts. In both cases, more overlap in host performance is associated with lower fitness values.

3.6 GP Implementation

The bid-based approach described in this chapter does not rely on a specific form of GP. This work used linear GP [20], in which instructions are encoded using fixed-length bit strings, to evolve the bidding behaviour. Specifically, each instruction consisted of four fields:

1. The *destination* field specified a register index x where the first operand was located and where the result of the operation was to be stored.

2. The *source* field specified an index y where the second operand was located. This index could refer to either a register or an input depending on the instruction format.

3. The *mode* field specified the instruction format, register-register or register-input. In the former y is interpreted as a register index, and in the latter y is interpreted as an input index.

4. The *operation* field specified the function to be applied to the values indexed by x and y.

A program was then represented as a linear sequence of instructions operating on a set of inputs (as defined by the problem) and *numRegisters* registers. The parameters associated with the GP implementation are listed in Table 3.2.

New programs were created by first selecting with uniform probability a program size, in terms of the number of instructions, in $\{1, 2, \ldots, maxProgSize\}$. Each bit in the new program was set to a random value selected with uniform probability. To

Parameter	Description
$numRegisters$	Number of registers
$maxProgSize$	Maximum program size
p_{delete}	Probability of instruction deletion
p_{add}	Probability of instruction addition
p_{mutate}	Probability of flipping a single bit
p_{swap}	Probability of swapping two instructions

Table 3.2: Parameters specific to the GP implementation.

generate a new program from an existing one (symbiont mutation step during host generation, line 7 of Algorithm 3.1), the following operations could then be applied stochastically in sequence:

1. **Instruction deletion.** If the program contains more than one instruction, with probability p_{delete} a uniformly selected instruction is removed.

2. **Instruction addition.** If the program contains fewer than $maxProgSize$ instructions, with probability p_{add} a new instruction is inserted into the program. The insertion location and the bit values of the new instruction are selected with uniform probability.

3. **Instruction mutation.** With probability p_{mutate}, the value of a uniformly selected bit in a uniformly selected instruction is flipped.

4. **Instruction swap.** With probability p_{swap}, two different instructions in the program exchange locations. The two instruction are selected with uniform probability.

The deletion and addition operations allow the new programs to increase or decrease in size compared to the original programs. The mutation operator affects a single bit and may therefore introduce small changes to the overall program, e.g., change one of the registers, whereas the instruction swap operation is included to address situations where the right instructions are present in the program but in the wrong order. It is also noted that during symbiont mutation all four operations are iteratively reapplied until a change in the genotype is triggered.

Prior to program execution, all register contents were initialized to zero. The GP function set, Table 3.3, consisted of eight operations. If the operation was unary, it was applied to the source operand indexed by y, and the result was stored in the register at index x. The same function set was used in all experiments, though

Function	Definition
Addition	$R[x] \leftarrow R[x] + R[y]$
Subtraction	$R[x] \leftarrow R[x] - R[y]$
Multiplication	$R[x] \leftarrow R[x] \times R[y]$
Division	$R[x] \leftarrow R[x] \div R[y]$
Cosine	$R[x] \leftarrow cos(R[y])$
Logarithm	$R[x] \leftarrow \ln \lvert R[y] \rvert$
Exponential	$R[x] \leftarrow e^{R[y]}$
Conditional	if $R[x] < R[y]$ then $R[x] \leftarrow -R[x]$

Table 3.3: GP function set used expressed in register-register mode, i.e., $R[x]$ refers to the contents of the register at index x. Register-input mode is analogous but y indexes an input. The conditional function, a special case, reverses the sign of the value in register $R[x]$.

common practice in GP is to use problem-specific functions [95]. If the result of the operation was undefined, e.g., division by zero, then zero was stored in the destination register.

Program output was defined as the value retained in register zero following execution. As such, there could potentially be a very wide range of bid values that a program could produce. In order to encourage individuals to compete over a common range of bids the Sigmoid function is employed, Figure 3.9. This resulted in the mapping of the raw program outputs to bid values over the unit interval.

In GP, introns can be bypassed during program execution to reduce the computational overhead of the fitness evaluation. They should not, however, be removed from the programs since the interaction between representation, search operators, and cost function may be complex thus an ineffective instruction may nonetheless turn out to be useful in the long run. A modified version of Algorithm 3.6 was used in this work to find structural introns. Algorithm 3.6 maintains a set of introns found so far, I, and a set of target register indices T representing variables relevant to the output of the program. It initializes the set of target registers T to contain the output register 0, line 3, and then considers instructions from last to first, line 4. If an instruction is found that affects one of the target registers, line 6, that instruction is effective. If an effective instruction is in register-register mode, line 7, the algorithm makes sure that the source register is also added to T. Instructions that do not affect registers in the target set T are marked as introns, line 12, and are skipped during program execution.

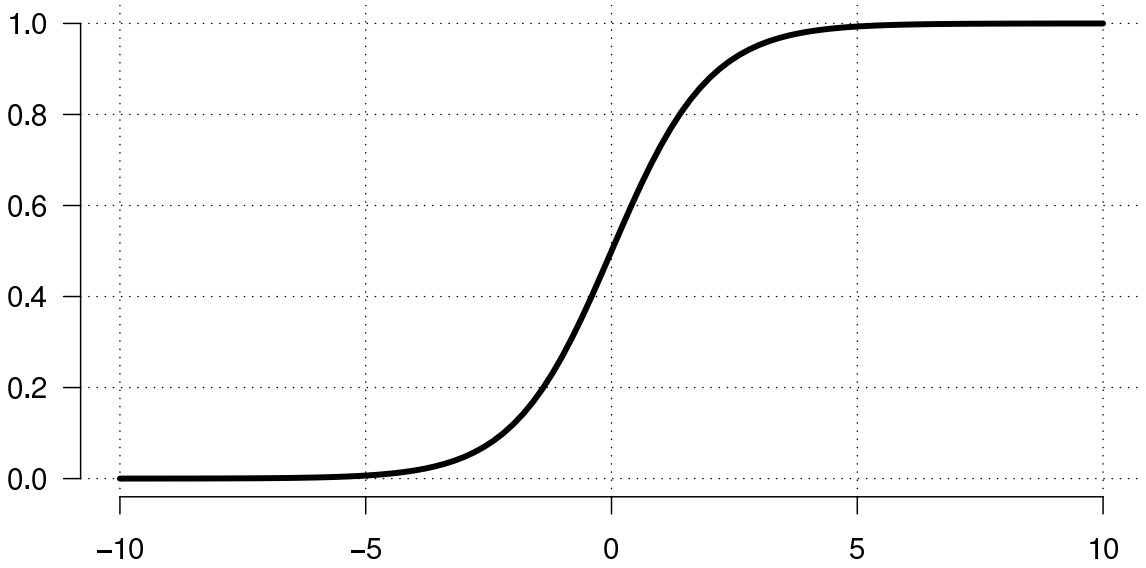


Figure 3.9: Sigmoid function $Sigmoid(z) = (1 + e^{-z})^{-1}$ over the range [-10, 10], z on the horizontal axis. The function can be viewed as consisting of three regions: low ($z < -5$), transition ($-5 \leq z \leq 5$), and high ($z > 5$).

Algorithm 3.6 represents a generic intron identification procedure [18]. In this work, it was modified as follows to account for the instruction format and the unary operations, Table 3.3. If a unary operation was applied in an instruction where the destination was in T, then the instruction would be marked as effective, however, the destination register would be removed from T. The source, if it was a register, was then added to T as before, i.e., the removal from T occurred after line 6 and before line 7 of the specification in Algorithm 3.6. This modification results in the identification of a larger number of introns. A trivial example of where this is useful is when the last instruction in the program applies a unary operation to an input and stores the result in register 0, in which case, all previous instructions are rightly identified as introns. The last instruction is the only one affecting the program output, however, Algorithm 3.6 as stated would proceed backwards through the program possibly marking additional instructions as effective.

Algorithm 3.6 Algorithm for identifying structural introns, assuming the output is extracted from register 0. The variable ip is an instruction pointer, $*ip$ denotes an instruction, and the program size is N. The procedures in lines 5 and 8 are used to extract the destination and source registers respectively, while line 7 tests if the instruction format is register-register. The introns are recorded in set I.

```
 1: procedure FindIntrons
 2:     I = ∅                                          ▷ Set of introns.
 3:     T = {0}                                        ▷ Set of target registers.
 4:     for ip = N − 1 to 0 do
 5:         x = Destination(∗ip)
 6:         if x ∈ T then                              ▷ Instruction ∗ip is effective.
 7:             if IsRegisterRegister(∗ip) then
 8:                 y = Source(∗ip)
 9:                 T = T ⋃ {y}
10:             end if
11:         else                                       ▷ Instruction ∗ip is an intron.
12:             I = I ⋃ {ip}
13:         end if
14:     end for
15: end procedure
```

Chapter 4

Comparison with Monolithic Genetic Programming on Binary Classification Problems

This chapter investigates three claims regarding the SBB framework in the context of a comparable monolithic approach:

- The first claim is that the solutions produced using the symbiotic[1] approach will yield improved classification performance.
- The second claim is that the monolithic solutions will be more complex in terms of instruction counts as well as the number of features accessed.
- The final claim is that evolving the SBB solutions require less computational overhead, where this may be a function of solution complexity.

Following a description of the experimental setup, Section 4.1, the three claims are tested empirically in Section 4.2.1.

Specifically, the symbiotic approach is compared with a monolithic approach on a set of four binary classification problems. The implementation of the monolithic approach is based on the SBB implementation, however, any elements associated with symbiosis are removed, e.g., the bid-based metaphor is no longer applied. Furthermore, the same parameters, including the limit on maximum solution complexity, is used throughout. Thus, with a high degree of confidence, significant differences in classification performance, solution complexity, or training overhead can be attributed to the use of the symbiotic metaphor.

In addition to the comparison of the symbiotic and monolithic approaches, this chapter presents a characterization of the SBB solutions with respect to host composition and behaviour, Section 4.2.2. The datasets forming the evaluation testbed include large training partitions and high feature counts, and as such, the ability to decompose the instance/feature space is expected to prove beneficial. An analysis of the inter-host problem decomposition, Section 4.2.3, motivates the hierarchical

[1]The terms 'SBB' and 'symbiotic' will be used interchangeably.

construction of solutions through symbiosis as described Section 3.4. The chapter concludes with a summary of the key results in Section 4.3.

The goal of this chapter is therefore to provide a comprehensive overview of the solutions evolved using the SBB approach. Given that the problem domain considered is binary classification, a fair comparison with the monolithic approach is facilitated. Furthermore, performing an analysis with respect to two actions is meant to make the results easier to digest and any ensuing conclusions more evident (multi-class problems are considered in the following chapter).

4.1 Experimental Setup

The implementation of the monolithic approach uses the SBB codebase and is described in more detail in Section 4.1.1. The datasets, parameters, and evaluation criteria are then summarized respectively in Sections 4.1.2, 4.1.3, and 4.1.4.

4.1.1 Implementation of the Monolithic Approach

To implement the monolithic approach, the code for the SBB approach as described in Chapter 3 was modified as follows. First, the minimum and maximum host sizes were both set to one so that a host always contained just a single symbiont[2]. A side-effect of this was that symbiont mixing did not occur during host initialization, line 4 of Algorithm 3.1, and that the symbiont removal and addition steps during host generation, line 7 of Algorithm 3.1, were never executed since the restrictions on host sizes were never met.

Since a solution now consisted of a single symbiont, the action selection procedure had to be modified. This was done by disregarding the action associated with the symbiont, and basing the decision on the symbiont's bid value alone. Specifically, if the bid value was less than 0.5, then the output was the first action, otherwise the output was the second action. As a result, the monolithic approach was restricted to problems with two actions, e.g., binary classification problems, where this is in line with canonical approaches to binary classification using GP [95].

Apart from the modifications described above, no other changes were made. In particular, the same GP implementation, Section 3.6, including the function set, was used to evolve the programs. Furthermore, coevolution between hosts and points still formed the basis for the dynamic evaluation of fitness.

[2]The terms 'host' and 'symbiont', which would normally have no association with the monolithic approach, are used since the description is with respect to the SBB implementation.

Dataset	Features	Training cases		Test cases	
		Total	Majority	Total	Majority
Census (cen)	40	199 523	187 141	99 762	93 576
Gisette (gis)	5 000	6 000	3 000	1 000	500
Bupa (bpa)	6	310	180	35	20
Pima (pma)	8	691	450	77	50

Table 4.1: Datasets used to compare the SBB and monolithic approaches. For Bupa and Pima, 10-fold cross-validation was used so the instance counts represent the mode values across all folds. The contents in parentheses following the dataset name denote the abbreviation used to refer to the dataset.

The decision to base the implementation of the monolithic approach on the code for the symbiotic approach was made so that, apart from the change in the action selection procedure, any other necessary changes would be as minimal as possible. The change in the action selection process is a result of the shift from a symbiotic to monolithic representation – the goal of this chapter is to investigate the effect of this shift.

4.1.2 Datasets

Four datasets, Table 4.1, were used to compare the SBB algorithm with the monolithic approach. Only binary problems were considered since the action selection procedure in the monolithic approach makes it directly applicable to two-class problems and because multi-class domains would complicate analysis of the results. All the datasets were obtained from the UCI Machine Learning Repository [3]. Furthermore, all were represented using a direct encoding, i.e., for each exemplar all the features were explicitly specified and subsequently read into memory when that exemplar was accessed, where this may be of significance if a dataset contains many attributes and is sparse.

Census refers to the KDD Census Income dataset with nominal attributes mapped to integer values; in this work, the training and testing partitions used were as they appear in the original distribution [3]. The Gisette dataset was designed for the 2003 Neural Information Processing Systems (NIPS) feature selection competition [60] in which only filter methods were applied to the dataset, i.e., no embedded approaches, such as GP, were submitted. It is sparse, with only thirteen percent of feature instances representing non-zero values, and half of its 5000 features are

Dataset	Action 0	Action 1
Census (cen)	† 0	1
Gisette (gis)	-1	1
Bupa (bpa)	1	† 2
Pima (pma)	† 0	1

Table 4.2: Action to class label mapping used in the experiments. The majority class label is marked with a †.

redundant in the sense that they are linear transformations of the other features. The training partition used in this work is the original training partition, while the test partition corresponds to what was originally the validation set[3].

The Census dataset was selected for this evaluation because it is large and unbalanced (the proportion of instance belonging to the majority class is 0.938) while Gisette was included because it contains a high number (5000) of features, Table 4.1. In both cases, useful problem decomposition is expected under the SBB approach, e.g., under Gisette different symbionts in the same host may associate themselves with different subsets of features. Here, it is noted that datasets as large as Census are not frequently encountered in GP benchmarking studies unless use is made of specialized hardware, exceptions being subset selection approaches [30, 111, 123, 168], while feature counts as high as those observed under Gisette have only recently been encountered [38].

Two other datasets that were used were the Bupa Liver Disorders dataset and the Pima Indian Diabetes dataset. Both datasets are smaller and so stratified 10-fold cross validation was used in the evaluation; the instance counts, Table 4.1, represent the most common values across all folds. Though not as large as Census and Gisette[4], both of these datasets are known to be difficult across a broad class of learning algorithms, with error rates between 0.28 and 0.43 observed on Bupa and error rates between 0.22 and 0.31 observed on Pima [116]. In addition, the inclusion of these datasets allows evaluation of the relative effectiveness of the point sampling heuristic on problems with fewer training instances.

Regardless of the problem, possible actions were restricted to a contiguous range of integers starting with 0. To address the inconsistency of class labels as used across different datasets, an action to class label mapping was constructed as follows. First,

[3]Access to the original test partition is restricted.

[4]Though still of moderate size as far as GP benchmarking is concerned.

the labels were sorted into ascending order. The action associated with each label was then defined as the index of the label in the sorted list assuming a starting index of 0, e.g., labels 1, 5, and 4 would be mapped to actions 0, 2, and 1 respectively. The action to class mapping used in the experiments is shown in Table 4.2.

4.1.3 Parameters

The SBB and GP parameter settings for the symbiotic approach are listed in Tables 4.3 and 4.4 respectively. Familiarity established through previous applications of the SBB algorithm [38, 111, 112] provided the basis for the settings adopted here, though no claims regarding optimality are made. The parameter settings under the symbiotic and monolithic approaches were identical with the exception of the maximum program size, $maxProgSize$. Since the hosts could consist of up to 10 symbionts/programs each with 48 instructions, while the monolithic solutions contained just a single program, the maximum program size in the monolithic approach was set to 480. As such, the upper limit on solution complexity in terms of the number of instructions was the same under both approaches; consequently, so was the upper limit with respect to the feature access counts.

While the parameter values in Table 4.3 were established through experience, some general insights were also developed. It was found that performance typically did not degrade with higher values for t_{max}, P_{size}, and H_{size}, possibly due to the elitist selection policies adopted, so the primary factor considered when setting these parameters was the amount of computational overhead that could be tolerated. The parameters P_{gap} and H_{gap}, relative respectively to P_{size} and H_{size}, reflect one of the main tradeoffs in exploration versus exploitation in the point and host populations. In particular, higher gap parameters suggest a more significant bias towards exploration of new individuals and less emphasis on exploitation of existing genetic material. It was found that, compared to the host population gap parameter, a lower point population gap parameter typically resulted in improved performance. Therefore, whereas H_{size} was set equal to P_{size}, H_{gap} was set at three times the value of P_{gap}. A possible reason for this observed behaviour may be that a higher P_{gap} value, supporting an increased rate of change in the point population, may not allow sufficient time for the host population to adapt to each influx of new points. At the same time, it was found that a relatively high H_{gap} parameter was often more effective, perhaps because it allows more host configurations to be explored. Given that the action set under classification contains two elements, the setting of ω allows some degree of redundancy in the sense

Parameter	Value
t_{max}	1000
P_{size}	120
P_{gap}	20
H_{size}	120
H_{gap}	60
ω	10
p_{genp}	–
p_{sd}	0.7
p_{sa}	0.7
p_{sm}	0.2
p_{am}	0.1
K	–
ϕ	50
τ_{init}	10

Table 4.3: SBB model parameters used in the comparison with the monolithic approach. The number of nearest-neighbours used in the point fitness calculation, K, is not applicable in classification problems since the distance between points is always one. The probability of generating an offspring point from a parent point, p_{genp}, is also not applicable under classification since a balanced sampling heuristic is always used. Parameter description summaries are provided in Table 3.1.

that multiple symbionts in the same host can be associated with the same action. Naturally, problems with more actions may require this value to be increased, e.g., the Rubik's Cube environment presented in Chapter 8. It was also found that the probabilities of applying the variation operators could be set relatively high, where the policy not to disrupt existing solution under an elitist selection strategy may help with the preservation of strong solutions in the population. The exception was the probability of action mutation, p_{am}, which was set at a relatively low value since the operator is viewed more as a safeguard to make sure that all actions would have a chance of appearing in the symbiont population[5].

4.1.4 Evaluation Methodology

Three claims regarding the monolithic and symbiotic approaches were made at the beginning of this chapter. Two of the claims relate to the classification performance

[5]Comparing probabilities of applying different variation operators will always be difficult given that the manner in which the operators are implemented varies greatly.

Parameter	Value
$numRegisters$	8
$maxProgSize$	48
p_{delete}	0.5
p_{add}	0.5
p_{mutate}	1.0
p_{swap}	1.0

Table 4.4: GP parameters used in the comparison with the monolithic approach. Parameter description summaries are provided in Table 3.2.

and the complexity of the solutions evolved under the two approaches. This multi-objective comparison is motivated by the view that if one cannot gain insight from the solutions post training, as is more likely if complexity is high, then their usefulness is limited. In addition, the training overhead incurred under the two approaches is also considered, where this may depend in part on the complexity of the individuals being evolved. By assessing solutions in terms of different factors, including two metrics quantifying classification performance, this methodology reflects recent calls for broader forms of evaluation in ML [83, 167]. The three criteria used in the evaluation, classification performance, complexity, and training overhead, are described below:

- **Classification performance.** The classification performance of a solution was measured in two ways. The first was *accuracy* defined for model i as

$$accuracy_i = \frac{1}{|E|} \sum_{e \in E} hit(i, e) \tag{4.1}$$

  where E is the set of problem exemplars over which performance is measured and $hit(i, e)$ returns 1 if model i labels e correctly and 0 otherwise. The classification accuracy, representing the fraction of instances labeled correctly, is commonly used to evaluate models in classification. However, it is not informative on problems where the label counts are unbalanced [83]. For example, if 98% of instances belong to class 0 and 2% belong to class 1, then a model could achieve an accuracy of 98% by labeling all instances as belonging to class 0 – this despite capturing none of the characteristics underlying the differences between the two classes.

  To provide an informative measure of classification performance on unbalanced datasets, a *multi-class detection rate* (mcdr) with respect to model i was defined

as

$$mcdr_i = \frac{1}{|L|} \sum_{l \in L} detection(i, l) \tag{4.2}$$

where L is the set of class labels, and $detection(i, l)$ is the detection rate of model i on class l, i.e., the fraction of all instances whose true label is l that were also assigned a label of l by the model. Thus, the quantity defined in Eq. 4.2 effectively considers the detection rate across all classes. Under binary problems, it is equivalent to combining sensitivity and specificity as in the balanced error metric used in the 2003 NIPS competition [60], though unlike the latter the mcdr also scales to problems with more than two classes. It associates degenerate behaviour with a value of $\frac{1}{|L|}$, e.g., on binary datasets, an mcdr value of 0.5 flags the type of degenerate behaviour described in the above example.

- **Solution complexity.** A meaningful complexity comparison between the symbiotic and monolithic approaches is possible because their underlying representation, linear GP, is the same. Specifically, complexity is considered in terms of the number of features accessed and the number of effective instructions, where both counts are obtained post intron removal. Unless otherwise noted, duplicate accesses to the same feature are not counted. Under the symbiotic approach, the complexities can be expressed with respect to both the symbionts and the hosts. In the latter case, instruction counts are taken over the bid programs of all symbionts in the host, and the feature access counts correspond to the set of unique features accessed by the symbionts in the host. The parameters used, Section 4.1.3, results in the same limit on the maximum solution complexity under both approaches.

- **Training overhead.** To evaluate the training overhead required under the monolithic and symbiotic approaches, process running times were measured using the *C getrusage* function and account for instructions executed in user mode. As such, these running times can be used to provide an intuitive measure of training efficiency – this can be significant given that training efficiency is often viewed as a major drawback of using GP [5]. It is noted that introns are bypassed during program execution, care of the intron identification algorithm described in Section 3.6, and their effect on the running times is therefore limited.

For each training partition, 60 initializations using different random seeds were performed. Thus, under Census and Gisette, the total number of initializations was

60, while under Bupa and Pima, the number of initializations was 600 as a result of using 10-fold cross validation. For each initialization, unless otherwise noted, the results presented correspond to the solution in the population at the end of training that yielded the highest mcdr on the entire training dataset. Therefore, when a reference to a host is made, it is implied that the host represents a single initialization. Unless otherwise noted, all of the presented classification performance results are with respect to the test data.

4.2 Results

The section following immediately compares the symbiotic and monolithic approaches with respect to classification performance, solution complexity, and training overhead. A further characterization of the symbiotic solutions, in terms of structure and behaviour, is then presented in Section 4.2.2. Finally, the population-wide performance under the symbiotic approach is analyzed in Section 4.2.3.

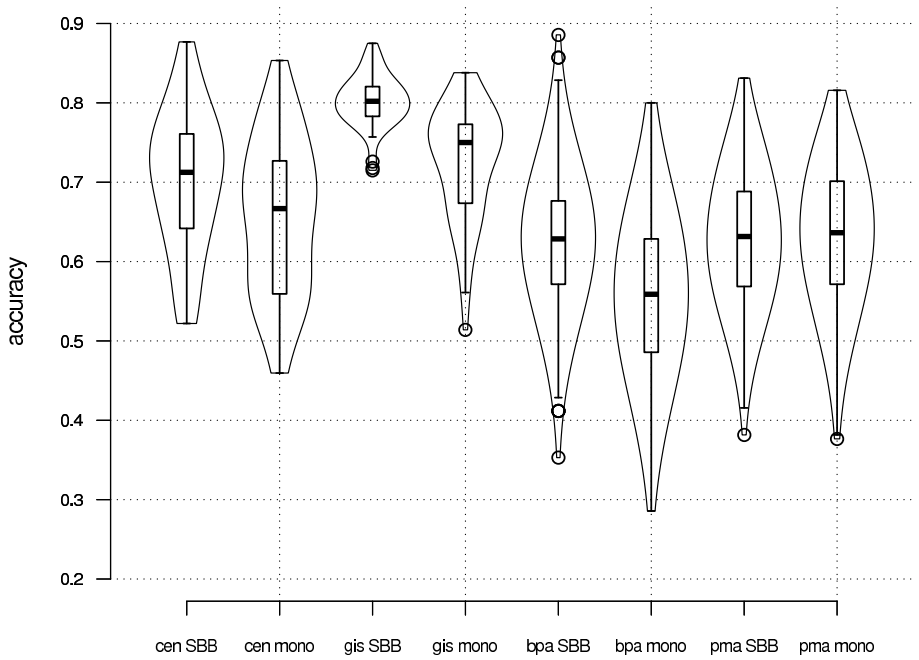
4.2.1 Comparison of Symbiotic and Monolithic Approaches

Classification Performance

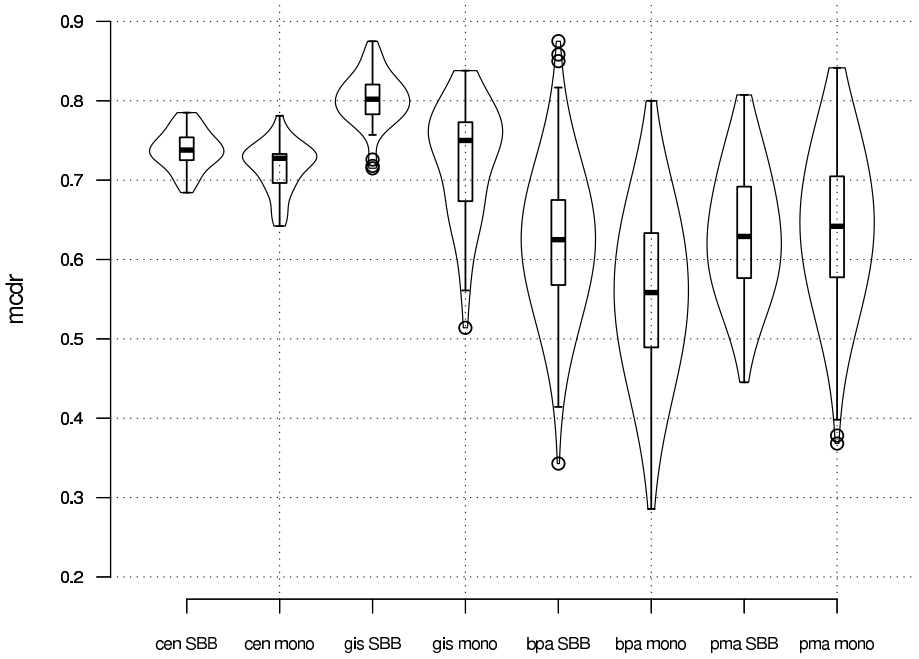
Figure 4.1 summarizes the classification performance of the symbiotic and monolithic approaches using a combination of boxplots and violin plots[6]. The boxplots show the distribution of values across all initializations for each pairing of dataset and approach, e.g., the Census distributions represent 60 points while the Bupa distributions contain 600 points. The box endpoints denote the first and third quartiles, and the horizontal line inside the box indicates the median. Whiskers extend from the box endpoints to cover any points within one-and-half times the interquartile range, and a 'o' denotes outliers beyond the whiskers. For each distribution, a violin plot [69] is also drawn representing the associated kernel density estimate with wider regions reflecting higher densities[7]. In all figures, 'mono' is used to refer to the monolithic approach.

The accuracy and mcdr results on the test data, Figure 4.1, support the claim that the symbiotic solutions yield improved classification performance. With respect to accuracy, Figure 4.1a, on Census, Gisette, and Bupa the median and third quartile values under the monolithic approach are always lower. On Pima, the corresponding

[6]Similar plots are used throughout the rest of this thesis.
[7]The 'vioplot' library in the statistical package R was used to generate the violin plots.

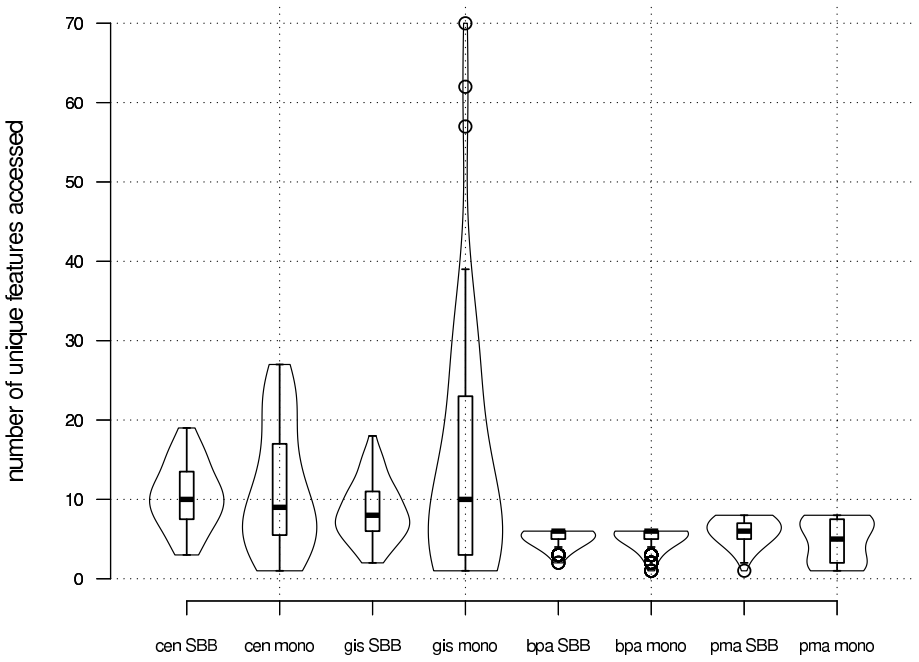


(a) Classification performance with respect to accuracy.

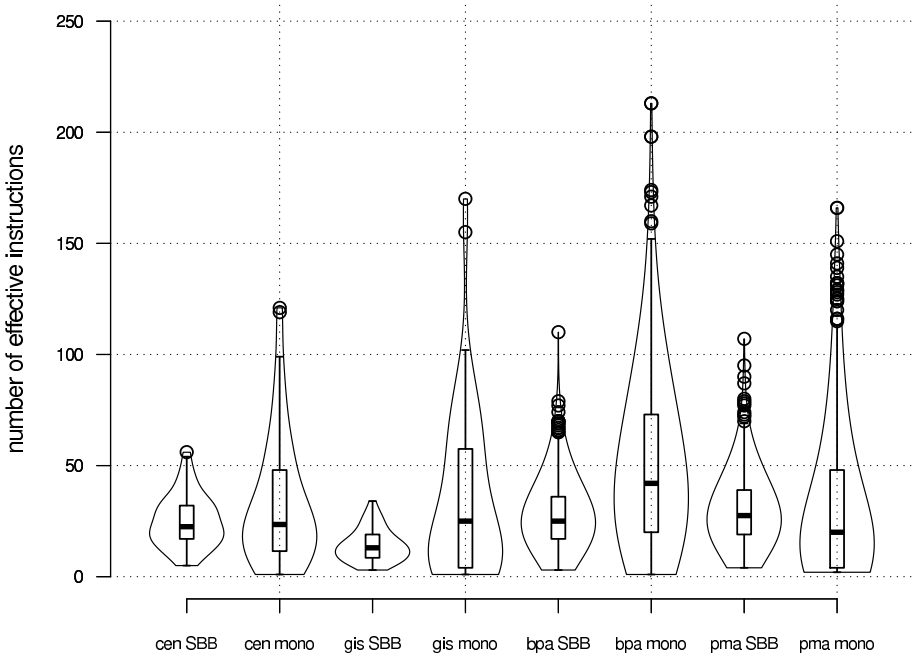


(b) Classification performance with respect to the mcdr.

Figure 4.1: Comparison of the SBB and monolithic approaches with respect to accuracy and mcdr on the test partition.



(a) Complexity with respect to the number of unique features.



(b) Complexity with respect to the number of effective instructions.

Figure 4.2: Comparison of the SBB and monolithic approaches with respect to unique feature and effective instruction counts. The SBB results are an aggregate across all symbionts in a host, e.g., number of unique features accessed by the host as a whole.

	Census	Gisette	Bupa	Pima
accuracy	0.01167⋆	1.103e-11⋆	≈0⋆	0.525
mcdr	0.0002131⋆	1.103e-11⋆	≈0⋆	0.02842

Table 4.5: Two-tailed Mann-Whitney test results comparing the SBB and monolithic approaches, test accuracy and mcdr. Shown are the p-values obtained when the test was applied to each pair of corresponding accuracy and mcdr distributions in Figure 4.1. Cases where the SBB median value is *higher* are noted with a '⋆'.

monolithic values appear to be higher, however, the absolute difference is small compared to the other three problems. Similar trends are observed with respect to the mcdr, Figure 4.1b.

A set of two-tailed Mann-Whitney tests was applied to pairs of corresponding distributions in Figure 4.1, e.g., the two accuracy distributions under Census, to determine if the observed values are significantly different. Assuming a significance level of 0.01, the p-values returned by the tests, Table 4.5, suggest that the accuracies under Census and Pima and the mcdr values under Pima are not significantly different. However, when a significant difference is observed in these test results, the median value under the SBB approach is always higher. This is in line with the qualitative observations noted earlier, and in particular, it suggests that on Pima, the problem where the monolithic approach appears slightly better, the difference is not significant.

Overall, the results in Figure 4.1 and Table 4.5 suggest that, on all four problems, the classification performance under the symbiotic approach is as good as or better than the classification performance under the monolithic approach. These results are therefore consistent with the first claim made at the start of this chapter.

Solution Complexity

Observing the median complexity results, Figure 4.2, the results are mixed in that sometimes the SBB solutions are simpler and sometimes the monolithic solutions are simpler. A visible trend, however, is that the SBB complexity values are much more consistent. With the exception of the feature counts under Bupa and Pima, Figure 4.2a, the highest SBB solution complexities never approach the highest monolithic solution complexities; the unpredictable level of complexity in the monolithic solutions is thus viewed as a drawback of the approach. The number of features under Bupa and Pima is low to start with so it is expected that both approaches would tend to

	Census	Gisette	Bupa	Pima
Feature counts	0.7385	0.3685⋆	0.0009292	1.288e-14
Instruction counts	0.6142⋆	0.06565⋆	≈0⋆	1.091e-08

Table 4.6: Two-tailed Mann-Whitney test results comparing the SBB and monolithic approaches, unique feature and effective instruction counts. Shown are the p-values obtained when the test was applied to each pair of corresponding accuracy and mcdr distributions in Figure 4.2. Cases where the SBB median value is *lower* are noted with a $\star$.

use most of the available features – as is the case – and even if a solution does use all of the features, this is still typically less than the counts observed under Census and Gisette.

A set of two-tailed Mann-Whitney tests, Table 4.6, indicates no significant difference in the unique feature and effective instruction counts at a 0.01 significance level on Census and Gisette. As a nonparametric procedure, the Mann-Whitney test does not consider the fact that some of the monolithic solutions may be very large, or alternatively, the greater consistency of the complexities under the SBB approach. Under Bupa and Pima, the p-values suggest that there is a significant difference in three of the four cases, where the difference appears to be in favour of the monolithic approach. Given these results, the second claim regarding the reduced complexity under the symbiotic approach cannot be substantiated.

Classification Performance versus Complexity

Given the choice, solutions that achieve both a high accuracy and a high mcdr are preferred over solutions that score high in just one of these objectives. However, in practice, strong performance with respect to one objective may require a compromise with respect to the other, especially under problems with unbalanced class distributions. Furthermore, the analysis so far has suggested that the classification performance of the SBB solutions is as good as or better than that of the monolithic solutions, but given the solution complexities observed in Figure 4.2, it is not clear if the improved performance comes at a cost of higher solution complexity. The aim of Figures 4.3 through 4.6 is therefore to shed some light on these issues by relating accuracy with mcdr results and solution complexities.

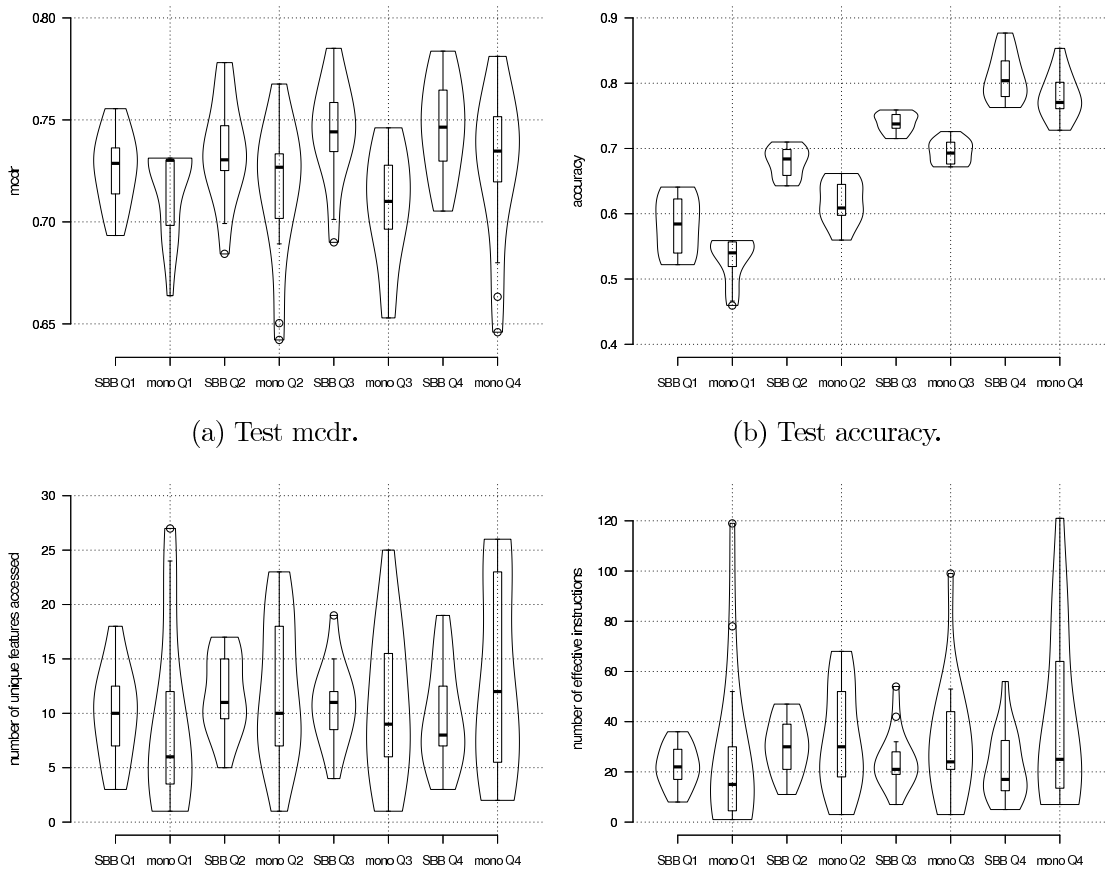
Figure 4.3a shows observed mcdr values as a function of observed accuracy values for the two approaches on the Census dataset. It is generated by first sorting the

solutions for each approach using accuracy as the sort key and using mcdr to break ties. Each sorted list is then divided into four equal contiguous sublists, Q1 through Q4, representing solutions that are progressively more accurate, i.e., quarters Q1 and Q4 respectively contain the least and most accurate solutions. For each quarter, the distribution of the solutions with respect to mcdr is then plotted in Figure 4.3a. The same sorted lists (one for each approach) are also used in Figures 4.3b through 4.3d, e.g., in each plot Q1 corresponds to the same subset of solutions, but each figure aims to characterize a different property. For completeness, Figure 4.3b shows the range of accuracy values that fall in each quarter; given the solutions are first sorted by accuracy the distributions under each approach do not overlap. Figures 4.3c and 4.3d respectively show the complexity in terms of the number of unique features and the effective instruction counts. Thus, from Figure 4.3 as a whole, it may possible to infer relationships between the mcdr values, unique feature counts, and effective instruction counts associated with solutions yielding different levels of accuracy. Figures 4.4 through 4.6 are analogous but shown results for the other three problems.

Based on the results presented in Figures 4.3 through 4.6, a number of observations are noted:

- Considering the relationship between accuracy and mcdr, subplot (a) in the figures, under both approaches higher accuracies tend to be associated with higher mcdr values across all the problems. Thus, solutions that achieve both a high accuracy and a high mcdr value are found by both methods. On Census, however, this trend appears to be the weakest, with the most overlap observed between distributions for consecutive quarters – this is likely due to the high class imbalance on this dataset. Under Gisette, a perfectly balanced dataset, the trend is the strongest with no overlap between consecutive quarters under each approach. Spearman's rank correlation results, Table 4.7, are consistent with these observations.

- Under the SBB approach, it appears as if increases in accuracy are not realized at the expense of higher complexity as measured both in terms the unique feature counts and the effective instruction counts. Given that, with the exception of Census, accuracy appears to have a strong positive correlation with mcdr, Table 4.7, the same relationship between mcdr and solution complexity can be inferred.



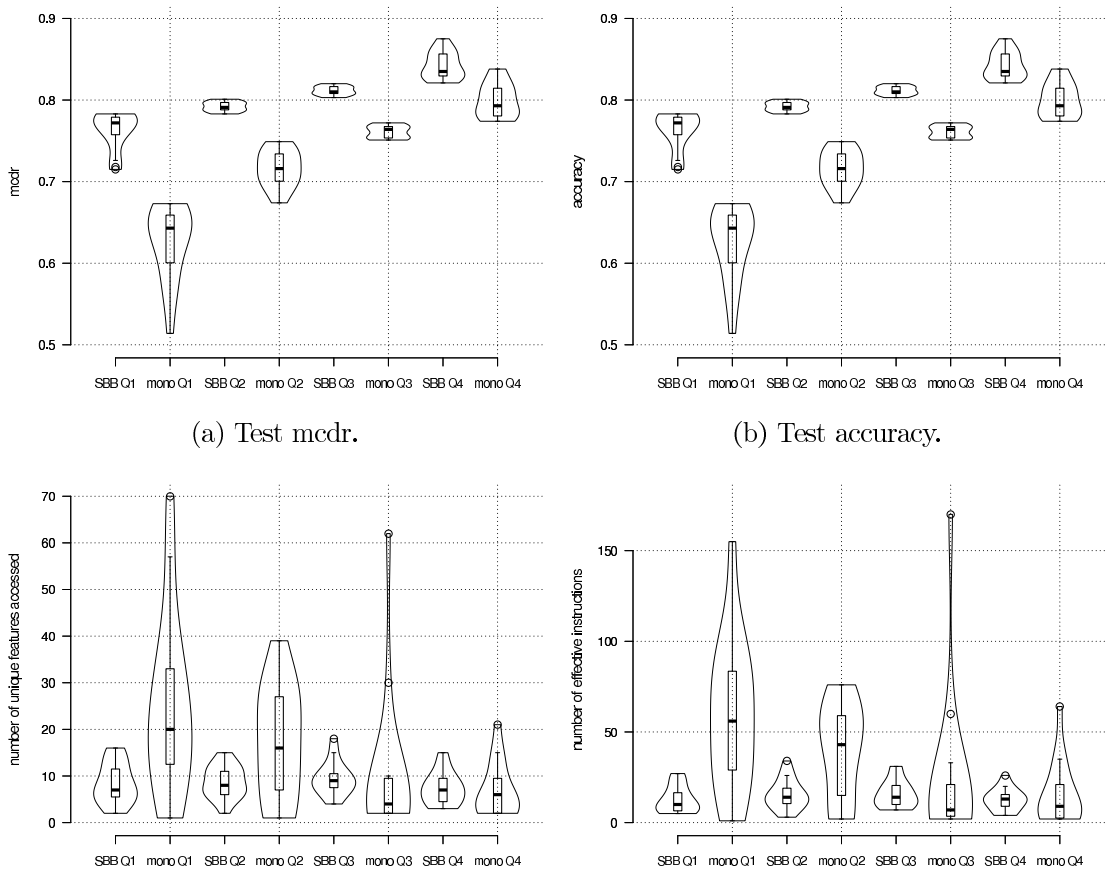
(a) Test mcdr.

(b) Test accuracy.

(c) Unique feature count.

(d) Effective instruction count.

Figure 4.3: Breakdown of Census results with respect to accuracy. The x-axis denotes quarters of solutions with progressively higher accuracy values, and for each quarter, the indicated quantity is plotted on the y-axis. For each approach, corresponding quarters across all subplots represent the same set of solutions.



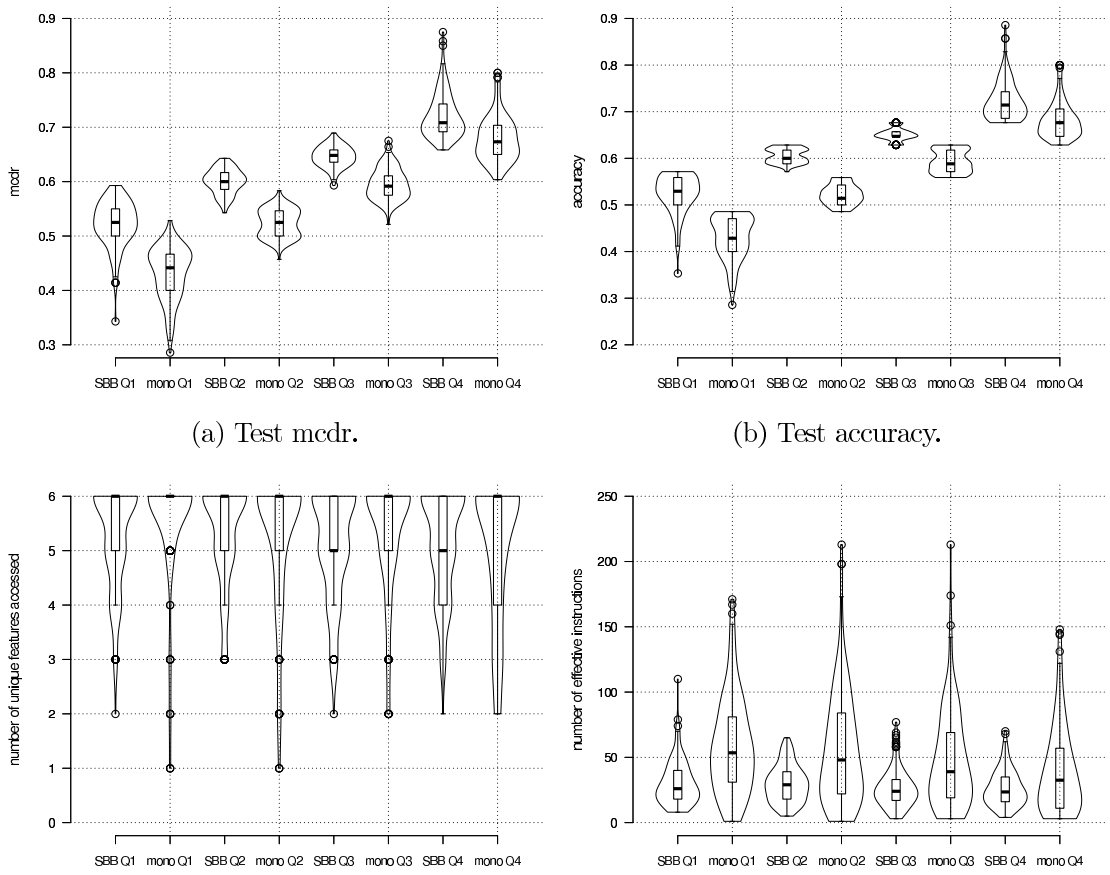
(a) Test mcdr.

(b) Test accuracy.

(c) Unique feature count.

(d) Effective instruction count.

Figure 4.4: Breakdown of Gisette results with respect to accuracy, analogous to Figure 4.3.



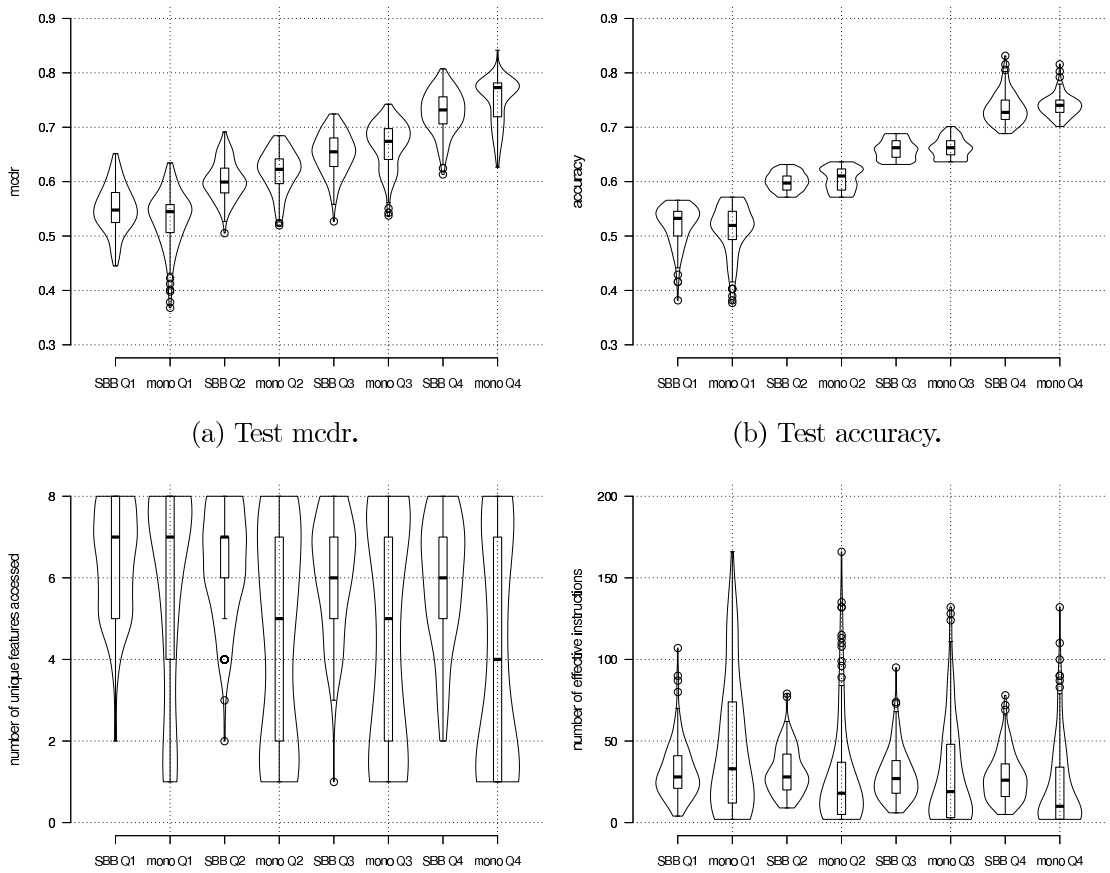
(a) Test mcdr.

(b) Test accuracy.

(c) Unique feature count.

(d) Effective instruction count.

Figure 4.5: Breakdown of Bupa results with respect to accuracy, analogous to Figure 4.3.



(a) Test mcdr.

(b) Test accuracy.

(c) Unique feature count.

(d) Effective instruction count.

Figure 4.6: Breakdown of Pima results with respect to accuracy, analogous to Figure 4.3.

	Census	Gisette	Bupa	Pima
SBB	0.35	1.00	0.97	0.88
Monolithic	0.08	1.00	0.98	0.88

Table 4.7: Spearman's rank correlation values with respect to accuracy and mcdr on the test partition under the SBB and monolithic approaches. The correlation values reflect the relationships between accuracy and mcdr visible in subplot (a) in Figures 4.3 through 4.6. In particular, with the exception of Census, a strong positive correlation between accuracy and mcdr is observed under both approaches. The correlation values themselves were calculated using the 'Stats' library in the R package.

- Whereas the solution complexity under the SBB approach tends to remain constant across all levels of accuracy/mcdr (quarters Q1 through Q4), the relationship between accuracy/mcdr and solution complexity under the monolithic approach is less consistent. Under Census, the monolithic Q4 feature and instruction counts tend to be the highest, while under Gisette and Pima, the opposite appears to be true. The latter two cases appear particularly counter-intuitive since one would expect improved classification performance to require increased complexity. Under Bupa, the monolithic solution complexities appear to be relatively constant across all levels of classification performance.

- Comparing the solutions yielding the highest classification performance, i.e., those in quarter Q4, the SBB solutions appear to be equally or less complex than the monolithic solutions on Census, Gisette, and Bupa. Under these three problems, the SBB solutions also tend to yield higher accuracy and mcdr values. Under Pima, the trend is reversed, nevertheless, the SBB approach appears to produce better solutions with respect to both the classification performance and complexity objectives on three of the four problems.

In summary, based on these results, it appears that under the two approaches both high accuracy and high mcdr values are attainable, i.e., one does not have to choose either between high accuracy or high mcdr. Furthermore, these results suggest that any improvement in classification performance observed under the SBB approach, in contrast to the monolithic approach, is independent of the level of solution complexity. The one exception where the best monolithic solutions appear to be both more effective and less complex is the Pima dataset, a result that is consistent with observations made in the previous sections.

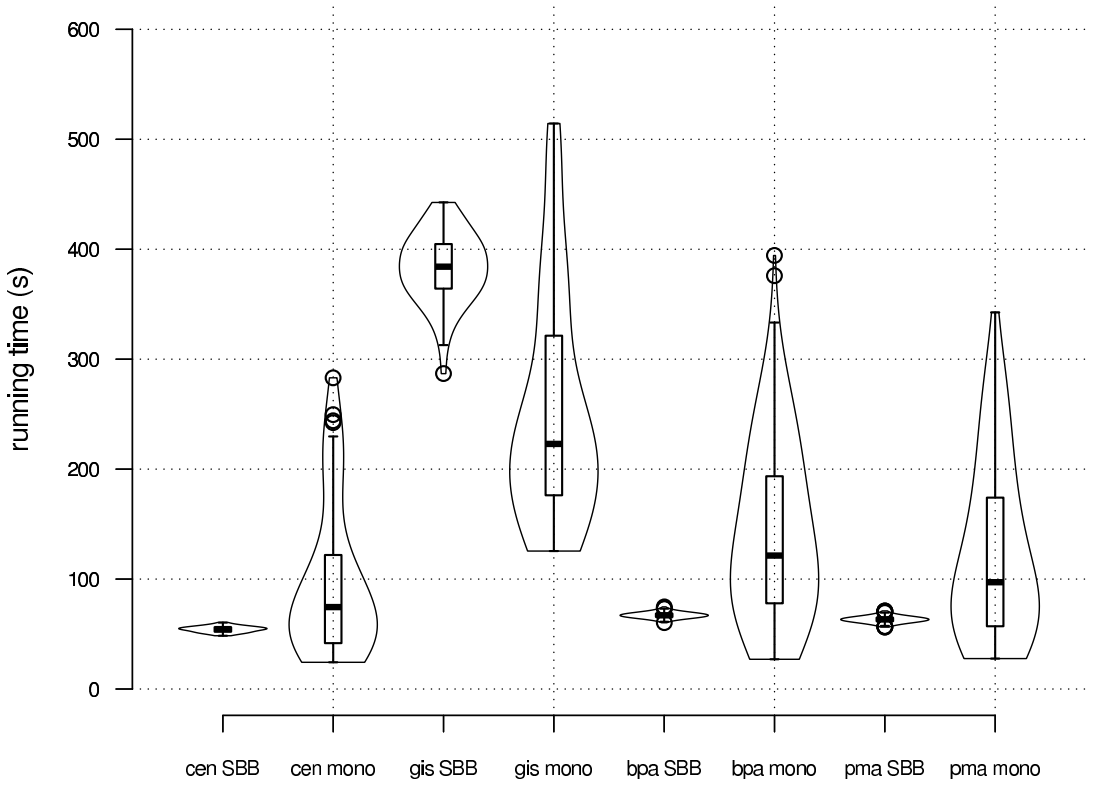


Figure 4.7: Comparison of the SBB and monolithic approaches with respect to the training times (in seconds) as measured by the C *getrusage* function. Only non-intron instructions executed in user mode are considered.

Training Overhead

The running time results, Figure 4.7, show that on Census, Bupa, and Pima, the SBB distributions are more consistent and suggest a lower training overhead. Furthermore, the distributions are similar across all three of these problems even though Census is much larger in terms of the number of instances – this can be attributed to performing fitness evaluation on a fixed-size, stochastically-sampled, subset of training exemplars. Results on Gisette appear to be an exception in two ways. First, the training times under both approaches appear to be higher where this is likely due to the large number of attributes that are present. Second, the SBB training times on Gisette are higher than the monolithic training times. In all cases, a substantial difference between corresponding SBB and monolithic distributions was identified by a two-tailed Mann-Whitney test at a 0.01 significance level.

It is natural to assume that the training time would be a function of the number of attributes used or the effective instruction count, Figure 4.2. However, differences

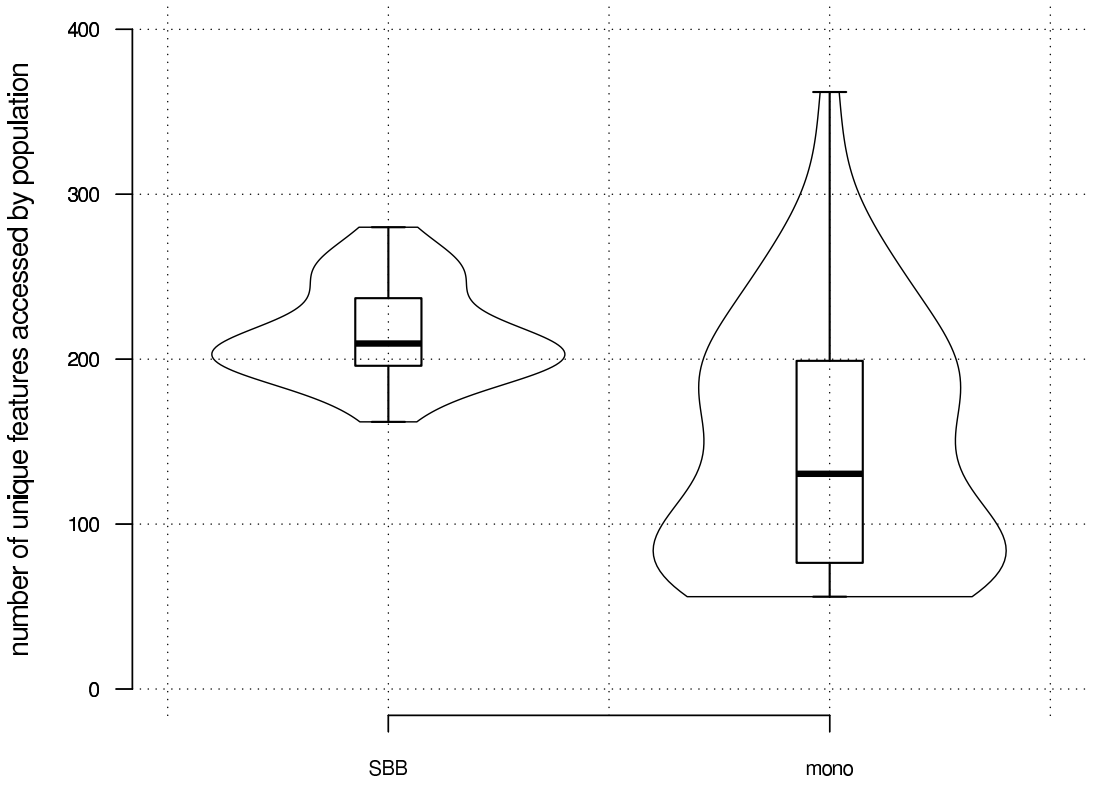


Figure 4.8: Number of unique features accessed across all solutions in the final population under Gisette.

from the expected may be attributed to the fact that the complexity results presented in Section 4.2.1 represent the best solution at the end of training, whereas the training times in Figure 4.7 represent all individuals across all generations. This makes it difficult to determine the specific cause for why this assumption may not hold.

However, in an attempt to gain insight into the training times on Gisette, where the total number of attributes is exceptionally high, additional results were collected with respect to the number of attributes used across all individuals in the final populations, Figures 4.8 and 4.9. These results indicate that the number of unique attributes used across all hosts in the final population is higher under the symbiotic approach than under the monolithic approach, Figure 4.8. In this case, a two-tailed Mann-Whitney test indicates a difference at a 0.01 significance level. However, as suggested in Figure 4.9, under the symbiotic approach the proportion of attributes used across the final population that are also used by the best individual in that population is smaller, where this observation is consistent with a two-tailed Mann-Whitney test at a 0.01 significance level.

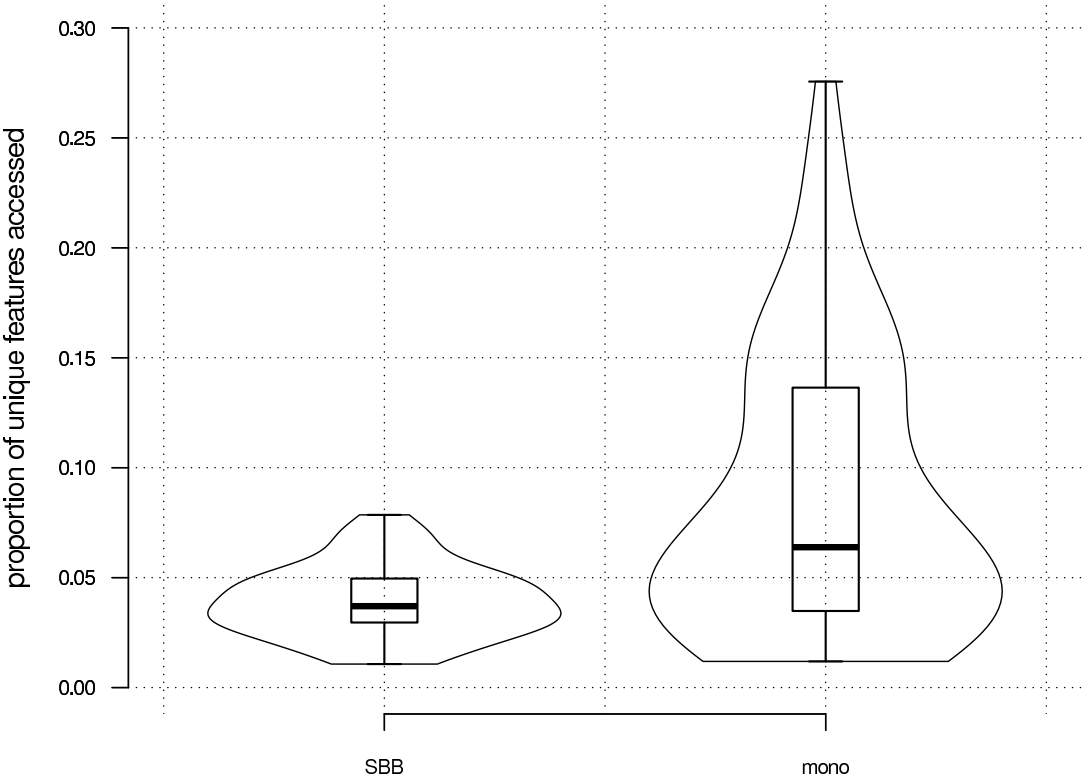


Figure 4.9: Proportion of unique features accessed across all solutions in the final population under Gisette that are also accessed by the best solution. Lower values suggest less overlap in the features used.

Based on these results, it appears as though the symbiotic approach is more effective at encouraging diversity in the attributes explored across the population; generally speaking, this is a desirable property since it suggests a greater degree of problem decomposition and a decreased likelihood of premature convergence. However, this may come at the expense of increased training times as additional operations need to be performed by the system to read data from storage, i.e., the rate of cache misses is increased. Given that the Census dataset involves a large number of attributes yet it is sparse (with eighty-seven percent of the feature instances representing zero values), an indirect encoding[8] may reduce the Gisette training times to the same levels as in the other three problems.

Returning to the final claim made at the beginning of this chapter, these results confirm that on three of the four problems evolving the SBB solutions incurs less computational overhead compared to the monolithic approach. The one exception where the SBB training times were on aggregate longer than the monolithic training times was Gisette, where it was suggested that the higher computational overhead may result from an increased diversity in the features that are explored combined with the direct encoding used to represent the data. However, on all datasets the SBB training times were found to be more consistent; as a result, the highest training times on Gisette were nevertheless observed under the monolithic approach. Given these considerations, the results presented in this section are viewed as consistent with the final claim, with the caveat that on problems with large attribute spaces care must be taken to efficiently represent the data.

4.2.2 Further Characterization of Symbiotic Solutions

In this section, the solutions generated using the SBB approach are further characterized. Specifically, the composition of the evolved hosts is presented, and an attempt is made to gain insight into the way in which symbionts in a host decompose the problem. As in the comparison with the monolithic approach, Section 4.2.1, distributions represent the hosts that yield the highest mcdr rates on the training partition at the end of training, with one host selected from the final population in each initialization, i.e., the Census and Gisette distributions account for 60 hosts while the Bupa and Pima distributions represent 600 points. Reference is also made to specific symbiont actions whose mapping to the problem-specific class labels is defined in Table 4.2.

[8]Specifying the indices and values of non-zero attributes only.

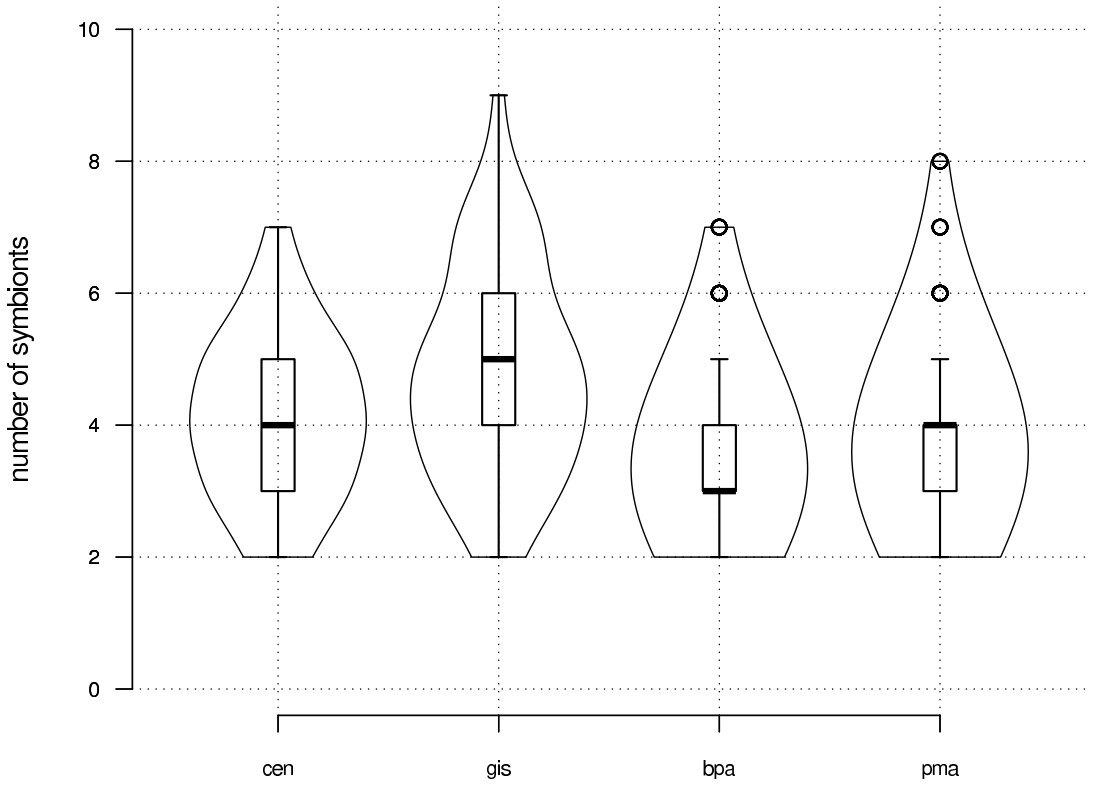


Figure 4.10: Host sizes on Census, Gisette, Bupa, and Pima, in terms of the number of symbionts after pruning of symbionts.

Host Composition

The number of symbionts found in a host, Figure 4.10, and the mean number of effective instructions per symbiont, Figure 4.11, tend to fall well below the upper limits of 10 (ω) and 48 ($maxProgSize$) respectively. The largest hosts, with respect to the median symbiont counts, are observed under Gisette, as is the greatest spread in the host sizes. The mean number of effective instructions per symbiont, Figure 4.11, is obtained by dividing the total effective instruction count in each host, Figure 4.2b, by the size of the host. The results in Figure 4.11 highlight the simplicity of the individual symbionts with respect to the instruction counts, particularity under Gisette where the median symbiont instruction count is about 2.8. Together, the distributions in Figures 4.10 and 4.11 suggest that under Gisette the hosts are larger but consist of simpler symbionts; in this sense, a greater degree of problem decomposition occurs here than under the other problems. In contrast, the host sizes under Bupa and Pima are smaller but result in larger per-symbiont instruction counts.

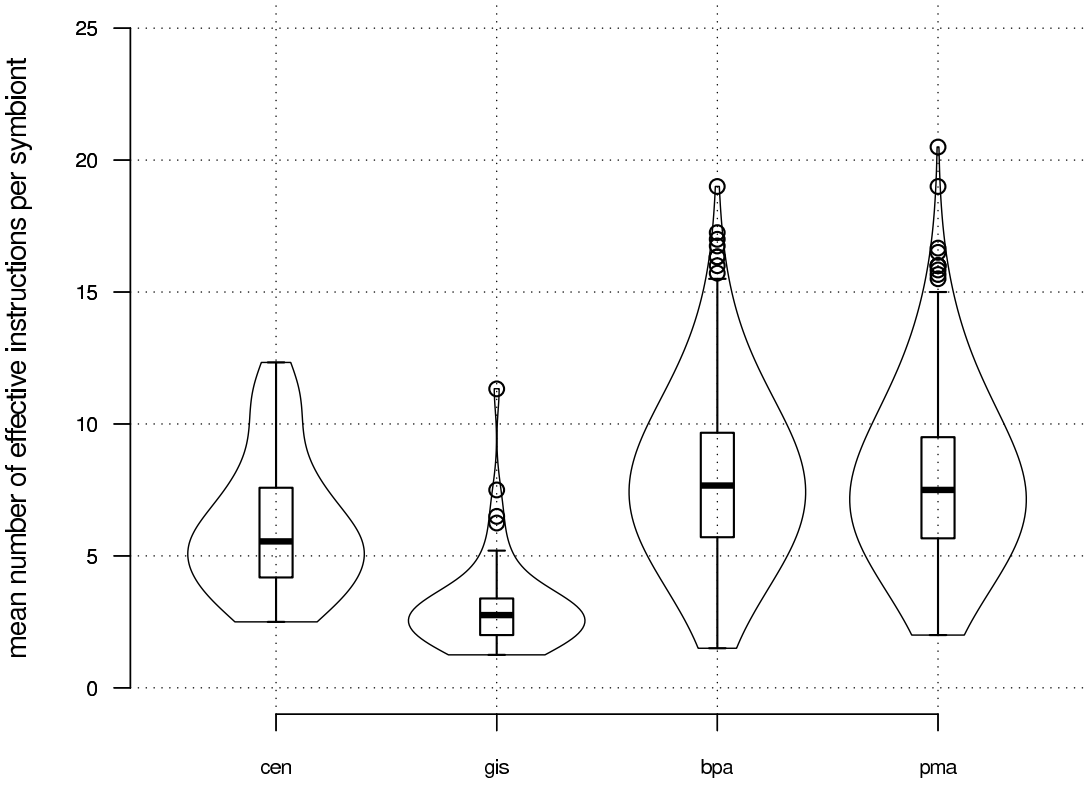


Figure 4.11: Mean number of effective instructions per symbiont on Census, Gisette, Bupa, and Pima.

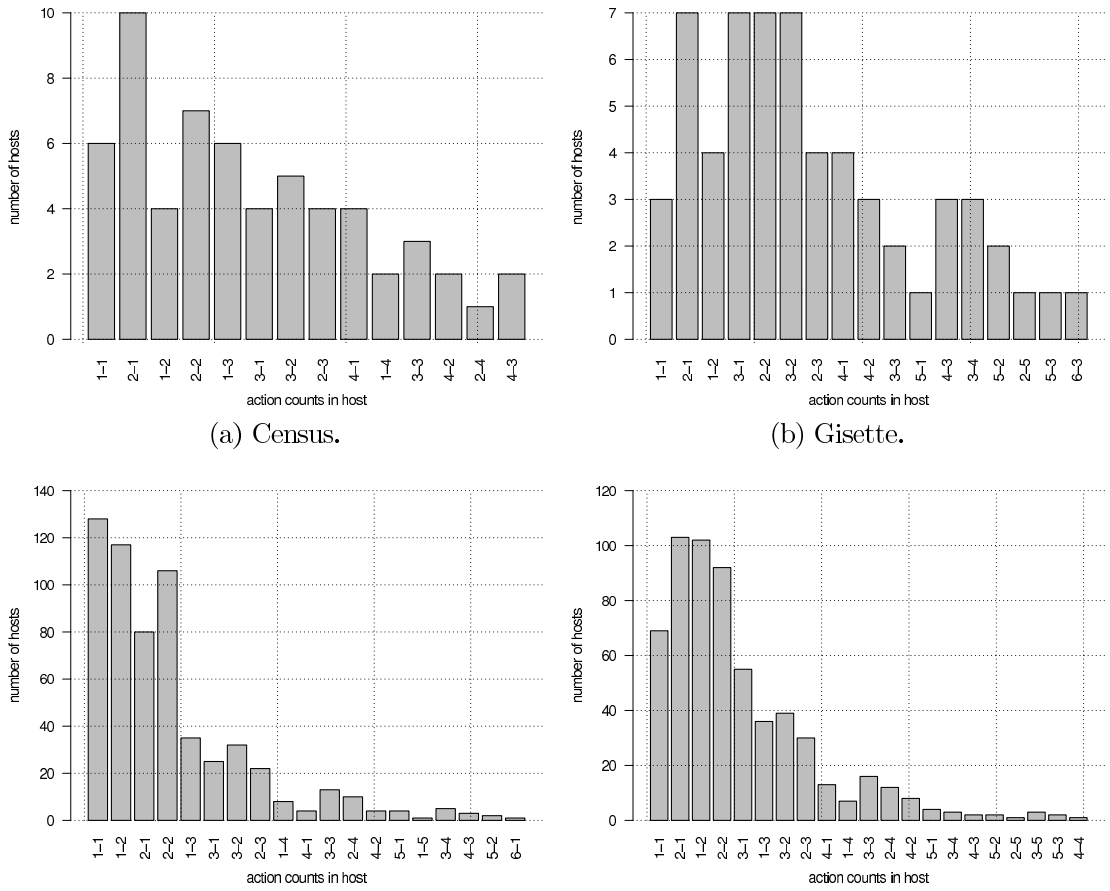
Results on the composition of the hosts that were evolved, with respect to the number of symbionts of each action, are shown in Figures 4.12 and 4.13. Given that there were two actions and at most 10 symbionts per host, it was reasonable to enumerate all possible combinations of action counts in a host, that is, how many symbionts of each action were present in the host. This exhaustive enumeration, excluding combinations that were never observed, is shown in Figure 4.12, where the number of hosts, y-axis, observed for each action combination, x-axis, is shown. For example, under Census, Figure 4.12a, there were two hosts observed containing four symbionts of action 0 and three symbionts of action 1 (rightmost column). As expected given the symbiont counts in Figure 4.10, there is a bias on all problems towards smaller hosts, though this bias is more apparent under Bupa and Pima. No tendencies towards symbionts of a certain action are apparent, instead, different combinations of actions are explored in different initializations.

Figure 4.13 summarizes the host makeup with respect to the action counts by showing the proportion of symbionts in a host whose action matches the majority class. There is a large spread in the distributions, though median values are observed at 0.5 for Census, Bupa, and Pima, and at 0.6 for Census. In all cases, both actions are present in the final solution despite the lack of an explicit mechanism for ensuring that this happens. Thus, it does not appear that any degenerate solutions, i.e., ones which always predict the same class label, are present in the final set represented in the figure[9].

Partitioning of the Instance Space

The bid-based approach to context learning adopted in this work, Section 3.1, supports the partitioning of the instance space as illustrated in the right side of Figure 1.1. Specifically, each problem instance can be associated with exactly one symbiont in a given host, i.e., the symbiont winning the bid auction on that instance; conversely, each symbiont in a host will identify a unique subset of exemplars. Figure 4.14 provides insight into the nature of this partition for the four problems that were considered. In particular, the figure shows the proportion of all test instances won by the symbiont winning the most instances (the top winner), and also the mean win count across all other symbionts in the same host (excluding the top winner). The results are broken down by action so that behaviour that may be specific to one

[9]Care of the symbiont pruning algorithm, any symbionts that do not win at least one bidding round during training are removed. Thus, it is not possible for an action to be represented in Figure 4.13 and never be output by the host.



(a) Census. (b) Gisette.

(c) Bupa. (d) Pima.

Figure 4.12: Host counts broken down by symbiont actions, after pruning of symbionts. The x-axis denotes combinations of action counts found in individual hosts with the first (second) element in each pair denoting the number of symbionts of action 0 (1). The y-axis corresponds to the number of hosts containing (precisely) the indicated combination of actions. The sum of counts across all combinations is 60 for Census and Gisette and 600 for Bupa and Pima.

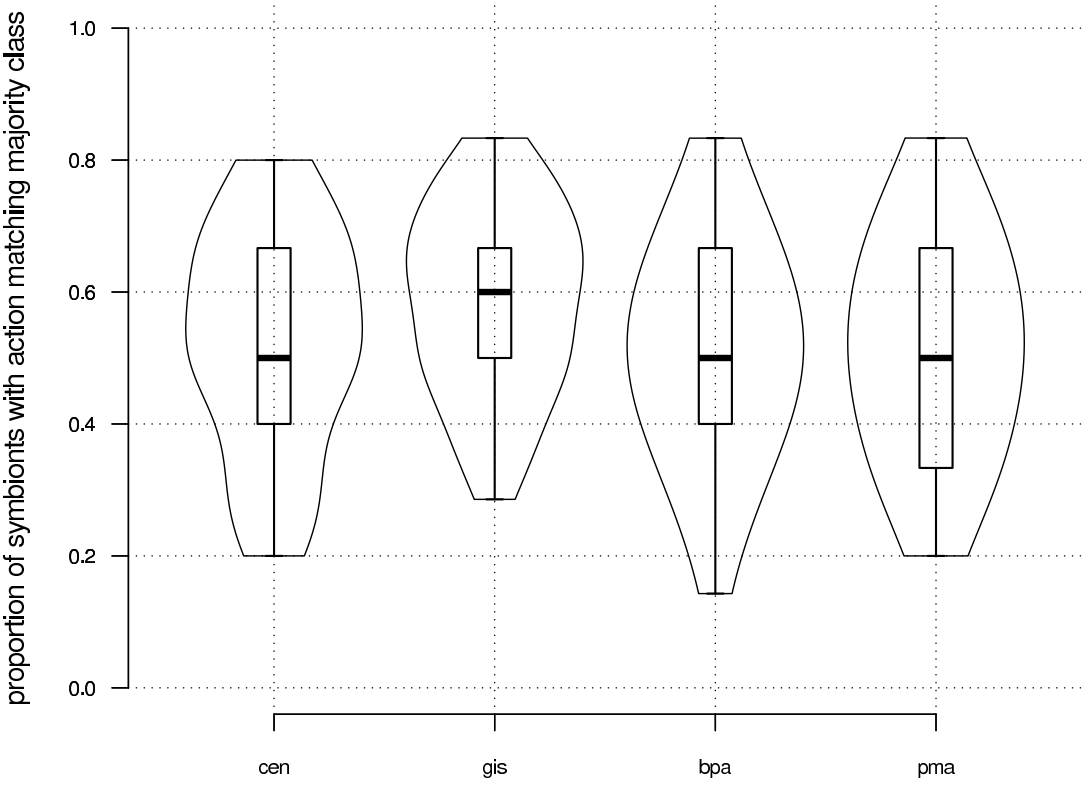


Figure 4.13: Proportion of symbionts in a host whose action matches the majority class, after pruning of symbionts.

action but not the other can be observed.

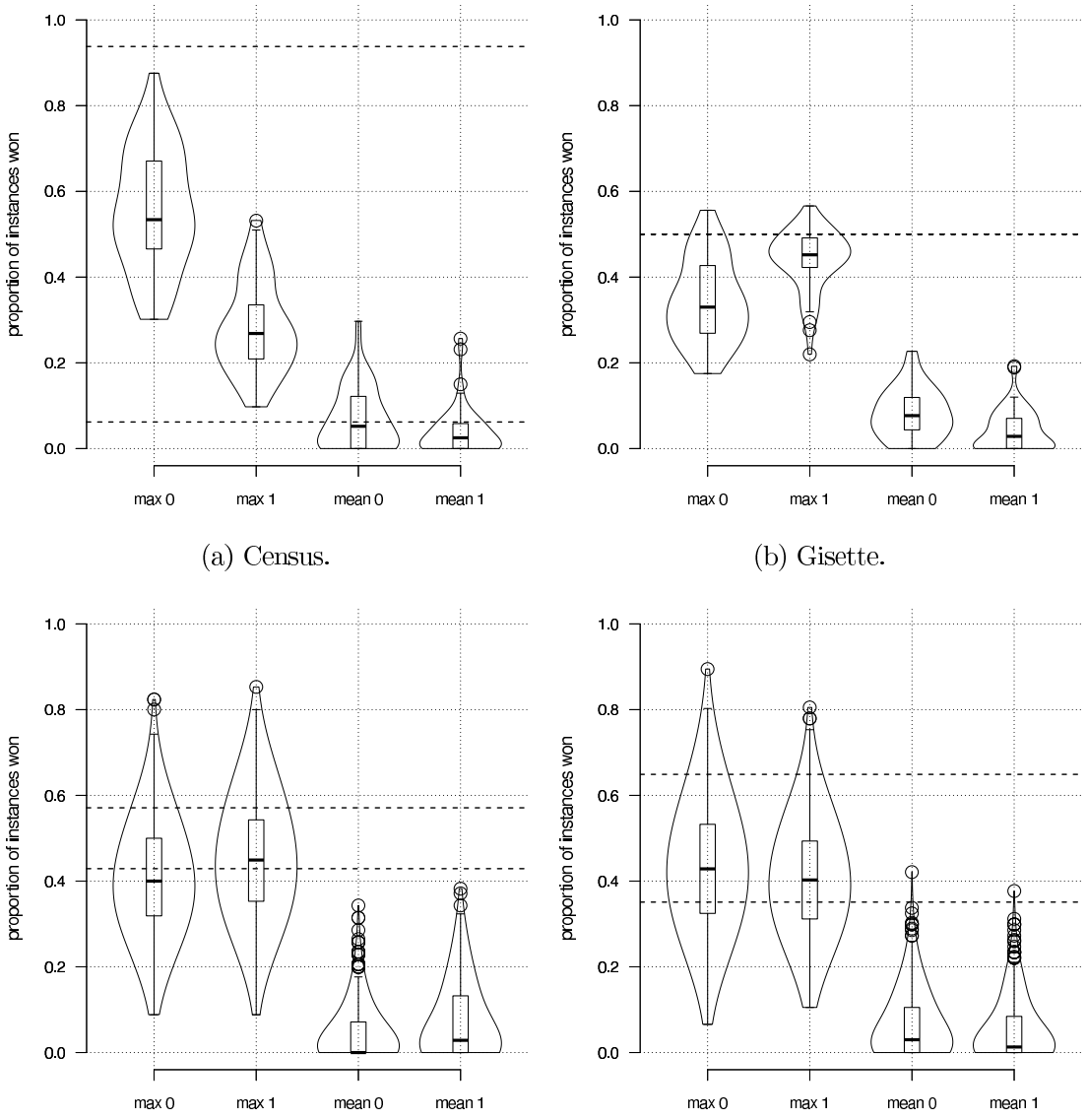
Figure 4.14 suggests that on all datasets the partition on the instance space is unbalanced. Specifically, within a typical host, there is one symbiont that wins a large number of instances, while the remaining symbionts win a relatively small number of instances. This behaviour is the strongest on Census, likely due to the unbalanced nature of the dataset. In addition, the observed values tend to be higher for actions corresponding to the majority class, namely, actions 0, 1, and 0 for Census, Bupa, and Pima respectively. This behaviour is consistent with the manner in which default hierarchies operate, Section 2.2.4, though it is difficult to determine whether or not this is actually the case without further evidence.

Unique Features Accessed

In terms of the mean number of unique features accessed per symbiont, Figure 4.15, the bidders evolved under the SBB approach are always simple regardless of the number of features in the problem. The feature counts are similar under Census, Bupa, and Pima with median values falling between 2.6 and 3.0. Under Gisette, the problem with 5000 attributes, the median number of features accessed by a symbiont is even less and stands at about 1.7. This is consistent with previous observations regarding the high level of problem decomposition under Gisette: compared to the other problems, the hosts evolved tend to consist of more members, Figure 4.10, but the members themselves are simpler containing fewer effective instructions, Figure 4.11, and accessing fewer features, Figure 4.15.

A peculiar observation in Figure 4.15 is that under Gisette the minimum mean number of features accessed per symbiont is below 1. Thus, there are certain symbionts that access no features, and in effect, always bid the same value regardless of the input exemplar. This suggests a thresholding approach whereby other symbionts learn to bid around this constant bid, effectively simulating a default hierarchy, Section 2.2.4. Due to the pruning process, Section 3.3.1, only one symbiont supplying a constant bid can exist in a host regardless of its action, and it must be the case that this symbiont wins at least one bid auction (otherwise it would be pruned).

The mean number of unique features accessed by each symbiont in a host, broken down by action, is shown in Figure 4.16. Under Gisette, Bupa, and Pima, the behaviour under both actions appears to be similar, an observation consistent with a two-tailed Mann-Whitney test at a 0.01 significance level. Under Census, symbionts associated with action 1 tend to access more features. Census is the least balanced



(a) Census.

(b) Gisette.

(c) Bupa.

(d) Pima.

Figure 4.14: Proportion of test instances won by the symbionts in a host, grouped by action. Distributions labeled 'max' show the proportion of test instances won by the symbiont winning the most instances, while those labeled 'mean' represent the mean number of wins for all other members in the same host. The counts are broken down by the action of the symbiont, Table 4.2, e.g., 'max 0' corresponds to the top winner of action 0. The dashed horizontal lines denote the proportion of test instances corresponding to the minor and major classes.

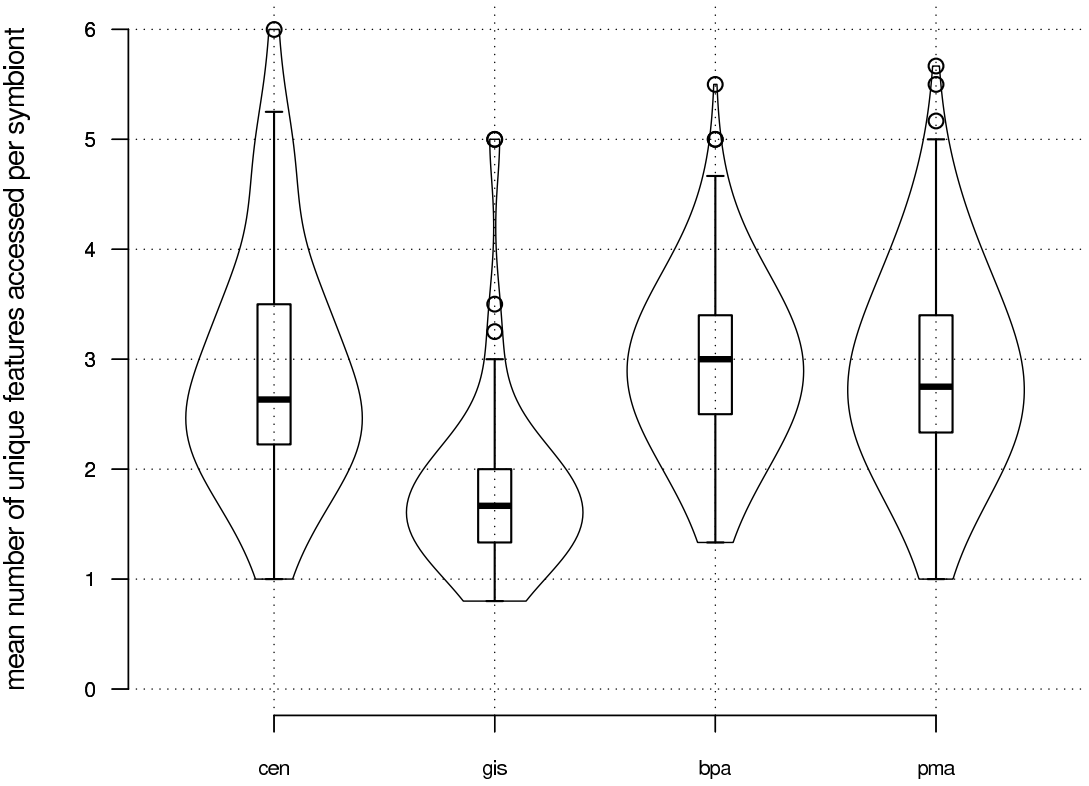


Figure 4.15: Mean number of unique features accessed per symbiont on Census, Gisette, Bupa, and Pima. For each symbiont in a host, the number of unique features that it accesses is considered, i.e., multiple accesses to the same feature by the symbiont contribute once to its count. The mean for a host is then taken over all symbionts in the host and represents one point the distribution.

of the datasets, with action 0 corresponding to the majority class, Table 4.2, and accounting for 0.9379 of the training data. Figure 4.16 also shows at least one host under Census where there is a single symbiont of action 0 and that symbiont bids a constant value[10]. This again suggests a thresholding approach but one where the symbiont of action 0 provides a constant bid, and the members providing dynamic bids are all associated with action 1. Since the class associated with action 0 accounts for such a large number of instances, this strategy, effectively defaulting to class 0, is viewed as reasonable. This is also consistent with the higher feature counts observed for hosts of action 1 since then the onus is on them to learn when to bid above the threshold.

Similar thresholding strategies are observed under all datasets, i.e., a host with a mean unique feature count below one is observed on all four problems in Figure 4.16. It is possible that this type of thresholding occurs even if the mean feature counts for a given action are above one, since, if other symbionts matching the constant bidder's action exist in the host, they may bring the average count up. Thus, the behaviour may be more prevalent that Figure 4.16 would suggest. However, the cases where the thresholding strategy is known to apply with certainty are viewed as strong evidence that default hierarchies are emerging, where the primary benefit of this is that it may lead to more compact solutions [71, 154].

The amount of overlap in the features accessed by member symbionts in a given host is shown in Figure 4.17. The overlap, calculated for each host separately, is defined as the proportion of features that are accessed by exactly one symbiont in the host out of all the unique features accessed by the host, or,

$$\frac{\text{number of features accessed by exactly one symbiont}}{\text{total number of unique features accessed across the host}}, \tag{4.3}$$

and is limited to the unit interval. If this proportion is equal to unity, this then means that each feature is identified by a single symbiont, and as the amount of overlap in the features accessed increases, this proportion decreases. Figure 4.17 suggests that on the two datasets with a low overall feature count, Bupa and Pima, the amount of overlap is relatively high. If the dataset contains more features then less overlap is observed, with the least overlap seen under Gisette. This is reasonable because

[10]Due to pruning, no other symbionts bidding a constant value are known to exist in the same host, and specifically, no other symbionts of action 0 bidding a constant value are known to exist. At the same time, all symbionts of action 0 in the host must bid a constant value, otherwise, the mean unique feature count for action 0 would be non-zero. This implies that there is only one host of action 0 and that host bids a constant value.

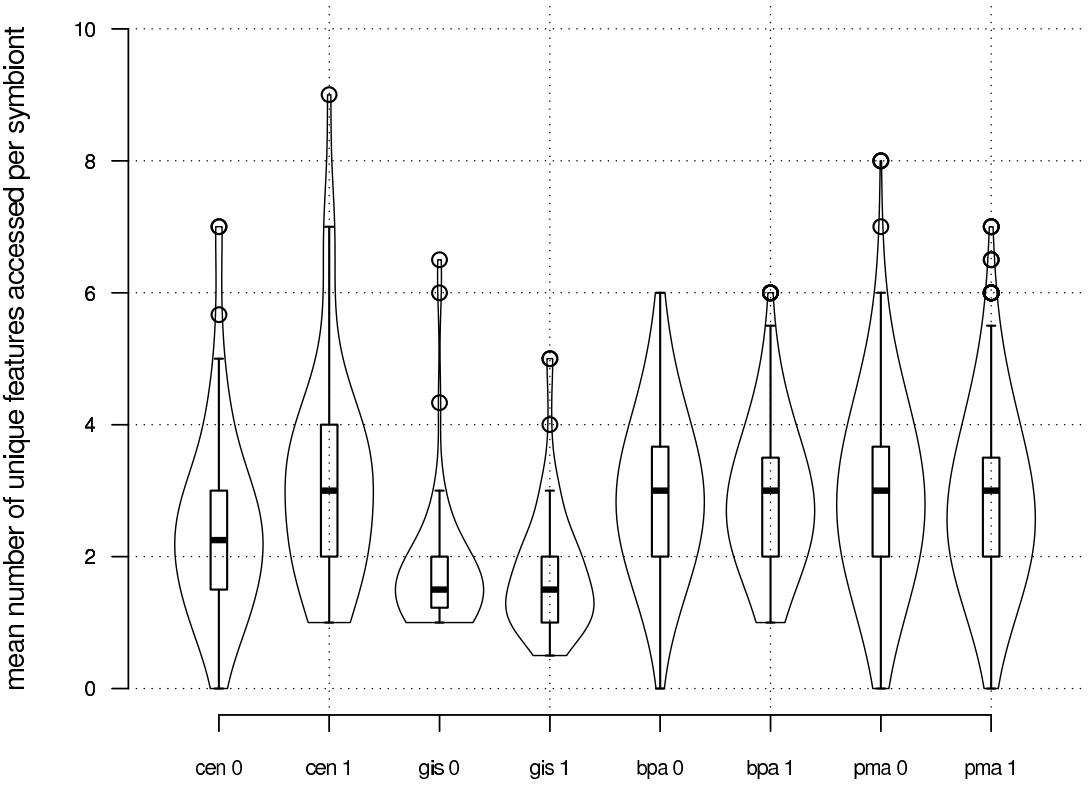


Figure 4.16: Mean number of unique features accessed per symbiont grouped by action. For each host, the symbionts in that host are grouped by action, and the mean number of unique features accessed by individuals in each group is calculated separately. For a given group, the mean is calculated as in Figure 4.15. This is done for each dataset, x-axis, where '0' and '1' denote the action groups. A uniqueness restriction means that multiple accesses to the same feature only contribute once to the count for a given symbiont.

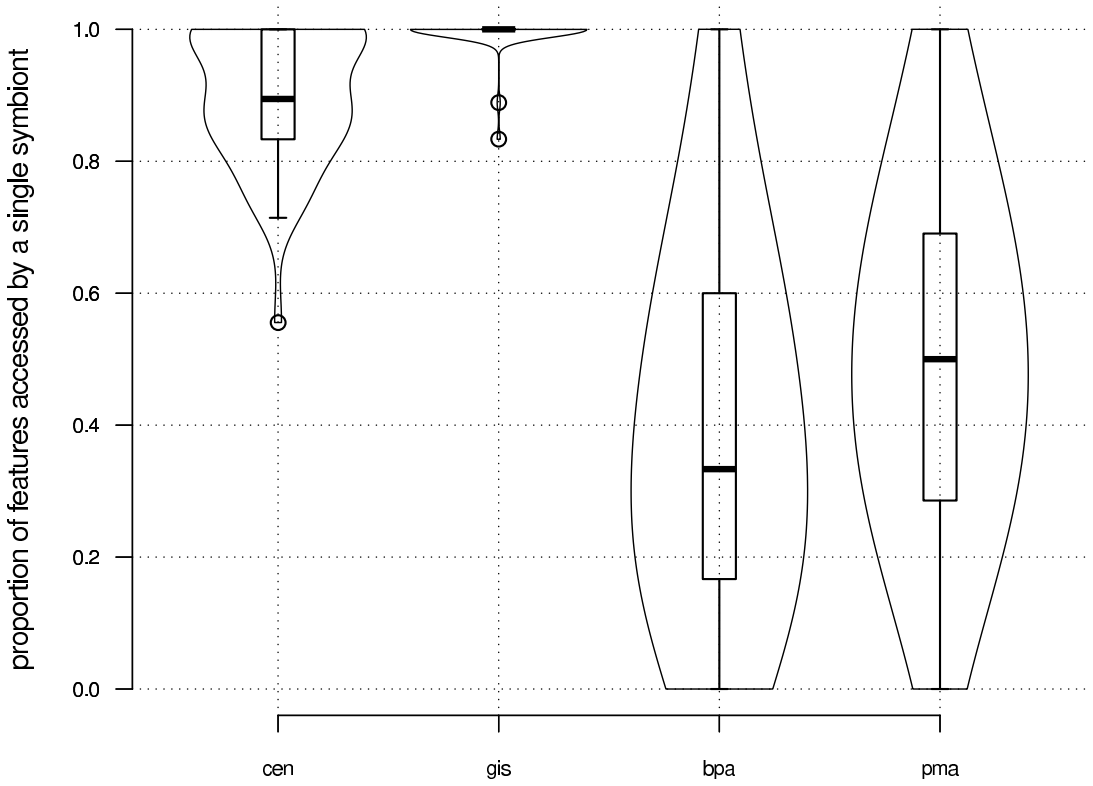


Figure 4.17: Proportion of features, with respect to the unique features accessed across a host, that are accessed by exactly one symbiont in that host under Census, Gisette, Bupa, and Pima. Lower values indicate more overlap in the features accessed by the symbionts in the same host.

as the number of features decreases, evolving symbionts whose feature sets do not overlap is likely to become more difficult (a trivial example being a problem with a single feature, in which case the same feature is always accessed). Gisette contains 5000 features of which 2500 have been linearly transformed to generate the other half, both factors likely leading to the very low levels of overlap that are observed. However, as seen under Census, the symbionts are able to identify unique subspaces even when the problem contains a moderate[11] number of features.

Finally, access counts to specific features are shown in Figure 4.18. The counts are broken down by action, where the overall tally for a given feature can be increased once for each action represented in a host, i.e., if the feature is accessed by a symbiont of that action. Under Gisette, of the 5000 features, only 425 were ever accessed. Of these, the 42 features with overall counts of two or more are shown, Figure 4.18b, which is

[11]Relative to Gisette, Bupa, and Pima.

why overall counts of less than two do not appear. Given that the vast majority of Gisette features were never accessed, and of the features that were accessed most were accessed only once, if a feature has a count of two or more it is viewed as significant. Some of the Gisette features are accessed between six and ten times and are therefore viewed as particularly informative in arriving at a classification decision – especially given that only 5 features out of 5000 are identified as such, and given that each host was evolved in an independent initialization. In addition, these frequently accessed features tend to be associated with action 0, though it is not clear why. Under the other problems, all the features have overall counts of two or more and are therefore all included. Under Census, some features are also identified as more significant than others, e.g., those with twenty or more references could be viewed as particularly useful. Under Bupa and Pima, the counts tend to be more balanced.

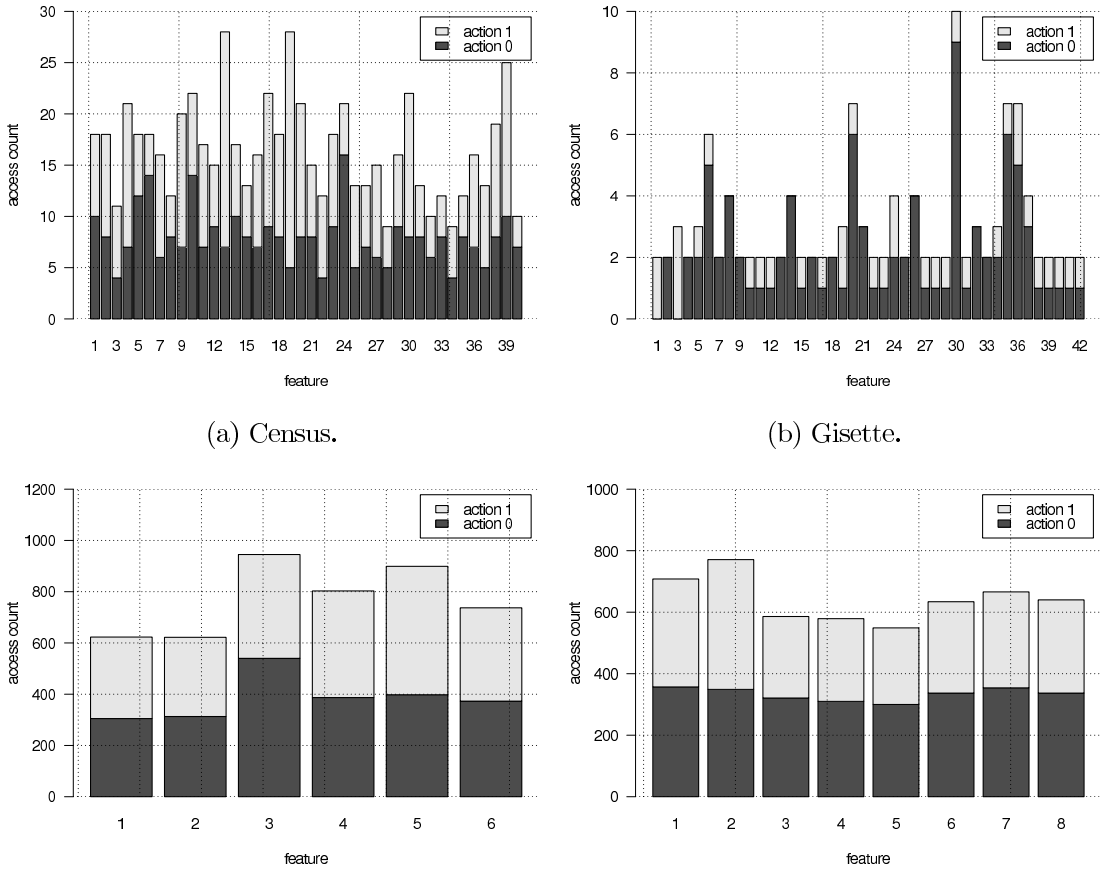
Bid Margins

Although the margin in bid values between the highest bidder and the second-highest bidder was not used as an objective in the host fitness calculation[12], the hosts that were evolved exhibited a wide range of bid margins, Figure 4.19. The bids were specified in terms of the C++ *double* data type, and though the precision of this primitive may vary depending on the computing platform, in general bid margins of 0.1 or more are viewed as considerable. A large bid margin is desirable because it suggests a greater degree of confidence in a host's decision. In the case of the SBB model, this result appears to emerge implicitly as part the evolutionary process, though there is also room for improvement, e.g., through the inclusion of an explicit bid margin objective.

4.2.3 Inter-Host Problem Decomposition

Figure 4.20 shows the amount of overlap in the behaviour observed in the final host populations. For each run, the hosts in the entire population at the end of training are sorted into descending order with respect to the number of correct classifications (hits) on the training data. The hosts are then considered from first (most hits) to last (least hits) and two quantities are recorded. The first is the absolute hit count defined as the number of correct classifications by the host currently under consideration. The second is the cumulative hit count defined as the number of

[12]A bid margin objective was previously effected through the use of a second-price auction under the bid-based metaphor [110].



(a) Census.

(b) Gisette.

(c) Bupa.

(d) Pima.

Figure 4.18: Feature access counts broken down by action. For each feature, x-axis, the number of times it was accessed by a host is shown, y-axis, taking into account symbiont actions. Specifically, for a given feature, a single host can contribute once to the tally under each action depending on whether or not it contains a symbiont of that action that accesses the feature. Under Gisette, there were 425 features with overall counts that were non-zero. Of these, any features with an overall count of less than two were omitted, leaving 42 features. The x-axis tick labels do not correspond to indices of features as they appear in the dataset but are included for reference.

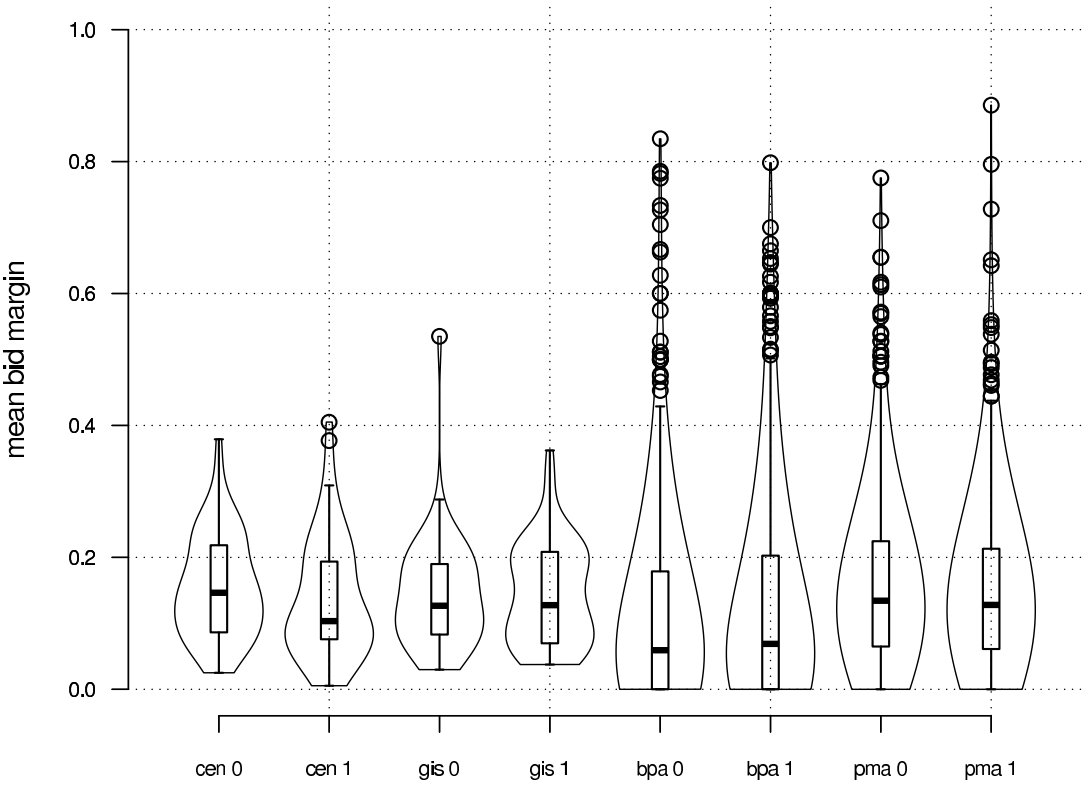


Figure 4.19: Bid margin on the test partition broken down by class. The mean bid margin, y-axis, is plotted for each combination of problem and class, x-axis. For a given host, the mean bid margin with respect to a class is defined as the mean absolute difference between the two highest bids on all instances of that class.

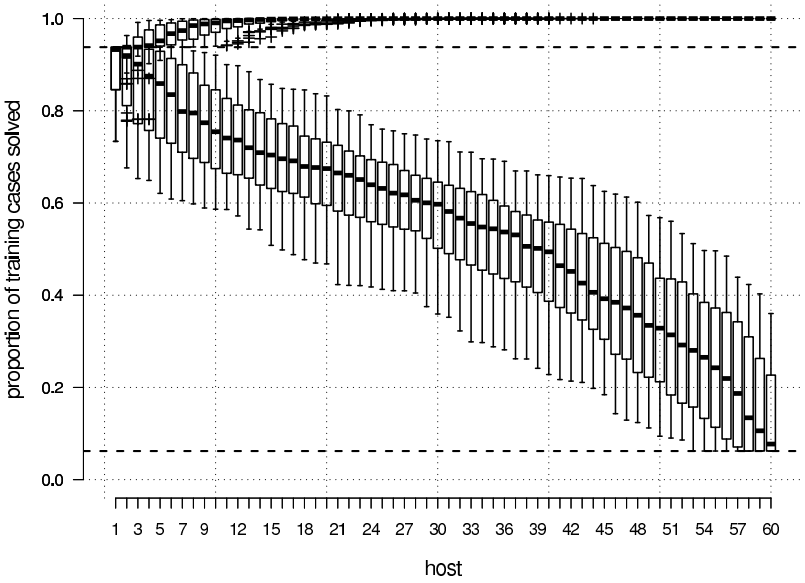
instances classified correctly by the current host or any previous host in the sorted sequence. As successive hosts are considered, the absolute hit count cannot increase and the cumulative hit count cannot decrease.

In Figure 4.20, the sorted hosts representing the final population are denoted on the x-axis. For a given run, the host with the most hits is at $x = 1$ and the host with the least hits is at $x = 60$. For each position on the x-axis, two distributions are plotted, a cumulative count distribution and an absolute count distribution. The number of points in each of these distributions equals the number of initialization, i.e., 60 for Census and Gisette and 600 for Bupa and Pima. Outliers are denoted with a '+' for the cumulative count and a 'o' for the absolute count, otherwise, no distinction is made in the notation of the two quantities; at $x = 1$, the cumulative hit count is equal to the absolute hit count, thereafter, the cumulative hit count always tends to be greater. The counts themselves, y-axis, are normalized by the training set size, and the proportion of training instances belonging to the two classes is indicated by the dashed horizontal line.

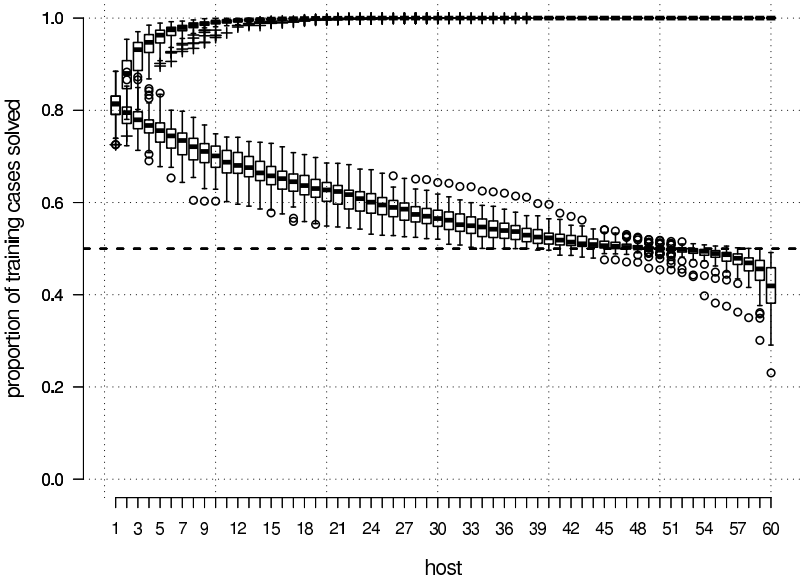
Across all problems and initializations, the final host population always manages to correctly classify all the training instances. This is indicated by the cumulative distributions in Figure 4.20 which always converge to unity. This behaviour motivates the construction of a host hierarchy as described in Section 3.4. Specifically, given the behaviour in Figure 4.20, even a single additional level of hosts should be able to significantly improve classification performance by combining multiple first-level hosts. Furthermore, the second-level hosts need not necessarily use all of the first-level hosts, e.g., at $x = 10$ the cumulative distributions already represent a significant improvement compared to their values at $x = 1$[13].

Under a binary problem, a cumulative count of unity can be trivially achieved by including a host that always assigns one of the labels and a host that always assigns the other label. However, on all problems, a cumulative count of or very close to unity is observed before such degenerate solutions are considered. For example, on Census, if a host labels all instances as belonging to the minority class then its absolute hit count (and sort key) will be at 0.0621 as indicated by the horizontal line at $y = 0.0621$ in Figure 4.20a. However, unity is reached at $x = 45$, well before hosts with such low absolute counts are included.

[13]The ordering of hosts along the x-axis that was used in Figure 4.20 is a heuristic that may not be the best for yielding the highest increase in the cumulative counts. Using a different ordering, it may be possible to yield cumulative counts close to unity with fewer hosts than shown.

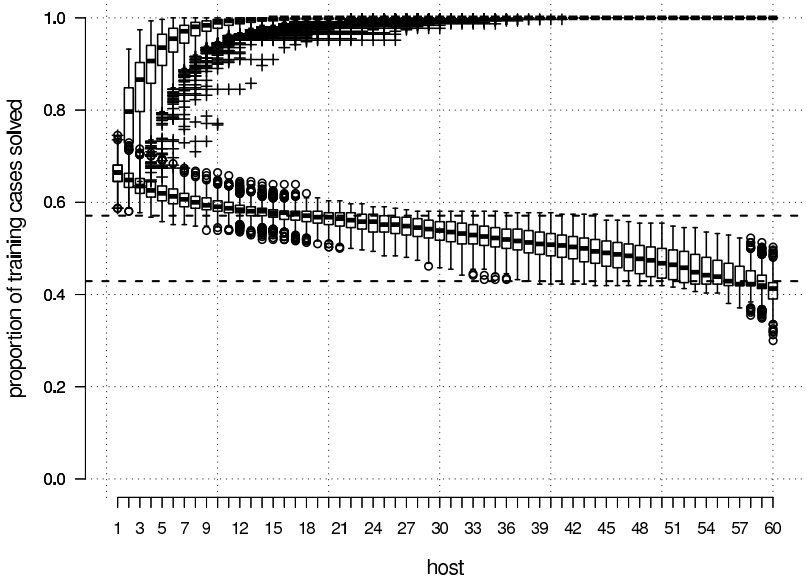


(a) Census.

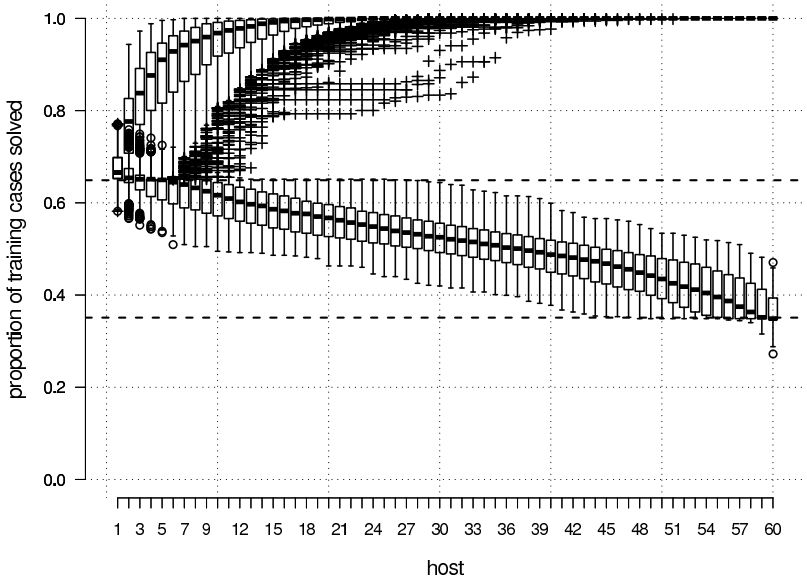


(b) Gisette.

Figure 4.20: Boxplots indicating cumulative and absolute hit counts on the training data across the final host population expressed as a fraction of the entire training dataset. At $x = 1$ the cumulative and absolute hit counts are the same, thereafter, the cumulative hit count always tends to be greater. Outliers are denoted by a '+' and a 'o' for the cumulative and absolute distributions respectively. The proportion of training instances of each class is also indicated by the dashed horizontal line.



(c) Bupa.



(d) Pima.

Figure 4.20: Continued.

4.3 Summary of Results

In summary, the results presented in Section 4.2.1 suggest that the symbiotic approach results in improved classification performance compared to the monolithic approach and that it also requires less training overhead. In addition, it was found that improved classification performance under the SBB approach was independent of the associated solution complexity, i.e., improved accuracy and mcdr did not come at the cost of more complex solutions. Support for the second claim made at the beginning of this chapter, regarding reduced solution complexities under the symbiotic approach, was not as strong; though the SBB complexities appeared to be smaller on the larger Census and Gisette datasets, no statistically significant differences were observed. However, the SBB distributions, including those corresponding to the solution complexities, were typically a lot more consistent, where this is viewed as a desirable property of any learning algorithm. Since minimal changes were applied to the codebase for the SBB approach in implementing the monolithic approach, these differences can be attributed, with confidence, to the inclusion or exclusion of the symbiotic metaphor.

The following are viewed as several key characteristics of the solutions evolved under the symbiotic metaphor as observed in Section 4.2.2:

1. In terms of the number of symbionts per host, the number of effective instructions per symbiont, and the number of unique features accessed per symbiont, the solutions appear to be simple.

2. Each symbiont accesses a small number of features regardless of the number of attributes in the underlying dataset.

3. Typically there is one symbiont that wins a large portion of instances, with the other members of the same host winning a relatively small portion.

4. Some solutions employ a thresholding approach that resembles the behaviour typically associated with default hierarchies, where this may allow for more concise solutions.

The same type of analysis as was presented in this section cannot be performed on the solutions evolved using the monolithic approach where explicit subcomponents are not defined.

A greater degree of problem decomposition, i.e., more symbionts per host, fewer effective instructions per symbiont, and fewer features accessed per symbiont, was

observed under Gisette, a dataset of moderate size and with a relatively high number of features. The SBB approach also did particularly well on this dataset compared to the monolithic approach. Thus, symbiosis may be especially appropriate for problems having these characteristics, where this is consistent with previous results involving the SBB algorithm [38]. However, the approach remains effective under datasets with tens of attributes such as Census as well as the smaller problems such as Bupa and Pima; in all three cases, solution quality was maintained while significant reductions in training overhead were observed.

Chapter 5

Additional Classification Results: Comparison with Boosting and Support Vector Machines

The primary goal of this chapter is to compare the SBB approach against well-established classification algorithms outside of the EC domain, providing what is perhaps a more conventional baseline from which the utility of the proposed approach can be assessed. This chapter builds on the results from Chapter 4 by considering two additional multi-class problems. Thus, the capacity of the symbiotic approach to support multi-class learning from a single population is also demonstrated.

The two classification algorithms used in the comparison with the proposed approach are a SVM [43] and the boosting algorithm AdaBoost [52]. The SVM algorithm was chosen because it is by some considered to be the state-of-the-art in classification [64], whereas AdaBoost, though perhaps not as cutting-edge, is nonetheless viewed as robust and capable of generating very strong classification results. Furthermore, the AdaBoost algorithm can be used to construct solutions consisting of multiple base classifiers which can be regarded as analogues to the symbionts in the SBB approach.

A premise underlying the comparison in this chapter is that both classification performance and solution complexity should be considered when evaluating the usefulness of an algorithm. As suggested in Section 1.1, solution complexity may affect solution transparency, i.e., it is likely that a smaller solution can be more easily interpreted. This then reflects the growing need for models that capture interesting relationships in potentially massive and complex datasets as opposed to focusing on classification performance alone [41, 122, 131, 186]. The proposed approach addresses this need by using GP to evolve the bid programs which assume a more human-readable form than, e.g., SVM or neural network classifiers. The programs evolved using GP could still be prohibitively complex so as to make them difficult to interpret; to this end, the proposed approach aims to break down the complexity by associating a single action with each bidding behaviour, i.e., through context learning. As suggested by the additional characterization of the SBB solutions in Chapter 4, this then results in very simple solution subcomponents which may lead to the desired levels of transparency.

The experiments leading to the results presented in this chapter are first described

in Section 5.1, including the boosting and SVM implementations, their parameters, and the datasets that were used. The three algorithms are then compared with respect to classification performance, Section 5.2.1, as well as solution complexity, Section 5.2.2; a summary of the results, including the observed tradeoff between classification performance and solution complexity, is presented in Section 5.3.

5.1 Experimental Setup

This section describes the experiments that were performed. It begins by presenting an overview of the two baseline algorithms and their key implementation details in Section 5.1.1. The datasets and parameters used are then described in Sections 5.1.2 and 5.1.3 respectively, while the criteria forming the basis of the evaluation are presented in Section 5.1.4.

5.1.1 Support Vector Machine and Boosting Implementations

Support Vector Machine

The typical, and in some ways idealized, application of the SVM training algorithm to a complex binary classification problem can be conceptually broken down into the following two steps [64]:

1. The non-linear mapping of the original input space to a new feature space with a higher, possibly infinite, dimensionality. As suggested by Cover's Theorem [28], the problem is more likely to be linearly separable in the new feature space than in the original input space[1].

2. A quadratic optimization to construct an optimal hyperplane in the higher-dimensional feature space that not only separates the points in one class from points in the other but also maximizes the *margin of separation*, i.e., the distance between the hyperplane and the closest data point.

Furthermore, the optimal hyperplane constructed through the above process can be characterized in term of a subset of the training data, i.e., the points that are closest to the hyperplane, while all other points can be discarded. These characterizing data points are referred to as the *support vectors*.

[1] In some cases, it may be sufficient to apply the non-linear transformation without necessarily increasing the dimensionality of the feature space [64].

The non-linear mapping from the original input space to the new feature space is commonly effected through the invocation of the *kernel trick* [64]. Specifically, the problem of finding the optimal hyperplane can be reformulated so that points in the new feature space appear only as factors in the inner product with other points in the same space. These inner product computations can then be implemented indirectly using an *inner-product kernel function* that specifies the outcome of the calculation for each pair of points in the original input space[2], e.g., the kernel may take the form of a square matrix containing an outcome for each such pair. Thus, even if the new feature space is infinite, the result of the inner product calculations can still be determined.

Despite the application of the non-linear transformation in the first step, it may still be the case that the problem is not linearly separable in the new feature space. The hard constraints related to separability in the quadratic program in the second step can therefore be relaxed through the introduction of *slack variables* which quantify the degree to which instances are misclassified [27]. This results in an additional penalty term in the objective function which now captures the tradeoff between a large margin of separation and a small classification error with respect to the training data. A cost parameter, C, can then be used to emphasize either quantity with higher values of C placing more emphasis on error minimization. Consequently, higher values of C typically result in more complex models as measured in terms of support vector counts, and may also lead to over-fitting.

The SVM implementation used in the comparison was LIBSVM Version 2.9 [24] employing Sequential Minimal Optimization for solving the underlying quadratic optimization problem [43]. LIBSVM was selected in this work because it is well documented, widely-utilized, and because it includes a suite of tools helpful in constructing classifiers. It supports various SVM types, and naturally, in this work the C-SVC formulation designed for binary classification was used. To implement the non-linear transformation, the radial basis function (RBF) kernel of the form $e^{-\gamma\|\mathbf{u}-\mathbf{v}\|^2}$ was chosen; this choice was based on previous experience with C-SVC on a number of the datasets used here [112]. In addition, the LIBSVM distribution includes a parameter selection tool which was used to find suitable values for the cost parameter C and the γ parameter that controls the width of the RBF kernel. To extend the C-SVC

[2]More formally, given an input space I and a transformation on the points in I, the kernel specifies the result of applying the transformation to two points in I and taking the inner product between the resulting images. However, the kernel typically abstracts this operation in that it does not explicitly apply the transformation nor perform the inner product calculation.

approach to multi-class problems, a one-against-one approach was employed where for an $|L|$-class problem $\frac{|L|(|L|-1)}{2}$ binary classifiers were constructed to distinguish between every pair of classes. To assign a class label to an input instance, each classifier was applied in turn voting for one of its associated classes, and the label with the most votes was output as the final result.

Boosting

The variant of boosting used in the comparison was AdaBoost.M1 [52] as implemented in version 3.6 of the WEKA data mining package [61]. It is common practice to use a decision stump, i.e., a single-level decision tree, as the base classifier in boosting. However, applied to multi-class problems, a decision stump is likely to yield accuracy rates below 0.5 and is therefore inappropriate for use with AdaBoost [160]. Instead of a decisions stump, this work therefore employed the C4.5 algorithm for generating decision trees [151] as implemented in the WEKA J48 class. In addition, resampling was used instead of reweighing to generate different training distributions in each boost iteration, i.e., the distributions were created by sampling the original data with replacement using probabilities proportional to the weights as assigned by the boosting algorithm[3]. Though typically reserved for base classifiers that are not compatible with instance weights, resampling was used instead of reweighing because under the WEKA implementation of AdaBoost it allows different initial conditions to be specified. This, in turn, allows multiple AdaBoost classifiers to be generated on a single dataset, facilitating comparison with the multiple SBB models. Furthermore, resampling is known not to adversely affect the boosting classification performance, and in fact, may actually outperform reweighing [162].

Details of the AdaBoost algorithm itself can be found elsewhere [52], while here, two properties that are relevant to the comparison in this chapter are noted. First, the algorithm stops adding base classifiers either when it reaches a limit that is fixed *a priori*, or when it produces a base classifier that yields a weighted error greater than 0.5 (at which point the base classifier is discarded). In the latter case, an early stop signals that the boosting process is failing, which is in contrast to other approaches where a premature stop often indicates that a solution of adequate quality has been found. With respect to the results presented in this chapter, the early stopping criterion may thus have an effect on the solution complexity because it affects how many base models are included in the final solution. The second property relates to

[3]In contrast to bagging where instances are uniformly sampled with replacement.

Dataset	Features	Classes	Training cases		Test cases	
			Total	Majority	Total	Majority
Census (cen)	40	2	199 523	187 141	99 762	93 576
Gisette (gis)	5 000	2	6 000	3 000	1 000	500
Bupa (bpa)	6	2	310	180	35	20
Pima (pma)	8	2	691	450	77	50
Thyroid (thy)	21	3	3772	3488	3428	3178
Shuttle (shu)	9	7	43500	34108	14500	11478

Table 5.1: Datasets used to compare the SBB approach with AdaBoost and a SVM. For Bupa and Pima, 10-fold cross-validation was used so the instance counts represent the mode values across all folds. The contents between the parentheses denote the abbreviation used to refer to the dataset.

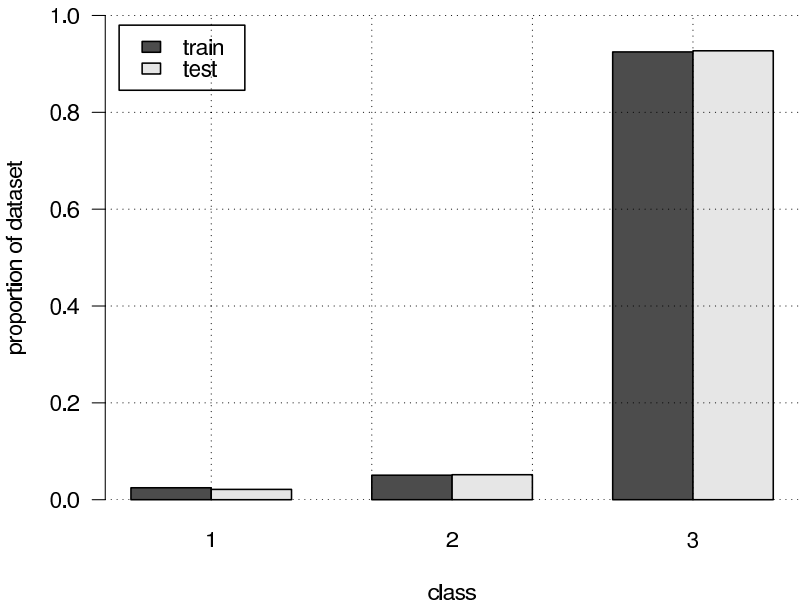
the way in which the base classifiers are combined into the final model, and in the case of AdaBoost, this is done through a weighted vote in which base classifiers that were found to be more accurate are assigned higher weights. This is representative of the ensemble learning methodology discussed in Section 2.2.3, in contrast to the bid-based action selection process used in the SBB approach, and may therefore result in base models with overlapping behaviours.

5.1.2 Datasets

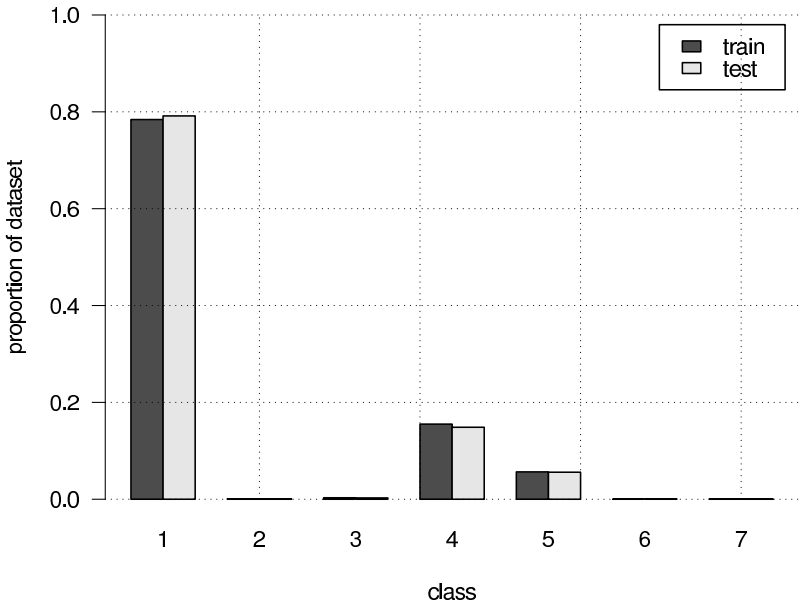
Six datasets, Table 5.1, were used to compare the SBB approach with AdaBoost and the SVM algorithm. All the datasets were obtained from the UCI Machine Learning Repository [3]. Census, Gisette, Bupa, and Pima were used in the comparison with the monolithic approach in Chapter 4; an overview of these datasets, including any preprocessing, is given in Section 4.1.2. The information from Table 4.1 is repeated here to provide a complete summary across all six problems.

Thyroid refers to the 'ann' dataset in the Thyroid Disease database. It contains fifteen binary attributes, six continuous attributes, and three classes. Shuttle, taken from the Statlog Shuttle database, contains nine attributes all of which are numerical. Both of these datasets are unbalanced, with the majority class accounting for close to 93% and 79% of all instances for Thyroid and Shuttle respectively, Figure 5.1. In both cases the original training and testing partitions were used in the evaluation, and with the exception the SVM, no additional preprocessing was performed.

For use with the SVM algorithm, attribute values in all *six* datasets were linearly scaled. First, the attributes in the training partition were transformed to fall within



(a) Thyroid.



(b) Shuttle.

Figure 5.1: Distribution of instances in the training and test partitions under Thyroid and Shuttle expressed as a proportion of the entire partition. From left to right, the number of Thyroid training (test) instances is 93 (73), 191 (177), and 3488 (3178), while the number of Shuttle training (test) instances is 34108 (11478), 37 (13), 132 (39), 6748 (2155), 2458 (809), 6 (4), and 11 (2).

± 1. The ranges used to scale the training data were then applied to the test data.

5.1.3 Parameters

Experiments involving the SBB approach on the binary datasets were performed earlier, Chapter 4, and those results are used in the comparison with the SVM and boosting algorithms presented in this chapter. For consistency, the same parameters as detailed in Section 4.1.3 were used when training the SBB solutions on Thyroid and Shuttle. In particular, 60 initializations of the SBB algorithm were performed on each training partition.

With one exception, the AdaBoost parameters were left at their default values as specified in WEKA 3.6. The one exception already mentioned was that resampling was used instead of reweighing. The default number of boost iterations, set at 10, matches the maximum number of symbionts in a host in the SBB experiments, Table 4.3. The parameters for the base classifier, C4.5, were also left at their default values, and in particular, the confidence threshold for pruning was set at 0.25. As in the case of the SBB parameters, 60 initializations of the AdaBoost algorithm were performed on each training partition, where this was made possible through the use of resampling.

The SVM parameter tuning follows the recommendations made in the LIBSVM documentation. Assuming the RBF kernel of the form $e^{-\gamma\|\mathbf{u}-\mathbf{v}\|^2}$, the LIBSVM parameter selection tool was used to find suitable values for the kernel parameter γ and the cost parameter C. Specifically, the selection tool was set to conduct a grid search across each combination of $\gamma \in \{2^{-15}, 2^{-13}, \ldots, 2^{3}\}$ and $C \in \{2^{-5}, 2^{-3}, \ldots, 2^{15}\}$, a total of 110 parameter pairs. Each combination of parameter values was evaluated using 5-fold cross-validation on instances of the training data selected as follows:

- Census, Shuttle. Given the size of the original training partitions in these two datasets, 10 000 instances were selected using stratified random subset selection.

- Gisette. Though not as large as some of the other datasets, Gisette contains a much larger number of features. To make the grid search computationally feasible, a subset of 1 000 instances, again selected using stratified random subset selection, was used to evaluate each combination of values.

- Bupa, Pima. As detailed in Section 4.1.2, 10-fold cross-validation was used to evaluate the test performance under Bupa and Pima. The grid search was

performed on each of the 10 training partitions in their entirety yielding a pair of parameter values for each partition.

- Thyroid. The parameter selection was conducted across the entire Thyroid training partition.

The result of each 5-fold cross-validation was used as an estimate of the classification performance associated with that particular setting of γ and C. The pair of values yielding the highest accuracy was then selected for use in the comparison, Table 5.2. As such, under Bupa and Pima, the SVM parameter values could vary across each of the ten folds.

In contrast to the SBB and AdaBoost (with resampling) algorithms which allow different initial conditions to be specified, a single SVM initialization was performed on each training partition resulting in 10 points under Bupa and Pima and one point under the other four problems.

5.1.4 Evaluation Methodology

The comparison between the SBB, AdaBoost, and SVM algorithms is made with respect to classification performance and solution complexity. Classification performance is analyzed in terms of accuracy, Eq. 4.1, and the mcdr, Eq. 4.2. As suggested in Section 4.1.4, mcdr is more informative on problems where the class distributions are unbalanced, and in particular, on the two multi-class datasets introduced in this chapter. However, accuracy is included in the evaluation since it was used in selecting the SVM parameter settings. Based on previous observations, Table 4.7, on the more balanced datasets such as Gisette, Bupa, and Pima, it is expected that these two measures will be correlated.

The way in which solution complexity is expressed depends on the learning algorithm. SBB solutions are characterized in terms of the number of effective instructions, the number of symbionts, and the number of unique features accessed, all calculated across the entire host. The ensembles produced by AdaBoost are characterized in terms of the number of decision trees (alternatively, boosting iterations[4]), nodes (internal and leaf), and unique features accessed, where all quantities are calculated across the entire ensemble. The SBB and AdaBoost unique feature counts

[4]If the ensemble contains fewer than the maximum number of trees, then the number of boosting iterations that was performed is actually one more than the final number of trees. In this case, the tree generated in the last boosting iteration is discarded.

	Fold	γ	C	Accuracy
Census	–	2^{15}	2^{-11}	0.9465
Gisette	–	2^{1}	2^{-11}	0.9450
Bupa	0	2^{9}	2^{-3}	0.7484
	1	2^{13}	2^{-5}	0.7419
	2	2^{5}	2^{1}	0.7290
	3	2^{5}	2^{-3}	0.7355
	4	2^{9}	2^{-5}	0.7387
	5	2^{13}	2^{-7}	0.7556
	6	2^{9}	2^{-3}	0.7428
	7	2^{9}	2^{-3}	0.7460
	8	2^{9}	2^{-5}	0.7492
	9	2^{15}	2^{-7}	0.7331
Pima	0	2^{9}	2^{-7}	0.7800
	1	2^{7}	2^{-7}	0.7800
	2	2^{1}	2^{-5}	0.7713
	3	2^{9}	2^{-13}	0.7844
	4	2^{13}	2^{-9}	0.7829
	5	2^{9}	2^{-7}	0.7771
	6	2^{-1}	2^{-3}	0.7685
	7	2^{3}	2^{-5}	0.7873
	8	2^{5}	2^{-5}	0.7760
	9	2^{3}	2^{-3}	0.7818
Thyroid	–	2^{13}	2^{-5}	0.9799
Shuttle	–	2^{15}	2^{-3}	0.9984

Table 5.2: SVM parameters used in the experiments under the RBF kernel as determined using a grid search on γ and C. For each specified parameter setting, the accuracy refers to the mean accuracy obtained in the 5-fold cross-validation on the training data, and represents the highest value obtained across all 110 combinations of the parameters. Under Bupa and Pima, a pair of parameters is associated with each training partition.

can be meaningfully compared, while the number of decision trees in an ensemble is analogous to the number of symbionts in a host. Counting the number of features accessed by the SVM solutions is unnecessary since all the SVM models always access all of the features. The other measure of SVM solution complexity considered is the number of support vectors.

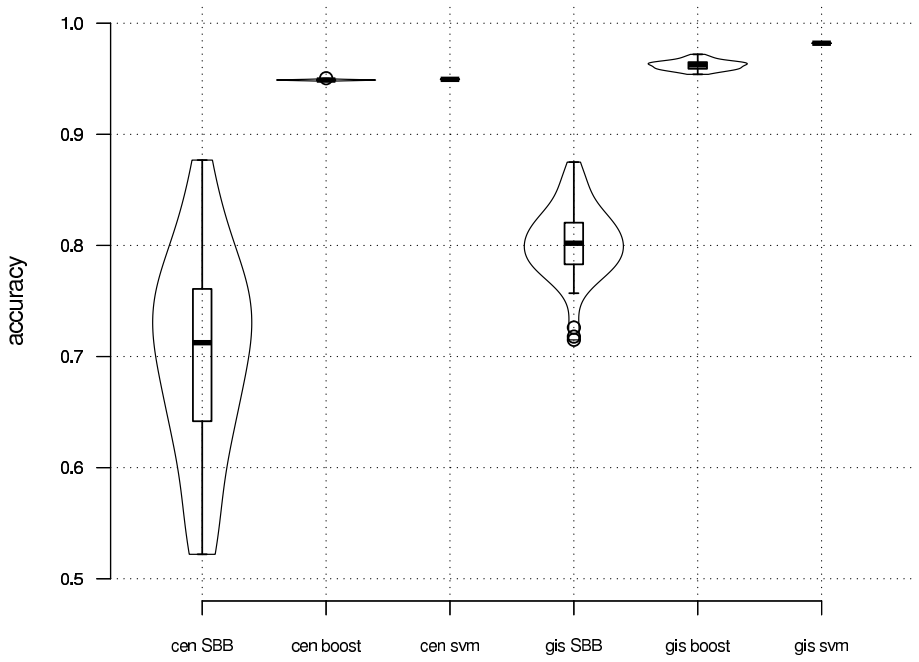
As in previous experiments, Section 4.1.4, the SBB results correspond to the host in the population at the end of training that yields the highest mcdr on the entire training set. In contrast, the SVM and AdaBoost algorithms yield a single solution per initialization, making post-training model selection not applicable. In all cases, the reported accuracy and mcdr values are with respect to the test partition.

5.2 Results

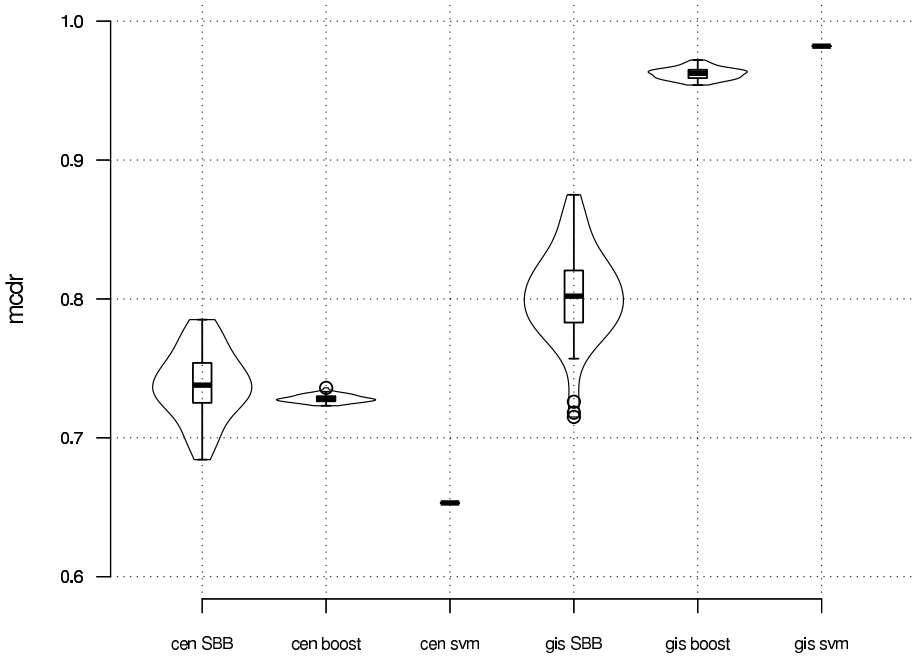
5.2.1 Classification Performance

Figures 5.2 through 5.4 compare the three algorithms with respect to accuracy and mcdr. The datasets are considered pairwise based on the following characteristics: binary and large instance/attribute space (Census and Gisette), binary and small but historically difficult [116] (Bupa and Pima), and multi-class (Thyroid and Shuttle). Performing 60 initializations of the SBB and AdaBoost algorithms on each training partition results in 600 points in the distributions under Bupa and Pima and 60 points under the other problems. In contrast, a single initialization of the SVM algorithm was performed on each training partition resulting in 10 points under Bupa and Pima and a single point on the other problems.

Compared to the SVM and AdaBoost algorithms, the SBB results appear to be competitive on Bupa, Pima, Thyroid, and Shuttle, Figures 5.3 and 5.4, but the relative SBB performance degrades on Census and Gisette, Figure 5.2. With the exception of the mcdr under Census, the SVM and AdaBoost distributions on Census and Gisette indicate significantly higher values, i.e., the AdaBoost and SVM distributions do not overlap with the SBB distributions, and they also exhibit significantly less variability. Under Bupa and Pima, Figure 5.3, the spread in the values in the distributions corresponding to all three approaches is greater than on the other problems. This, combined with a lot of overlap in the distributions, makes the identification of a dominant approach less obvious. Though the median SBB values are lower than the median SVM and AdaBoost values, the higher spread in the SBB distributions results in the top hosts performing at similar levels as top solutions under the other two

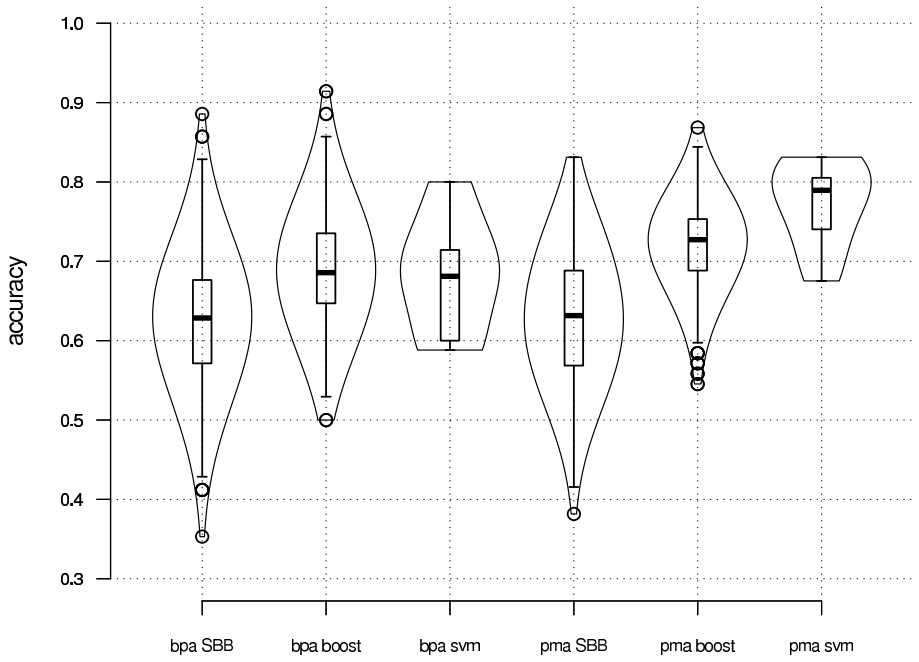


(a) Classification performance with respect to accuracy.

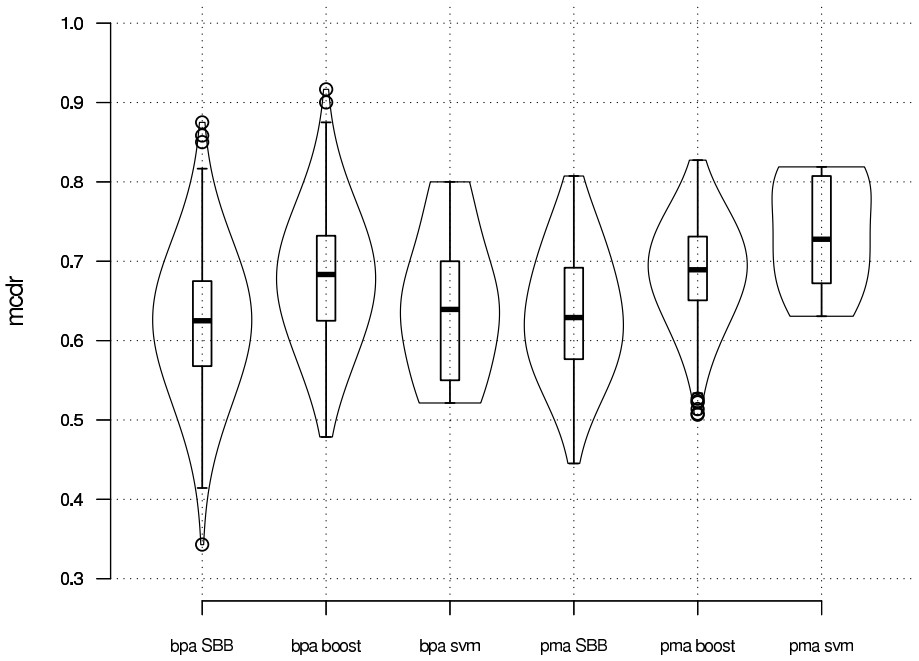


(b) Classification performance with respect to mcdr.

Figure 5.2: Comparison of the SBB approach with AdaBoost and SVM, test partition accuracy and mcdr on Census and Gisette. The boosting and SVM distributions are labeled 'boost' and 'svm' respectively.

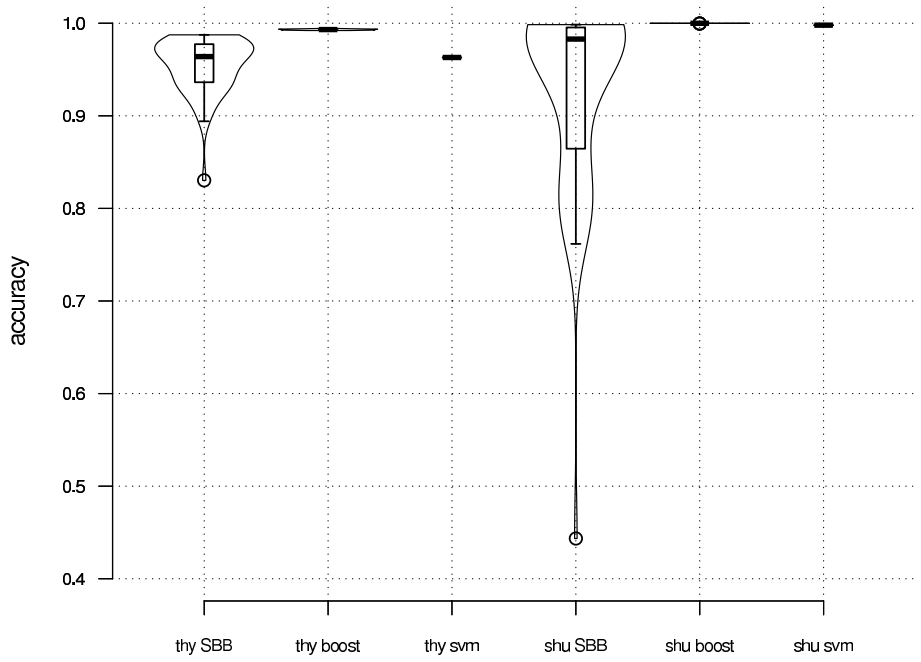


(a) Classification performance with respect to accuracy.

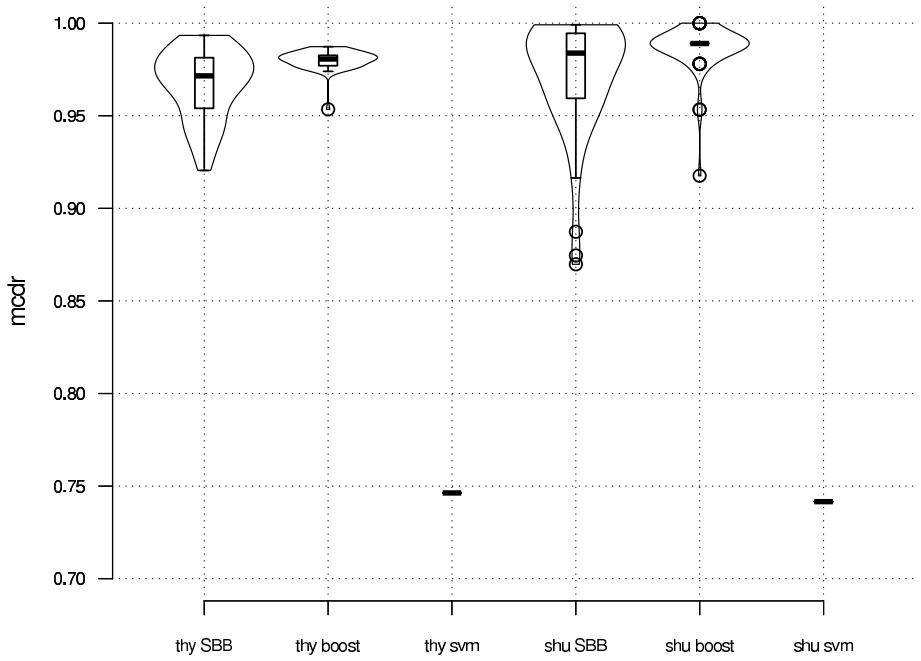


(b) Classification performance with respect to mcdr.

Figure 5.3: Comparison of the SBB approach with AdaBoost and SVM, test partition accuracy and mcdr on Bupa and Pima. The boosting and SVM distributions are labeled 'boost' and 'svm' respectively.



(a) Classification performance with respect to accuracy.



(b) Classification performance with respect to mcdr.

Figure 5.4: Comparison of the SBB approach with AdaBoost and SVM, test partition accuracy and mcdr on Thyroid and Shuttle. The boosting and SVM distributions are labeled 'boost' and 'svm' respectively.

	Census	Gisette	Bupa	Pima	Thyroid	Shuttle
accuracy	≈0	≈0	≈0	≈0	≈0	≈0
mcdr	0.0003842⋆	≈0	≈0	≈0	1.273e-05	0.003053

Table 5.3: Two-tailed Mann-Whitney test results comparing SBB solutions with AdaBoost on six classification problems, test partition accuracy and mcdr. Shown are the p-values obtained when the test was applied to each pair of corresponding accuracy and mcdr distributions, Figures 5.2 through 5.4. Cases where the SBB median value is *higher* are noted with a '⋆'. The comparison does not include the SVM results where only a single data point is available for each problem.

approaches. Finally, under Thyroid and Shuttle, Figure 5.4, both the SBB approach and AdaBoost achieve high accuracy and mcdr values. In contrast, the SVM solutions perform well with respect to accuracy but their associated mcdr values are poor.

The SVM solutions yield low mcdr values on Census, Thyroid, and Shuttle, suggesting that the SVM algorithm is less effective when the class distributions are unbalanced. A contributing factor may be that the SVM parameters were tuned with respect to accuracy. As noted, for this reason, accuracy was included in the comparison, and naturally the SVM solutions are expected to have an advantage when accuracy is considered. Furthermore, the SBB and AdaBoost solutions on Thyroid and Shuttle achieve both high accuracy and mcdr values, so the two objectives are not mutually exclusive.

The classification performance of the SBB solutions was compared with AdaBoost using a set of two-tailed Mann-Whitney tests, Table 5.3. The tests do not include the SVM results because for each training partition only a single data point is available under that algorithm. The hypothesis tests indicate that the AdaBoost classification performance is considerably better than the SBB performance in all but one comparison. The sole exception, consistent with previous observations, is the mcdr result on Census. Given that the AdaBoost and SVM algorithms are well established approaches tailored towards classification, their strong classification performance is to be expected. Perhaps through a more involved application of the algorithm, e.g., using domain-specific optimizations as is generally the case in GP [198], stronger SBB performance could be obtained. However, it is nonetheless emphasized that in certain situations the end-user may be willing to compromise classification performance in favour of a simpler solution, where this tradeoff is further discussed in Section 5.3.

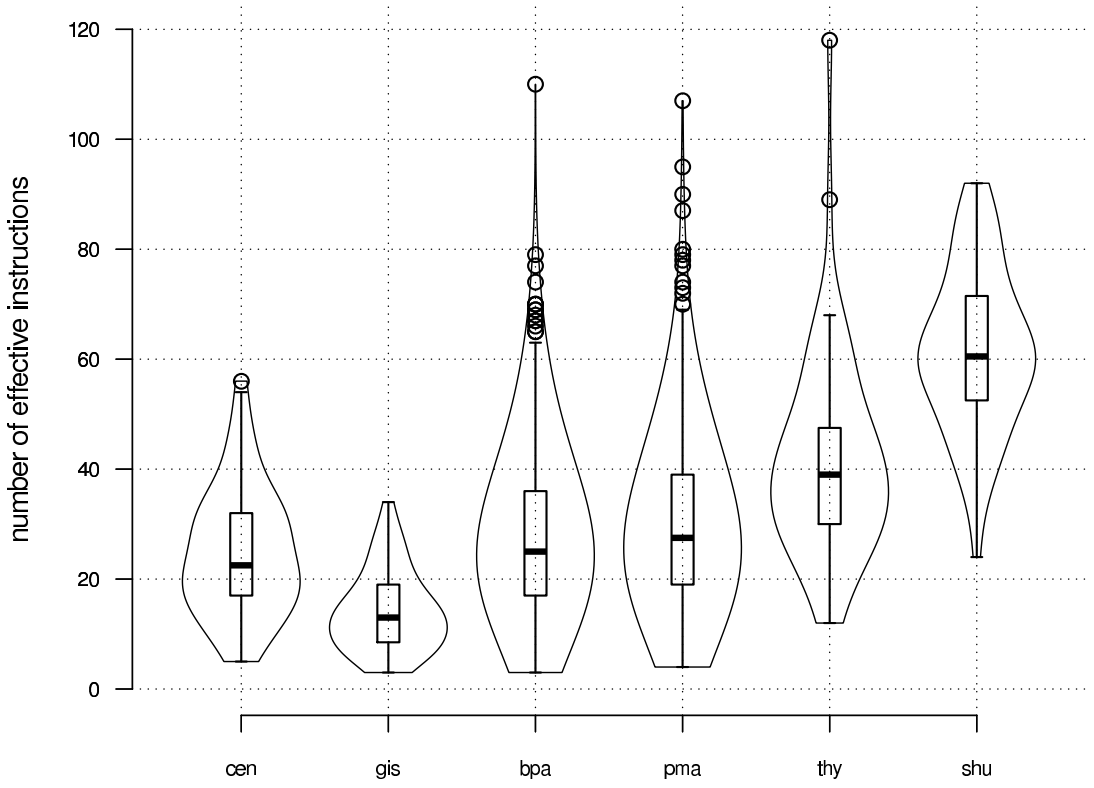


Figure 5.5: Number of effective instructions in the SBB solutions under six classification problems aggregated across the entire host.

5.2.2 Solution Complexity

Figures 5.5 through 5.9 summarize the complexity of the solutions generated by the three approaches with respect to the algorithm-specific building components making up the solution, i.e., instruction/symbiont, decision tree, or support vector counts. The complexities of the SBB solutions, Figures 5.5 and 5.6, are higher on Thyroid and Shuttle compared to the other four datasets. This is likely because Thyroid and Shuttle are multi-class, and as such, minimalist solutions require a greater number of symbionts in order to represent each class. Indeed, the highest host sizes are observed under Shuttle which contains seven classes – on this problem, the minimum observed host size is in fact seven. However, on the two multi-class problems, the sizes of the individual symbionts in terms of effective instruction counts appear to remain low. Specifically, once host-wide instruction counts normalized by host sizes are considered, less than 10 instructions per symbionts are typically observed.

The number of support vectors in the final SVM solutions, Figure 5.7, varies

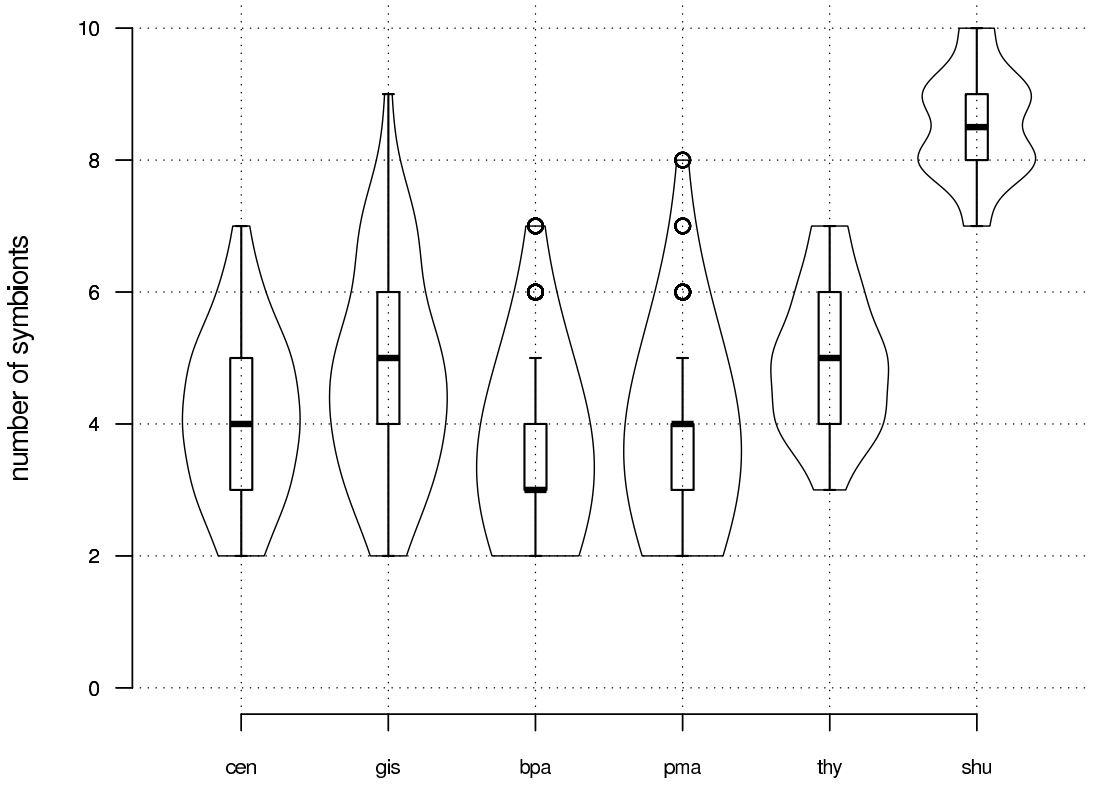


Figure 5.6: Host sizes under six classification problems in terms of the number of symbionts after pruning of symbionts.

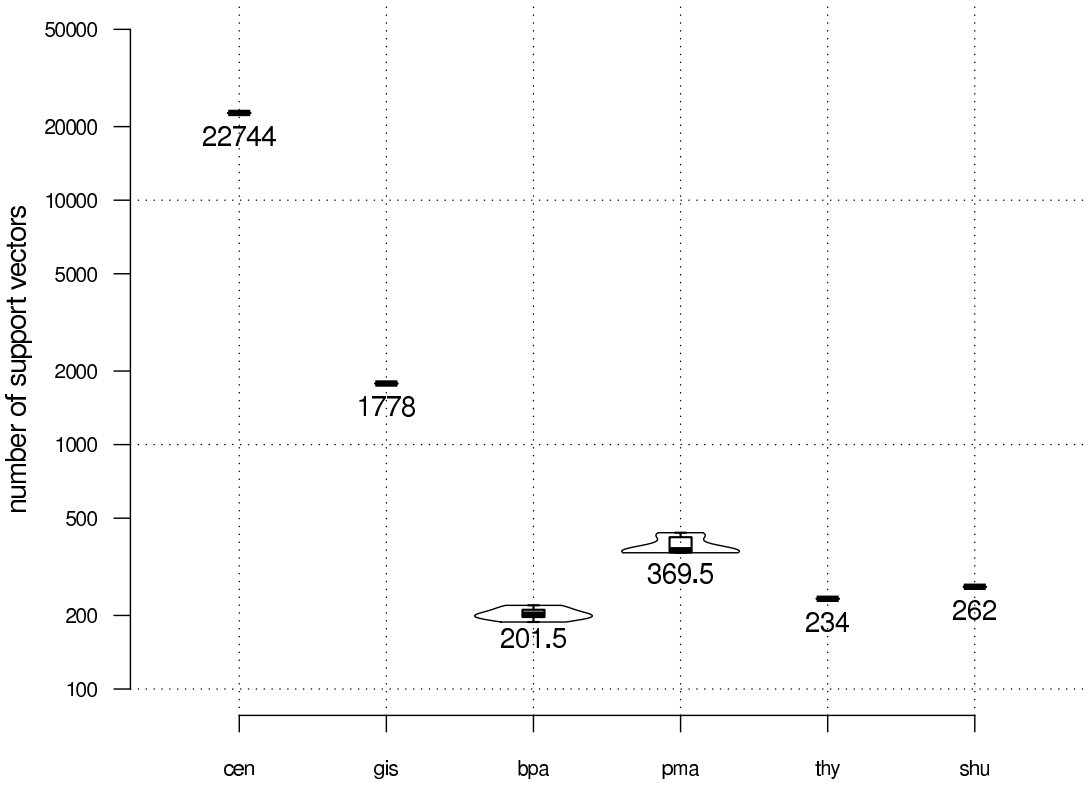


Figure 5.7: Number of support vectors in the SVM solutions under six classification problems. Median values are noted below each distribution.

significantly depending on the dataset. This is reflected in the use of a logarithmic scale on the y-axis. Specifically, the counts tend to be high when the dataset is large either in terms of the number of training instances, as in Census, or the number of attributes, as in Gisette. Such a high spread in complexities under different problems is not observed under the SBB approach, Figures 5.5 and 5.6, suggesting that the SBB approach scales better to larger problems such as Census and Gisette. This analysis may be over-simplified because on Shuttle, the second largest dataset in terms of the number of instances with 43500 training cases, relatively low support vector counts are observed. In this case, two contributing factors may also be that Shuttle contains only nine attributes, as well as the exceptional SVM performance on this problem with respect to accuracy, Figure 5.4a. Indeed, the SVM performance on this dataset with respect to the mcdr is poor, Figure 5.4b, and improvements in this regard may require substantial increases in model complexity.

Figures 5.8 and 5.9 respectively show the number of decision trees and the number of nodes across the AdaBoost ensembles. Given that in this context symbionts and

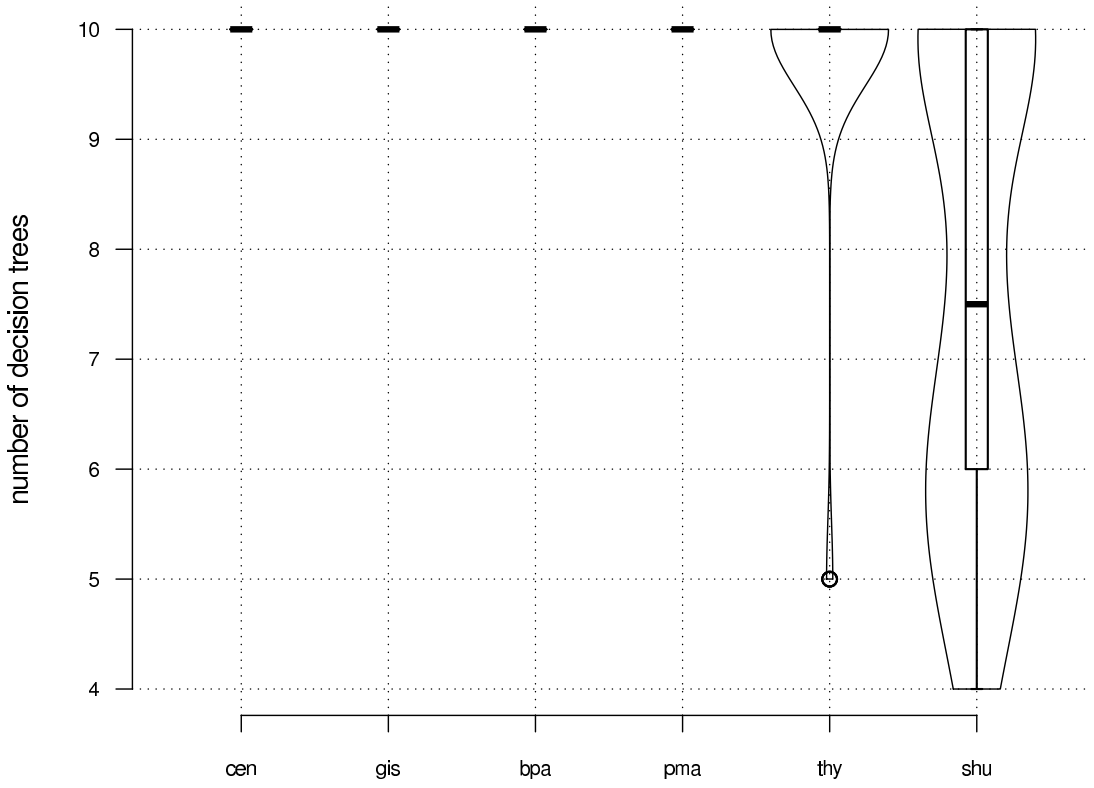


Figure 5.8: Number of C4.5 decision trees in the AdaBoost ensembles under six classification problems.

decision trees can be viewed as counterparts, the maximum number of these units was set at 10 in both cases. Figure 5.8 suggests that boosting tends to use more trees on the binary datasets and fewer trees on the multi-class problems. This is the reverse of the behaviour observed under the SBB approach, Figure 5.6, where more symbionts in a host are observed under the multi-class problems. Thus, as the number of problem classes increases, the likelihood that boosting produces a base classifier that yields a weighted error greater than 0.5 appears to increase as well, suggesting an upper limit beyond which additional boosting iterations are not applicable. In contrast, these results suggest that the SBB approach has the capacity to decompose the problem among multiple symbionts so that the investment of additional computational resources may lead to improved performance.

With respect to the number of nodes across all trees in the AdaBoost ensembles, Figure 5.9, the trends observed are similar to the trends in the support vector counts, Figure 5.7. In particular, the complexity of the boosting models varies significantly across different problems, e.g., as in Figure 5.7 a logarithmic scale is used on the

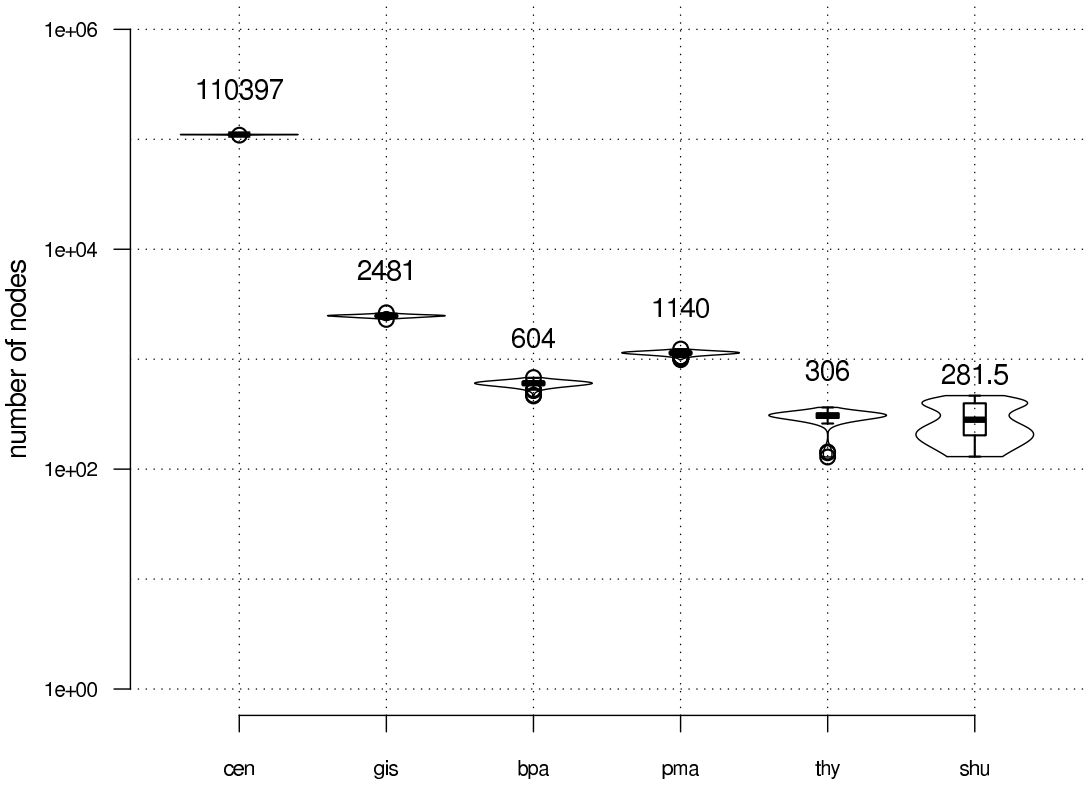


Figure 5.9: Number of nodes in the AdaBoost ensembles under six classification problems. The counts are aggregated across all trees in an ensemble, and include internal and leaf nodes. Median values are noted above each distribution.

y-axis, and boosting does not appear to scale as well as the SBB algorithm when the problems are large in terms of the number of attributes or training instances.

A comparison between the algorithms is also made with respect to the number of unique features accessed across the entire final solution, e.g., across all symbionts in a host or decision trees in an ensemble. The counts, excluding the SVM solutions, are shown in Figure 5.10; the SVM models always access all the attributes in each problem and are therefore omitted from the plots. On the two problems with the most features and where consequently maintaining low feature access counts may be most important, Figure 5.10a, the SBB solutions require fewer features compared to the AdaBoost and SVM solutions. This observation is supported by a two-tailed Mann-Whitney test at a 0.01 significance level, Table 5.4. Under the other four problems, where the total number of attributes is not as large and the unique feature counts may therefore be less relevant, the results across the three algorithms appear to be more similar. However, with the exception of Thyroid where no difference is indicated

Census	Gisette	Bupa	Pima	Thyroid	Shuttle
≈0⋆	≈0⋆	≈0	≈0⋆	0.5819	4.899e-05

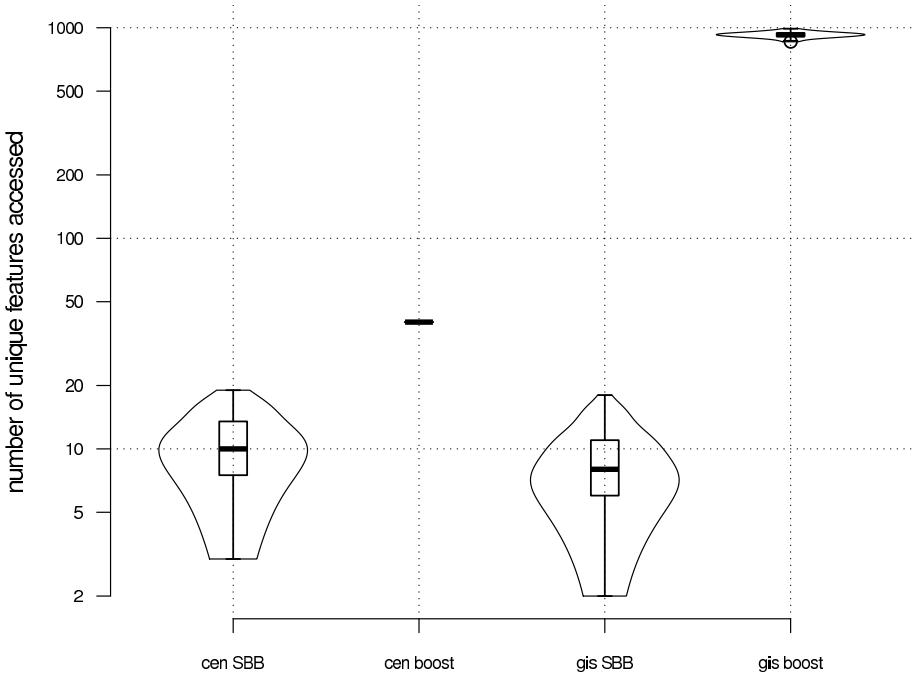
Table 5.4: Two-tailed Mann-Whitney test results comparing the SBB solutions with AdaBoost on six classification problems with respect to the number of unique features accessed. Shown are the p-values obtained when the test was applied to each pair of corresponding distributions in Figure 5.10. Cases where the SBB median value is *lower* are noted with a '⋆'. The comparison does not include the SVM results where only a single data point is available for each training partition.

between the SBB and AdaBoost counts at a 0.01 significance level, Table 5.4, the results suggest that the SBB feature access counts on these other four problems are also significantly lower.

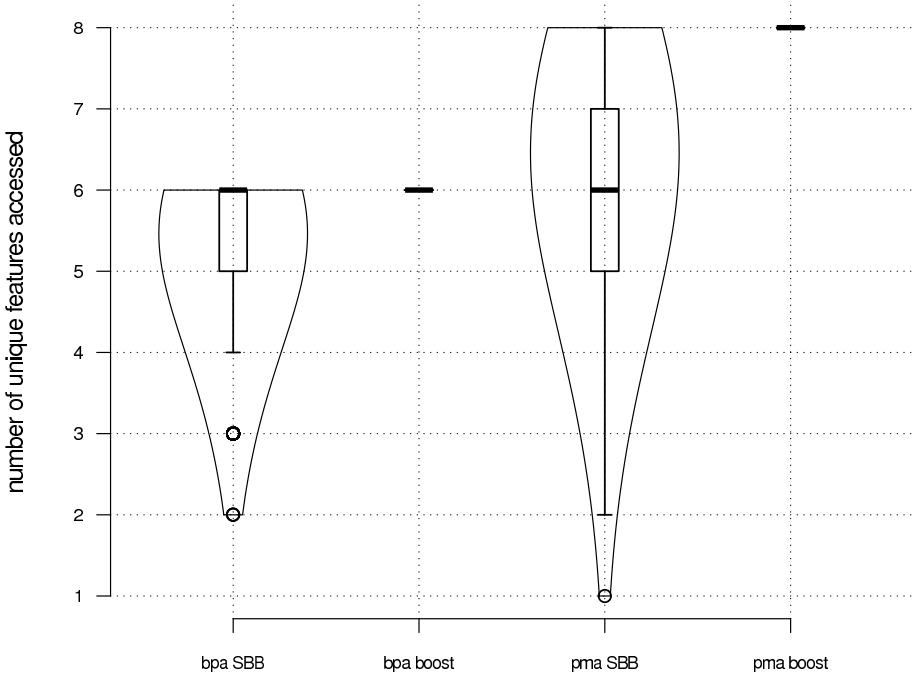
Finally, for the SBB and AdaBoost solutions, the amount of overlap in the features accessed was calculated, Figure 5.11. The SVM solutions were not considered since all support vectors in a model access the same features, i.e., all the available features. Depending on the algorithm, the overlap was considered with respect to symbionts in a host or decision trees in an ensemble, and the calculation itself was performed as per Eq. 4.3. Figure 5.11 indicates that on all problems the amount of overlap in features accessed under AdaBoost is greater than under the SBB approach. On some of the problems, i.e., Census, Bupa, and Pima, no feature is ever accessed by just one of the trees in an AdaBoost ensemble. These results highlight the difference in which the two approaches construct a solution. Whereas the SBB algorithm tends to decompose the problem among the symbionts, successive boosting iterations tend to result in a refinement of solutions found in previous iterations. More concretely, the SBB approach encourages diversity through the application of an explicit penalty for overlapping behaviours, whereas the AdaBoost approach to ensemble construction encourages overlapping behaviour to provide a pool of collaborating voters, i.e., overlap is desired when the majority in the pool makes the correct decision. These contrasting behaviours are then reflected in the attribute association under the two approaches.

5.3 Summary of Results

It was suggested at the end of Section 5.2.1 that if the SBB algorithm is competitive with the AdaBoost and SVM algorithms with respect to classification performance while providing simpler solutions, then the SBB solutions may be favoured by the

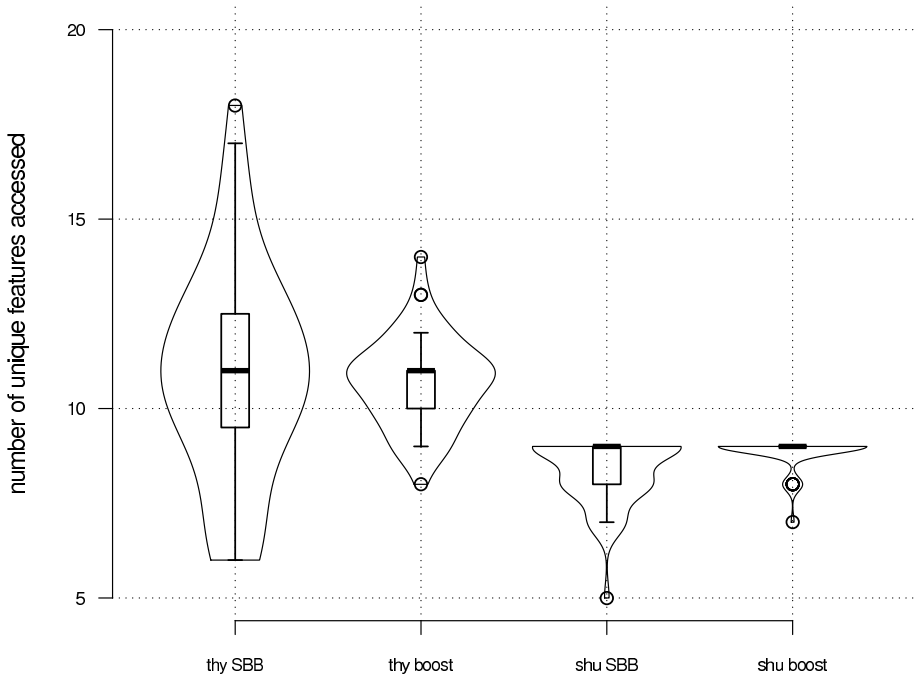


(a) Census and Gisette.



(b) Bupa and Pima.

Figure 5.10: Number of unique features accessed by the SBB and AdaBoost solutions under six classification problems. For each solution, all symbionts or decision trees in the solution are considered.



(c) Thyroid and Shuttle.

Figure 5.10: Continued.

end-user even if they are slightly less effective. The problem domains where the SBB algorithm brings the greatest advantage in terms of solution complexity were found to be Census and Gisette. However, under Gisette, the SBB algorithm clearly resulted in weaker classification performance than either of the other two approaches, and was competitive under Census but only with respect to the mcdr. Thus, on these two problems, the complexity-performance tradeoff is especially apparent.

Viewing instructions, support vectors, and decision tree nodes as very rough equivalents, the SBB instruction counts can be compared with the SVM support vector counts and the AdaBoost node counts. Across all problems, the SBB instruction counts were observed to be lower – however, no hypothesis testing was performed since the respective units were viewed to be insufficiently similar. With respect to the unique feature counts, a significant difference was found in favour of the SBB approach on all problems except Thyroid. Comparing the overlap in features accessed by the symbionts in a host and the trees in an AdaBoost ensemble, it was found that the amount of overlap among the trees was greater. This highlighted the differences in the two approaches whereby one relies on problem decomposition and the other relies on solution refinement.

In summary, compared to the solutions produced by the AdaBoost and SVM

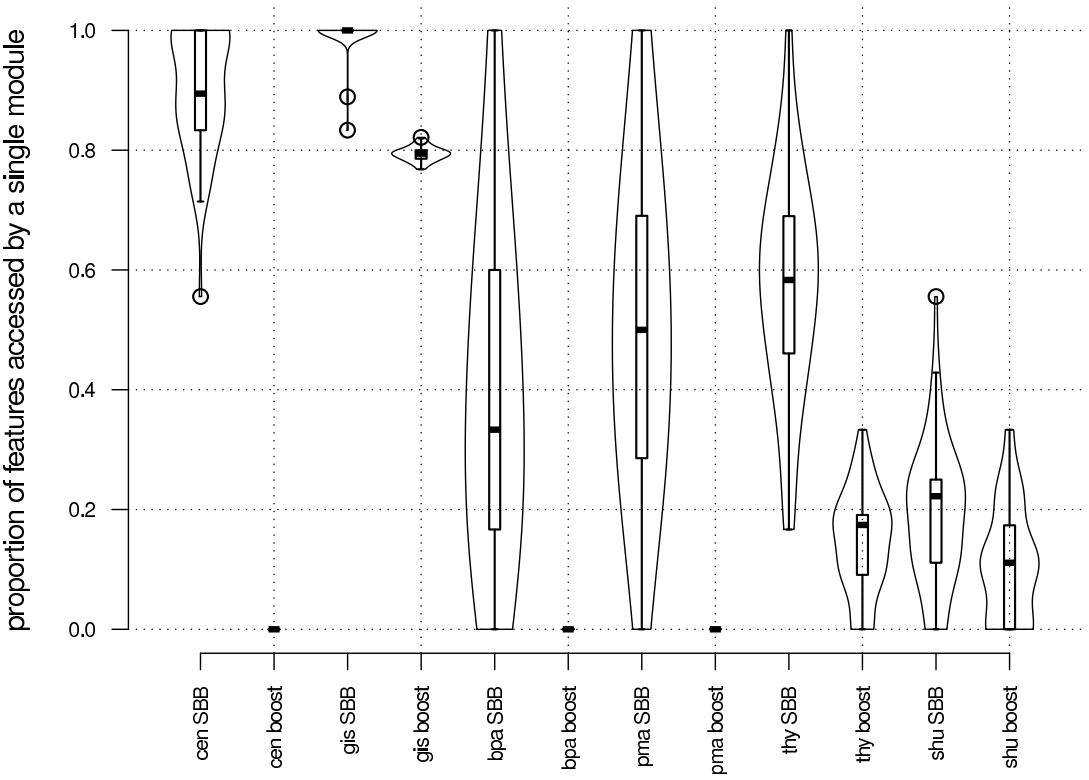


Figure 5.11: Proportion of features, with respect to the unique features accessed across a solution, that are accessed by exactly one module in the solution. The modules correspond to symbionts and trees under the SBB and AdaBoost approaches respectively. Lower values indicate more overlap in the features accessed by the modules in the same solution, i.e., fewer features accessed by just a single module. The figure is an extension of Figure 4.17 that also includes AdaBoost and considers feature counts on Thyroid and Shuttle.

algorithms, the SBB solutions were found to be simpler, particularly on the larger problem domains. In many cases, the tradeoff between classification performance and solution complexity was found to be present, with the two baseline algorithms yielding stronger classification performance. If more consistent performance could be achieved at the upper end of the spectrum, either through parameter tuning or modifications to the algorithm itself, the SBB algorithm could become a useful alternative to the mainstream classification algorithms provided solution complexity remains low. Finally, it is emphasized that the relative simplicity of the SBB solutions is attributed to the focus in the approach on evolving non-overlapping behaviours, e.g., through implicit or explicit fitness sharing, whereas standard practice in ML tends to emphasize solution subcomponents whose behaviours overlap as in the case of AdaBoost.

Chapter 6

Test Case Selection in Temporal Sequence Learning under Binary and Real-Valued Rewards

A key component of the SBB algorithm is the coevolutionary interaction between the host and point populations, center-left of Figure 3.3, which allows the approach to scale to problems with many training cases. The design of this component involves many decisions, and in this chapter, the focus is on the policies for managing the contents of the point population. In particular, the choice of a binary versus a real-valued cost function is investigated under temporal sequence learning, where this then forms the basis for the environment definition assumed in Chapters 7 and 8.

Binary cost functions, indicating either success or failure, are typically easier to specify but provide no information about the quality of partial solutions, i.e., they often result in a search process analogous to that of a 'needle in a haystack'. Real-valued functions, on the other hand, tend to be a lot more informative in that they credit partial solutions, but this also means that they may not be as direct in guiding the search, e.g., a high fitness may be associated with an individual that partially solves many training cases but does not actually solve the problem. As such, more care must be taken in the specification of real-valued functions. Furthermore, whereas a real-valued function may perform adequately under a random test case sampling heuristic, i.e., an arbitrarily selected point will often be informative, binary cost functions are likely to require more sophisticated mechanisms for maintaining an informative set of training cases [113].

To investigate some of the issues involved in the design of the cost function, in this chapter, the SBB algorithm is compared against random and static subset selection strategies under binary and real-valued rewards. In addition, the effectiveness of the proposed point fitness function, Section 3.2.6, is established through a comparison with a distinction-based formulation, as well as a formulation where genotypic point diversity is not explicitly encouraged. Based on the results that are presented, the assertion is made that the additional cost of calculating the genotypic diversity component in the point fitness function is worthwhile, especially under temporal sequence learning where a lot of computational overhead is already expanded in performing

simulations.

In the process, as this chapter introduces the truck reversal environment, the problem domain changes to temporal sequence learning. In this environment, the inputs and output are continuous[1], where this provides a contrast to the discrete Rubik's Cube environment that the SBB algorithm is applied to in Chapter 8. In addition to providing an instance of a continuous problem, a key motivation behind the choice of problem in this chapter was that the associated states can be easily visualized, thus facilitating analysis of the host and point populations that are evolved. The difficulty of a given problem formulation depends on many factors including the particular problem parameters and constraints; the performance of the Neuroevolution of Augmenting Topologies (NEAT) algorithm [177] on the truck reversal problem, presented in Chapter 7, establishes the difficulty of the formulation used here.

The core SBB algorithm and the associated parameter settings remain unchanged compared to the previous experiments in the classification domain. This illustrates the robustness of the algorithm, in terms of its applicability to different problems, as well as its associated parameter settings.

The structure of this chapter is as follows. The SBB algorithm components that differ relative to the version of the algorithm used in previous chapters are briefly summarized in Section 6.1. Here, it is noted that the changes are only with respect to the point population, and that these changes are also in effect in Chapters 7 and 8. The truck reversal environment, including the point search operators and point distance functions used, is described in detail in Section 6.2. Section 6.3 describes the parameter settings, evaluation methodology, and variants of the SBB algorithm that are compared. Finally, the results and the conclusions that they lead to are presented respectively in Sections 6.4 and 6.5.

6.1 Algorithm Details Specific to Temporal Sequence Learning

In contrast to classification, test cases under temporal sequence learning represent only the starting states, and more importantly, this space of starting conditions is typically structured so that variation operators can be meaningfully applied to generate offspring points from parent points. Thus, as a result of the switch in this chapter from classification to temporal sequence learning, a number of changes to the SBB algorithm restricted to point initialization, generation, and selection have been made. These changes, made relative to the version of the algorithm as used in Chapters 4

[1]However, in order to apply the SBB algorithm, the angle of the front wheels was discretized.

and 5, also apply in Chapters 7 and 8. They are as follows:

1. Instead of using a class-balanced point sampling heuristic, the point population is initialized to contain a random set of points.

2. The class-balanced point sampling heuristic is also no longer used during point generation. Instead, in the majority of cases, new points are generated from existing points through a variation operator, with an occasional random point added to help maintain diversity.

3. The distance between all points is no longer assumed to be one. Therefore, the normalization rewarding genotypically unique points is no longer bypassed.

These changes have already been discussed in general terms in Sections 3.2.1, 3.2.3, and 3.2.6. In the context of the truck reversal environment, the first two items, point initialization and point generation, are described in Section 6.2.3. The distance metric used in point removal is detailed in Section 6.2.4. It is emphasized that the core SBB algorithm remains unchanged.

6.2 Truck Reversal Environment

This section describes the truck reversal environment used in this and the following chapter, beginning with an overview of the problem in Section 6.2.1. The equations of motion, used to define successive states, are then detailed in Section 6.2.2. The point initialization/generation procedures and the genotypic distance used are described in Sections 6.2.3 and 6.2.4 respectively, and are followed by a specification of the binary and real-valued reward schemes in Section 6.2.5.

6.2.1 Overview

Problem Formulation

The truck reversal environment used here is a modified version of an earlier formulation [1]. The problem involves a cab and semi, moving in reverse at a constant speed, that must be steered from an arbitrary location into a loading dock. Thus, the end result tends to be sensitive to steering decisions made early on, or alternatively, the effect of specific decisions is not immediate, making temporal credit assignment difficult. In addition, the problem is highly non-linear. Steering is achieved by setting the angle of the front wheels, and the goal is to learn a successful strategy for setting

this angle. The main modification in this work compared to the previous formulation [1] is to include an obstacle into the environment.

The environment variables defining a configuration of the cab and semi, Figure 6.1, are the location of the back of the semi, (x, y), the angle of the semi, Θ_s, and the angle of the cab, Θ_c. These four variable are input to the controller which must learn a strategy for setting the angle of the front wheels u. Angles Θ_s and Θ_c are measured from the positive x-axis, while the steering angle u is measured relative to the alignment of the cab. In all cases, counter-clockwise corresponds to the positive direction. The angle u, continuous in the original problem formulation [1], is discretized and may assume values in $\{0°, -30°, 30°\}$; defining three actions in this way keeps the problem definition simple while allowing solutions that are not strictly 'bang-bang'-style[2] controllers to be evolved. The 0° action allows the cab and semi to move in a straight line provided that Θ_s is equal to Θ_c.

The cab and semi move in a plane, Figure 6.2, in which a rectangular obstacle has been placed with upper-left and lower-right corners at (45, 50) and (55, -50) respectively. From an initial state, the cab and semi travel a fixed distance during each time step. A maximum of 600 time steps is allowed in a single simulation, and the episode is stopped under the following conditions:

1. The x-coordinate marking the back of the semi assumes a negative value, i.e., the semi crosses the y-axis.

2. The (x, y)-coordinates of the back of the semi fall within the obstacle.

3. The difference between Θ_s and Θ_c is more than 90°, i.e., the cab and semi jackknife.

4. Not enough time steps remain to return from the current location to the origin along a straight line.

Comparison with Previous Formulations

Initial ML approaches to the truck reversal problem focused on the use of neural networks [136, 137]. Jackknifing of the cab and semi was allowed, supporting 'bang-bang'-style policies and making the learning task significantly easier, and training was based on increasingly more difficult 'lessons' which were specified *a priori*. In general,

[2]Whereby the controller resorts to the use of two, often symmetric, actions, e.g., hard left and hard right.

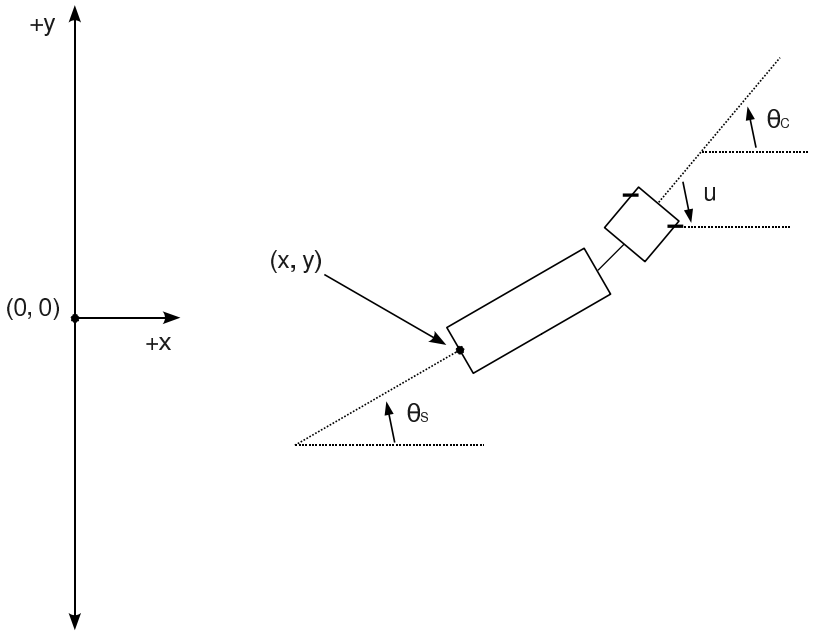


Figure 6.1: Truck reversal cab and semi configuration. The state is defined in terms of the coordinates of the back of the semi, (x, y), the angle of the cab, Θ_c, and the angle of the semi, Θ_s. The cab and semi angles are measured from the positive x-axis, while the steering angle u is measured relative to the cab orientation. In all cases, the counter-clockwise direction corresponds to positive angle values. Distances are measured in meters.

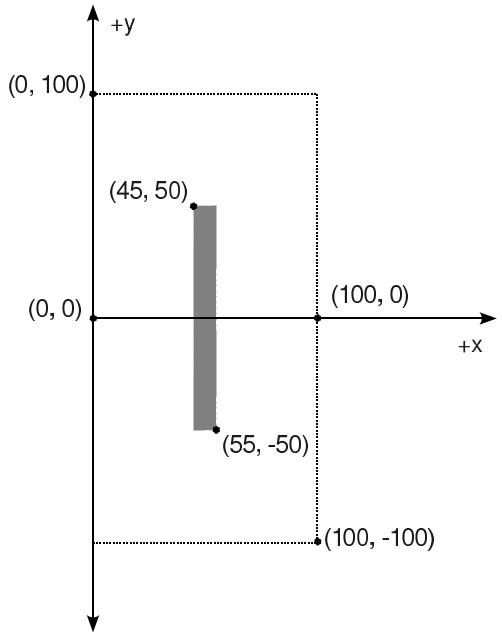


Figure 6.2: Truck reversal environment plane and obstacle. Training cases are restricted to the rectangular area bounded by the y-axis and the dotted line.

the starting configurations were much closer to the goal state than in the formulation used here, with the back of the semi never more than two truck lengths away from the origin in the x direction and one truck length away in the y direction. Subsequently, the difficulty of the original problem formulation was questioned [56], although in doing so significant domain knowledge was assumed including the transformation of the state space to polar coordinates and a manual override in case of jackknifing. In contrast to the initial attempts, other approaches have relied on a manual task decomposition, e.g., separating the subtask of aligning the semi with the x-axis from directing the vehicle towards the y-axis [84].

More recently, the truck reversal problem has attracted attention from the fuzzy systems research community [26, 143, 153]. However, the underlying problem definition in this case is fundamentally different as it involves a cab with no semi, altogether eliminating the possibility of jackknifing. As in this work, a recurring theme in the fuzzy systems research appears to be knowledge-driven learning versus data-driven learning, where the proposed approach is viewed as an instance of the latter.

EC approaches to the truck reversal problem include the optimization of neural network weights using a GA [161] and the evolution of GP-based controllers [94]. In both cases, a hand-crafted set of starting configurations was used as a basis for the

fitness function, and no evaluation on an independent test set was performed. In the latter case, the domain-specific *arctangent* function used in the equations of motion for the problem, Section 6.2.2, was included in the GP function set. As in the other formulations discussed above, jackknifing represented a legal condition.

In summary, relative to previous formulations of the truck reversal problem, the proposed work differs in the following key ways:

1. The cab and semi are not allowed to jackknife, a constraint that is enforced by terminating the episode and returning minimum (zero) reward when jackknifing does occur. This makes the learning task more difficult, particularly given that the effects of decision made are delayed for a relatively long time.

2. An obstacle is included in the environment which must be avoided in steering the cab and semi towards the origin – this again increases problem difficulty.

3. The goal is to evolve a policy that is effective across a diverse set of starting conditions. The set of fitness cases, i.e., the point population, is therefore not specified *a priori* but rather selected stochastically, and post-training evaluation is performed on an independent test set.

It is also noted that the formulation of the truck reversal problem used in this work differs from the style of problems often encountered in robotics. Specifically, there is a relatively limited amount of global information that can be used to control the truck, namely, the (x, y)-coordinates of the back of the semi and the two angles. Thus, there is no way to detect the obstacle, and any successful policy must infer its location indirectly. In contrast, problems commonly encountered in robotics often make use of extensive local information, e.g., sensory inputs that can detect a wall.

6.2.2 Equations of Motion

The equations of motion [1] define the state at time step $t + 1$ given the state and action at time step t and are included for completeness:

$$\begin{aligned} A &= r \cos u[t] \\ B &= A \cos(\Theta_c[t] - \Theta_s[t]) \\ C &= A \sin(\Theta_c[t] - \Theta_s[t]) \\ x[t+1] &= x[t] - B \cos \Theta_s \\ y[t+1] &= y[t] - B \sin \Theta_s \end{aligned}$$

$$\begin{aligned}\Theta_c[t+1] &= \tan^{-1}\left(\frac{d_c \sin\Theta_c[t] - r\cos\Theta_c[t]\sin u[t]}{d_c\cos\Theta_c[t] + r\sin\Theta_c[t]\sin u[t]}\right)\\ \Theta_s[t+1] &= \tan^{-1}\left(\frac{d_s \sin\Theta_s[t] - C\cos\Theta_s[t]}{d_s\cos\Theta_s[t] + C\sin\Theta_s[t]}\right)\end{aligned}$$

Here, parameters r, d_c, and d_s are respectively the distance traveled by the front tires in a single time step, the length of the cab, and the length of the semi. The tan^{-1} function is the arctangent of two variables[3] with a range of (-π, π].

6.2.3 Point Initialization and Generation

The operators used to produce points, described below, result in initial configurations where the (x, y)-coordinates marking the back of the semi are restricted to fall within a rectangular region with upper-left and lower-right corners at (0, 100) and (100, -100) respectively but outside of the obstacle, Figure 6.2. The cab and semi always start in an aligned state. Given the way points are created, some initial configurations may arise from which it is impossible to guide the cab and semi to the origin, though identifying these configurations is not trivial.

During point initialization and point generation, Sections 3.2.1 and 3.2.3, random points are created as follows:

1. The x- and y-coordinates marking the back of the semi are selected with uniform probability from (0, 100) and (-100, 100) respectively.

2. If the (x, y)-coordinates selected in the first step fall within the obstacle, the first step is repeated.

3. A value for Θ_s is selected uniformly from (-π, π], and the same value is used to initialize Θ_c.

To generate an offspring point from a parent point, Section 3.2.3, the parent is cloned, and the individual components of the clone's state are mutated as follows:

1. The x-coordinate is mutated through the addition of a normally distributed deviate with zero mean and standard deviation σ_x.

2. If the x-coordinate after the previous step does not fall within (0, 100), the mutation is reversed, and the previous step repeated.

[3]Implemented in C as the *atan2* function.

3. The y-coordinate is mutated through the addition of a normally distributed deviate with zero mean and standard deviation σ_y.

4. If the y-coordinate after the previous step does not fall within (-100, 100), the mutation is reversed, and the previous step repeated.

5. If the resulting (x, y)-coordinates fall within the obstacle, the mutations are reversed, and the process reverts back to the first step.

6. Θ_S is mutated through the addition of a normally distributed deviate with zero mean and standard deviation of σ_Θ, and Θ_c is set to equal Θ_s.

6.2.4 Genotypic Distance

Denoting two arbitrary starting states by $(x, y, \Theta_s, \Theta_c)$ and $(x', y', \Theta'_s, \Theta'_c)$, the raw distance between them as used in calculating the genotypic diversity reward, Eq. 3.5 in Section 3.2.6, was calculated as

$$\left(\frac{x - x'}{100}\right)^2 + \left(\frac{y - y'}{200}\right)^2 + \left(\frac{\Delta(\Theta_s, \Theta'_s)}{\pi}\right)^2$$

where $\Delta(\Theta_s, \Theta'_s)$ is the difference in angles reduced to [0, π], i.e., the minimum of the original difference or 2π less the original difference. The denominators 100, 200, and π represent the maximum possible value for each respective numerator and are used to normalize each term to the unit interval. The angle Θ_c is not included in the calculation since for each starting state it is equal to Θ_s.

6.2.5 Reward Functions

As suggested in the preliminaries to this chapter, binary reward schemes tend to provide less information than real-valued functions, but they are often easier to specify, e.g., the end user typically knows what the desired end condition is but may not be able to characterize the subgoals required to reach it[4]. The choice of reward scheme is therefore likely to have an effect on the credit assignment policies inherent in the learning algorithm, and consequently in this work, the selection and generation of both the hosts/symbionts and points.

[4]The Rubik's Cube environment that is considered in Chapter 8 is a good example of this, i.e., the specification of a solved cube is trivial and intuitive.

In this chapter, two reward schemes defining the outcome of applying a host to a point, Section 3.2.5, were considered in the experiments. Both were based on the final configuration of the cab and semi. Under the binary reward scheme, the reward was defined as

$$\begin{cases} 1 & \text{if } |x| < 1.0,\ |y| < 1.0, \text{ and } |\Theta_s| < 45^\circ \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

assuming final values of x, y, and Θ_s for the coordinates and angle of the semi. The binary reward scheme thus represents an all-or-nothing approach where partial solutions, e.g., solutions that are very close to the goal but do not meet the criteria in Eq. 6.1, do not receive any credit. For example, under this reward scheme, no distinction would be made between a policy that directs the cab and semi into the obstacle and a policy that runs out of time but otherwise successfully directs the vehicle towards the origin.

The real-valued reward scheme, on the other hand, provides a continuum of rewards based on the distance of the final configuration to the goal state. Specifically, given final values of x, y, and Θ_s the associated reward under the real-valued scheme is defined as

$$\frac{1}{\sqrt{x^2 + y^2 + \Theta_s^2 + 1}} \tag{6.2}$$

yielding values closer to unity as the final configuration approaches the goal state.

Under both reward schemes, jackknifing is treated as a special case in which a reward of zero is returned. In this way, the controller cannot take advantage of jackknifing the cab and semi, e.g., terminating the episode when the truck is close to the origin. In addition, the low reward value associated with a jackknifed configuration discourages solutions that would repeatedly swing the semi hard in one direction then the other, i.e., 'bang-bang'-style policies.

6.3 Experimental Setup

The sections that follow describe the parameter settings, Section 6.3.1, the evaluation methodology, Section 6.3.2, and the algorithm formulations used in the comparison, Section 6.3.3.

Parameter	Symbol	Value
Distance traveled in one time step	r	1.0 m
Length of the cab	d_c	6.0 m
Length of the semi	d_s	14.0 m
Mutation size, x-coordinate	σ_x	5.0 m
Mutation size, y-coordinate	σ_y	10.0 m
Mutation size, angles	σ_Θ	18°
Maximum number of time steps	-	600
Actions	-	$-30°, 0°, 30°$

Table 6.1: Truck reversal environment parameters.

6.3.1 Parameters

The same parameter settings were used in these experiments as in earlier chapters, Tables 4.3 and 4.4. Due to the switch from classification to temporal sequence learning, point generation, Section 3.2.3, and point selection, Section 3.2.6, has changed to take advantage of the structure in the state space. Thus, two additional parameters became applicable. The first was the probability of generating an offspring point from a parent point (as opposed to creating a random point), P_{genp}, which was set to 0.9. The second was K, the number of nearest neighbours used in the point fitness calculation, which was set to 13[5]. As before, 60 initializations were performed using each configuration.

Parameters specific to the truck reversal environment are summarized in Table 6.1. The length of the cab and semi were kept as in the original problem definition [1], while the distance traveled in one time step was increased from 0.2 meters to 1.0 meter so that the same distance could be covered in one episode as in previous studies [25, 94] while reducing the number of time steps. The mutation standard deviations were set at five percent of the range of each respective initial state variable.

6.3.2 Evaluation Methodology

The quality of a solution was measured in terms of the number of test cases that it solved. A test case was considered solved if in the final configuration the state variables were as follows: $|x| < 1.0$, $|y| < 1.0$, and $|\Theta_s| < 45°$. Thus, at the end of the episode, the back of the semi had to be within one meter of the origin and aligned within 45° of the positive x-axis. This definition of a solved state matches the criteria

[5]Calculated as 1 plus one-tenth of the size of the point population.

used in the binary reward function, Eq. 6.1. It is also noted that the term 'solved' is not limited to post-training evaluation but is also used in reference to members of the point population during training, i.e., when the final configuration associated with a point contributes to a high fitness value for a given host.

A solution could be viewed as either the best host in the population at the end of training, or as the population of hosts as a whole. Naturally, different hosts are expected to solve different problem instances, hence the population-wide counts will tend to be larger than the count for any single host. Considering population-wide counts is motivated in part by the capacity of the SBB algorithm to combine solutions hierarchically, where this is considered in Chapter 7.

To determine which host in the final population was identified as the best, a validation set consisting of 1000 random points was used. Each point in this set was initialized as described in Section 6.2.3 with one exception: if the selected Θ_s and Θ_c were such that, following a straight line, the truck would run into the obstacle, the angles were reselected. This additional constraint was included to reduce the number of cases that were not feasible[6]. The best host was then defined as the individual in the population with the highest mean reward on the validation set. An independent test set, consisting of 1000 problem instances initialized in the same way as the validation set, was then used to evaluate the performance of the best host and the population as a whole. The results presented in Section 6.4 are with respect to this independent test set. It is also noted that the validation and test sets were the same across all initializations.

6.3.3 Algorithm Formulations

Five variants of the SBB algorithm were compared in the experiments, differing only in the way in which the point population is managed, with the goal of establishing the significance of the formulation of specific algorithm components. Relative to the original algorithm specification, the relevant components include point initialization, Section 3.2.1, point generation, Section 3.2.3, and the removal of points, Section 3.2.6. Not all of the variants, described below, involve modification in all three areas:

1. **SBB**. The original SBB algorithm for temporal sequence learning as described in Chapter 3, i.e., no modifications were made. Point generation adds points

[6]Given a random set of points generated in this way, it is likely that on average a certain proportion of the cases remain infeasible, e.g., a case may require a turn that is physically too tight. A one-hundred percent success rate on the test set is therefore not expected, though the exact proportion of cases that are not feasible is unknown.

that are predominantly obtained through the domain-specific mutation of existing points with the occasional inclusion of a random point aimed to help maintain diversity. Removal of points deterministically discards points based on a fitness calculation that considers two factors: the degree to which points identify unique and useful host behaviours, and the degree to which points are themselves genotypically unique. Furthermore, the latter is with respect to the domain-specific distance measure.

2. **SBB-RSS**. The SBB algorithm with coevolution between the point and host populations replaced with random subset selection (RSS). Point generation still adds P_{gap} points to the population but each point is generated as in point initialization. During removal of points, P_{gap} points, each selected with uniform probability, are removed.

3. **SBB-SSS**. The SBB algorithm with coevolution between the point and host populations replaced with static subset selection (SSS). The point population is initialized to contain P_{size} points and the same set of points is used to evaluate hosts across all generations. As such, point generation and point removal are bypassed.

4. **SBB-DST**. This variant uses distinctions to establish the raw fitness of each point in place of the function used in the SBB formulation. Specifically, for each point, a binary distinction vector is calculated, Eq. 2.2. Based on this set of distinction vectors (one vector per point), a shared fitness is calculated for each point in a way that is analogous to the host shared fitness calculation, Eq. 3.7. Thus, points are rewarded for making distinctions that few other points make. The genotypic reward is then applied to the shared score for each point as before.

5. **SBB-RAW**. A version of the SBB algorithm that uses the raw point fitness function, Eq. 3.3, without applying the reward based on the points' genotypes.

It is stressed that relative to the original SBB formulation, all other components in the latter four formulations remain unchanged. Inclusion of SBB-RSS and SBB-SSS is meant to establish the significance of the coevolutionary component between the point and the host populations by comparing it to two naive point sampling and removal strategies. SBB-DST represents an established formulation of competitive coevolution [46, 139] against which the proposed raw point fitness function can be

compared. Finally, comparison between SBB and SBB-RAW should isolate the effect of the genotypic diversity reward on the populations that are evolved.

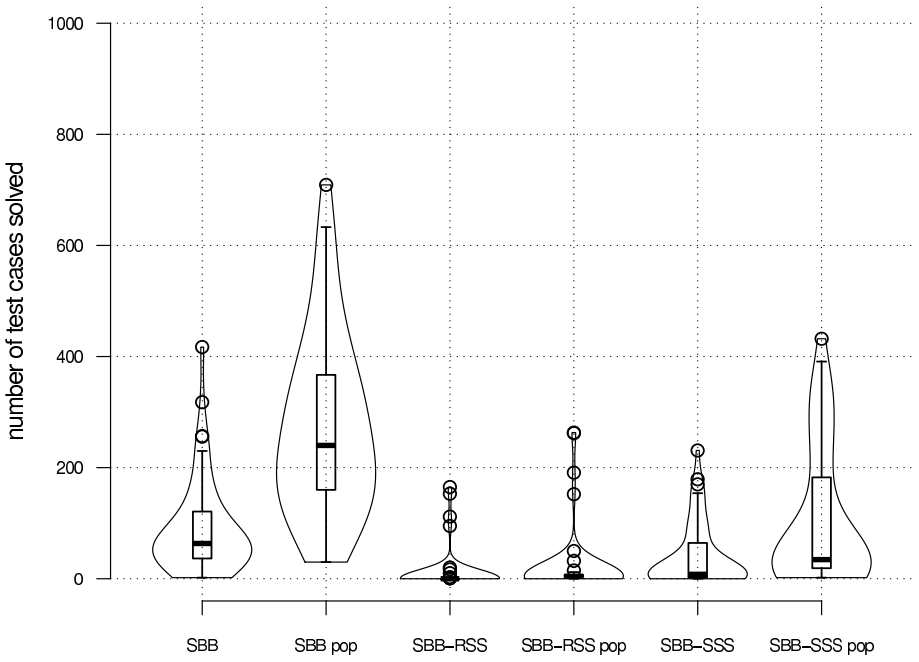
6.4 Results

Results related to the comparison between the original SBB algorithm and the formulations using random and static subset selection are presented in Section 6.4.1, whereas comparison with distinction-based fitness and fitness with no genotypic reward are presented in Section 6.4.2. Section 6.4.3 involves additional analysis of the difference in behaviour under the SBB and SBB-RAW formulations.

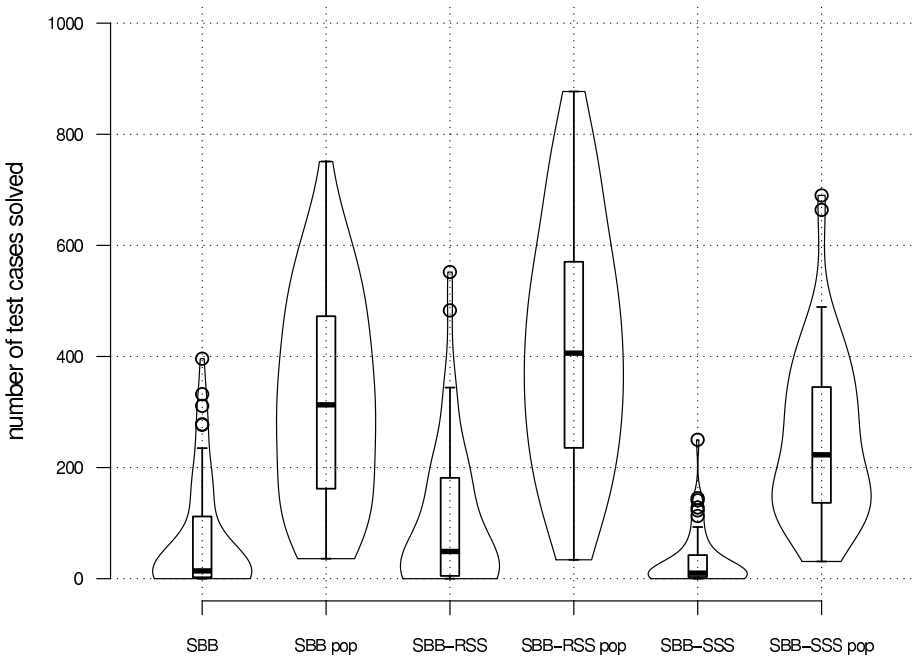
6.4.1 Comparison with Random and Static Subset Selection

The number of test cases solved using the SBB, SBB-RSS, and SBB-SSS formulations is shown in Figure 6.3. Under binary rewards, Figure 6.3a, the SBB formulation does significantly better than the other approaches both with respect to the best host and population distributions. In addition, it exhibits a visible improvement in the population-wide performance compared to the best host counts. With the exception of a handful of values, the SBB-RSS counts tend to be close to zero, both with respect to the best individual as well as the population as a whole. The SBB-SSS performance is slightly better though well below that of SBB. Under real-valued rewards, Figure 6.3b, SBB-RSS appears to perform the best, with SBB yielding solution counts that are slightly lower. The SBB-SSS performance appears to be the worst. Furthermore, across all three approaches, the population-wide solution counts tend to be greater under real-valued rewards than under binary rewards.

To determine if the observed differences between the SBB counts and the counts for the other two approaches are significant, a set of two-tailed Mann-Whitney tests was applied to each pair of corresponding distributions in Figure 6.3, e.g., the best individual counts using SBB and SBB-RSS under binary rewards. The resulting p-values are reported in Table 6.2. Under binary rewards, the tests suggest a difference between SBB and the other two approaches at a 0.01 significance level, and here, the difference is clearly in favour of the SBB formulation. Under the real-valued reward scheme, the observed differences are not meaningful at a 0.01 significance level. These results, together with the qualitative observations made earlier, suggest that under binary feedback only the SBB formulation is appropriate. In this case, the critical factor appears to be the coevolutionary relationship between the point and host populations.



(a) Binary rewards.



(b) Real-valued rewards.

Figure 6.3: Number of truck reversal test cases solved under SBB, SBB-RSS, and SBB-SSS under binary rewards and real-valued rewards. Distributions corresponding to the population-wide counts are labeled 'pop'.

		Binary	Real-valued
SBB vs SBB-RSS	Best host	≈0⋆	0.06329
	Population	≈0⋆	0.02107
SBB vs SBB-SSS	Best host	7.002e-07⋆	0.3193⋆
	Population	1.176e-09⋆	0.03374⋆

Table 6.2: Two-tailed Mann-Whitney test results comparing SBB solution counts with SBB-RSS and SBB-SSS solution counts. Shown are the p-values obtained when the test was applied to each pair of corresponding distributions in Figure 6.3. Cases where the SBB median value is *higher* are noted with a ⋆.

The contrasting performance of SBB-RSS under binary and real-valued rewards suggests that under binary rewards there is not enough host fitness information to form and maintain a learning gradient. In particular, given an initial set of random points and hosts, it is likely that the number of hosts solving one or more points is small, and that the maximum number of points solved by any given host is also small. Thus, most hosts are likely to always receive a reward of zero, resulting in little or no information as to whether one host is better than another. As the point population evolves, the points that do provide information on host fitness are just as likely to be removed as any other point, leading to a reduction in the fitness of any hosts that solve those points. Furthermore, there is no process for exploiting points that do provide useful information on the quality of the hosts. Effectively, using the SBB-RSS formulation under binary rewards, it appears that the host and point populations are not suitably engaged. Under real-valued rewards, on the other hand, a learning gradient is always likely to be present because a non-zero value is associated with all non-jackknifed states.

The poor performance of SBB-SSS compared to SBB under binary rewards can also be rationalized in similar terms. However, unlike under SBB-RSS, once a point is found to be solvable under SBB-SSS that point will not be arbitrarily discarded since the point population is not turned over. The point population under static subset selection thus represents a stationary target making it easier to maintain engagement, which, once established, depends only on the ability of the host population; naturally, this is assuming that the static point population represents a challenging set of starting configurations. This is consistent with the second-best performance of SBB-SSS observed in Figure 6.3a.

To gain insight into the degree to which different hosts in the same population solve different test cases, the number of test cases solved by the best host is compared

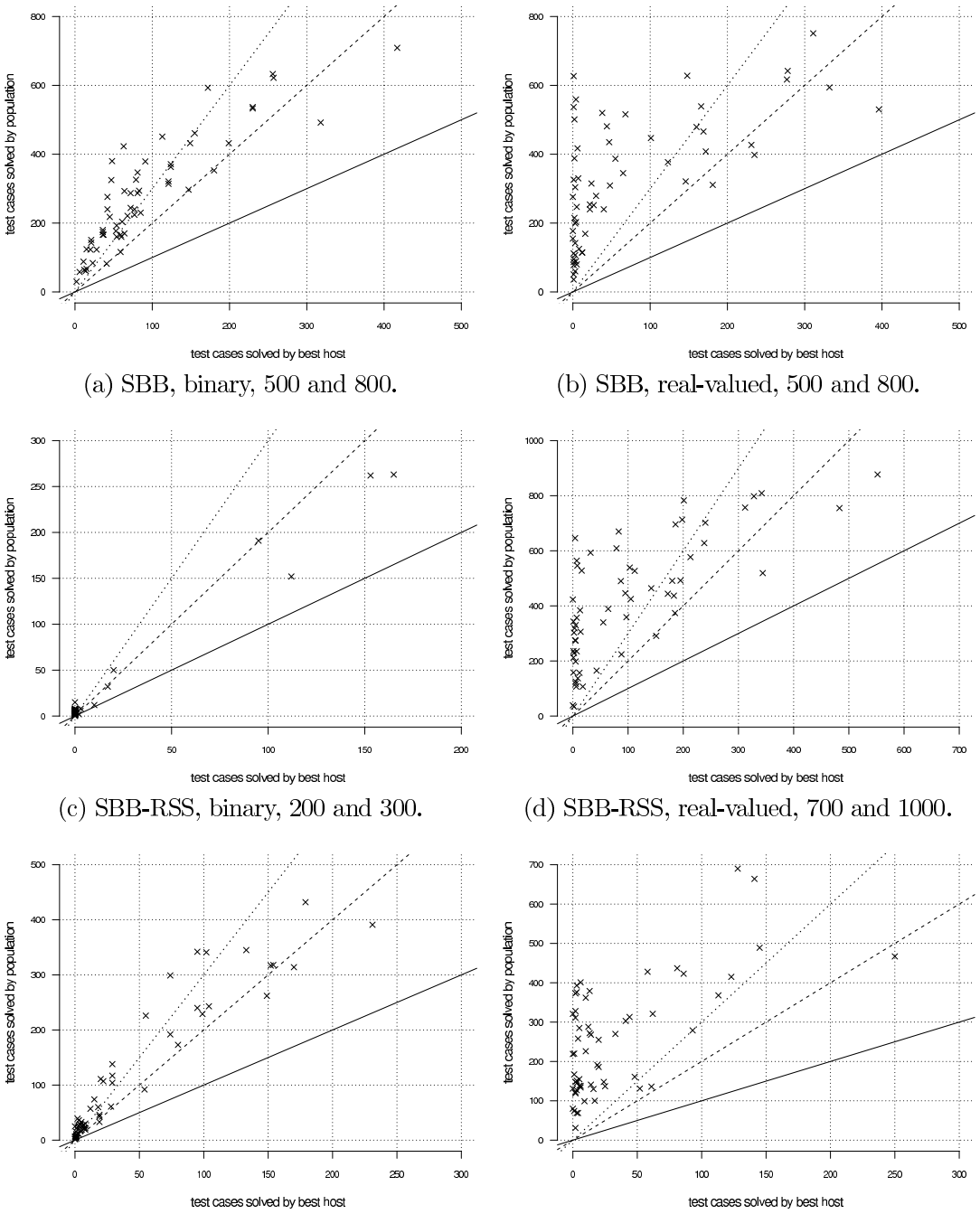
to the number of test cases solved by all the hosts in the population at the end of training, Figure 6.4. In a given initialization, if the best host solves a large portion of the cases solved by the population as a whole, the point in the plot will approach the $y = x$ line. This is interpreted as suggesting overlapping behaviour in terms of the test cases solved, i.e., different hosts tend to solve the same test cases. However, no points are observed below $y = x$ since no matter how good it is the best host cannot solve more cases than the population that it is a part of. Lines at $y = 2x$ and $y = 3x$ are also included indicating thresholds at which the population solves at least twice and three times as many cases as the best host.

In the case of the SBB algorithm under binary rewards, Figure 6.4a, different hosts in the same population do appear to solve different test cases. However, the best host tends to win an increasingly large proportion of test cases as its quality increases; this is observed in the high number of relatively poor solutions above the $y = 3x$ line. With respect to the best overall solution across all initializations, the best individual solves 417 cases while the population as a whole solves 709 test cases, i.e., the best individual accounts for nearly 60% of all cases solved. This trend is consistent with the SBB-RSS and SBB-SSS binary results, Figures 6.4c and 6.4e, though it is perhaps not as visible given the high number of very poor solutions.

Under real-valued rewards, Figures 6.4b, 6.4d, and 6.4f, many solutions are observed where the best host count is close to zero but the population wide count is comparatively high. However, this may not necessarily suggest a high degree of problem decomposition among the hosts in the same population with respect to the test cases solved. Instead, it indicates that the criterion for selecting the best host under real-valued rewards may not be a good choice. Specifically, the best host is selected as the one with the highest mean reward on the validation set. Under real-valued rewards, a host can receive a high reward on average, but due to the crisp limits defining a solved state, it may not actually solve many test cases, Figure 6.5. In contrast, under binary rewards, using the mean reward to select the best host is equivalent to using the count of solved cases.

6.4.2 Comparison with Raw and Distinction-Based Point Fitness

A comparison between SBB, SBB-DST, and SBB-RAW with respect to the number of test cases solved is shown in Figure 6.6. Compared to the other two approaches, SBB appears to do better under binary rewards, Figure 6.6a, both in terms of the single best host and the population as a whole. Under real-valued rewards, Figure



(a) SBB, binary, 500 and 800.

(b) SBB, real-valued, 500 and 800.

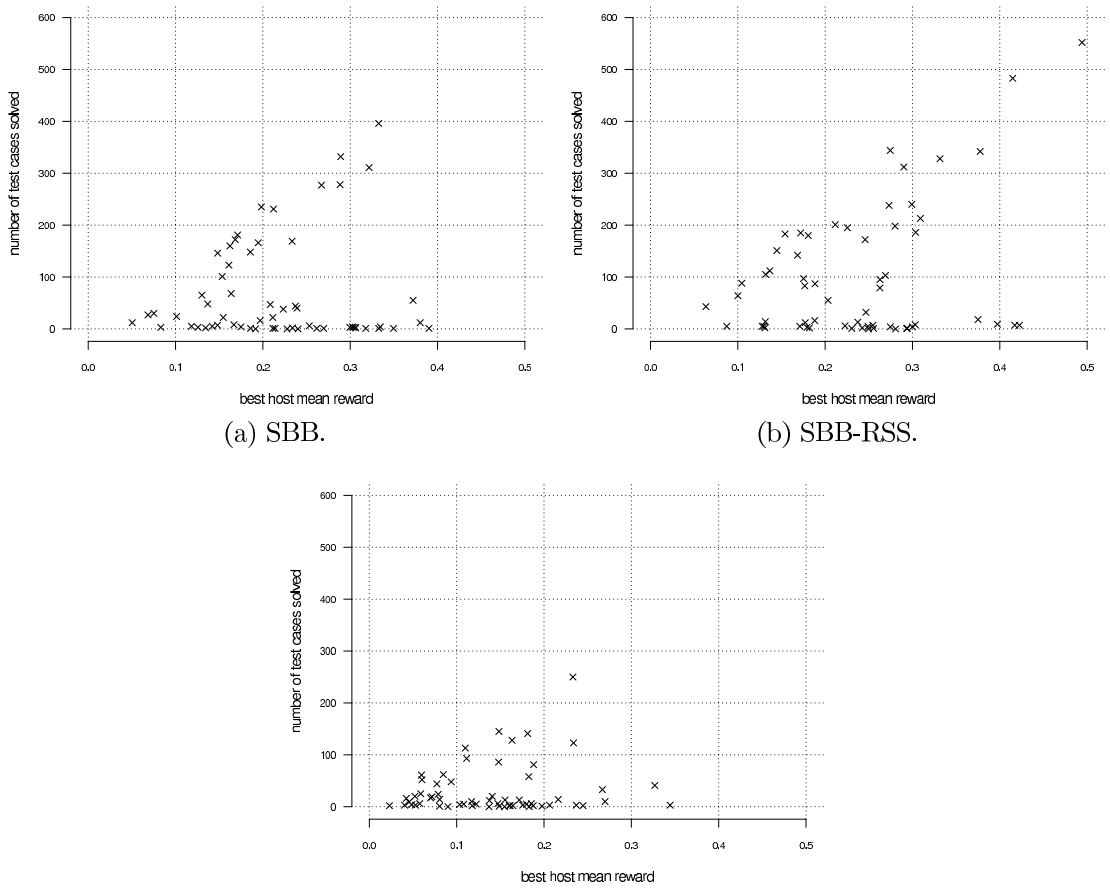
(c) SBB-RSS, binary, 200 and 300.

(d) SBB-RSS, real-valued, 700 and 1000.

(e) SBB-SSS, binary, 300 and 500.

(f) SBB-SSS, real-valued, 300 and 700.

Figure 6.4: Comparison of SBB, SBB-RSS, and SBB-SSS with respect to the number of test cases solved by the best host, x-axis, and across the population, y-axis. Each point represents the two quantities at the end of training for one initialization. The plots include a solid, dashed, and dotted line drawn at $y = x$, $y = 2x$, and $y = 3x$ indicating when the population solves one time, two times, or three times as many cases as the best host. Each subfigure caption denotes respectively the algorithm, reward scheme, and x- and y-axis ranges.



(a) SBB.

(b) SBB-RSS.

(c) SBB-SSS.

Figure 6.5: Mean reward and number of test cases solved associated with the best host under real-valued rewards. The mean reward received by the best host, x-axis, is plotted against the number of test cases solved by that individual, y-axis.

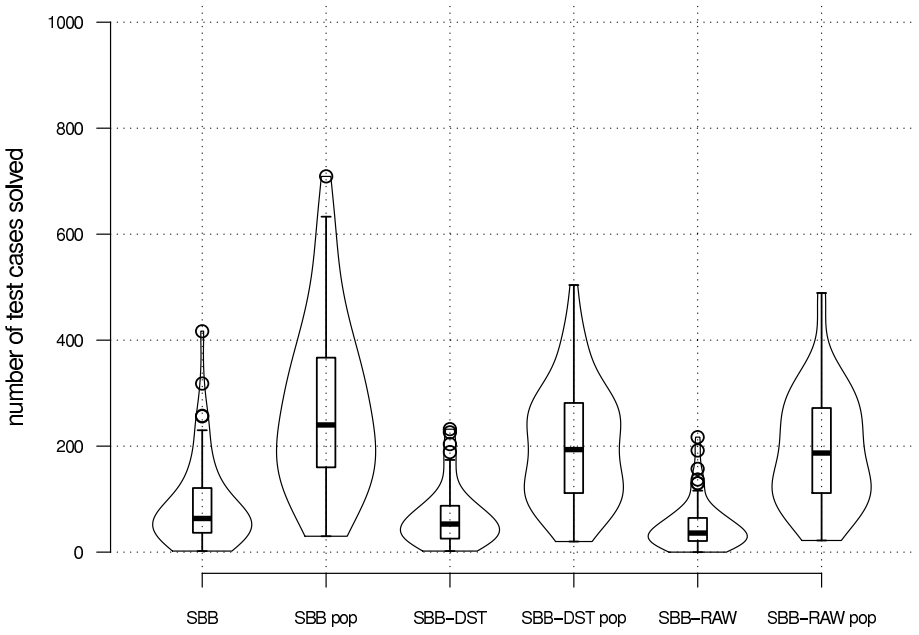
		Binary	Real-valued
SBB vs SBB-DST	Best host	0.1467⋆	0.5472
	Population	0.02241⋆	0.5322
SBB vs SBB-RAW	Best host	0.001211⋆	0.7766
	Population	0.004516⋆	3.188e-05⋆

Table 6.3: Two-tailed Mann-Whitney test results comparing SBB solution counts with SBB-DST and SBB-RAW solution counts. Shown are the p-values obtained when the test was applied to each pair of corresponding distributions in Figure 6.6. Cases where the SBB median value is *higher* are noted with a ⋆.

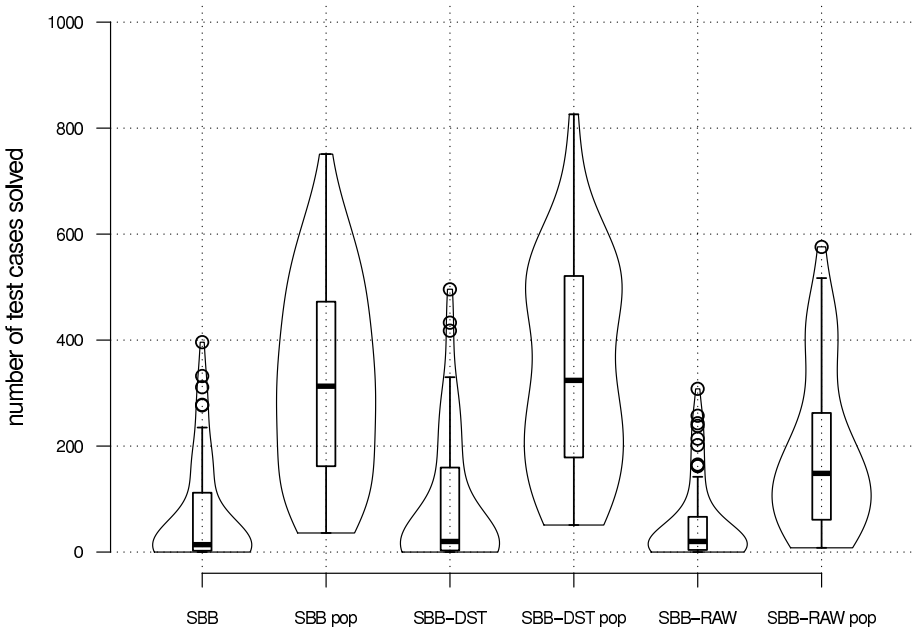
6.6b, SBB appears to outperform SBB-RAW in terms of the population-wide counts, although the highest values in terms of both the best host and population-wide counts are observed under SBB-DST. It is surprising to see that compared to the best host counts, the population-wide counts observed under SBB-RAW appear to show significant improvement under both reward schemes. It was expected that due to the lack of an explicit mechanism for enforcing genotypic diversity in the point population, the SBB-RAW host populations would converge resulting in more overlap in the test cases solved. However, the SBB-RAW performance still tends to fall behind the SBB performance in all but the best host counts under real-valued rewards, the latter approach taking into account the genotypic diversity of the point population.

The results of a set of two-tailed Mann-Whitney tests comparing SBB to each of the other two approaches is shown in Table 6.3. At a significance level of 0.01, no meaningful differences are observed between SBB and SBB-DST in any of the tests – this is out of line with the qualitative observations noted earlier whereby the SBB counts were observed to be higher under binary rewards but lower under real-valued rewards. With respect to the population-wide counts, this result may be due in part to the slightly bimodal shape of the SBB-DST distributions. Compared to SBB-RAW, the SBB formulation appears to be better with respect to both sets of counts under binary rewards at a 0.01 significance level, and also under real-valued rewards but only with respect to the population-wide counts. Thus, even though the SBB-RAW population-wide counts show an unexpected improvement over the corresponding best host counts, the overall level of performance achieved by including an explicit genotypic diversity component in the point fitness is higher.

Given that the difference in performance between SBB and SBB-DST was not shown to be statistically significant, the motivation for using the proposed point



(a) Binary rewards.



(b) Real-valued rewards.

Figure 6.6: Number of truck reversal test cases solved under SBB, SBB-DST, and SBB-RAW under binary rewards and real-valued rewards. Distributions corresponding to the population-wide counts are labeled 'pop'.

fitness function instead of the already established distinction-based fitness may not be readily apparent. However, a disadvantage of using distinctions is that given n individuals to be evaluated, the total number of possible comparisons is $n^2 - n$. This has two negative consequences. First, performing the point fitness computation becomes increasingly expensive as the number of individuals n increases. Second, as the number of individuals grows, they all tend to fall in the non-dominated set with respect to the Pareto-dominance relation that is the basis of Pareto-coevolution and several of the associated theoretical results. As such, a tie breaking heuristic must be used, at which point it becomes difficult to decide which distinctions are meaningful and which are not. In contrast, the proposed fitness function is more efficient, i.e., linear in the number of individuals n, and as specified, yields a single scalar fitness value which can be used directly without the application of additional heuristics.

6.4.3 Genotypic Diversity in the Points: Further Analysis

To gain insight into the effect of the genotypic diversity component in the point fitness function, contents of the point population were analyzed for select runs under the binary reward scheme, Figures 6.7 and 6.8. Each subplot in the figures shows the initial states corresponding to the 100 points at the end of the indicated generation. A state is represented by two conjoined line segments, indicating the cab and the semi, with the back of the semi denoted by a $\bullet$. Since initial states are generated such that $\Theta_s = \Theta_c$, in these figures the two segments are always aligned (though this is not true in all figures where this representation is used).

Figure 6.7 represents the SBB-RAW run where the highest number of test cases was solved across the population. An SBB run where roughly the same number of test cases was solved is also shown in Figure 6.8, though this is clearly not the best SBB run observed. Comparing the two, the effect of the genotypic diversity reward can be clearly visualized. In the SBB-RAW run, Figure 6.7, after ten generations most of the population has already converged to a relatively small region of the state space. The points tend to remain in a single cluster, although the lump itself does not remain stationary. In particular, occasionally, e.g., as seen in generations 300, 400, 600, and 800, some of the points break off from the main cluster. If these seceding points survive, it appears as if they may then be able to draw the cluster to a different region of the space space.

A different dynamic is observed when genotypic diversity is explicitly promoted, Figure 6.8. Initially, the points also converge to a single region of the state space,

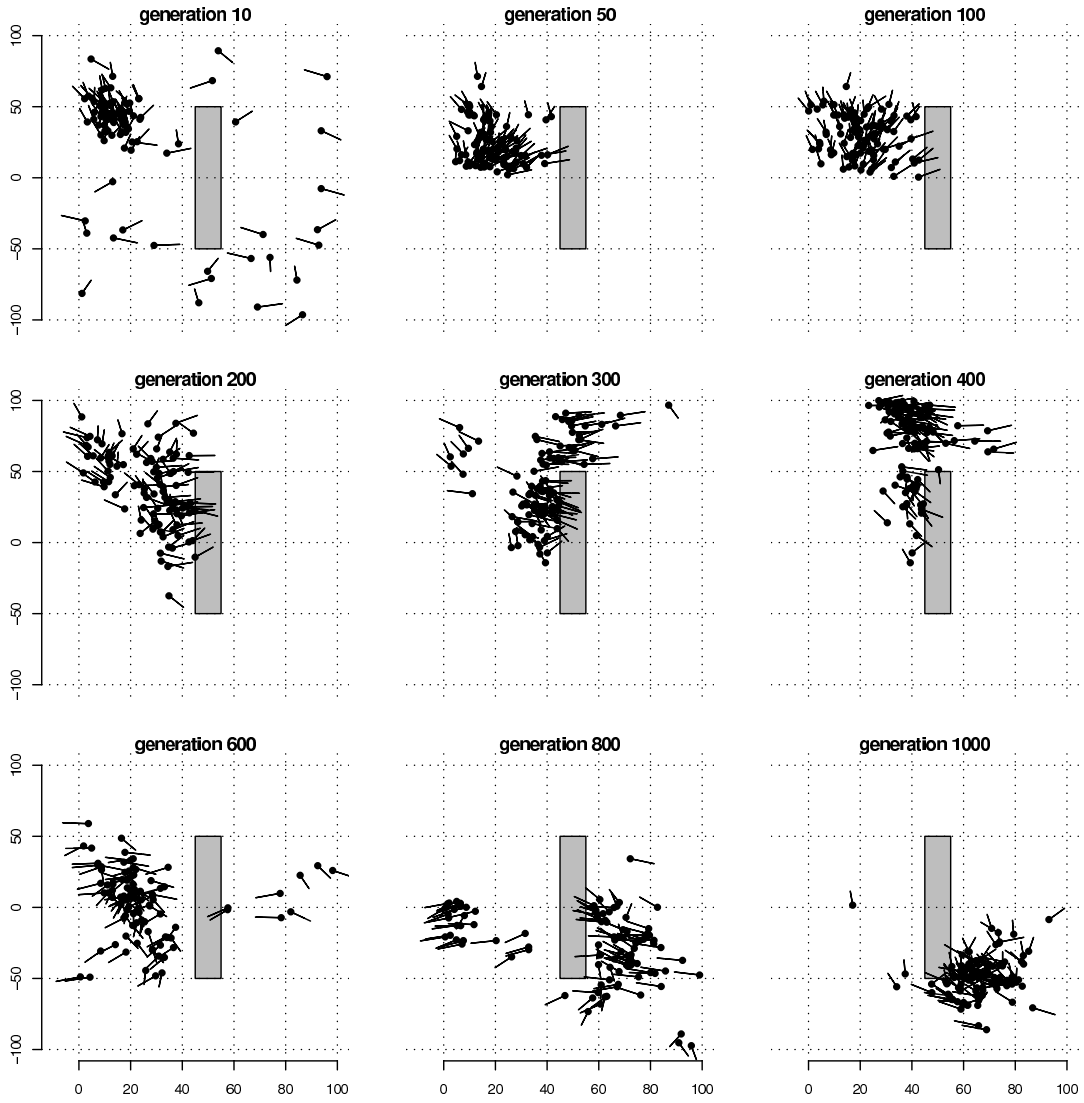


Figure 6.7: Evolution of points in an SBB-RAW run, good initialization (489 of 1000 test cases solved by population).

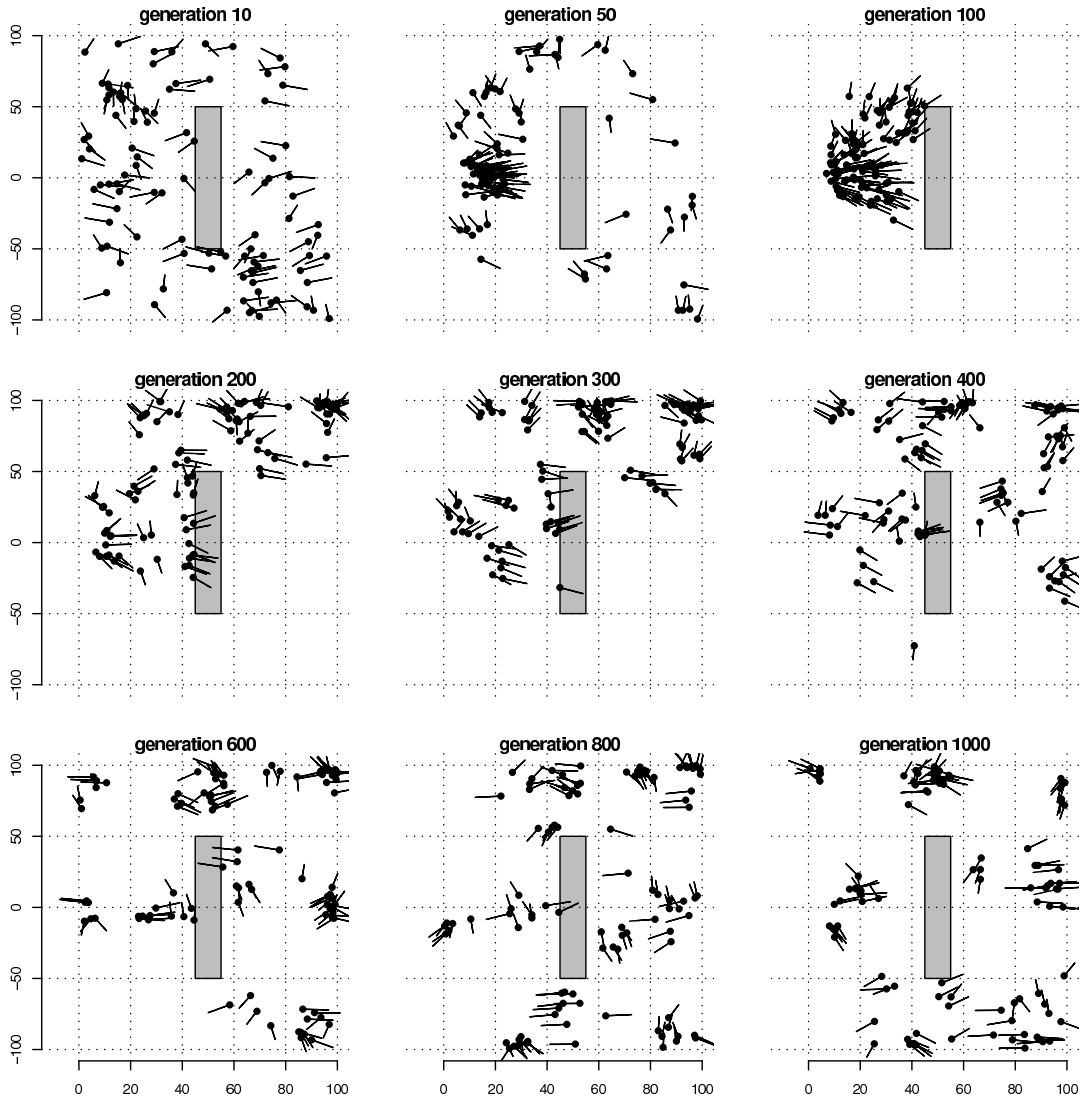


Figure 6.8: Evolution of points in an SBB run (492 of 1000 test cases solved by population).

although the rate of convergence appears to be slower. In particular, at generation 100 in Figure 6.8, the point population is very similar to the point population at generations 50 and 100 in Figure 6.7. Following this initial convergence, however, the points do not remain in a single cluster but rather they tend to disperse and remain spread out – this behaviour is attributed to the genotypic diversity factor applied in calculating point fitness.

Both Figures 6.7 and 6.8 indicate an initial convergence of the point population to a region near the origin, where this is due to the coevolutionary interaction between the host and point populations. The exact dynamic at play cannot be determined given the available data, but one possibility is as follows. Specifically, the initial host population is likely to contain relatively weak individuals that can successfully reach the goal state only from easy starting states, that is, those close to the origin and where the direction of motion is towards the origin; indeed, this characterization of easy points matches the initial clusters that are formed. Thus, initially, only these easy points will be supported with the more difficult points receiving a minimum (zero) fitness value. As the behaviour of the hosts grows stronger, they will gain the ability to reach the goal state from other regions of the state space, including points that were previously regarded as too difficult. At the same time, as points with non-zero fitness appear in regions of the state space away from the initial cluster, the genotypic diversity component in the SBB formulation is likely to assume a more significant role; this is viewed as a secondary effect, however, since the critical factor is undoubtedly the ability of the point population to differentiate between hosts of different ability, or in coevolutionary terminology, Section 2.4.1, that engagement between the host and point populations is maintained[7]. For example, under the SBB-RAW initialization in Figure 6.7, it appears that engagement is maintained even though the points tend to represent similar conditions suggesting that perhaps they are not as discriminating as one would like.

The test cases solved by the population at the end of the runs shown in Figures 6.7 and 6.8 are shown in Figures 6.9 and 6.10 respectively. The plots are similar, where this is likely due to selecting an SBB run that solves roughly the same number of test cases as the SBB-RAW run[8]. It is surprising, however, that the convergent point behaviour observed in Figure 6.7 results in similar levels of performance as the diverse behaviour observed in Figure 6.8. One would expect more diversity in the

[7]Alternatively, if raw point fitness, Eq. 3.3, is zero, then the genotypic reward factor is irrelevant.

[8]In both runs the unsolved cases tend to be close to the y-axis. This is reasonable if those test cases require the cab and semi to initially pull away form the y-axis.

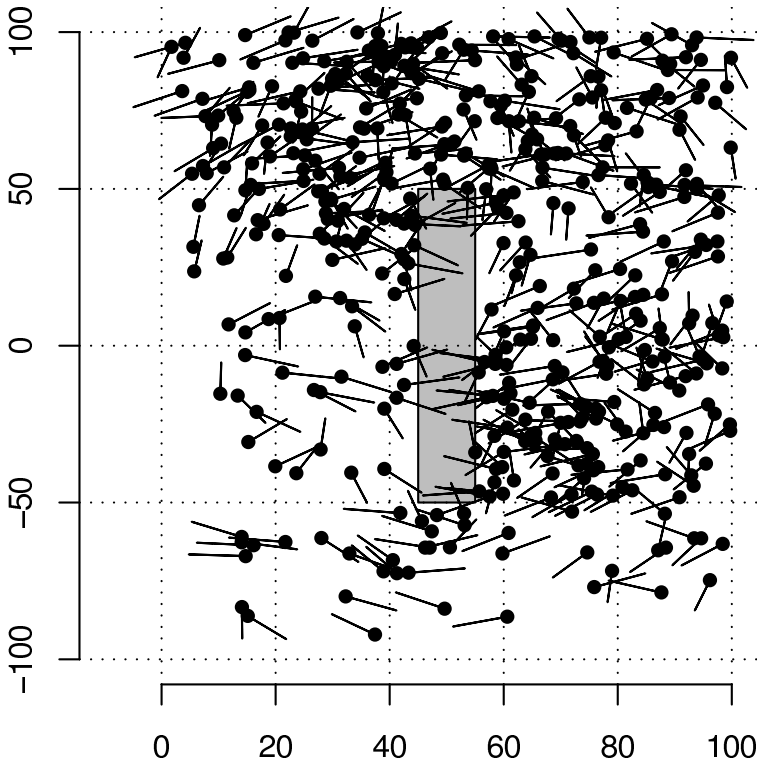


Figure 6.9: The 489 test cases solved in the SBB-RAW run in Figure 6.7.

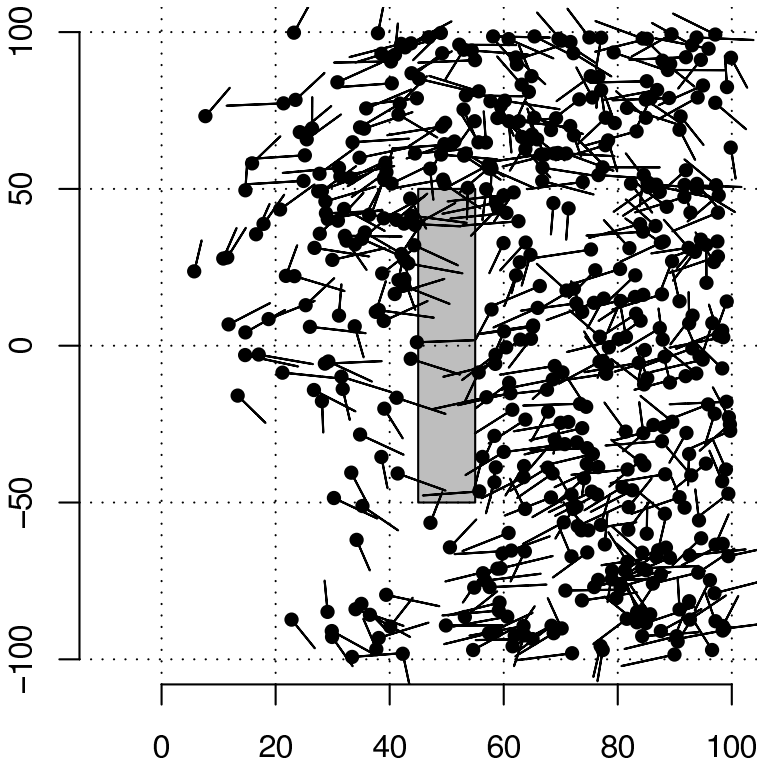


Figure 6.10: The 492 test cases solved in the SBB run in Figure 6.8.

point population to produce much better population-wide host performance, since points are supported only if they are solved by one or more hosts. A reason for the relatively strong performance of the SBB-RAW run may be that the point cluster in Figure 6.7 is not stationary. Thus, if the knowledge learned in previous generations is retained in the host population, the models will incrementally get better with respect to the entire state space. It is also noted that the best host in the SBB and SBB-RAW runs solved 318 and 192 of the test cases respectively, suggesting that in the former case the best individual is responsible for a greater proportion of the solved points. This is counter-intuitive given the higher diversity in the point population under the SBB run where one would expect different individuals to associate with different regions of the state space.

Analogous to the presentation above, the evolving point population and the test cases solved in a bad SBB-RAW run are shown in Figures 6.11 and 6.12 respectively. These figures are consistent with the hypothesis that the SBB-RAW algorithm can produce relatively strong solutions provided that the cluster of points does not remain stationary. Specifically, in Figure 6.11, the cluster of points tends to remain fixed to the left of the obstacle. This convergent and stationary behaviour is reflected in that only 22 of the 1000 test cases are solved across the final population, Figure 6.12.

To observe more general trends in the point populations evolved using the SBB and SBB-RAW formulations under both binary and real-valued rewards, a rough measure of the area covered by the points was considered, Figure 6.13. In contrast to Figures 6.7, 6.8, and 6.11, which illustrate behaviours in specific runs, Figure 6.13 allows trends to be observed across all initializations. Given the point population at the end of a generation, the minimum and maximum coordinates in the x and y directions were determined across all the points in that population. A rectangle was then defined with vertical sides parallel to the y-axis and horizontal sides parallel to the x-axis; the width of this rectangle was defined by the minimum and maximum x-coordinates, and the height was defined by the minimum and maximum y-coordinates. If the area of this rectangle, calculated as the width times the height, is small, this means that the points have converged to a small region of the plane. A large area, however, does not necessarily mean that the points have not converged as there may be outliers. In addition, this measure of point diversity is not complete in the sense that it does not account for the angles, however, it nonetheless provides useful information about the behaviour of the point population.

Despite being a very rough measure of point diversity, this area-based metric supports the observations made so far under binary rewards, Figures 6.13a and 6.13b,

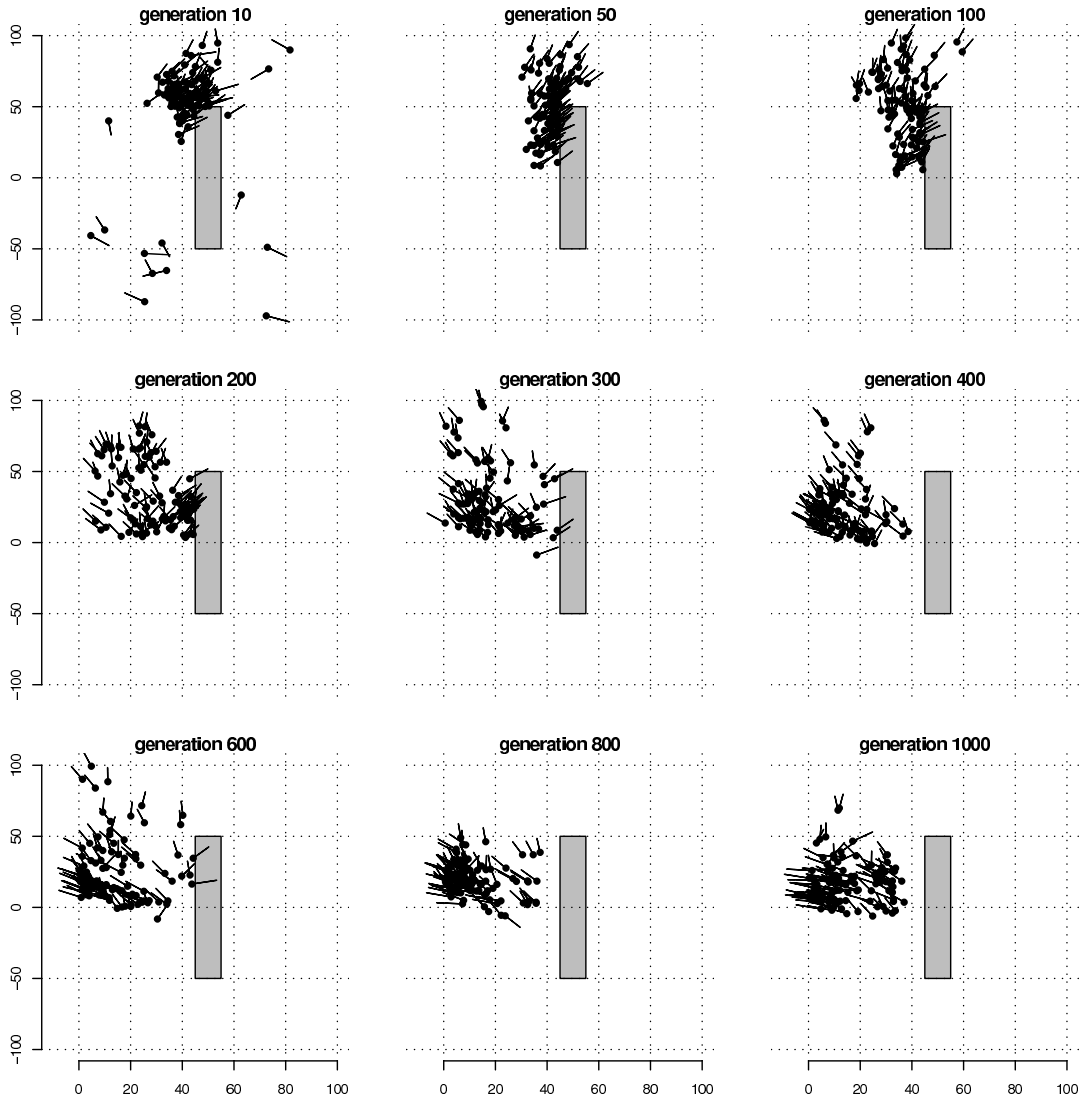


Figure 6.11: Evolution of points in an SBB-RAW run, bad initialization (22 of 1000 test cases solved by population).

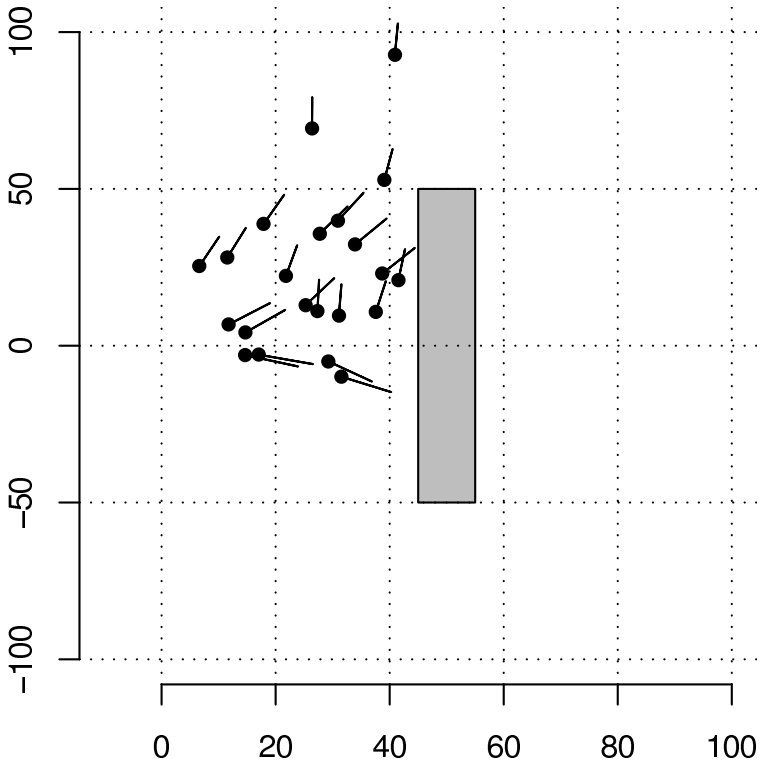
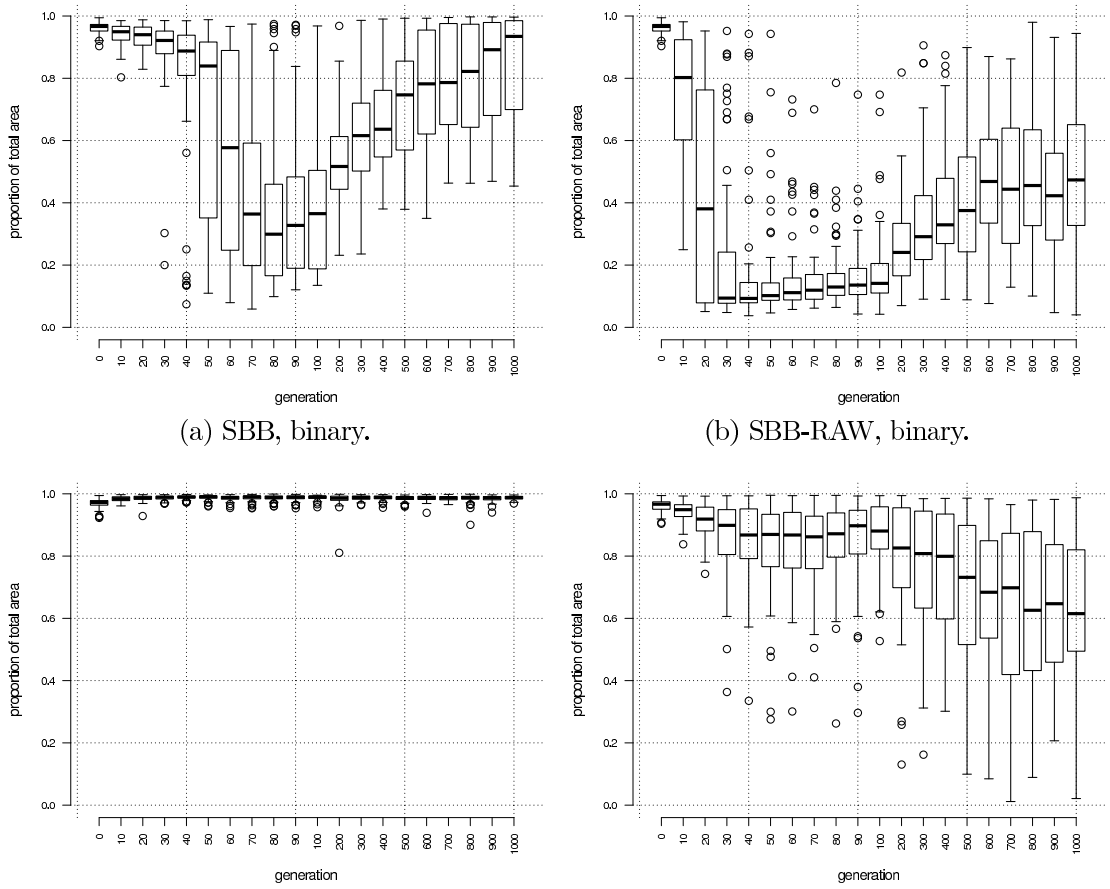


Figure 6.12: The 12 test cases solved in the SBB-RAW run in Figure 6.11.



(a) SBB, binary.

(b) SBB-RAW, binary.

(c) SBB, real-valued.

(d) SBB-RAW, real-valued.

Figure 6.13: Area of a rectangle containing all the points in the point population expressed as a proportion of the maximum possible area. At the end of each generation, x-axis, the minimum and maximum values are calculated across the x-coordinates of all points in the point population. Similarly, the minimum and maximum values are calculated across the y-coordinates. These values delimit the width and height of the containing rectangle and are then used to calculate its area, y-axis. The maximum possible area corresponds to the rectangle with upper-left and lower-right corners at (0, 100) and (100, -100) respectively, Figure 6.2. Each distribution represents 60 initializations.

suggesting that the behaviour is common across multiple initializations. In particular, both the SBB and SBB-RAW populations tend to converge early in the run (the scale on the x-axis changes at 100). However, the SBB-RAW populations appear to converge earlier and to a much greater degree. In addition, having initially converged, the SBB points then tend to spread out a lot more with median values at 1000 generations observed that approach unity. In contrast, towards the end of the run, the SBB-RAW points typically cover about half of the plane, e.g., this may correspond to the area to the left of the obstacle. This is consistent with the higher SBB population-wide solved counts, Figure 6.6a, since points are only supported if they are solved by one or more hosts.

The areas observed under real-valued rewards, Figures 6.13c and 6.13d, indicate different trends than under binary rewards. Specifically, though the behaviour under the two approaches is different, neither exhibits an initial decrease in the point diversity followed by a recovery. The area covered by the points in the SBB runs, Figure 6.13c, tends to account for the whole plane throughout. In comparison, using the SBB-RAW variant, the points tend to slowly converge as the runs progress with no eventual improvement. This difference in the behaviour is attributed to the lack of a genotypic diversity component in point fitness under the SBB-RAW formulation. As in the binary case, the higher diversity in points under the SBB approach is reflected in improved test performance with respect to the population-wide counts, Figure 6.6b. The steady decrease in point population diversity under the SBB-RAW formulation suggests that the coevolutionary point fitness function is too exploitative, resulting in a significant decrease in the number of test cases solved across the population (the best host counts under SBB and SBB-RAW are similar under the real-valued reward scheme, Table 6.3).

Figure 6.13 also illustrates a fundamental difference between binary and real-valued reward functions. Specifically, under binary rewards, Figures 6.13a and 6.13b, point population diversity initially decreases and is then (at least partially) restored. This suggests that in the early generations the host and point populations are effectively disengaged, but that a learning gradient is eventually established. In contrast, under real-valued reward schemes, Figures 6.13c and 6.13d, it appears that a learning gradient is present from the start, where this results from the capacity of the real-valued approach to credit partial solutions. As suggested earlier, the initial convergence observed using both algorithms in the binary case is consistent with the identification of a small region of the state space that can be solved by one or more

hosts. If no other points are solved at that time, then only the points in that region receive a non-zero reward, and a rapid convergence to that region is expected. Without an incentive for points to be physically different, however, the SBB-RAW populations are then unable to regain earlier levels of genotypic diversity.

In summary, by providing partial reward values, the real-valued reward scheme appears to support greater levels of genotypic diversity in the point population. In particular, the same initial convergence as is observed using both SBB and SBB-RAW under binary rewards does not occur under real-valued rewards. Furthermore, under both reward schemes, the level of genotypic diversity in the point population appears to be greater when the genotypic diversity component is included in the point fitness function. This, in turn, results in a tendency towards an increased number of test cases solved under the SBB approach, where the difference is greatest with respect to the population-wide counts.

6.5 Summary of Results

Based on the results presented in this chapter, the following conclusions can be drawn regarding test case selection under temporal sequence learning:

1. Compared to the SBB-RSS and SBB-SSS formulations, a significant difference was observed in favour of the SBB approach under binary rewards but not under real-valued rewards. Thus, in the latter case, the naive sampling heuristics represent a reasonable option, but under binary rewards they are inappropriate.

2. A statistically significant difference between the performance of traditional distinction-based coevolution, SBB-DST, and the SBB approach was not found. Distinction-based coevolution, however, is less computationally efficient, suggesting a preference for the proposed approach.

3. Including a genotypic diversity component in the point fitness promotes better coverage of the state space. This results in better overall performance under binary rewards and in terms of population-wide counts under real-valued rewards.

In general, the less sophisticated point selection policies, i.e., SBB-RSS, SBB-SSS, and SBB-RAW, may represent suitable alternatives under real-valued cost functions which tend to provide more information about solution quality than binary cost functions. When less information is available, as in the case of binary rewards, then greater effort

must be taken to maintain a set of discriminating and diverse training points. With respect to the third point noted above, including a genotypic diversity component in the point fitness requires a suitable distance metric to be available and incurs additional computation overhead. However, the extra effort applied in finding a good set of points may be more easily justified when evaluation is costly, as is the case under temporal sequence learning.

Chapter 7

Truck Reversal: Comparison with Neuroevolution of Augmenting Topologies

In this chapter, the SBB framework is compared against NEAT, an evolutionary algorithm for generating neural networks of arbitrary topologies [177]. When it was first introduced, NEAT was shown to outperform leading neuroevolutionary approaches on both Markovian and non-Markovian temporal sequence learning tasks. Today, NEAT is still regarded as state-of-the-art[1], a status reflected in the amount of active research and publications related to the approach, e.g., [63, 108, 182].

The primary goal of this chapter is therefore to determine how the quality of the SBB solutions compares with the state-of-the-art in the temporal sequence learning domain. Since the comparison is made on the truck reversal environment as presented in Chapter 6, the difficulty of this particular problem formulation is also established. Furthermore, for the first time in this thesis, use is made of hierarchical model building, Section 3.4, so any benefits associated with this approach to layered learning are also investigated.

The chapter first summarizes the key concepts behind the NEAT algorithm in Section 7.1. The experimental setup, including the choice of NEAT implementation and its parameters, is then described in Section 7.2. Results are presented in Section 7.3, including an in-depth analysis of a solution evolved under the SBB algorithm, and summarizing remarks are made in Section 7.4.

7.1 Neuroevolution of Augmenting Topologies: Key Concepts

NEAT encodes neural networks of arbitrary topologies by using a variable-length linear representation consisting of *connections genes* and *node genes*. The node genes define the input, hidden, and output nodes, while the connection genes specify the (possibly recurrent) network links, i.e., the link endpoints and weight. Through a series of ablation experiments, three components of NEAT were shown to be critical to its effectiveness [177]:

[1]Though it is noted that recently a new incarnation of the NEAT algorithm has been proposed [178].

1. **Historical markings**. Whenever a new connection gene is introduced into the population, it is marked using a unique innovation identifier which is then inherited by all its descendants. These markings are meant to enable a more meaningful crossover to occur by allowing genes representing the same network structure to be aligned, i.e., crossover aims to model homologous recombination.

2. **Speciation.** A form of explicit fitness sharing is used to prevent a single niche of organisms (topological structures and associated weights) from dominating the population, and to protect novel structures which may require time to be optimized to the point where they become competitive. The innovation identifiers associated with each connection gene form a basis for the distance function between two organisms which in turn is used to define niche membership.

3. **Incremental growth.** The NEAT populations is initialized to contain small (minimal) networks. It is then suggested that since increases in complexity must be justified through natural selection, the search space is bounded thus helping to increase efficiency.

These components, including their inter-dependencies, are detailed in the original description of the NEAT algorithm [177]. Here, it is noted that the historical markings and speciation are two mechanisms whose aim it is to support diversity in the NEAT populations.

7.2 Experimental Setup

After a brief remark on the truck reversal environment in the section following immediately, the NEAT implementation that was used is detailed in Section 7.2.2. The parameters used under both the NEAT and SBB algorithms are then presented in Section 7.2.3, and the evaluation methodology is summarized in Section 7.2.4.

7.2.1 Problem Domain

The truck reversal problem as described in Section 6.2 was used in the evaluation. For reasons noted below, fitness evaluation was based on the real-valued reward scheme, Section 6.2.5.

7.2.2 Neuroevolution of Augmenting Topologies: Implementation

The NEAT C++ package [175], also used in the original NEAT dissertation [177], was used in the comparison. This implementation does not include a component for coevolving a set of fitness cases, whereas the SBB algorithm coevolves the points and hosts allowing the approach to scale to large state spaces. To address this issue, NEAT was compared with the SBB-RSS formulation as specified in Section 6.3.3, and used the same strategy for managing the set of training fitness cases[2]. Specifically, under both approaches, each candidate solution was evaluated over P_{size} points, and every generation P_{gap} of these points were uniformly selected and replaced with random points. In the case of NEAT, the fitness of an individual in a given generation was taken as the sum of the rewards over all the points in the corresponding point population.

The rationale behind the decision to use RSS as a point sampling heuristic was to make minimal changes to the NEAT implementation. Given this choice of point sampling heuristic, the real-valued reward scheme, Section 6.2.5, was used to define the outcome of applying an individual to a single point. The decision to do this was based on the results of the previous chapter where it was found that random sampling performed adequately under a real-valued reward scheme.

Unless otherwise noted, NEAT was configured as in the chosen distribution [175], which in turn reflects the algorithm as originally described [177]. In particular, the discrete multi-point crossover was used, as was the steepened Sigmoid transfer function $f(x) = (1 + e^{-4.9x})^{-1}$. The NEAT parameter settings used here are further detailed in Section 7.2.3.

The NEAT starter genome, Figure 7.1, consisted of five input nodes, three output nodes, and no hidden nodes. Each input was connected to all the outputs. The first four inputs corresponded to the state variables (x, y, Θ_s, and Θ_c), and the fifth input represented the bias (value fixed at 0.5). The three outputs corresponded to possible settings of the steering angle u assuming values of $0°$, $-30°$, and $30°$, i.e., the possible actions under the SBB algorithm and NEAT were the same. Given the three outputs, at each time step during an episode, the action associated with the output having the highest activation was used to set the steering angle u. This starter genome, representing the connectivity of all networks in the initial population, is consistent with the NEAT philosophy of starting with minimal structures and at the same time

[2]In the remainder of this chapter, references to the SBB algorithm assume the RSS point sampling heuristic.

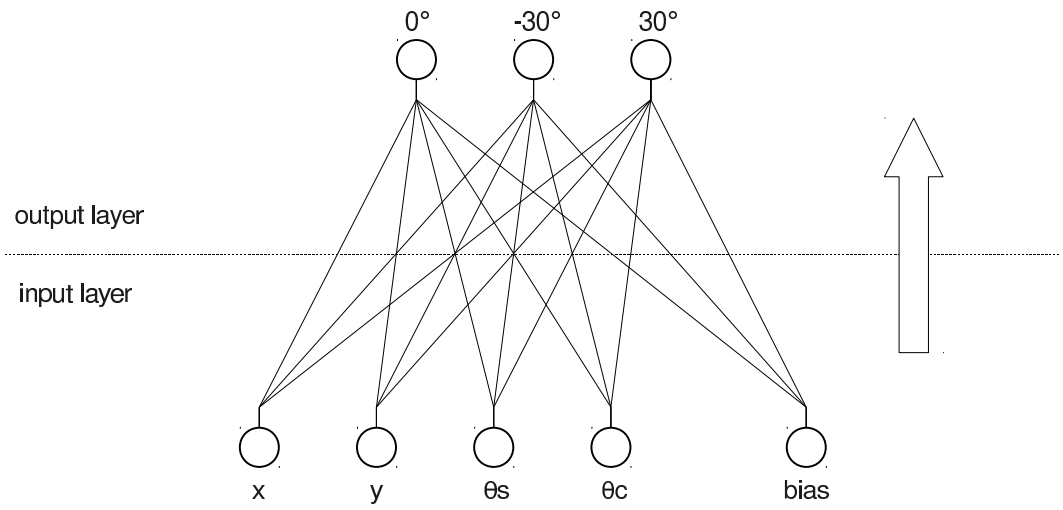


Figure 7.1: NEAT starter genome under the truck reversal domain representing the connectivity of all initial networks. The block arrow indicates the direction of activation.

does not make assumptions about the initial connections.

7.2.3 Parameters

With one exception, the SBB parameters were set as in Section 6.3.1; to determine if it would lead to improved solution quality, in this chapter, two levels were evolved instead of one. Each level was trained using the same parameters, Tables 4.3 and 4.4, which included the same number of generations. No claims are made regarding the optimality of these settings, and it is acknowledged that different parameter values at each level may be more appropriate. Given that the number of SBB fitness evaluations per level was kept as before, the total number of fitness evaluations was a multiple of the number of levels evolved.

The NEAT parameter settings used here were based on those in the original double pole-balancing with velocities experiments [177], i.e., unless otherwise noted, the same values were employed. In the original experiments, similar settings were also used across different problems, and it was suggested that for the most part the system was robust to non-trivial variation in the parameters. However, three mutually-sensitive sets of parameters were identified as especially important: the population size, the compatibility threshold (used to determine if two individuals are in the same species),

	Non-recurrent	Recurrent
High mutation	H-NR	H-R
Low mutation	L-NR	L-R

Table 7.1: NEAT parameter tuning configurations.

and the weights used to calculate distances between individuals. By using already established values for these parameters, the core set of settings used here was assumed to be appropriate.

Given the settings employed in the original experiments as a starting point, some tuning was done with respect to the *link weight mutation probability* and the *recurrent link probability*. Tuning of the former was explicitly suggested by the authors of NEAT [176], whereas the latter was tuned to determine if the system could benefit by having the ability to maintain an internal state (a capability not present in the current version of the SBB algorithm). Two values for the link weight mutation probability were explored, 0.9 (high mutation) and 0.1 (low mutation), and two values were also explored for the recurrent link probability, 0.1 (recurrent) and 0.0 (non-recurrent). This resulted in four possible configurations, Table 7.1, whose performance over 20 initializations is summarized in Figure 7.2. This evaluation, including the selection of the best-of-run individual, was performed as described in Section 7.2.4, i.e., both best individual and population-wide counts were considered. Given these results, it is clear that high mutation rates result in better performance. The H-NR and H-R configurations yield very similar performance with respect to the best individual, but better best-case population-wide counts are obtained by allowing recurrent connections. As such, the H-R configuration was selected for comparison with the SBB algorithm.

The set of NEAT parameter settings that were ultimately used in the comparison are detailed in Table 7.2. The *excess*, *disjoint*, and *weight difference coefficients* are the weights used in calculating the distance between individuals. The distance is compared against the *compatibility threshold* to determine species membership. Together with the *population size*, these parameters represent the core set mentioned above that is viewed as critical to the successful operation of NEAT. The *survival threshold* determines the proportion of individuals that form the parent set in each generation. The *link weight mutation probability* (one of the parameters that was tuned) controls how often weights are mutated, while the magnitude of the mutations is controlled using the *weight mutation power*. The *toggle gene enable probability* and

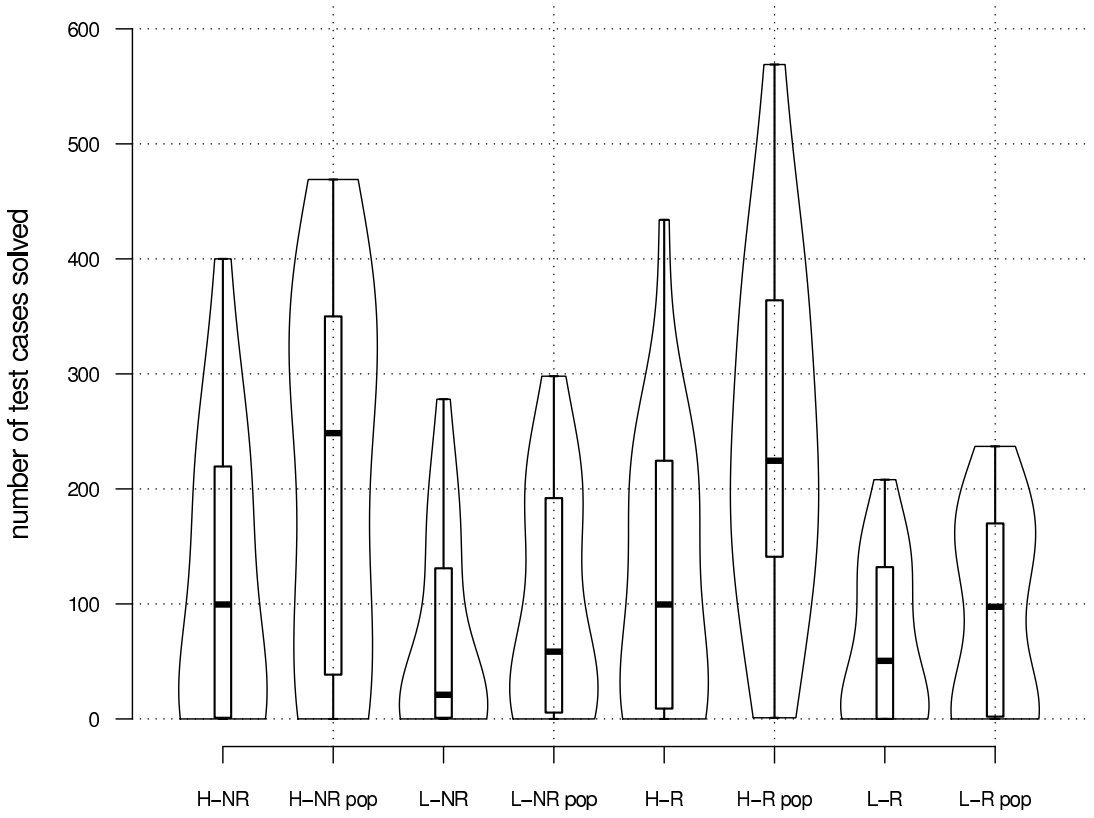


Figure 7.2: NEAT parameter tuning results. For each configuration, Table 7.1 and x-axis, shown is the number of test cases solved across 20 initializations, y-axis. Population-wide counts are labeled 'pop'. The configuration viewed as the best, H-R, was used in the comparison with the SBB algorithm.

the *gene re-enable probability* refer to, respectively, the rate at which the active status of genes is flipped and the rate at which inactive genes are re-enabled[3]. The *recurrent link probability* (the other parameter that was tuned) controls how often recurrent links are added. The *dropoff age* defines how long before a species or the entire population is considered to be stagnant (used to determine when to penalize species or to perform delta coding). Finally, *new link tries* refers to the number of times an attempt is made to find an open set of endpoints during an add link operation. The *number of generations* was set so that the total number of fitness evaluations performed under NEAT and across all levels of the SBB algorithm was approximately the same[4]. More information on the NEAT parameters can be found in the original dissertation [177] and in the NEAT implementation documentation [175].

Under both the NEAT and SBB algorithms, P_{size} and P_{gap} was set at 120 and 20 respectively[5], i.e., the fitness of each individual was assessed over 120 test cases and 20 of those test cases were replaced every generation as per the SBB-RSS formulation, Section 6.3.3. Using each algorithm, 60 initialization based on different random seeds were performed.

7.2.4 Evaluation Methodology

Using the same parameter settings at each level, training two levels under the SBB algorithm requires twice as many fitness evaluations as training just a single level. Thus, to determine if any observed improvement in solution quality at the second level was not merely a result of doubling the training effort in terms of the number of fitness evaluations, the two-level SBB solutions were also compared to single-level solutions that were evolved for twice as long. The latter, denoted SBB-2k, were obtained by training just the one SBB level but for 2000 generations, and otherwise using the same parameters.

The three approaches that were compared, SBB, SBB-2k, and NEAT, were then evaluated as described in Section 6.3.2. In particular, under each of the three approaches, a validation set of size 1000 was used to select a single best individual from

[3]It is not clear how these two parameters were set in the original NEAT experiments, hence, the values used here may be different.

[4]The number of fitness evaluations performed in each NEAT and SBB run was 16 812 000 and 16 800 000 respectively.

[5]These parameters do not appear in the original NEAT formulation [177] but result from the inclusion of the random point sampling heuristic.

Description	Notation	Value
Population size	n	150
Excess coefficient	c_1	1.0
Disjoint coefficient	c_2	1.0
Weight difference coefficient	c_3	0.4
Compatibility threshold	δ	3.0
Survival threshold		0.2
Link weight mutation probability		0.9
Weight mutation power		2.5
Toggle gene enable probability		0.01
Gene re-enable probability		0.001
Add node probability		0.03
Add link probability		0.05
Recurrent link probability		0.1
Interspecies mating rate		0.001
Mate only probability		0.2
Mutate only probability		0.25
Dropoff age		15
New link tries		20
Number of generations		934

Table 7.2: NEAT parameters used in the truck reversal experiments. Where applicable, the notation used for a parameter in the original NEAT description [177] is shown.

the final population, i.e., the individual yielding the highest mean reward on the validation set. This best individual was then applied to an independent test set of size 1000, and populations-wide counts on this test set were also considered.

Given the parameters used, at the end of training, the SBB host populations contained 60 individuals while the NEAT network populations contained 150 individuals. Thus, with respect to the population-wide solved counts, NEAT is viewed as having an advantage; in calculating the population-wide counts, the tally cannot be reduced as more individuals are considered, but it can always increase.

7.3 Results

The presentation of the results begins with the performance on the test set and the solution complexities in Section 7.3.1. These are followed by an overview of the attributes used in the SBB solutions, Section 7.3.2, as well as an analysis of the best SBB solution that was found, Section 7.3.3.

7.3.1 Performance and Complexity

Performance

With respect to the number of test cases solved, the second-level SBB solutions tend to outperform the first-level SBB solutions in terms of the best individual and the population-wide counts, Figure 7.3. The second-level SBB counts also tend to be higher than the corresponding NEAT and SBB-2k counts. Specifically, with respect to the best individual, the second-level SBB counts, SBB 1, tend to be higher that the NEAT and SBB-2k counts, NEAT and SBB-2k respectively. In addition, the second-level SBB population-wide counts, SBB 1 pop, appear to be higher than the population-wide NEAT and SBB-2k counts, NEAT pop and SBB-2k pop respectively. A two-tailed Mann-Whitney test applied to pairs of corresponding distributions, Table 7.3, supports these observations at a significance level of 0.01.

Thus, it appears that the SBB 0 distribution in Figure 7.3 represents the lowest performance point in terms of best individual counts, however, the focus in the SBB framework at the first level is diversification as is apparent in the corresponding first-level population-wide counts, SBB 0 pop. This is especially evident when comparing the NEAT and NEAT pop distributions with the SBB 0 and SBB 0 pop distributions. The individual hosts at the second SBB level are then able to combine the population

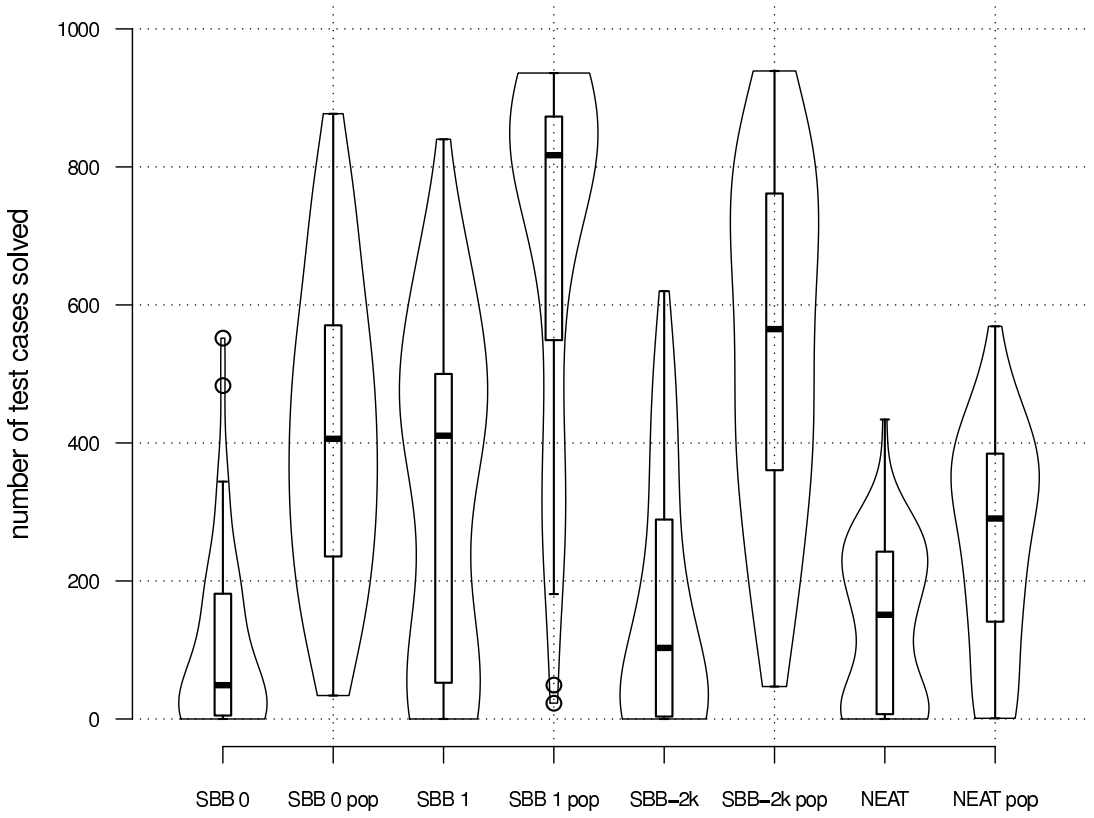


Figure 7.3: Comparison of the NEAT and SBB algorithms on the truck reversal problem in terms of the number of test cases solved by the best individual and across the final population. In all cases, RSS is used to sample training points. The comparison includes the single-level SBB configuration SBB-2k, and where applicable SBB performance is also broken down by level. Distributions corresponding to the population-wide counts are labeled 'pop', while SBB results at the first and second level respectively include '0' and '1' in the label. All distributions represent 60 initializations.

	Best individual	Population
SBB 1 vs SBB 0	2.683e-07⋆	7.76e-09⋆
SBB 1 vs SBB-2k	3.774e-05⋆	1.386e-04⋆
SBB 1 vs NEAT	4.636e-06⋆	1.783e-13⋆

Table 7.3: Two-tailed Mann-Whitney test results comparing second-level SBB solutions with NEAT, SBB-2k, and first-level SBB solutions with respect to the number of test cases solved. Shown are the p-values when corresponding distributions in Figure 7.3 are compared, e.g., SBB 1 versus NEAT and SBB 1 pop versus NEAT pop. Cases where the SBB median value is *higher* are noted with a ⋆.

diversity in the first level, resulting in best host performance at the second level that appears to match the population-wide counts at the first level; this leads to SBB 1 counts that represent the best individual-wise performance observed across the three formulations. Likewise, the second-level SBB results represent the highest levels of performance in terms of population-wide counts.

These results underlie the role that layered learning – through a symbiotic model of inheritance – may play when the difficulty of the problem domain is such that suitable levels of performance cannot be reached in a single run. Furthermore, the improvement observed at the second level is not due entirely to the application of twice the training effort as measured in terms of the number of fitness evaluations. Comparing the second-level SBB distributions with the corresponding SBB-2k distributions, Figure 7.3, the improvement appears to be especially significant when the best individual counts are considered, SBB 1 versus SBB-2k. The strong SBB host performance at the second level also results in higher and more consistent population-wide counts, e.g., the difference in median counts between the SBB 1 pop and SBB-2k pop distributions is well over 200. This suggests that the proposed hierarchical model building approach is more effective compared to a flat, single-level, approach.

It is also interesting to note that the second-level hosts are able to combine the set of fixed, first-level, behaviours in novel ways, i.e., the maximum number of test cases solved population-wide at the second level is greater than at the first level. This implies that the second-level hosts are finding ways to effectively use different first-level hosts at different time steps.

A good way to visualize the impact of hierarchical model building is to compare, in Figure 7.3, the improvement from SBB 0 to SBB 1 versus the improvement from SBB 0 to SBB-2k. The former improvement results from adding a second level, whereas the latter improvement results from training for twice as many generations. A similar comparison can be made with respect to the population-wide counts, but because the SBB 0 pop values are already relatively high, the difference in improvement is reflected more in the shape of the distribution as opposed to the observed range of values, i.e., the SBB 1 pop values are more consistent than the SBB-2k pop values.

The SBB solutions at the second level also tend to do considerably better than the NEAT solutions both with respect to the best individual as well as the population-wide solved counts. Despite relying on a speciation mechanism, thus encouraging diversity, the NEAT population-wide counts appear to be relatively low even compared to the first-level SBB counts. This is especially significant given that the NEAT

counts are taken over 150 individuals whereas the SBB counts are taken over 60 individuals. However, excluding the second-level SBB results, the best individual counts under NEAT appear to be competitive. This suggests that the NEAT speciation mechanism is more effective at establishing context for crossover, resulting in relatively strong individuals but weak population-wide behaviour. In comparison, the SBB framework is able to produce strong population-wide behaviour by supporting diversity at multiple levels, i.e., through a combination of implicit fitness sharing applied to the symbionts and explicit fitness sharing applied to the hosts, and then exploiting this diversity at the second level through context learning by way of the bid-based metaphor. As such, the SBB framework appears to be more capable of leveraging the potential benefits of layered learning.

Complexity

Though, given the parameter settings that were used, the SBB and SBB-2k configurations incur the same amount of computational overhead in terms of the number of fitness evaluations, the complexity of the SBB solutions has the potential and is likely to be higher. Specifically, a solution under the SBB-2k configuration can consist of as many as 10 symbionts. A solution under the SBB configuration, on the other hand, can consist of 10 symbionts at the second level, each referencing a host of size 10 at the first level, for an overall upper bound of 110 symbionts. However, even though the upper bound on the complexity of the SBB solutions is more than ten times the upper bound on the complexity of the SBB-2k solutions, the difference in computational effort as measured in terms of the maximum number of instructions executed *during training* is only one-and-a-half-fold[6]. More generally, because they are structured hierarchically, not all subcomponents in a given multi-level SBB solution are engaged at each time step – this is in contrast to ensemble methods where all ensemble members participate in a decision. This results in a runtime computational overhead *post-training* that is linearly proportional to the number of levels in, and not the complexity of, the SBB solutions.

Given the higher upper bound on the complexity of the SBB solutions, an additional set of runs was carried out to investigate the effect of increasing the maximum

[6]This can be seen by equating the maximum number of instruction executions required to train the SBB-2k configuration for one generation (proportional to $H_{size} \times \omega \times maxProgSize$) with one unit of overhead. Thus, the total effort expanded in each SBB-2k run is 2000 units of overhead. For a given SBB run, the effort is 1000 units expanded on training the first level (same as training SBB-2k but for 1000 generations) and 2000 units expanded on training the second level (since, in computing an action, only one host is applied at each of the two levels), for a total of 3000 units.

solution complexity under the SBB-2k configuration, Figure 7.4. Keeping all other parameter settings the same, the maximum host size ω was doubled from 10 to 20 resulting in a 'big host' configuration SBB-2k-bh. Based on the results, no statistically significant difference in performance was observed between corresponding distributions. In fact, the median solution counts decreased when the maximum host size was increased[7]. This suggests that increasing the maximum solution complexity under a single-level approach will not necessarily lead to improved performance. This may be due in part to the fact that, without the means to structure the additional complexity as in (for example) a hierarchy, the single-level approach is not well suited to the expanded search space.

Thus, when analyzing solutions, it may be useful to consider the way in which increases in complexity come about. On the one hand, increased complexity may be a side-effect of the learning process with no real contribution to the bottom line. This appears to be the situation in the case when the maximum host size is increased under the SBB-2k configuration, Figure 7.4. On the other hand, under the hierarchical SBB approach, complexity is methodically added in a structured and controlled way with the goal of improving performance. As such, in the latter case, increases in complexity may be more easily justified, especially if accompanied by significantly higher levels of performance.

The best single-individual SBB solutions are characterized with respect to the number of effective instructions per host, the number of symbionts per host, and the number of effective instructions per symbiont, Figure 7.5. Here, the SBB 1 counts do not include the SBB 0 counts, e.g., the instruction counts at the first level are not counted again at the second level. In both SBB levels, SBB 0 and SBB 1, the complexities are comparable to the SBB-2k complexities, but because the final SBB solutions span two levels, it is clear that the aggregate multi-level solution complexities are significantly greater. The SBB-2k-bh instruction counts across the entire host, Figure 7.5a, and the host sizes, Figure 7.5b, tend to be higher than the corresponding SBB-2k counts. However, there appears to be a net decrease in the complexity of the SBB-2k-bh solutions compared to the SBB-2k solutions when individual symbionts are considered, Figure 7.5c, with a two-tailed Mann-Whitney test returning a p-value of 0.001806. Thus, increasing the limit on the number of symbionts that can be combined in a single host appears to result in more symbionts per host, though the symbionts themselves appear to be simpler, i.e., simpler solution subcomponents

[7]The two highest values under the SBB-2k-bh distribution in Figure 7.4 are 604 and 766, suggesting that the latter is an outlier.

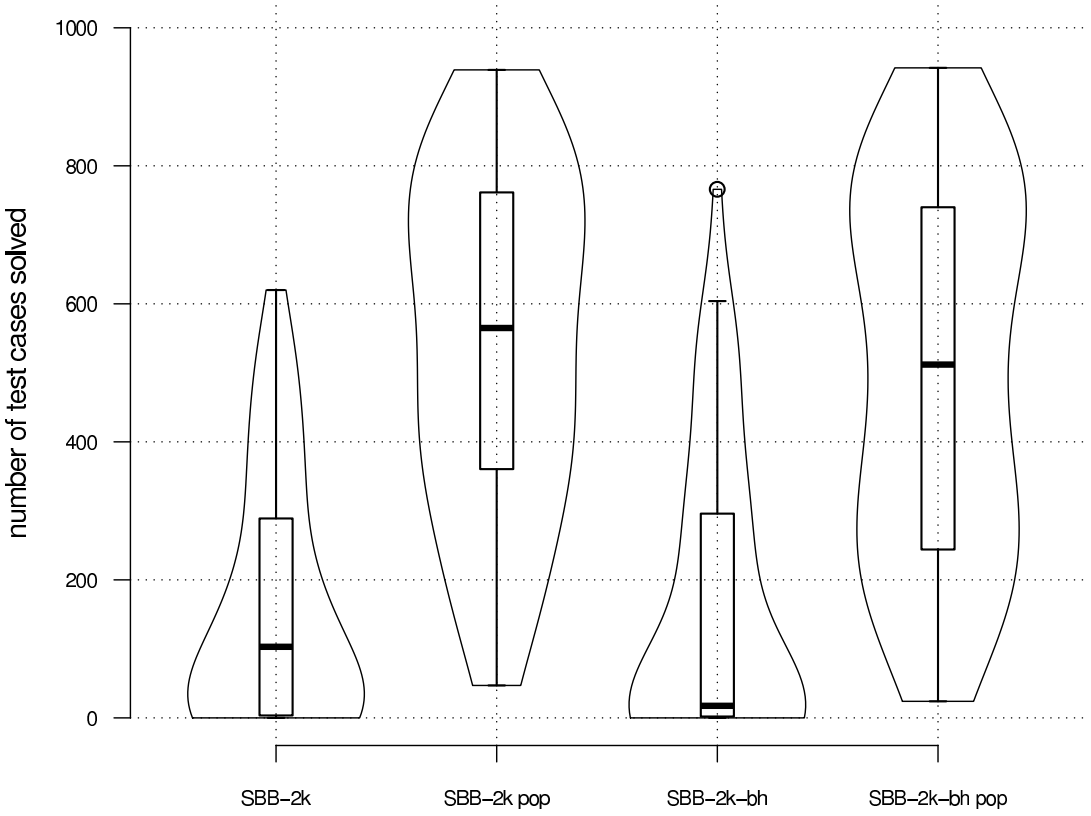


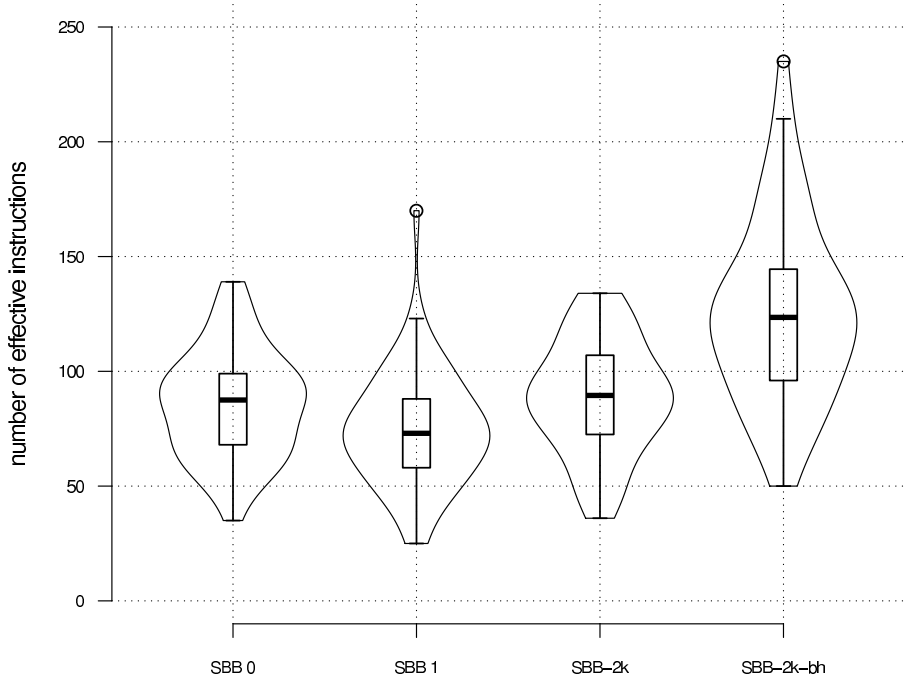
Figure 7.4: Comparison of SBB-2k performance under two maximum host size thresholds. Shown is the number of test cases solved by the single best host and across the population, with the population-wide counts labeled 'pop'. Distributions labeled SBB-2k-bh represent a configuration where the maximum host size, ω, was doubled relative to the original configuration of SBB-2k (all other parameters remained the same). Each distribution represent 60 initializations. Two-tailed Mann-Whitney tests applied to the two single-individual distributions and the two population-wide distributions return p-values of 0.6457 and 0.3694 respectively.

appear to be identified, though additional data would be required to confirm this trend.

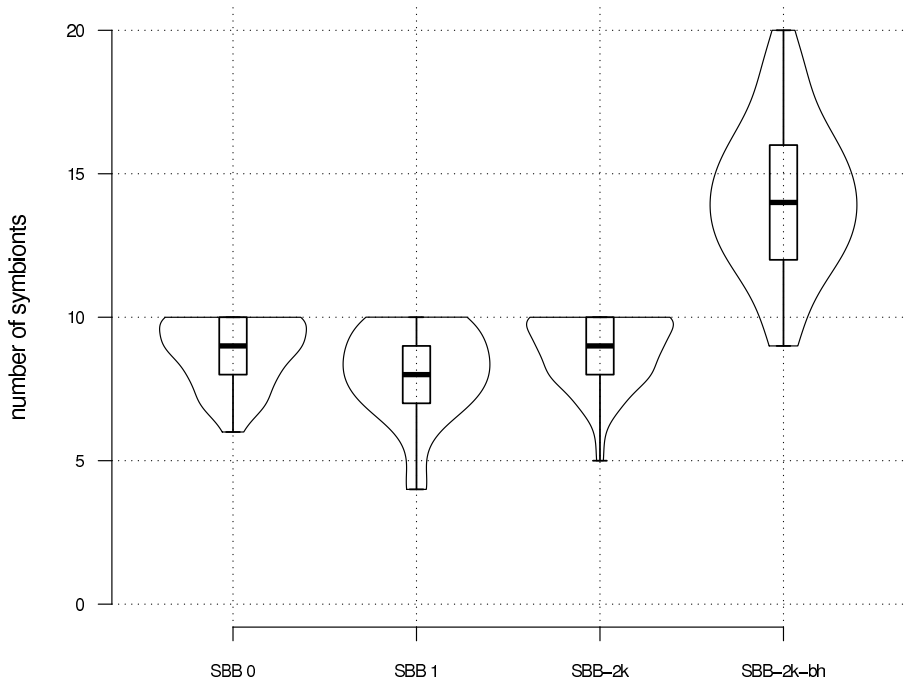
The complexities of the NEAT solutions, in terms of the number of nodes and links in the single best individual, are shown in Figure 7.6. For comparison, the figure includes a distribution representing SBB complexities expressed in terms of the number of unique symbionts, i.e., combinations of bid program and action, across both levels of the two-level solution. This includes the level 0 symbionts, where the associations involve one of the domain-specific actions, as well as the level 1 symbionts, where the associations involve an action defined with respect to the first-level hosts. Alternatively, given that there is a one-to-one correspondence between symbiont and program, the symbiont counts are equal to the total number of unique programs in the solution. The SBB counts in Figure 7.6 appear to be lower, though the significance of this result is open to interpretation given the different representations used by the two approaches.

7.3.2 Attributes Used

Additional analysis was performed with respect to the attributes used by the symbionts in the best hosts across all runs, Figures 7.7 through 7.9. The mean number of unique features accessed per symbiont, Figure 7.7, suggests that bidders typically do not use all of the features and that fewer features are used at level 1, though no difference in the two distributions was detected by a two-tailed Mann-Whitney test at a 0.01 significance level. Considering the rate at which symbionts access specific features, Figure 7.8, at level 0 the focus tends to be more on the angles. This is consistent with behaviour where the primary goal is not to jackknife the cab and semi, even if this comes at the cost of running into the obstacle or the y-axis away from the origin. This is a reasonable strategy since jackknifing results in a reward of zero. An additional property of the solutions is that every feature is used by at least one symbiont in the best host as no zero values are observed. Further breaking down the level 0 counts by considering the actions associated with the symbionts, Figure 7.9, it becomes clear that the programs associated with the $\pm 30°$ actions tend to use the x and y coordinates less than the programs associated with the $0°$ action. Thus, a strategy common to some of the solutions may be to set the steering angle straight unless there is a risk of jackknifing, a condition identified by considering the angles, at which point the wheel would be set to the left or to the right to mitigate the risk.

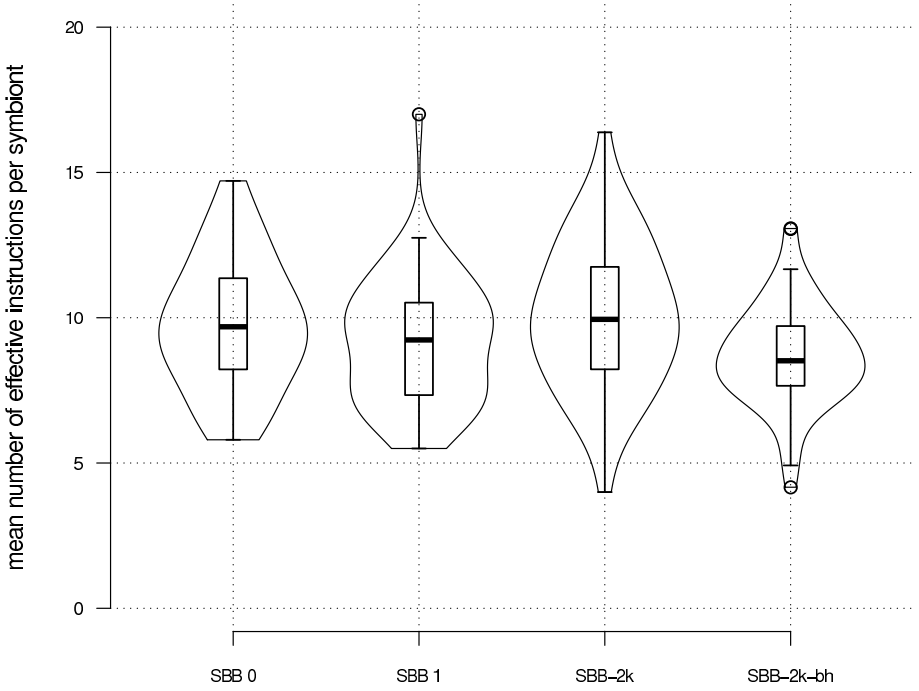


(a) Number of effective instructions per host.



(b) Number of symbionts per host.

Figure 7.5: Complexity of the SBB solutions evolved under the truck reversal domain with respect to the single best individual broken down by level (the best SBB individual at each level was selected independently as described in Section 7.2.4). Analogous SBB-2k as well as SBB-2k-bh counts are included for comparison. Each distribution represents 60 initializations.



(c) Mean number of effective instructions per symbiont.

Figure 7.5: Continued.

7.3.3 Analysis of a Solution

The structure of the best SBB solution, chosen as the one solving the most test cases, is illustrated in Figure 7.10. Using rectangles to represent hosts, the figure shows a level 1 host that contains eight symbionts referencing a total of five level 0 hosts. Thus, there are certain level 0 hosts which are referenced by more than one symbiont at level 1, indicated by a count next to the connector, e.g., the level 0 host labeled 2 appears three times as an action in the level 1 host, though the bid program of each symbiont is unique. The level 0 symbionts are grouped into sets, denoted by the ovals at the bottom of the figure, to indicate if there is overlap in the contents of the level 0 hosts. Each level 0 symbiont appears in exactly one of these subsets, and for each subset, the actions that is contains are indicated. For example, the leftmost subset contains six symbionts of action 30° and one symbiont of action −30°, and furthermore, the symbionts in that subset are unique to the host labeled 1. The host labeled 1 contains one additional symbiont, of action −30°, which is also referenced by host labeled 5. In total, there are thirty level 0 symbionts and eight level 1 symbionts in the solution, where each symbiont provides a unique context, i.e., bid program, for its associated action.

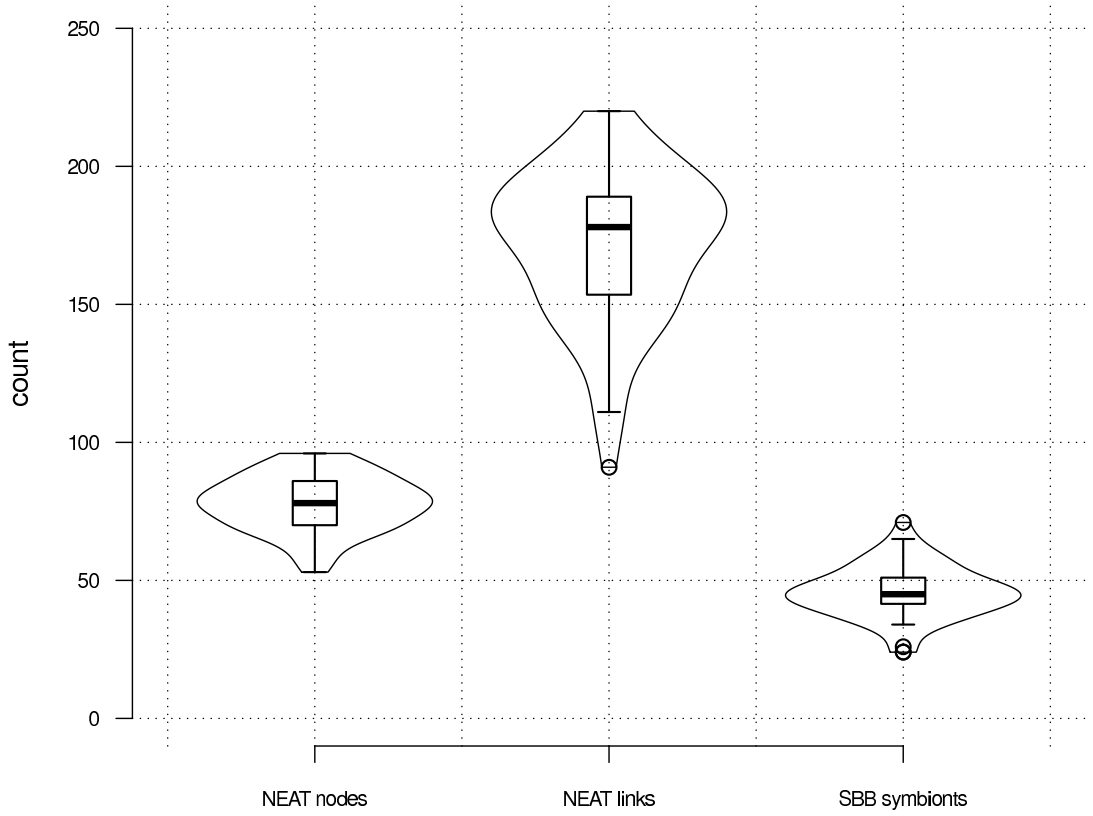


Figure 7.6: Complexity of the best single-individual SBB and NEAT solutions evolved under the truck reversal domain. The NEAT complexities are expressed in terms of the number of nodes and links in the network. The SBB complexities are expressed in terms of the number of unique symbionts across both levels of the solution, or equivalently, the number of bid programs in the solution. Each distribution represents 60 initializations.

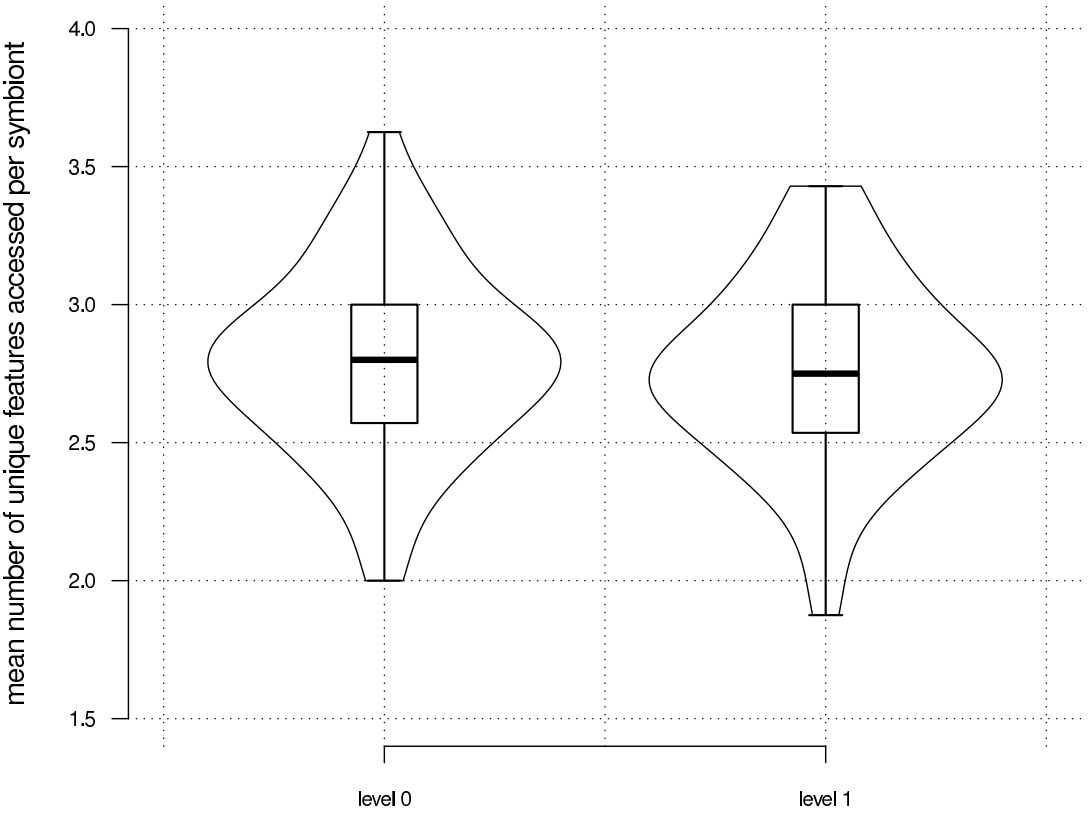
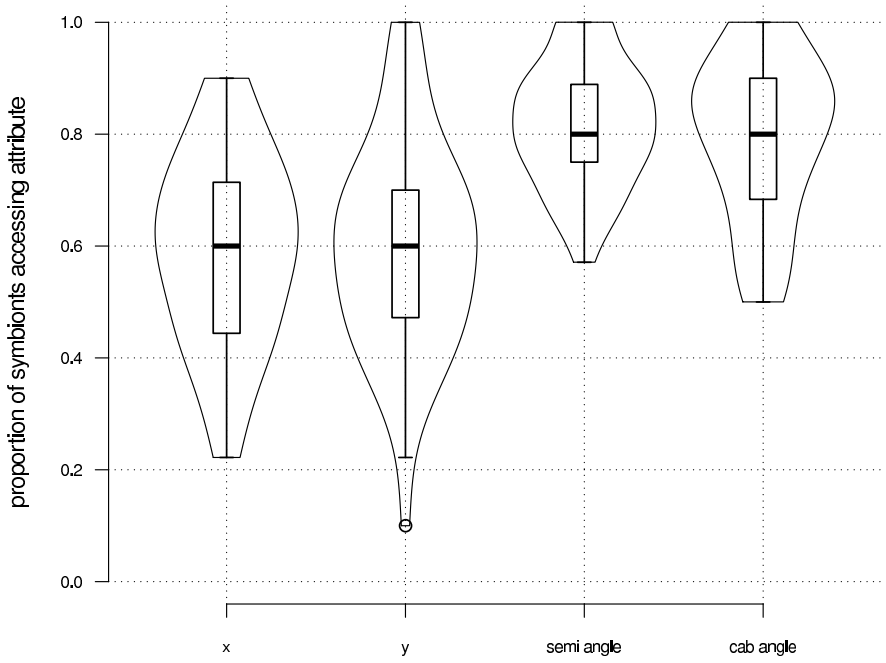
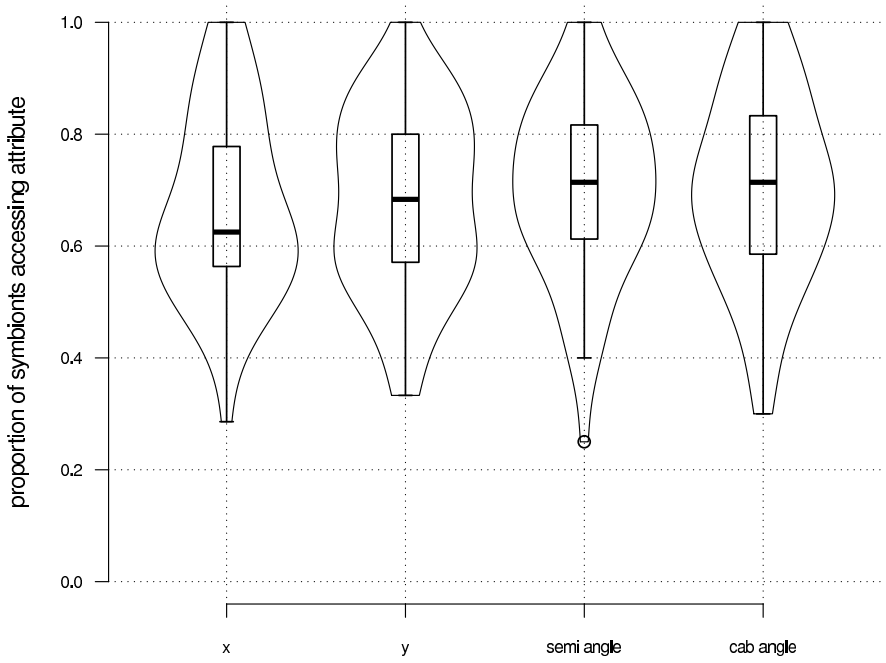


Figure 7.7: Mean number of unique features accessed per symbiont in the best host broken down by level (the best SBB individual at each level was selected independently as described in Section 7.2.4). Each point represents the mean taken across all symbionts in a given best host, and each distribution contains a point for each of the 60 initializations.



(a) Level 0.



(b) Level 1.

Figure 7.8: Proportion of symbionts in the best hosts accessing each attribute (the best SBB host at each level was selected independently as described in Section 7.2.4). For each attribute, x-axis, the proportion of symbionts in the host accessing that attribute is shown, y-axis, i.e., the calculation is performed across the symbionts in each host separately. The distributions, broken down by level, represent the best host in each of the 60 initializations.

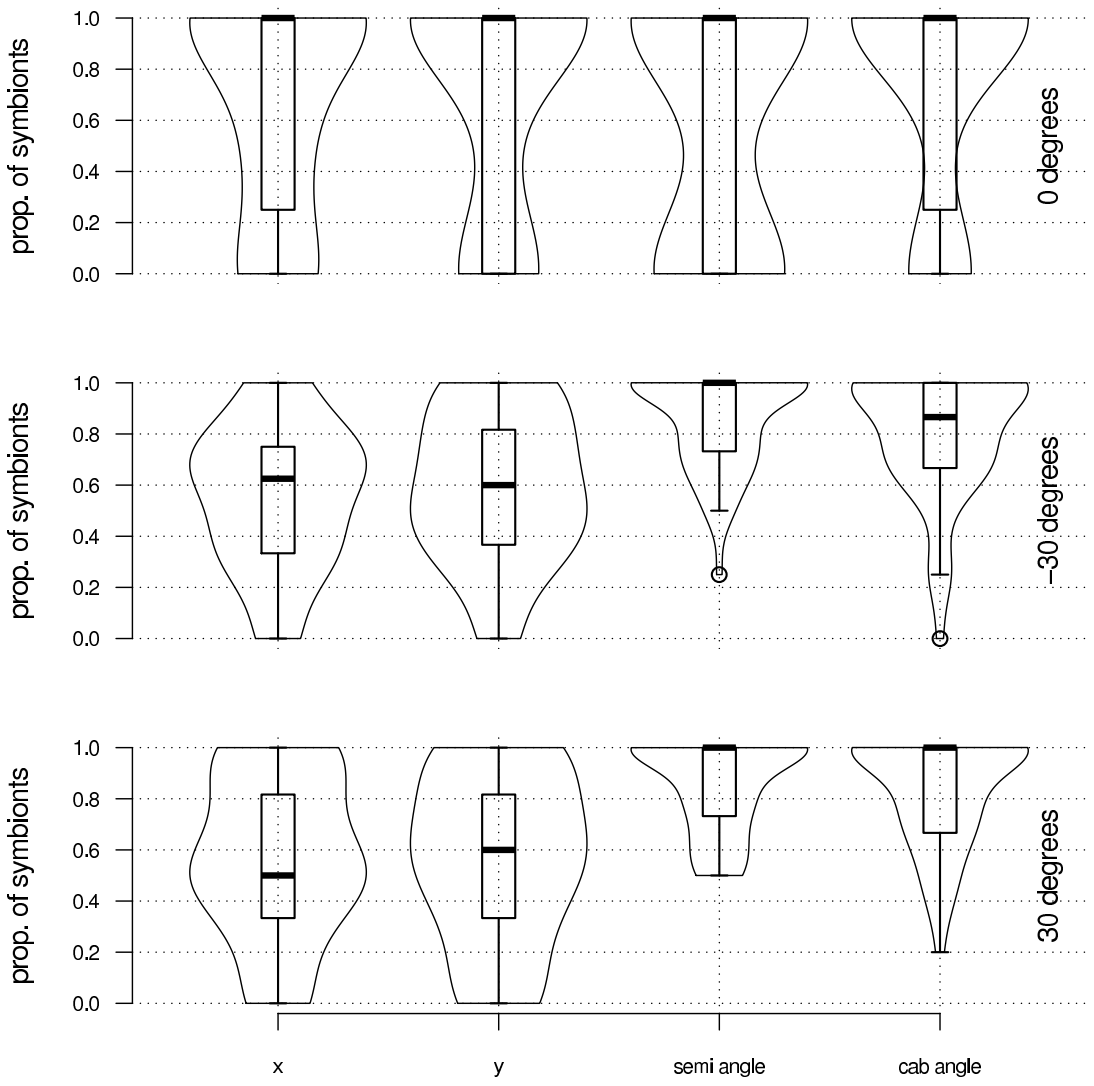


Figure 7.9: Proportion of symbionts in the first-level best hosts accessing each attribute broken down by action, distributions are otherwise as in Figure 7.8. Specifically, the mean is calculated as in Figure 7.8, but when considering a given action the symbionts in the best host associated with the two other actions are not counted. Only 28 points are plotted under the 0 degree action since that action was not represented in 32 of the 60 runs, and only the first level is considered since there are 60 actions at the second level.

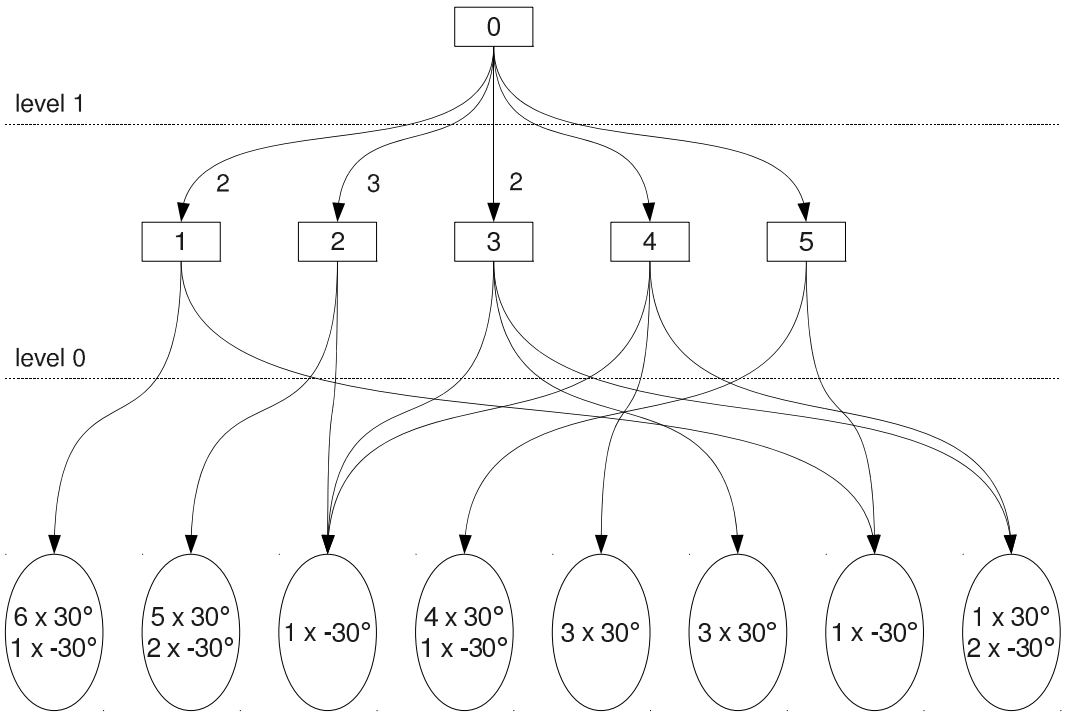


Figure 7.10: The best SBB solution under the truck reversal domain. The level 1 host (rectangle labeled 0) contains eight symbionts which reference a total of five level 0 hosts (rectangles labeled 1 through 5); multiple references to the same level 0 host are indicated by a count next to the connector, though the program associated with each reference is unique, i.e., it associates a unique context for the action represented by the level 0 host. The five level 0 hosts then reference a total of 30 level 0 symbionts which are grouped into sets to indicate which subsets of level 0 symbionts are common across the level 0 hosts. For each group, the number of times each action is represented within it is shown, i.e., the sum of the counts reflects the number of symbionts in the group. The solution therefore contains a total of 38 symbionts (unique programs): the 8 symbionts at level 1 and the 30 symbionts at level 0.

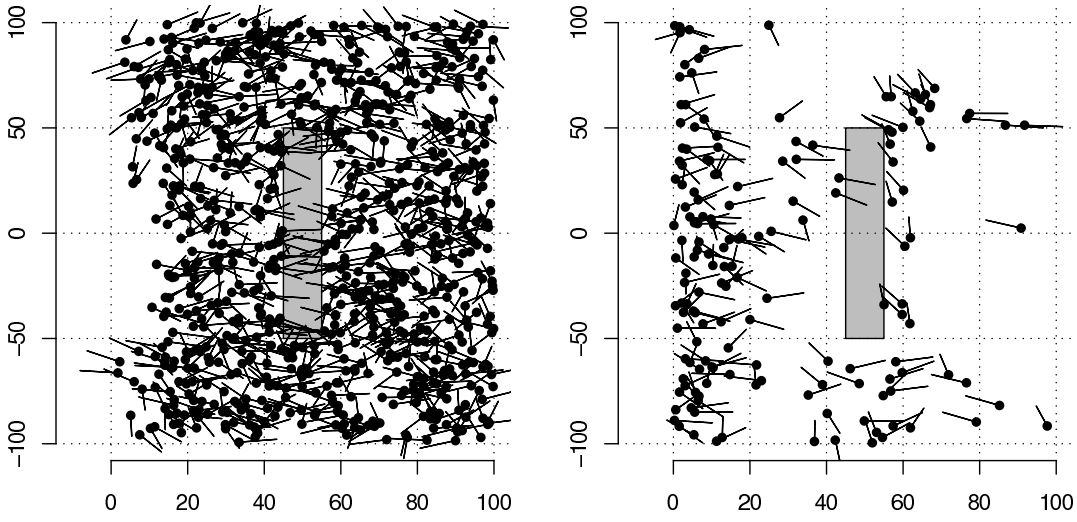
The best host, depicted in Figure 7.10, solved 840 of the 1000 test cases, Figure 7.11a. Of the 160 episodes corresponding to the tests where the solution failed, Figure 7.11b, 92 ended when the back of the semi crossed the y-axis, 45 ended when the back of the semi ran into the obstacle, and 17 ended when the cab and semi jackknifed, leaving a number of states when the episode was terminated because the time ran out[8]. The configurations at the moment of failure are shown in Figure 7.11c.

To gain insight into the control strategy employed by the best solution, the cab and semi configurations at times steps 100, 200, 300, and 400 are plotted across all the test points, Figure 7.12. The number of configurations that are plotted decreases as the time step count goes up since episodes may be terminated before the limit of 400 time steps is reached, e.g., if a configuration representing a solved state is reached. From the figure, it appears as though the strategy used is to approach the origin at a diagonal starting in the region to the left of the top of the obstacle. If the cab and semi are not in that region, they are steered towards it. For example, if the cab and semi start in the lower-right of the plane, they tend to circle around the top of the obstacle in a counter-clockwise direction.

For six different test points that were solved by the best solution, configurations of the cab and semi were plotted every 10 time steps resulting in a set of trajectories, Figure 7.13. The trajectories are consistent with the strategy interpreted from Figure 7.12, but they also reveal a peculiar behaviour. On two of the points, points 39 and 615, a spiraling trajectory is observed. This behaviour was likely learned in order to avoid jackknifing, a terminal condition that always results in the minimum reward. It appears to be exploited in order to direct the cab and semi towards a region of the state space where the solution knows how steer the cab and semi back to the origin, that is, the region to the left of the top of the obstacle. Though it is not the most efficient strategy, under the environment formulation that was used, Section 6.2, it results in a high number of solved test cases.

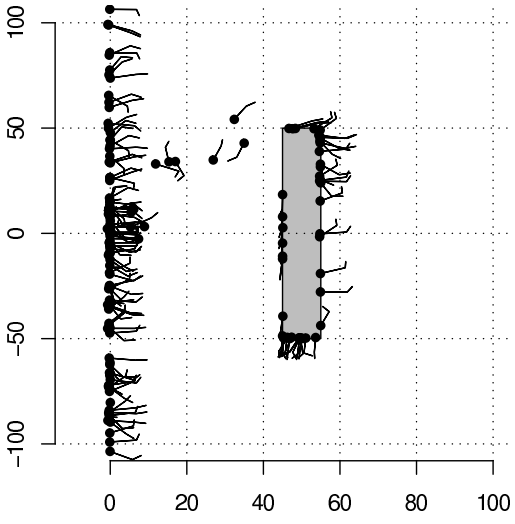
For two of the trajectories in Figure 7.13, additional information is plotted, Figures 7.14, and 7.15. Figure 7.14 shows the actions used to solve the point for which the trajectory loops around the top of the obstacle, including the level 0 host chosen at each time step. Based on the figure, certain roles become apparent among the first-level hosts. For example, the host denoted by the $\bullet$ is the cornerstone of the solution as it guides the cab and semi towards the origin along the diagonal as shown. The host denoted by $+$ appears to take control when the cab and semi move roughly up

[8]Sometimes two of these terminating conditions were in effect at the same time.



(a) Solved cases, initial states.

(b) Unsolved cases, initial states.



(c) Unsolved cases, final states.

Figure 7.11: Best SBB solution on the truck reversal problem, solved and unsolved test cases. The solution, depicted in Figure 7.10, solved 840 of the 1000 test cases. Shown are the initial states for the cases that were solved, and the initial and final states for the cases that were not solved. As noted before, the back of the semi is denoted by a •, and the cab and semi are aligned in all initial states.

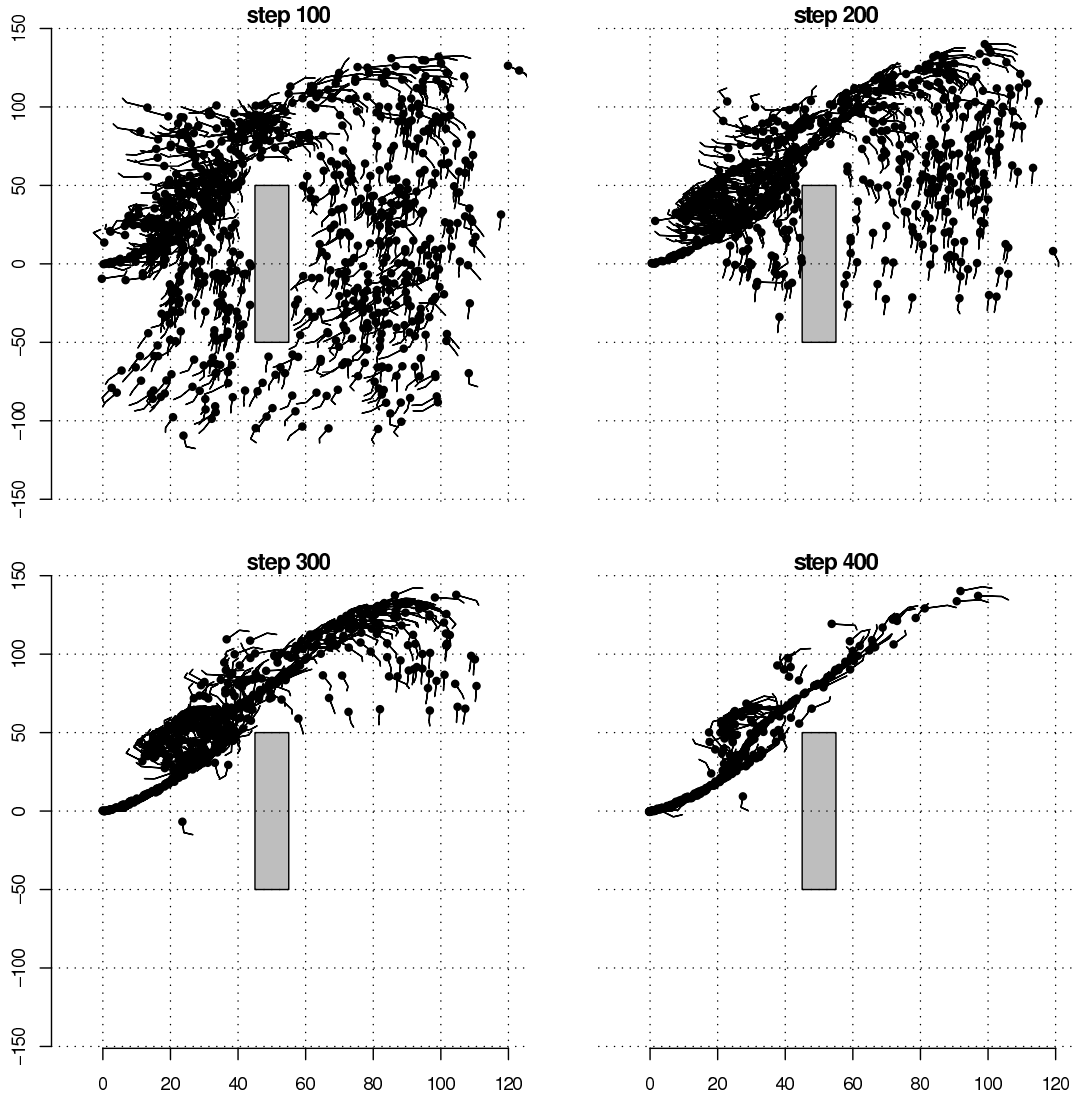


Figure 7.12: Cab and semi configurations at four different time steps across all test points. For the best single-individual SBB solution, Figure 7.10, the cab and semi configurations across all points are plotted at time steps 100, 200, 300, and 400. If, given a specific initial configuration, the episode ended earlier than the indicated time step, the associated state is omitted from the subplot.

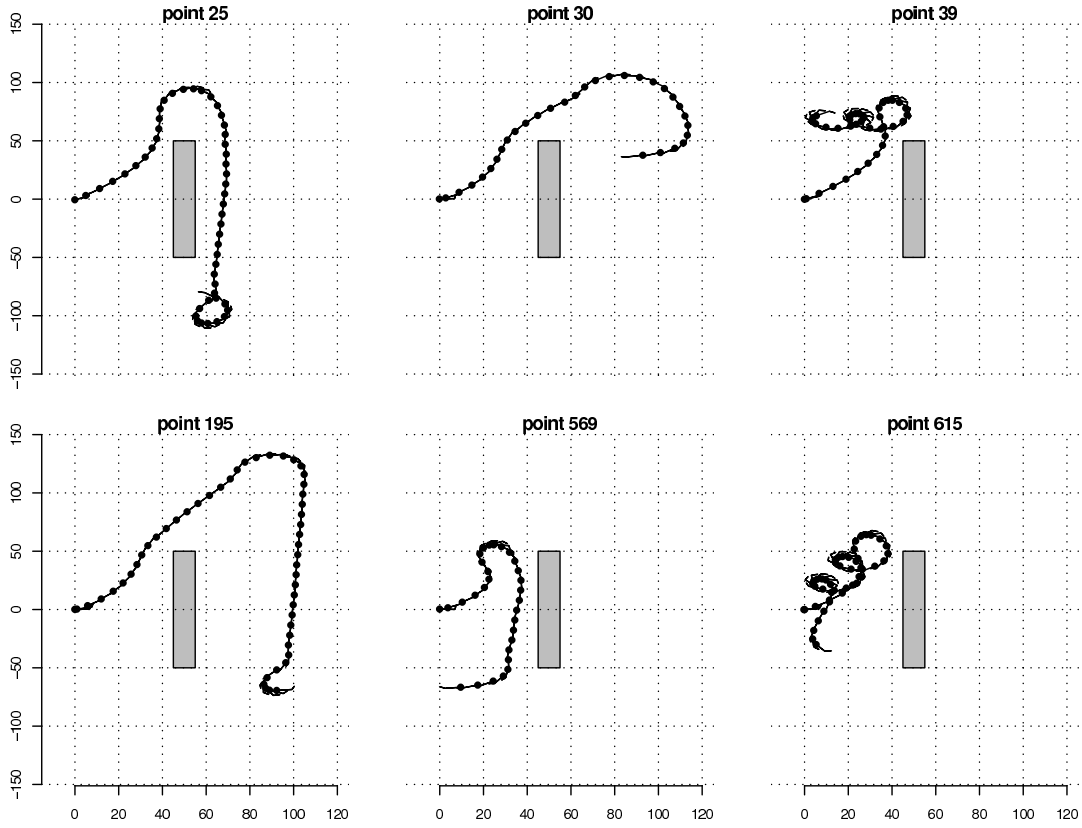


Figure 7.13: Solution trajectories across six test points. The states resulting from the application of the single best SBB solution, Figure 7.10, to six different test points are plotted every 10 time steps.

and down on the plane, while the host denoted by $\triangle$ tends take control when the motion is roughly side to side. Similar information is plotted for point 615, Figure 7.15, where control tends to switch between the level 0 hosts more often likely due to the more intricate trajectory. This figure suggests that the spiraling behaviour is learned at the second level since no single first-level host takes control during the entire execution of a single loop.

Similar to Figures 7.7 through 7.9 presented in Section 7.3.2, additional analysis was performed on the attributes used, but here it was done with respect to the final host population in the run that generated the best solution in Figure 7.10. Figure 7.16, analogous to Figure 7.7, shows the mean number of features accessed by the symbionts in each host in the final population broken down by level. In contrast to the counts observed in solutions across multiple runs, Figure 7.7, when the solutions at the end of this single run are considered the feature counts increase from level 0 to level 1. A two-tailed Mann-Whitney test applied to the level 0 and level 1 counts confirms a difference at a 0.01 significance level. This in turn suggests more sophisticated bidding behaviours at level 1.

Figures 7.17 and 7.18 are analogous respectively to Figures 7.8 and 7.9. Compared to Figure 7.8, the distributions in Figure 7.17 suggest that the solutions rely less on the y-coordinate, while the cab and semi angles appear to remain relatively important. This is again observed in Figure 7.18, where in addition, the different proportions observed under the two non-zero degree actions suggest a level of asymmetry in the control strategy. These counts are consistent with the behaviour observed in Figures 7.11 through 7.15. In particular, once the cab and semi reach the region to the upper-left of the obstacle, the trajectory is always similar and so control need not rely on the y-coordinate. At the same time, the trajectories are highly asymmetric. Figure 7.18 also reveals a number of hosts that employ the zero degree action even though this action was not present in the best solution, Figure 7.10, where these solutions may complement the single-best individual in contributing to the population-wide solved counts.

7.4 Summary of Results

This chapter compared multi-level SBB solutions against NEAT as well as single-level SBB solutions evolved using equivalent computational overhead (in terms of the number of fitness evaluations). Based on the results presented in Section 7.3, the following conclusions can be drawn:

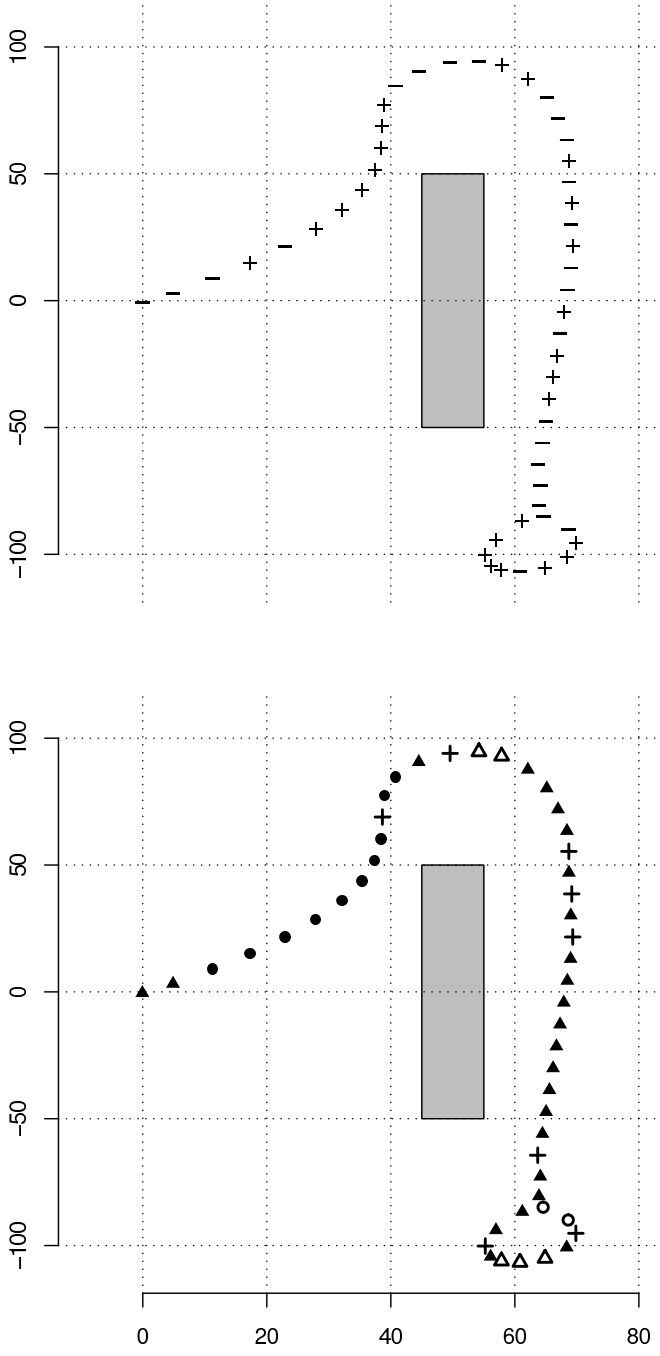


Figure 7.14: Actions used by the best SBB solution in solving test case 25. At each time step plotted in the trajectory for point 25, Figure 7.13, shown is the steering angle (negative or positive, top of figure), and the level 0 host employed (each denoted by a different symbol, bottom of figure).

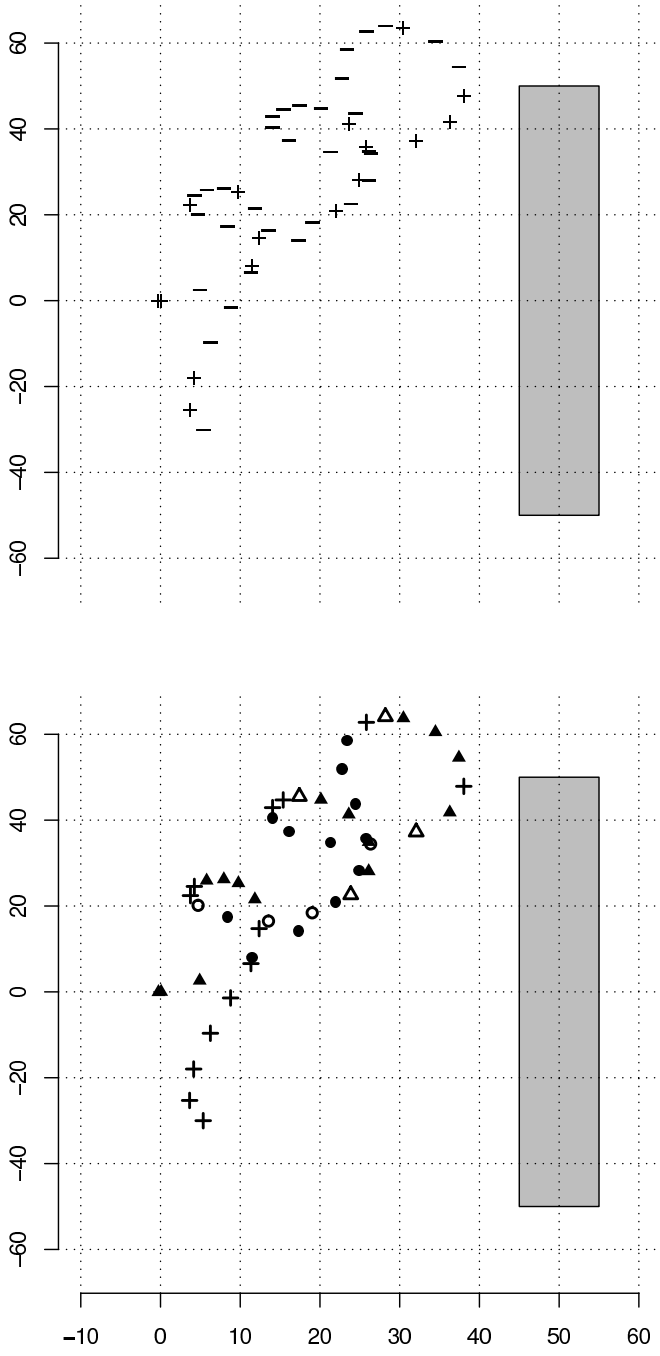


Figure 7.15: Actions used by the best SBB solution in solving test case 615. At each time step plotted in the trajectory for point 615, Figure 7.13, shown is the steering angle (negative or positive, top of figure), and the level 0 host employed (each denoted by a different symbol, bottom of figure).

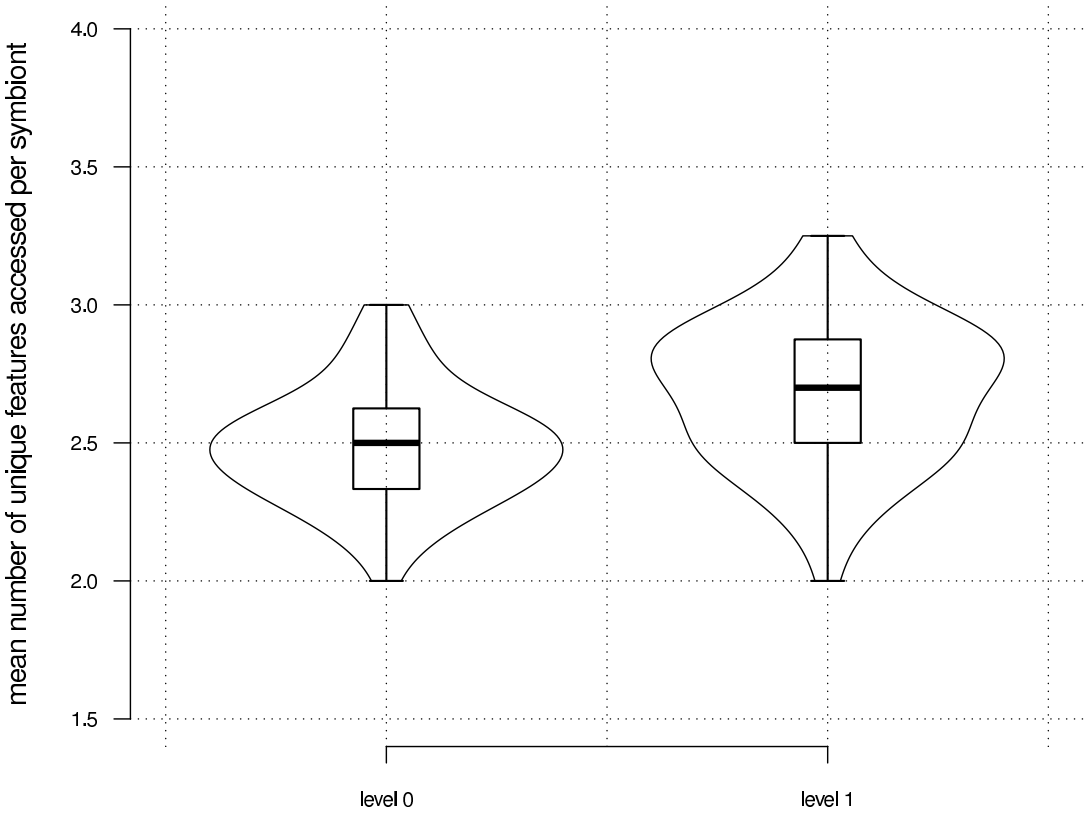
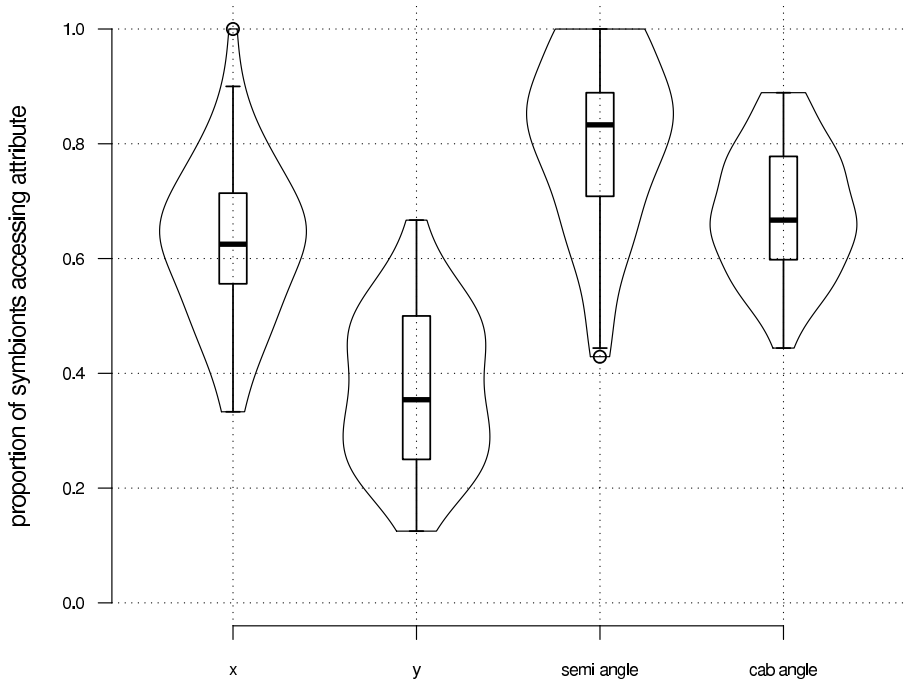
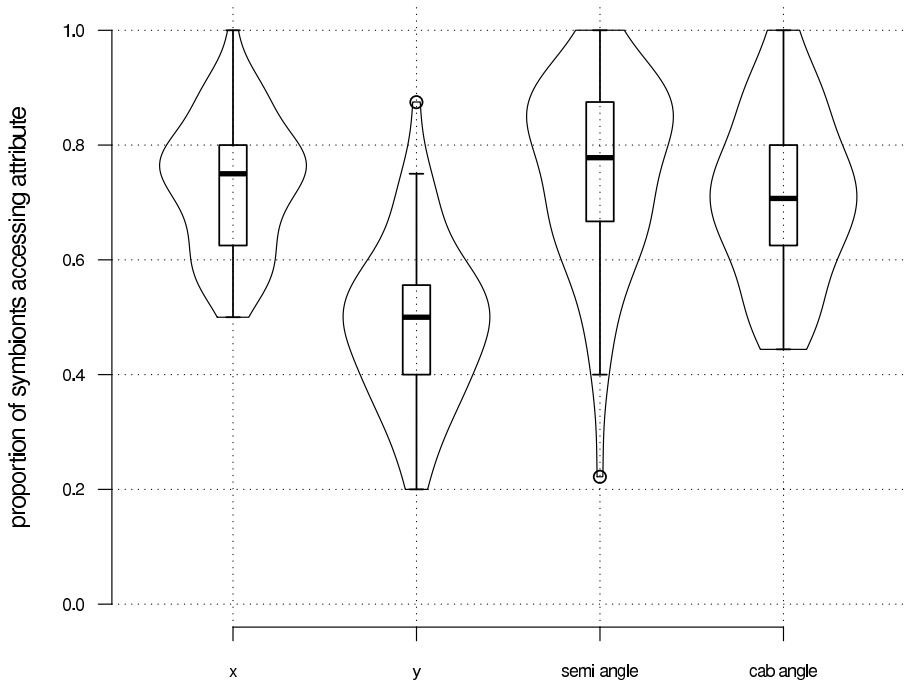


Figure 7.16: Mean number of unique features accessed per symbiont in each host in the final population broken down by level. The distributions represent the 60 individuals in the host population at the end of the run that generated the best solution, Figure 7.10.



(a) Level 0.



(b) Level 1.

Figure 7.17: Proportion of symbionts accessing each attribute. For each attribute, x-axis, the proportion of symbionts in the host accessing that attribute is shown, y-axis, i.e., the calculation is performed across the symbionts in each host separately. The distributions, broken down by level, represent the hosts in the final population at each level in the run that generated the best solution, Figure 7.10.

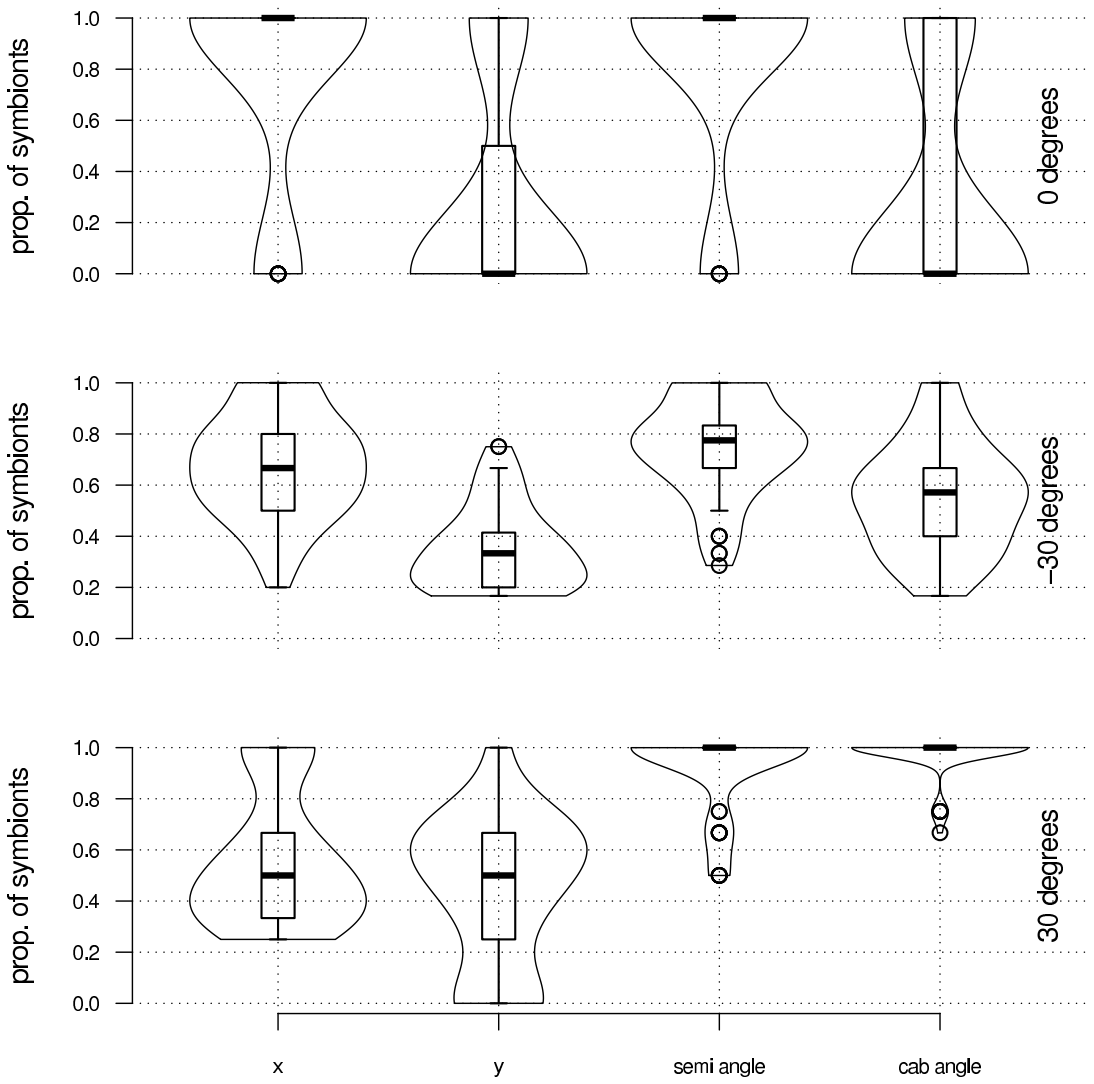


Figure 7.18: Proportion of symbionts in the first-level hosts accessing each attribute broken down by action, distributions are otherwise as in Figure 7.17. Specifically, the mean is calculated as in Figure 7.17, but when considering a given action the symbionts in the host associated with the two other actions are not counted. Only 28 points are plotted under the 0 degree action since that action was not represented in 32 of the 60 hosts (the same count is observed in Figure 7.9 by coincidence), and only the first level is considered since there are 60 actions at the second level.

1. As illustrated in the case of the SBB framework, hierarchical model building can lead to significantly better performance compared to the construction of single-level solutions. Naturally, the hierarchical models tend to be more complex.

2. The SBB algorithm outperformed NEAT on the truck reversal environment both in terms of the single best individual and the population as a whole. This establishes the relatively high effectiveness of the SBB algorithm in this temporal sequence learning task, though a more complete evaluation would involve additional problems.

3. Under the SBB algorithm, higher-level hosts were observed to combine multiple lower-level hosts in order to evolve behaviour that was both more complex and more effective. This provides additional support in favour of hierarchical model construction given challenging tasks.

4. The relatively poor performance of NEAT established the high difficulty of the truck reversal problem formulation used in this thesis, Section 6.2.

Furthermore, it is suggested that in contrast to classification, under temporal sequence learning increased solution complexity may be warranted – this is because in general temporal sequence learning represents a more difficult class of problems. This is especially true if the best of current practice, in this case NEAT, is not able to produce satisfactory results. Moreover, the multi-level SBB models, even if they are more complex, are not 'black box' solutions since they can be relatively easily analyzed in terms of their constituent parts and behaviours, e.g., as is done in Figures 7.10 and 7.14. Here 'relative' is primarily with respect to traditional ensemble approaches where individuals with highly overlapping behaviours are combined under a voting mechanism, making it difficult to interpret the contributions of individual solution subcomponents. Finally, it is emphasized that the increased complexity, in this case through layered learning, is made possible through the specific properties of the SBB architecture, i.e., the ability to learn context for specific actions and then perform a combinatorial search for a mutualistic mix of behaviours. Thus, complexity in this case is not a side-effect of evolution as in, e.g, the case of code bloat, but is instead a deliberate mechanism built into the framework, which, as demonstrated in this chapter, makes the approach effective on challenging temporal sequence learning problems.

Chapter 8

The Rubik's Cube

The Rubik's Cube presents a challenging problem for ML algorithms and is therefore the subject for this last investigative chapter. Though hand-crafted approaches for solving the cube exist, learning to solve arbitrary instances with minimal *a priori* knowledge is hard for the following reasons:

1. The state space, consisting of $4.33 \cdot 10^{19}$ possible configurations, is vast.

2. Using existing representations, a small number of cube twists typically results in large perceived changes.

3. An efficient distance function between two cube states is not obvious.

Together, these characteristics of the problem make the specification of an appropriate cost function, and consequently the learning process, challenging. Though symmetries with respect to the colours exist and could be taken advantage of, finding a way to exploit them poses a challenge in itself.

In this chapter, the SBB algorithm is applied to the 3x3, six-coloured, Rubik's Cube. In contrast to the previous chapter, where a random point sampling heuristic was used, the algorithm formulation used here reverts back to the original version that takes advantage of the structure in the state space, i.e., the formulation used in Chapter 6. In contrast to most previous computational methods, where solutions take the form of a sequence of actions and require relearning on each each new problem instance, the goal of this chapter is to evolve models capable of solving arbitrary Rubik's Cube configurations.

This chapter is organized as follows. Section 8.1 summarizes previous computational approaches to solving the Rubik's Cube. In Section 8.2, the environment formulation that is used is described along with the associated domain-specific functions. Section 8.3 presents the experimental methodology and also attempts to characterize the problem with respect to a random solver. Results are presented in Section 8.4, and the findings are summarized in Section 8.5.

8.1 Previous Approaches to Solving the Rubik's Cube

Numerous hand-crafted approaches to solving the Rubik's Cube, varying in difficulty and efficiency, have been developed. The most famous of these is likely the layer-by-layer approach [164]. These approaches apply precise combinations of moves under specific conditions and tend to result in long overall move sequences. In contrast, the focus of the computational approaches described below has been to reduce the maximum number of twists that is required[1] given an arbitrary starting configuration.

An upper bound of 52 moves required for solving any configuration of the Rubik's Cube was obtained in the early 1980s by Morwen Thistlethwaite. Thistlethwaite's strategy involved organizing the space of cube configurations hierarchically into five nested sets, where each set was defined by restricting the moves that could be used in generating its members (assuming the solved state as a starting point). The largest of these sets contained all $4.33 \cdot 10^{19}$ cube configurations, whereas the smallest was the singleton containing the solved state. To solve a given instance of the Rubik's Cube, the smallest of the sets containing that instance was considered. A lookup table was then used to obtain a sequence of moves that, when applied to that instance, resulted in a configuration that was also in the next-smallest set – in this way, the cube configuration was transformed towards the solved state[2]. The key behind Thistlethwaite's strategy was to organize the state space into nested sets, restricting the moves allowed in each of the four stages, and limiting the size of the lookup tables to enable the efficient identification of the required move sequences.

The way in which Thistlethwaite's algorithm structured solutions meant that, in terms of the number of twists, the resulting move sequences were likely to be sub-optimal. The first approach to compute optimal solutions to arbitrary configurations of the Rubik's Cube used an iterative-deepening-A* search [93], though only a limited number of initial states were considered. This form of depth-first search requires a heuristic for pruning excessively long branches. In this case, the heuristic was based on precomputed tables known as pattern databases which stored lower bounds on the number of moves required to solve particular sub-problems. Of the ten random cubes that were solved, the median number of required moves was eighteen. More recently, it was shown that no more than twenty-six twists[3] are required to solve any instance

[1]The sequence length is normally expressed in terms of the number of face turns, regardless of whether the rotation is a quarter- or half-turn.

[2]The longest move sequences at each of the four possible stages involved 7, 13, 15 and 17 twists resulting in the upper bound of 52 moves.

[3]Lower bounds have since been obtained but, as of this writing, the work has not yet been

of the Rubik's Cube problem [99]. This approach used insights from group theory and relied on parallel computation, high-performance hardware, and lookup tables to calculate the number of moves required to solve arbitrary problem instances.

In summary, what could be called traditional approaches to solving the Rubik's Cube tend to formulate the solution as a search for the sequence of moves that transform an initially scrambled cube to the solved state. The search is repeated from the start for each new problem instance, i.e., the solution is not a general strategy. To deal with the massive search space, they tend to use ideas from Thistlethwaite's original algorithm, in particular, they use domain knowledge, e.g., group theory, to organize the space of cube configurations and rely on lookup tables that contain precomputed information. Despite efforts to tame the search space, the resulting algorithms still require large amounts of computational resources, in terms of processing capacity, memory, or bandwidth, or in some cases all three [99].

There are a handful of documented attempts at applying techniques from EC to the Rubik's Cube problem. Early work [66] relied on turn patterns that, unlike half- and quarter-turns, resulted in gradual changes to the state of the cube. Since a turn pattern consists of multiple twists, the resulting move sequences were very long. More recently, a four-stage algorithm based on Thistlethwaite's approach was proposed [40]. The algorithm effectively replaced the lookup table used to transform the cube at each stage by the execution of an ES, concatenating the four subsequences of moves obtained. In both cases, as in the previously-described non-evolutionary methods, the approach is formulated as a search for a specific sequence of moves as opposed to a policy.

The SBB algorithm used in this work differs from these previous computational approaches in two significant ways. First, whereas all of the previous approaches relied in part on expert domain knowledge, the proposed approach remains generic. Second, the solution output by previous approaches was always a sequence of moves, i.e., it was only applicable to a single starting cube configuration, requiring a separate application of the algorithm to solve a different configuration. In this work, the solution output by the SBB algorithm is a model, and as such, it is likely to incorporate knowledge that is discovered automatically. Given that the goal is to evolve models that can solve multiple cube configurations, the compromise made is that move sequences of sub-optimal length are likely to be required if general strategies can be discovered.

The single known exception where models were evolved on the Rubik's Cube

---

thoroughly documented.

problem was the economy-based Hayek framework [12]. Though minimal domain knowledge was used, several cube models, point presentation schemes, and fitness functions were investigated. This led to little success, and a underlying result was initial progress followed by an indefinite period of stagnation. This was attributed to the 'step back to move forward' nature of the Rubik's Cube problem, namely, that as the cube configuration approaches the solved state it becomes increasingly difficult to advance further without (at least temporarily) undoing previous progress.

8.2 The Rubik's Cube Environment

In the following sections, the Rubik's Cube environment assumed in this work is detailed. An overview of the environment, including the structure of the state space, is first presented in Section 8.2.1. Sections 8.2.2 and 8.2.3 respectively describe point creation processes as well as the function used to measure the distance between two cube configurations; the latter forms the basis for the reward function described in Section 8.2.4. It is noted that in the version of the SBB algorithm used in this chapter the genotypic point diversity reward is applied in calculating point fitness, Section 3.2.6. For completeness, the implementation-specific state representation that was used is described in Section 8.2.5.

8.2.1 Overview

The 3x3 Rubik's Cube, Figure 8.1, has six *faces* each organized into a grid of nine *facelets*. Each facelet is assigned one of six colours, and in the solved state, all the facelets on a given face share the same colour. The cube can be broken down into twenty-six three-dimensional *cubies* each of which exposes one, two, or three facelets. There are six center cubies, one on each face, which expose a single facelet. The center cubies remain fixed and therefore define the colour associated with the face they appear on. There are twelve side cubies and eight corner cubies each exposing two and three facelets respectively. The set of colours on the facelets of a given cubie is unique and as such determines the location and orientation of the cubie in the solved state.

The Rubik's Cube is manipulated by rotating its faces. In the environment formulation used here, twelve moves were defined that could be used to manipulate the state of the Rubik's Cube. The first six moves, denoted as F, B, U, D, L, and R, turn the front, back, up, down, left, and right face respectively by ninety degrees in the clockwise direction (when viewing the face head-on). The remaining six moves,

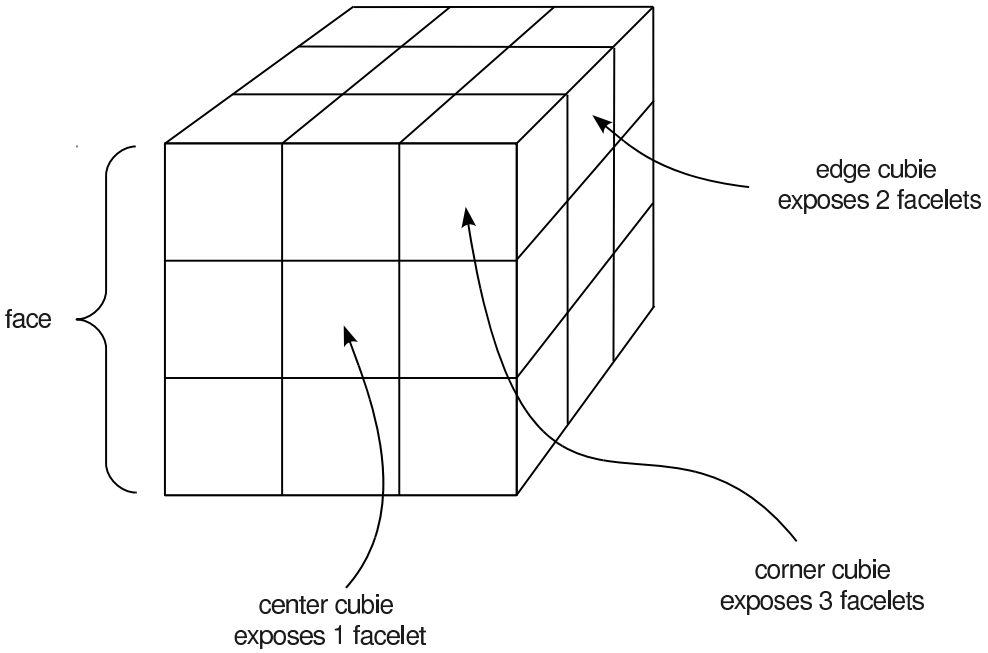


Figure 8.1: The 3x3 Rubik's Cube consists of six faces each associated with one of six colours. Each face in turn consists of nine facelets. The colour of a given face is defined by the single facelet on the center cubie.

Number of twists	Number of states
0	1
1	12
2	114
3	1068
4	10011
5	93840
6	878880

Table 8.1: Number of unique states reachable under the Rubik's Cube environment using quarter-turn move sequences of different lengths. States counted under shorter sequences are not included in the counts corresponding to longer sequences, i.e., the counts exclude sequences that include moves that cancel each other out.

denoted as F′, B′, U′, D′, L′, and R′, are inverse operations that turn their respective faces by ninety degrees in the counter-clockwise direction. Each of the moves affects four side and four corner cubies. Using these twelve moves, roughly $4.33 \cdot 10^{19}$ unique cube states can be reached from the solved state.

The choice was made not to include the half-turns, which would increase the number of moves to eighteen, but potentially reduce the length of the move sequences that would be required. As a result, under this scheme, the upper limit of twenty-six moves required to solve the cube [99] likely does not hold. By eliminating half-turns from the set of legal moves, the size of the action set was reduced without limiting the capability of the models that could be evolved. In effect, the space of possible models has been reduced due to limited computational resources.

The number of unique states reachable under the Rubik's Cube environment using the twelve quarter-turn moves described above is shown in Table 8.1. For example, starting at any state (including the solved state), 1068 different configurations can be reached using exactly three twists. The counts exclude move sequences that contain twists that cancel each other out. Even after restricting the move set from eighteen moves to twelve moves, the number of reachable states grows very quickly. The counts in Table 8.1 were obtained using a brute-force calculation, and as such, calculating the number of states reachable using seven or more twists was found to be infeasible given the available computation resources.

In the experiments presented in this chapter, a maximum of twenty time steps, i.e., moves, was allowed in a single simulation. If the cube was solved before this twist limit was reached, the episode was terminated. At the end of a simulation, the

cube was solved, the twist limit was reached, or both of these conditions were true. At that point, the reward corresponding to the final state, Eq. 8.2, was returned.

8.2.2 Point Initialization and Generation

In contrast to Chapter 7, where the SBB algorithm was modified so that the point population was updated using a random point sampling heuristic, the SBB formulation used in this chapter reverts back to the original which employs point operators that take advantage of the structure in the state space; here, point initialization and generation is described. The genotypic distance measure and the reward function, both forming the basis for the full-fledged point fitness function as described in Section 3.2.6, are described in subsequent sections.

All points were created using some sequence of legal moves, Section 8.2.1, applied to the solved state. Illegal configurations, such as those that would be reachable with the disassembly of the cube, were therefore not possible.

During point initialization and point generation, Sections 3.2.1 and 3.2.3, random points were created as follows:

1. A twist count cnt_{init} between one and $initTwists$ inclusive was selected with uniform probability.

2. Starting with the solved state, a total of cnt_{init} moves, each selected with uniform probability, was applied to the cube.

3. If, after the previous step, the cube was in the solved state, the previous step was repeated.

Using this point creation procedure, it was possible that some twists canceled each other out. A point created using cnt_{init} twists could therefore be solvable in fewer than cnt_{init} moves.

To generate an offspring point from a parent point, Section 3.2.3, a sequence of moves was applied starting from the parent's configuration as follows:

1. A twist count cnt_{gen} was selected as one plus the absolute value of a normally distributed deviate with a mean of zero and a standard deviation of $genTwists$.

2. Starting at the parent's state, a total of cnt_{gen} moves, each selected with uniform probability, was applied to the cube.

3. If, after the previous step, the cube was in the solved state, the previous step was applied to the solved state.

In the experiments, *initTwists* was set at 10 while *genTwists* was set at 3. Points that were created randomly could therefore always be solved under the upper limit of twenty moves, and in particular, this was true of the points used to evaluate test performance. Points created from other points, however, could potentially be unsolvable given the maximum number of steps that was allowed.

8.2.3 Genotypic Distance

Defining a meaningful distance between two configurations of the Rubik's Cube is not a straightforward task. For example, using a count of the facelets on each face that match the center facelet may be misleading because, under this definition, a cube configuration may be assigned a relatively large distance from the solved state even if it is close to being solved. This is due in part to the fact that a single twist may result in a disproportionate perturbation in the mix and positioning of the facelets on the affected faces (and why previous approaches resorted to the use of turn patterns to mitigate this effect [66]).

In this work, the raw distance between states $cube_i$ and $cube_j$ used in calculating the genotypic diversity reward, Eq. 3.5 in Section 3.2.6, as well as the reward function described in the next section, was calculated as

$$d_{rubik}(cube_i, cube_j) = \begin{cases} 0 & \text{if } cube_i \text{ is 0 twists away from } cube_j, \\ 1 & \text{if } cube_i \text{ is 1 twist away from } cube_j, \\ 4 & \text{if } cube_i \text{ is 2 twists away from } cube_j, \\ 16 & \text{otherwise,} \end{cases} \tag{8.1}$$

where the twist counts refer to the minimum number required. Under this definition, pairs of points that are three or more twists away are viewed as equidistant. This distance function could be made more informative, e.g., by checking points that are three, four, five or more twists away, though this would result in significantly higher computational overhead as more points would need to be considered, Table 8.1.

8.2.4 Reward Function

The reward scheme, used during evaluation to define the outcome of applying a host to a point, Section 3.2.5, was based on the genotypic distance between points, Eq.

8.1. Specifically, the reward associated with a final cube configuration $cube_f$ was a function of the genotypic distance between $cube_f$ and the configuration corresponding to the solved state, $cube_s$, defined as

$$\frac{1}{d_{rubik}(cube_f, cube_s)^2 + 1} \tag{8.2}$$

where d_{rubik} is the genotypic distance function, Eq. 8.1. Thus, final configurations farther away from the solved state were associated with a lower reward. Since the genotypic distance function is sensitive to within two twists, i.e., pairs of states three or more twists apart are viewed as equidistant, the reward function can distinguish between solutions that transform a given starting configuration to any of the 127 configurations closest to the solved state and all other solutions. A maximum reward of one is awarded for solving the cube, thereafter the reward decreases exponentially.

Despite being informative on only a minuscule region of the state space, this reward function was found to be more useful than some alternatives, e.g., a measure of entropy reflecting the number of facelets on each face matching the center cubie and a function that explicitly considers the location of facelet combinations [66]. This is due in part to the fact that, as noted earlier, a single twist of the Rubik's Cube can result in a disproportionate change in the configuration of the cubies. Thus, the degree to which a cube is scrambled often provides little information about how close the cube is to the solved state[4]. Equivalently, there are many states where the cube is scrambled to a similar extent but whose distance to the solved state varies significantly. In contrast, the reward function used here provides an accurate indication of how close the cube is to being solved when the configuration is in the immediate vicinity of the solved state, but no information in all other situations.

The more general problem faced in defining an appropriate cost function under the Rubik's Cube problem is that it is not obvious how to efficiently provide an accurate monotonic reward scheme, i.e., one which accurately reflects the distance to the solved state, with respect to even one percent of all the possible states. The cost function in Eq. 8.2, based on the genotypic distance defined in Eq. 8.1, provides a gradient only with respect to the 127 of the $4.33 \cdot 10^{19}$ states closest to the solved configuration, leaving approximately $4.33 \cdot 10^{19}$ states where the cost function provides no information. Alternatively, functions that compare cubie locations and orientations tend not to be monotonic. Finally, it is noted that even though the state space is

[4]Assuming, for example, the count of facelets that match the center as the measure quantifying how scrambled the cube is.

massive, the point population in the SBB algorithm will tend to support points that provide a learning gradient, so it is expected that given the reward function specified above the focus will be on starting states in the vicinity of the solved configuration.

8.2.5 State Representation

The cube state was represented using a 54-dimensional vector specifying the colour at each location on the cube's surface, Figure 8.2. Integers 0 through 5 were used to represent each of the colours, and there were always nine instances of each colour in a state vector. Since the six center facelets on each face remain fixed, the corresponding vector components were constant; these were included in the state vector regardless as they identified the colour associated with each face. This representation is an initial attempt and therefore it is acknowledged that it may not be optimal, and in particular, alternate representations that consider symmetries in the cube may lead to improved performance.

8.3 Experimental Setup

This section details the experimental setup including the SBB parameters used, Section 8.3.1, as well as the overall evaluation framework, Section 8.3.2.

8.3.1 Parameters

With two exceptions, the SBB parameters were set as in Chapter 7, Section 7.2.3; both changes were made in order to encourage a greater degree of problem decomposition. The first change was to increase the maximum host size, ω, from 10 to 24. The second change to was to decrease the maximum program size, $maxProgSize$, from 48 to 24. Together, these changes enabled larger solutions in terms of symbiont counts, although the bid programs themselves were more restricted in terms of the instruction counts. By limiting the capability of the symbionts, the goal was to have each associate itself with a smaller portion of the state/feature space. As in previous experiments, 60 initializations were performed using each algorithm configuration (described in the following section).

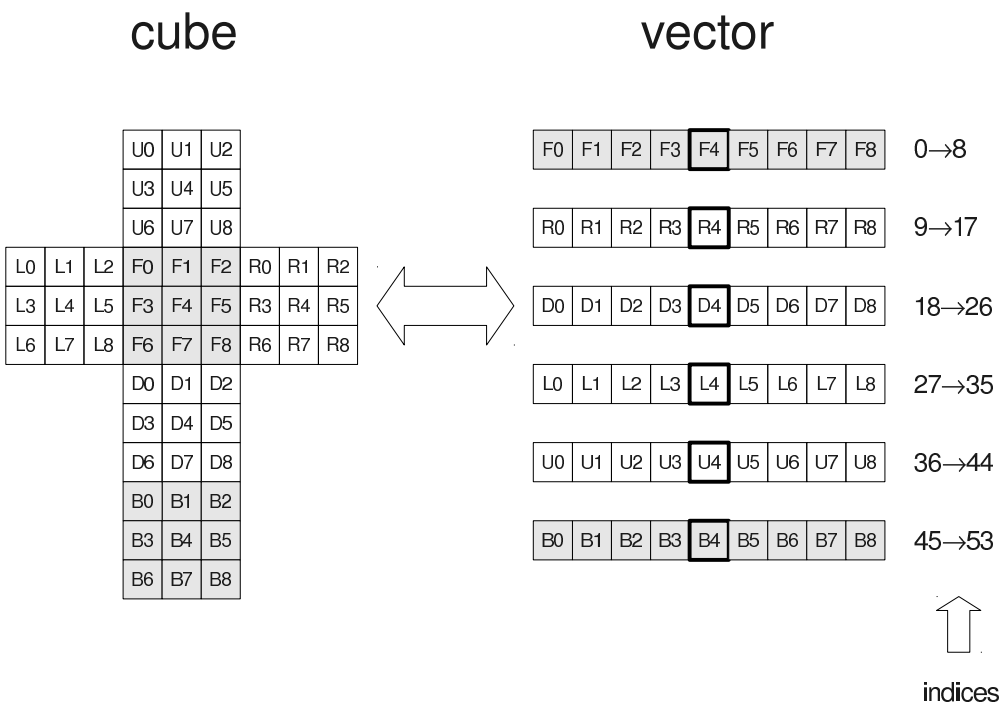


Figure 8.2: The 3x3 Rubik's Cube state representation. Shown is how the cube configuration is mapped to a 54-dimensional vector, with the left-hand side showing a cube that has been 'unfolded'. Front, right, down, left, up, and back facelets are denoted by F, R, D, L, U, and B respectively. The vector components with the bold border remain fixed during state transitions and define the colour of their associated face. For example, the first nine indices of the vector represent colours on the front of the cube.

	SBB	SBB-bh	SBB-bp
levels	2	1	1
t_{max}	1000	2000	2000
ω	24	36	24
$maxProgSize$	24	24	36

Table 8.2: Three SBB configurations compared under the Rubik's Cube problem. The two-level configuration is compared against a big host and big program configuration, with the differences in parameter settings highlighted. All other parameters were set as described in Section 8.3.1.

8.3.2 Evaluation Methodology

A two-level SBB configuration was compared with two single-level configurations, SBB-bh and SBB-bp, Table 8.2. In all three cases, the total number of fitness evaluations was kept the same. Under the SBB-bh and SBB-bp configurations, the number of training generations, t_{max}, was set at 2000 since they only involved a single level. The SBB-bh, or 'big host', configuration, was included in the comparison to determine the benefit of having two levels versus evolving just a single level but increasing the maximum host size. Thus, under this configuration, the maximum host size, ω, was increased to 36 from 24. The SBB-bp, or 'big program', configuration, was included to allow comparison against a single level with an increased maximum program size, $maxProgSize$. In this case, the maximum program size was increased to 36 from 24. In all three cases, parameters were set as indicated so that the upper bound on the computational overhead in terms of overall instructions executed was the same[5]. In addition, the performance of the first level of the two-level SBB configuration was also considered.

Solution quality was measured as in the experiments on the truck reversal domain, Section 6.3.2. In particular, the number of test cases solved was considered with respect to the best individual and the population as a whole. In contrast to the truck reversal domain where thresholds on the real-valued state components were set, under the Rubik's Cube there is only a single state corresponding to a solved state. As before, the best individual was defined as the individual with the highest mean

[5]For example, assuming host and point populations of size one, the maximum number of instructions that can be executed under the SBB configuration is $1000 \cdot 24 \cdot 24 + 1000 \cdot 2 \cdot 24 \cdot 24$, with the two main terms corresponding to the overhead at the first and second level respectively. Under the SBB-bh formulation, the maximum execution count is $2000 \cdot 36 \cdot 24$. Naturally, this analysis assumes one step per episode.

reward, Eq. 8.2, on the validation set, where this may not necessarily correspond to the individual solving the most cube configurations[6], i.e., individuals that nearly solve many cubes may also receive a high reward on average.

The validation set consisted of 1000 random points where each point was created as in the point initialization procedure, Section 8.2.2. As points were added to the validation set, if a new point was found to be a duplicate it was discarded and the creation process for that point was restarted, i.e., including the re-selection of the number of twists cnt_{init}. Two sets generated independently from the validation set were then used to evaluate the test performance of the solutions. The first test set consisted of all points 1-, 2-, and 3- twists away from the solved state and was used to determine the performance on what would be assumed to be easy instances. In this set, the number of 1-, 2-, and 3-twist points was 12, 114, and 1068 respectively, Table 8.1. Here, a point will be referred to as a k-twist point if the lower bound on the number of moves required to solve it is k. The second set consisted of 5000 unique random points generated in the same way as the validation set. As under the truck reversal problem, the same validation and test set was used across all initializations.

The distribution of points in the random test set, with respect to the number of twists cnt_{init} used to create each point, is shown in Figure 8.3. The number of 1-twist and 2-twist points in the random test set is low because there are, respectively, only 12 and 114 of these points in total. Even the 3-twist counts appear low compared to the higher twist values. This is because, given the way in which the test set is created, it is relatively likely that if a cnt_{init} of three is selected, then the resulting configuration will be a duplicate of an existing point (and in this case a new twist count cnt_{init} is reselected). Inclusion of the test set containing all 1-twist, 2-twist, and 3-twist points compensates for the relatively low count of these point in the random test set.

8.4 Results

In order to provide a performance baseline, a stochastic guesser was applied to the random test set, with the associated results presented in Section 8.4.1. The performance of the SBB algorithm on the 1-, 2-, and 3-twist points is then presented in Section 8.4.2, while SBB performance on the random test set is presented in Section 8.4.3. Complexity results are discussed in Section 8.4.4, with an additional analysis

[6]Though the non-linear distance, Eq. 8.1, and the squaring of the distance in the reward function, Eq. 8.2, is meant to mitigate this.

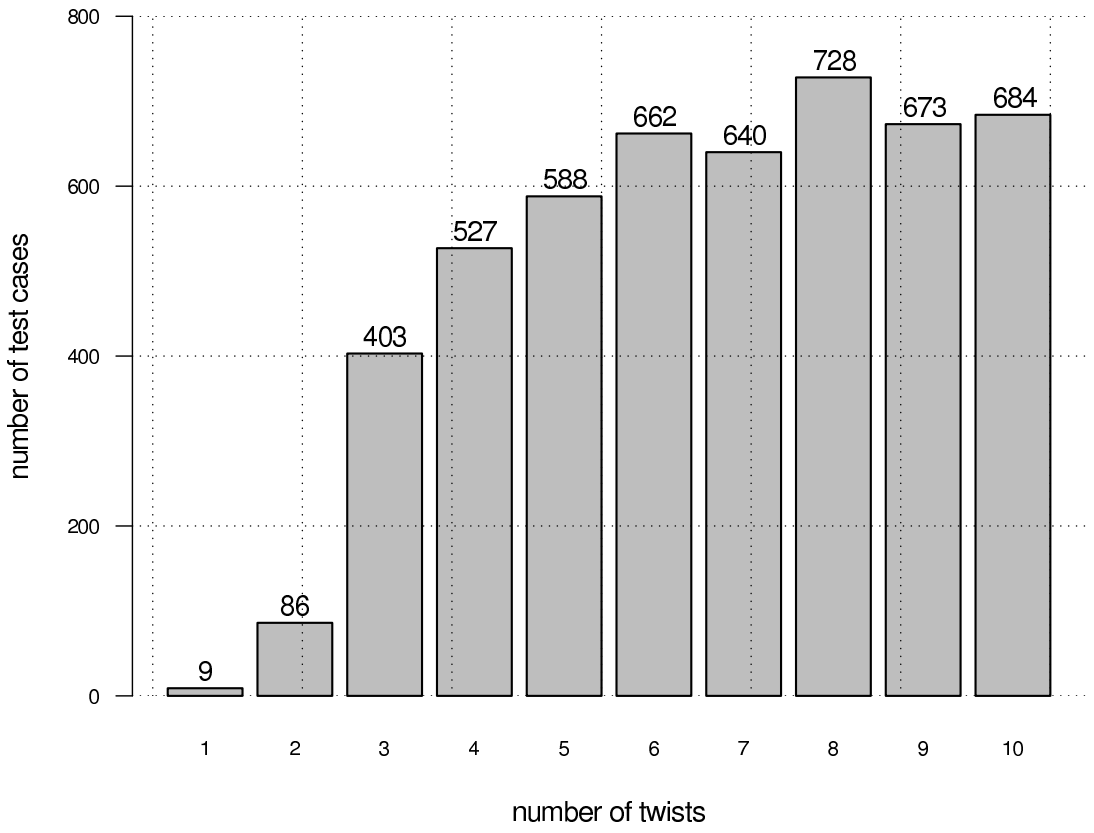


Figure 8.3: Distribution of points in the random test set with respect to the twist counts. The 5000 test points are binned by the number of twists, cnt_{init}, used to create each point. For each possible number of twists, x-axis, shown is the number of associated points, y-axis.

of performance under limited resources summarized in Section 8.4.5.

8.4.1 Random Solver

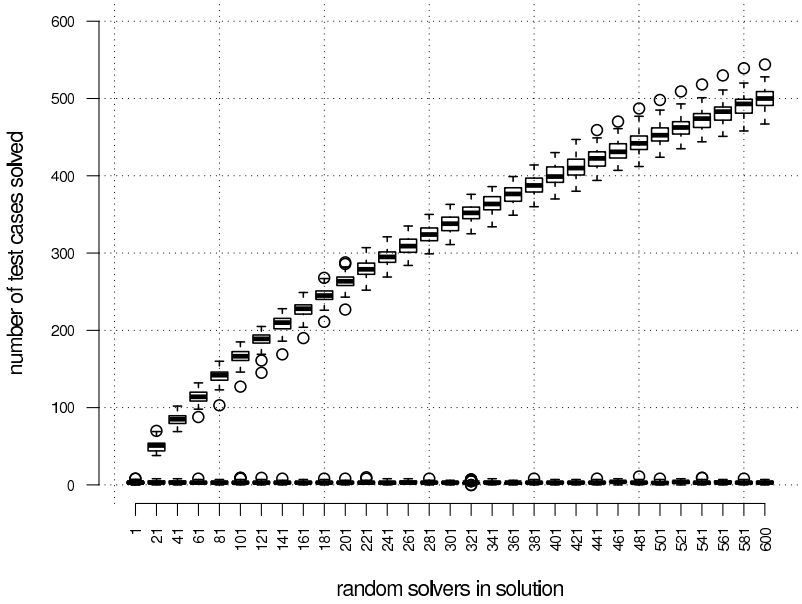
In order to characterize the problem, the performance of a set of random solvers on the random test set was considered, Figure 8.4. Here, a random solver is one that, at each time step, selects a move with uniform probability. Since each move is stochastically selected, the same random solver is likely to result in two different action sequences when applied to the same initial cube configurations, i.e., the action sequence given an initial state is not deterministic.

Performance was measured in terms of an absolute count and a cumulative count, much like in Section 4.2.3. The cumulative count is defined relative to some ordering of the solvers and refers to the number of different test cases solved by a solver and all solvers before it in the sorted list. It effectively provides a snapshot of the population-wide counts as additional solutions are included. The absolute count refers to the number of different test cases solved by a particular solver. A set of 600 random solvers was applied to the random test set[7]. Since each random solver follows the same move selection strategy, any difference in performance observed across the set was due to chance. This was repeated 60 times, the same as the number of SBB initializations used, leading to the distributions in Figure 8.4.

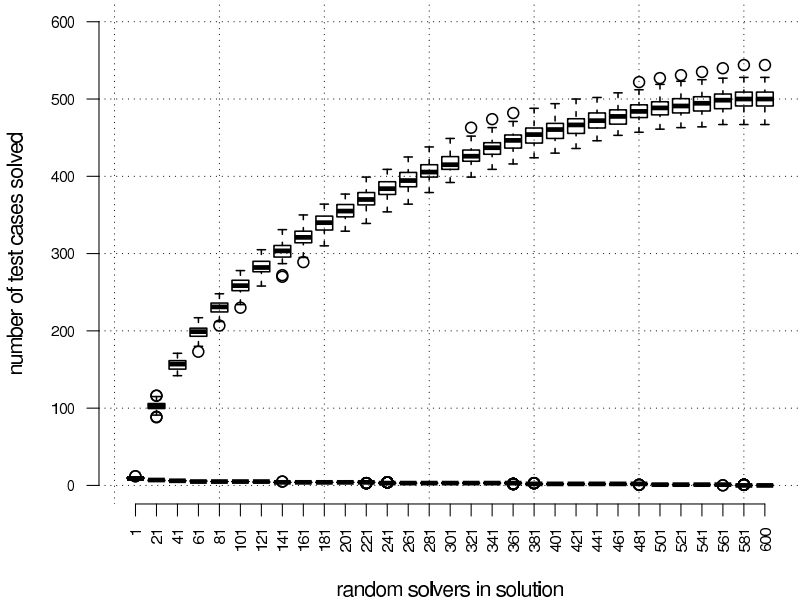
In Figure 8.4a, the solvers are considered in arbitrary order, i.e., a particular solver's index along the x-axis is arbitrary. Here, there is a steady increase in cumulative counts, though some diminishing returns are observed as more and more solvers are considered. In particular, the median cumulative count at $x = 300$ is 337, while at $x = 600$ it is 500. Regardless, this suggests a relatively low degree of overlap in the test cases solved by different solvers since the cumulative counts continue to increase despite the low absolute counts – if there was significant overlap, the curve would quickly plateau. In Figure 8.4b, the same sets of solvers are considered but a decreasing ordering with respect to the absolute counts is assumed. That is, solvers with higher absolute counts are associated with lower indices on the x-axis. There is a sharper increase in the cumulative counts at lower indices again suggesting that different solvers solve different test cases. In addition, the absolute counts highlight that a single solver solves only a handful of cases; the median absolute counts at $x = 1$, $x = 100$, and $x = 300$ are 9, 5, and 3 respectively.

Assuming a population size of 600, the random solvers were able to collectively

[7]Equivalently, a single random solver was applied 600 times to the random test set.



(a) Unsorted.



(b) Sorted.

Figure 8.4: Random solver performance on the Rubik's Cube problem. Shown is the count of solved cases with respect to the random test set, y-axis, as additional random solvers are considered, x-axis. The top curve in each subplot denotes cumulative performance, while the bottom curve denotes absolute performance. In Figure 8.4a, the solvers are considered in arbitrary order, while in Figure 8.4b, the solvers are sorted into descending order according to each solver's absolute count of test cases solved. The set of solvers in both subplots is identical.

	1-twist	2-twist	3-twist
SBB 1 vs SBB 0	0.04458	0.0008872⋆	0.0001368⋆
SBB 1 vs SBB-bh	0.2158	0.003993⋆	0.003636⋆
SBB 1 vs SBB-bp	0.4199	0.3693	0.7647⋆

Table 8.3: Two-tailed Mann-Whitney test results comparing second-level SBB solution counts with SBB 0, SBB-bh, and SBB-bp on the 1-, 2-, and 3-twist cases. Shown are the p-values when corresponding distributions in Figure 8.5 are compared, e.g., SBB 1 versus SBB-bh on all 1-twist points. Cases where the SBB 1 median value is *higher* (better) are noted with a ⋆.

solve almost ten percent of the random test cases. To do so, they relied on non-overlapping behaviour, indicating that population diversity is likely to be important if the SBB algorithm is to be successful on this problem. In particular, the median cumulative count at 60 solvers in Figure 8.4a is 113; this represents a population of random solvers that is the same size as the final SBB population[8]. If the models evolved under the SBB algorithm surpass this baseline level of performance, this would imply that learning is taking place.

8.4.2 Performance on 1-, 2-, and 3-Twist Points

The proportion of 1-, 2-, and 3-twist points that are solved by the best host decreases sharply as the lower bound on the required number of twists increases, Figure 8.5. This is indicative of the relative difficulty of the Rubik's Cube problem. The SBB 1 solutions appear to outperform the other three on the 1-twist points with 49 of the 60 initializations producing a best individual that solves all 12 points. On the 2-twist and 3-twist points, SBB 1 appears to outperform SBB 0 and SBB-bh, but its performance is similar to that of the big program configuration SBB-bp. A two-tailed Mann-Whitney test comparing SBB 1 with the other three configurations suggests no difference on the 1-twist points at a significance level of 0.01, Table 8.3. This may be due in part to the fact that there are only twelve 1-twist points. On the 2-twist and 3-twist points, the tests support the observations made regarding the relative performance of the four configurations assuming a significance level of 0.01, and in particular, the similar performance of SBB 1 and SBB-bp.

The number of moves used to solve the 1-, 2-, and 3- twist points, Figure 8.6,

[8]The cumulative count at 60 solvers in Figure 8.4b is not considered as this would imply selecting 60 individuals with the highest absolute solved counts out of a population of 600.

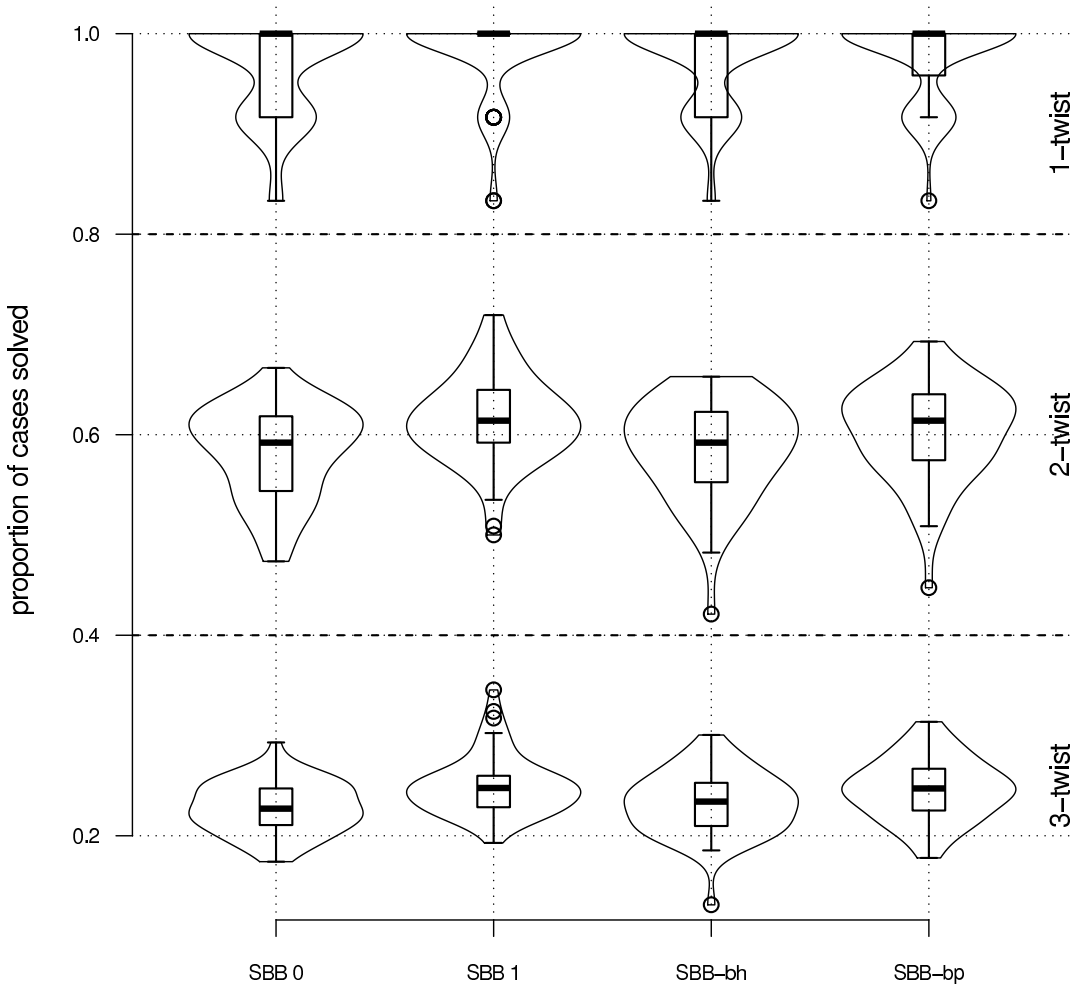


Figure 8.5: Proportion of 1-, 2- and 3-twist cases solved. For each configuration, x-axis, shown is the number of cases solved by the best host in the final population, y-axis. The first and second SBB levels in the two-level configuration are denoted by SBB 0 and SBB 1 respectively, i.e., SBB 1 is constructed on top of SBB 0. The cases are grouped by the minimum number of twists required to solve them, and each distribution represents 60 initializations.

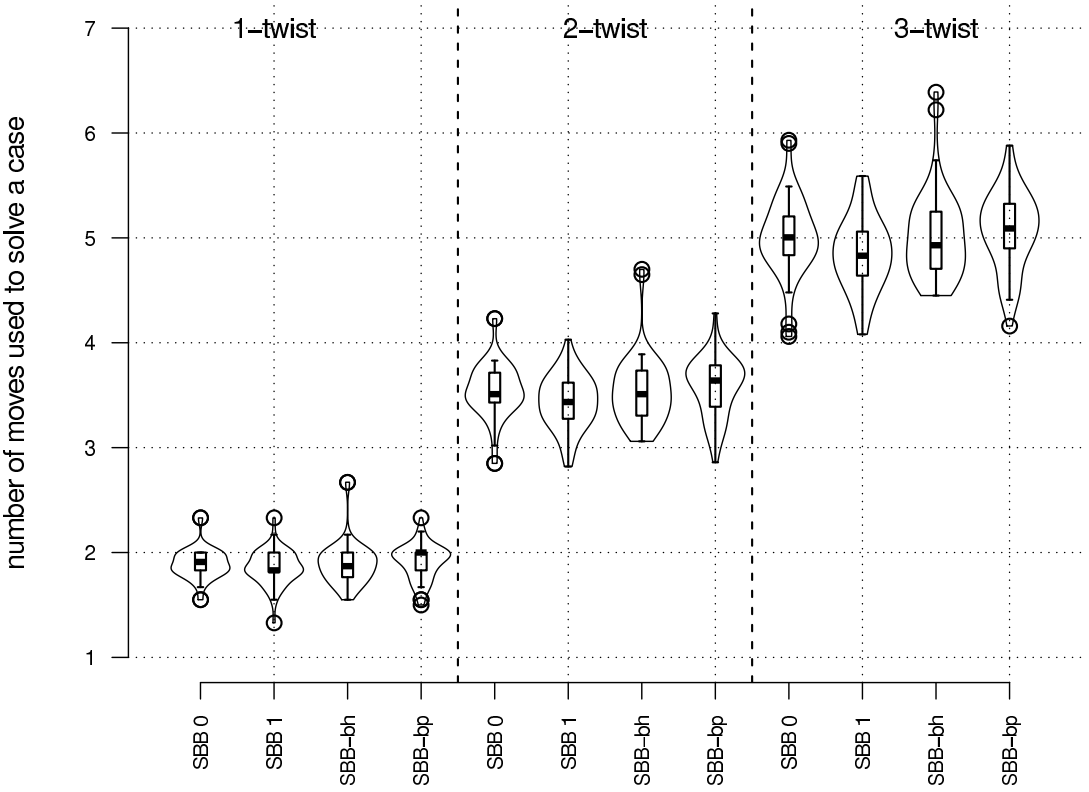


Figure 8.6: Number of moves used by the best host in solving the 1-, 2-, and 3-twist points. For each configuration, x-axis, the mean number of moves used in solving the test cases in each subset is shown, y-axis. The distributions are grouped by subset, and test cases that are not solved are not considered. Each distribution represents 60 initializations.

provides a measure of how efficient the evolved strategies are. The move counts for all configurations appear to be similar, though the median counts for the two-level formulation always fall below that of the other configurations – naturally, lower move counts are preferred. In all cases, the number of moves used appears to be considerably higher than the lower bound. Application of a two-tailed Man-Whitney test, Table 8.4, suggests that the SBB 1 move counts are lower than the SBB-bp move counts on the 2-twist and 3-twist points at a 0.01 significance level. Thus, even though both approaches achieve similar performance on the 2-twist and 3-twists cases, Table 8.3, it appears as though the layered approach is able to do so more efficiently.

	1-twist	2-twist	3-twist
SBB 1 vs SBB 0	0.2401⋆	0.01921⋆	0.009153⋆
SBB 1 vs SBB-bh	0.4976⋆	0.1374⋆	0.0534⋆
SBB 1 vs SBB-bp	0.02737⋆	0.001951⋆	0.0007957⋆

Table 8.4: Two-tailed Mann-Whitney test results comparing second-level SBB solutions with SBB 0, SBB-bh, and SBB-bp with respect to the number of moves used in solving the 1-, 2-, and 3-twist cases. Shown are the p-values when corresponding distributions in Figure 8.6 are compared, e.g., SBB 1 versus SBB-bh on all 1-twist points. Cases where the SBB median value is *lower* (better) are noted with a ⋆.

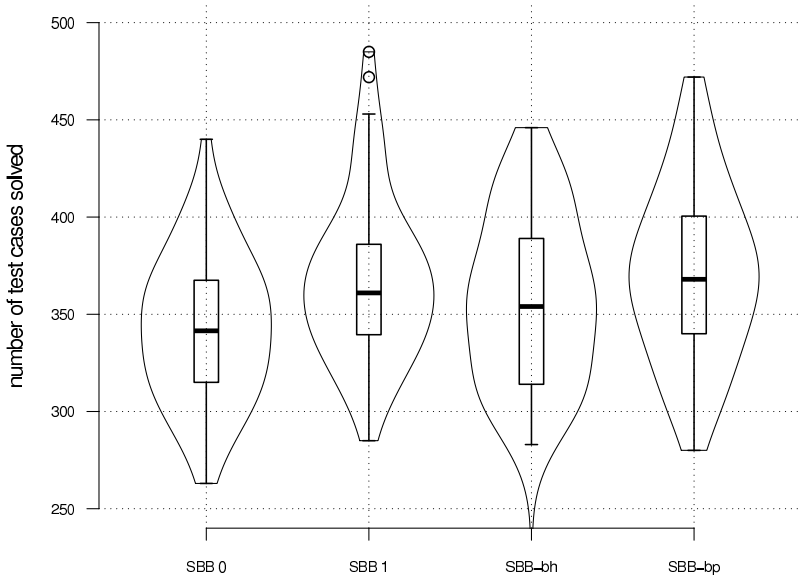
	Best individual	Population
SBB 1 vs SBB 0	0.002499⋆	3.109e-15⋆
SBB 1 vs SBB-bh	0.1519⋆	1.003e-10⋆
SBB 1 vs SBB-bp	0.5566	0.0002617⋆

Table 8.5: Two-tailed Mann-Whitney test results comparing second-level SBB solutions with SBB 0, SBB-bh, and SBB-bp on the random test set. Shown are the p-values when corresponding distributions in Figure 8.7 are compared, e.g., SBB 1 versus SBB-bh with respect to the population. Cases where the SBB 1 median value is *higher* (better) are noted with a ⋆.

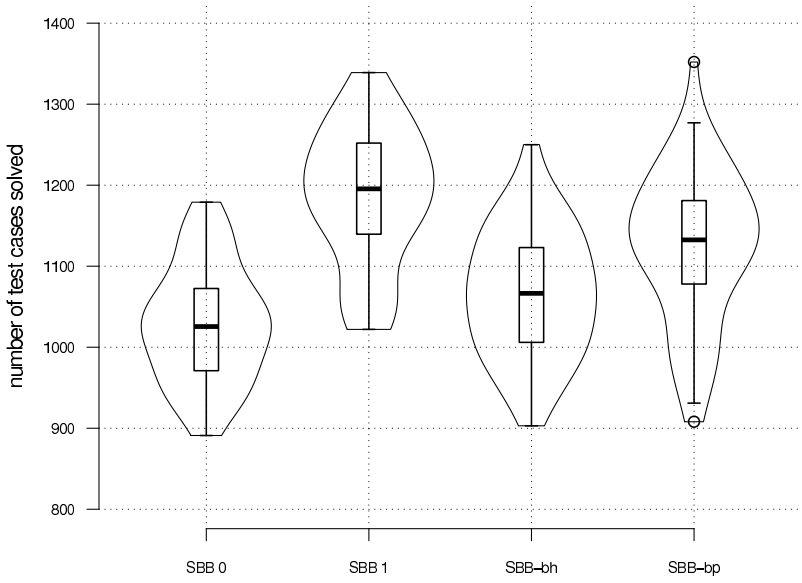
8.4.3 Performance on a Random Test Set

Solution counts on the test set containing 5000 random cases are shown in Figure 8.7. The two-level approach, SBB 1, appears to outperform SBB 0 and the big host configuration SBB-bh both in terms of the single-best individual and the population as a whole. Compared to the big program configuration SBB-bp, the two-level approach appears to result in similar single-individual counts, Figure 8.7a, but higher population-wide counts, Figure 8.7b. A two-tailed Mann-Whitney test, Table 8.5, suggests that indeed the population-wide performance under SBB 1 is better than that of the other configurations at a 0.01 significance level. With respect to the single-best individual, the tests indicate a significant difference only against SBB 0. Thus, the strength in the multi-level approach appears to lie in its ability to maintain more diverse behaviours in the population of solutions.

The median value under the SBB 1 distribution in Figure 8.7a is 361 – this can be interpreted as the expected best host performance on the random test set. In contrast, no random solver was able to solve more than 12 of these cases, Figure 8.4. Assuming a random solver population of size 60, the population-wide solved count



(a) Best host.



(b) Population.

Figure 8.7: Number of random test cases solved. For each approach, x-axis, shown is the number of cases solved, y-axis. The first and second SBB levels are denoted by SBB 0 and SBB 1 respectively, i.e., SBB 1 is constructed on top of SBB 0. The counts are with respect to the best host as well as across the population at the end of training, and each distribution represents 60 initializations.

on the random test set was estimated at 113, i.e., when the median cumulative value at $x = 60$ in Figure 8.4a was considered. In comparison, the median population-wide SBB 1 count, Figure 8.7b, is 1195.5. Thus, the models evolved under the SBB approach are clearly able to improve upon random behaviour.

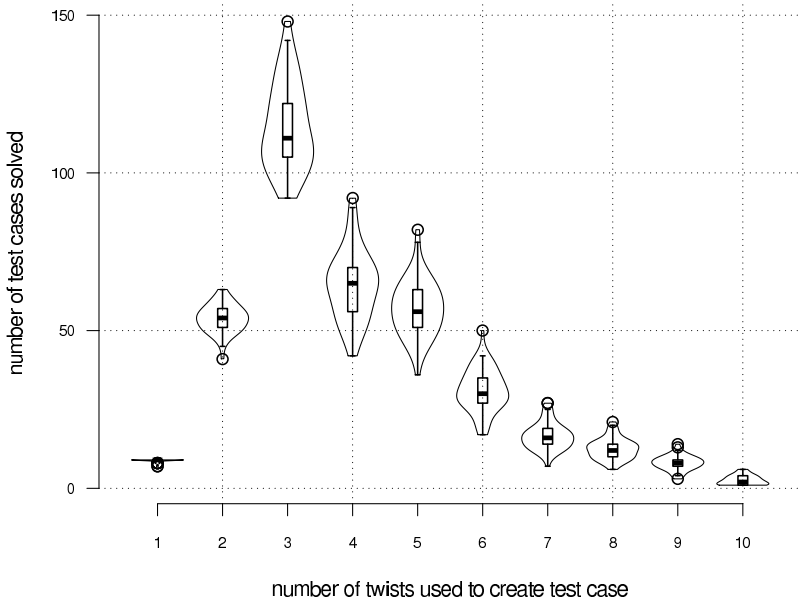
Based on the number of moves, cnt_{init}, used to create each instance in the random test set, the test cases were partitioned into ten subsets. The number of cases in each subset that was solved, as well as the mean number of moves used in solving the cases, was then considered, Figure 8.8. Here, only the SBB 1 results are shown, though the trends under SBB-bh and SBB-bp were observed to be similar; with respect to the test cases solved under each twist count, i.e., considering pairs of distributions under each of the ten twist counts, a two-tailed Mann-Whitney test did not indicate a difference at a 0.01 significance level in all ten comparisons with each of the two other approaches.

Much as in Figure 8.5, a sharp decrease in the number of test cases solved is observed as the upper bound on the required number of twists in Figure 8.8a increases. As indicated earlier, Figure 8.3, the total number of points in the random test set generated using a given number of twists levels off at about five twists, while the counts in Figure 8.8a continue to decrease. In addition, there appears to be some limit on the difficulty, as measured in terms of the number of moves from the solved state, of the cases that the approach is able to solve. That is to say, above $x = 6$, the median mean values in Figure 8.8b remain relatively steady and below the twist count cnt_{init} indicated on the x-axis. Thus, at best, the approach is able to consistently solve cases that are six twists away from the solved state.

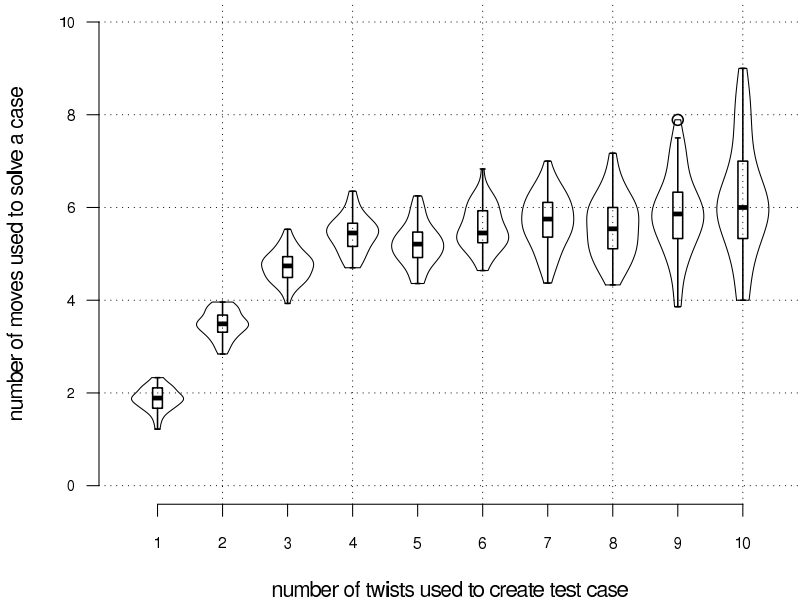
The goal of partitioning the random test set with respect to the number of twists used to generate the individual cases was to obtain subsets of increasing difficulty. As evidenced by Figure 8.8, after this partitioning process, subsets associated with higher twist counts still tend to contain easy cases, i.e., cases that can be solved in fewer twists than were used in their creation. This results because later moves may undo the effects of previously applied moves. This effectively defeats the purpose of performing the partitioning in the first place. Thus, a more comprehensive comparison with respect to these subsets was not performed.

8.4.4 Complexity of Evolved Solutions

The complexity of the best hosts is summarized with respect to the effective instruction counts across the host, the number of symbionts in the host, as well as the



(a) Number of test cases solved.



(b) Number of moves used in solving test cases.

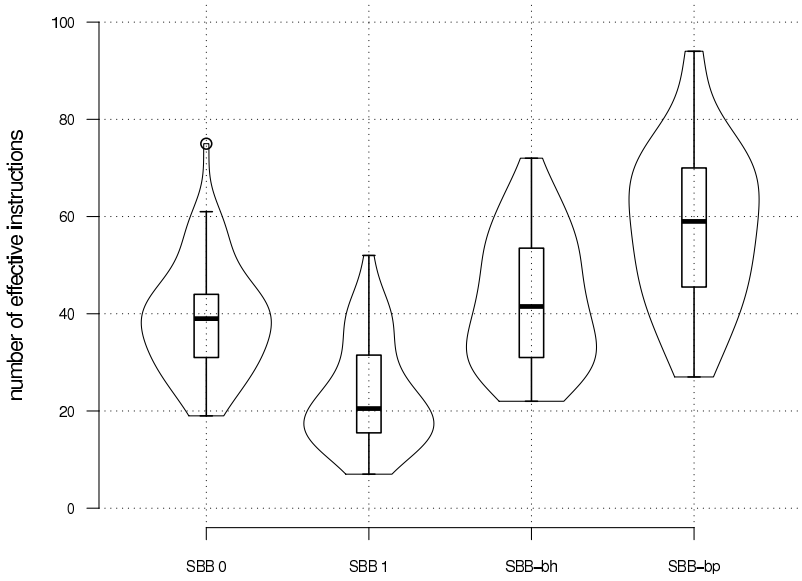
Figure 8.8: Number of test cases solved by the two-level SBB solutions on the random test set and the associated mean number of moves used. The test cases are partitioned into ten subsets, x-axis, based on the number of twists used in their creation. All counts reflect the best host in the final population. Three initializations were found where the count in one or more subsets was zero and these were excluded, therefore, 57 initializations are plotted.

number of effective instructions per symbiont in the host, Figure 8.9. In all cases, the SBB 1 counts do not include the SBB 0 counts, e.g., the instruction counts at the first level are not counted again at the second level. The SBB 0 and SBB 1 distributions in Figure 8.9a indicate the number of instructions across the best host at the first and second level. Here, compared to the first level, the hosts at the second level appear to be simpler, though it is not clear why. Despite having the ability to evolve solutions that consist of the same number of instructions, the SBB-bh solutions appear to be less complex than the SBB-bp solutions in terms of the instruction counts. This reduction in complexity appears to be a direct result of the increased capacity for compartmentalization under the SBB-bh configuration, though in general the performance of SBB-bh was found to below that of SBB-bp.

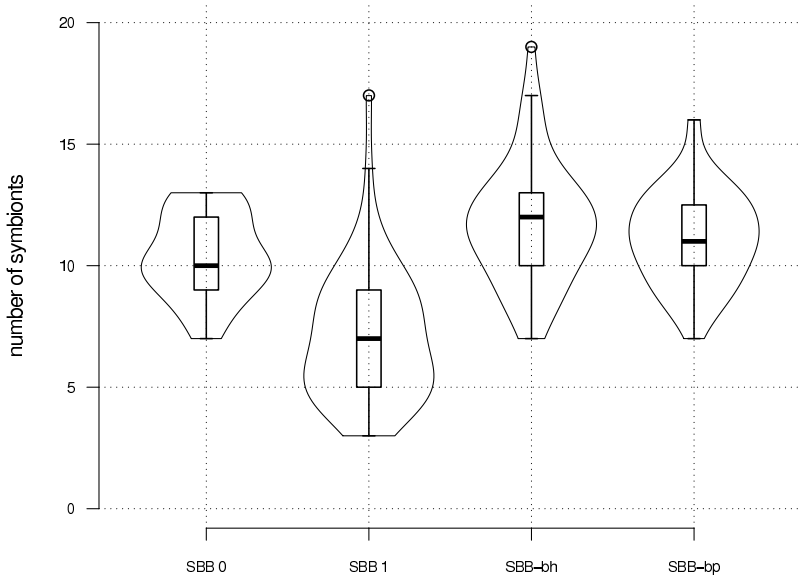
Considering the two-level configuration in Figures 8.9a and 8.9b, the median host size at the second level is 7 while the median effective instruction count at the first level is 39. Thus, the instruction count across both levels can be estimated as $7 \times 39 + 20.5 = 293.5$, assuming a median instruction count of 20.5 at the second level. This is considerably higher than the overall instruction counts observed under SBB-bh and SBB-bp, though it is likely an over-estimate that does not account for lower-level programs that are referenced multiple times. In terms of the number of instructions that are executed in order to select an action, using the median values once more the two-level count can be estimated as $39 + 20.5 = 59.5$ where this is more competitive with the other two approaches.

With respect to the number of symbionts per host, Figure 8.9b, comparing SBB 0 with SBB 1 a reduction in complexity is again observed at the second level. No difference between SBB-bh and SBB-bp is detected at a 0.01 significance level using a two-tailed Mann-Whitney test, despite the increased upper limit on the host size under the big host configuration. Given median SBB 0 and SBB 1 symbiont counts of 10 and 7 respectively, the number of symbionts across both levels in the two-level configuration can be roughly estimated at 77 – this is clearly higher than the counts observed under the big host and big program configurations. However, the number of symbionts that have to be considered in selecting an action in the two-level approach can similarly be estimated at $10 + 7 = 17$, making the runtime cost competitive.

The number of effective instructions per symbiont, Figure 8.9c, highlights the simplicity of the evolved individuals. Once again, the SBB 1 counts appear lower than the SBB 0 counts. The big program configuration, SBB-bp, results in the highest counts, though the third quartile value is still below six.

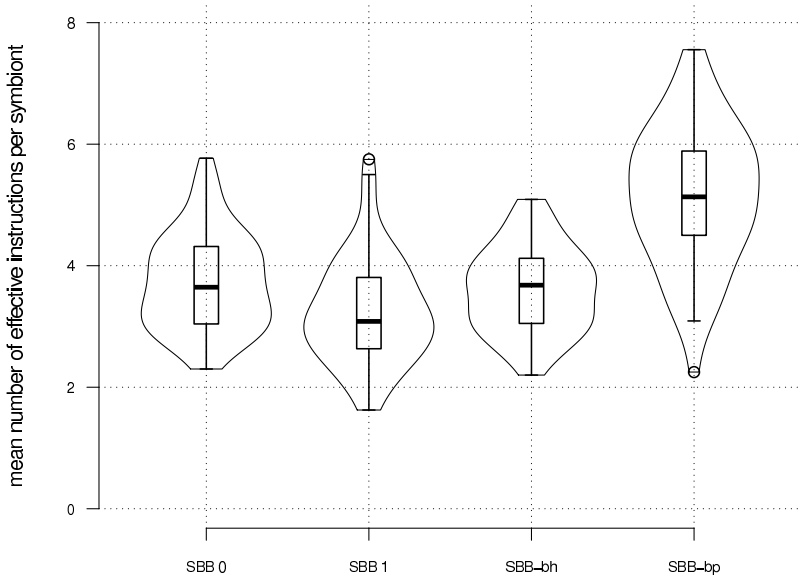


(a) Number of effective instructions per host.



(b) Number of symbionts per host.

Figure 8.9: Complexity of the solutions evolved on the Rubik's Cube problem with respect to the best host. For each configuration, x-axis, shown is the number of instruction across the host, Figure 8.9a, the number of symbionts in each host, Figure 8.9b, and the mean number of instructions per symbiont, Figure 8.9c. First and second SBB levels are denoted by SBB 0 and SBB 1 respectively. All instruction counts are post intron removal, and each distribution represents the best individual in each of the 60 initializations.



(c) Mean number of effective instructions per symbiont.

Figure 8.9: Continued.

In summary, when considering the complexity of the entire solution, i.e., the symbiont or instruction counts across all levels, the solutions under the two-level configuration appear to be more complex than the big host and big program configurations. However, in terms of the number of instructions or symbionts that are engaged during a single application of a host, i.e., in calculating an action at one time step, the overhead under the two-level approach is competitive; this is reflected in the similar training times under the three configurations, Figure 8.10, which one would expect given the parameters used, Section 8.3.2. This is a result of the hierarchical organization of the solution subcomponents in the multi-level configuration. Furthermore, the complexity at the second level in the two-level approach tends to be smaller than at the first level, though it is difficult to say why from the current data.

8.4.5 Layering under Limited Computational Resources

No significant difference was found in the best host solved counts when the second-level SBB individuals were compared with individuals evolved under the big program configuration, Tables 8.3 and 8.5. Both configurations assume a common set of design choices such as the fitness and distance functions, the cube state representation, and the GP instruction set. These design decisions, though independent of the core SBB algorithm, may ultimately determine the effectiveness of the approach. In particular,

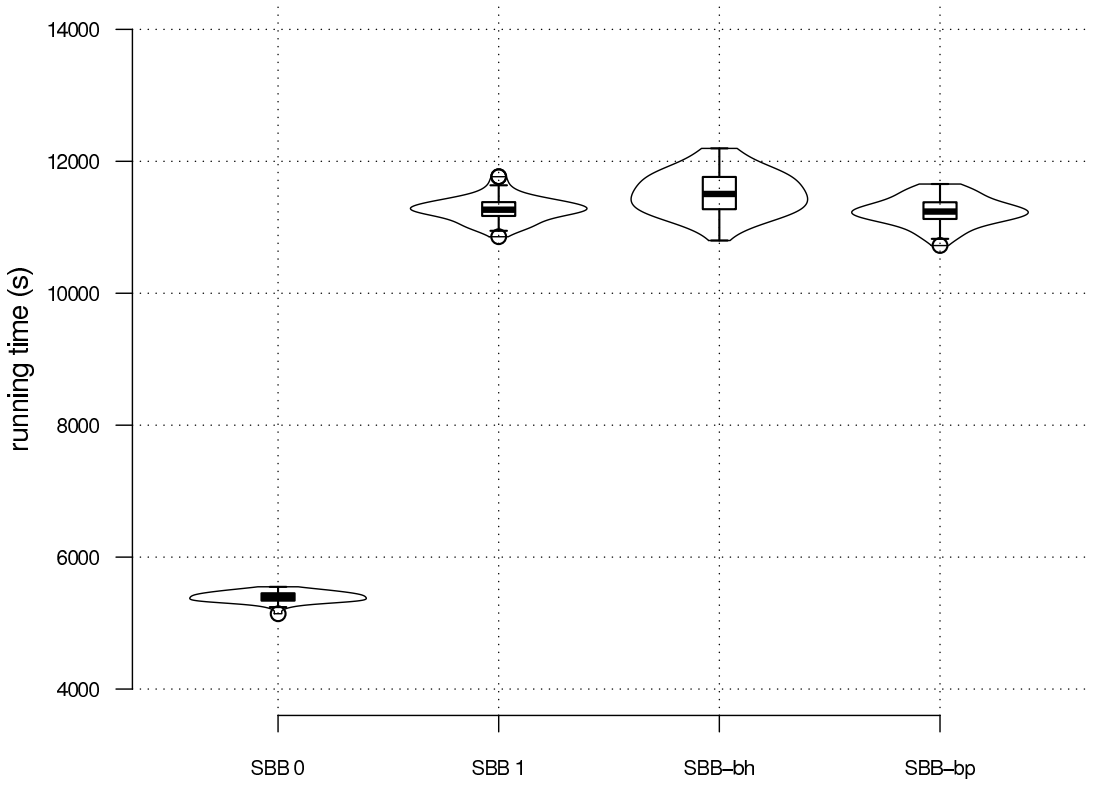


Figure 8.10: Comparison of the training times on the Rubik's Cube problem. Training overhead was measured as described in Section 4.1.4, i.e., accounting for effective instructions executed in user mode. A two-tailed Mann-Whitney test detected a difference between the SBB 1 and SBB-bh distributions at a 0.01 significance level, but no such difference was detected between the SBB 1 and SBB-bp distributions.

they may impose a ceiling on performance, i.e., a plateau after which no significant progress can be made without applying a disproportionate amount of computational effort. A reason for the similar levels of performance may therefore be that given their common implementation the plateau was reached under both configurations. Similar behaviour was encountered in previous work on evolving solutions to the Rubik's Cube where little progress was made beyond a certain point [12].

Hitting such a ceiling would depend on the choice of parameters because sufficient computational effort would need to have been applied. It may also be that, if the ceiling was reached, too much computational effort was applied; since only end-of-run performance was considered, it is impossible to determine at which point during training the plateau was reached. With these considerations in mind, one more set of runs was performed under parameter settings where it was known that further effort would yield improved performance, that is, the target level of performance in these additional runs was below previously established levels.

To this end, two SBB configurations were considered. The first, denoted 'SBB short', involved three levels, training each level for 10 generations; otherwise, parameter settings were as before, Section 8.3.1. By reducing the number of generations from 1000 to 10, the performance levels established previously were unlikely to be reached. The second configuration, denoted 'SBB-bp short', was a single-level configuration analogous to the SBB-bp configuration, Section 8.3.2, but where the maximum program size was increased to 48 and the generational limit set at 30 to match the number of fitness evaluations allowed under the SBB short configuration. The choice was made to increase the maximum program size instead of the maximum host size because earlier in this chapter the former was found to perform better.

The single-best individual results on the random test set are shown in Figure 8.11. The median values under the SBB short 2 and SBB-bp short distributions fall well below the median SBB 1 and SBB-bp values established previously, Figure 8.7a. Thus, significant progress can be made by training for more than 10 generations so any ceiling, if it exists, is not yet reached. Furthermore, layering is now observed to have a much stronger impact. Specifically, a two-tailed Mann-Whitney test now suggests a difference between each successive pair of SBB levels in the three-level configuration as well as the third SBB level and the SBB-bp short distribution at a 0.01 significance level[9].

The results presented thus far, in addition to those in Figure 8.11, suggest that

[9]In all three cases the p-values fall below 10^{-10}.

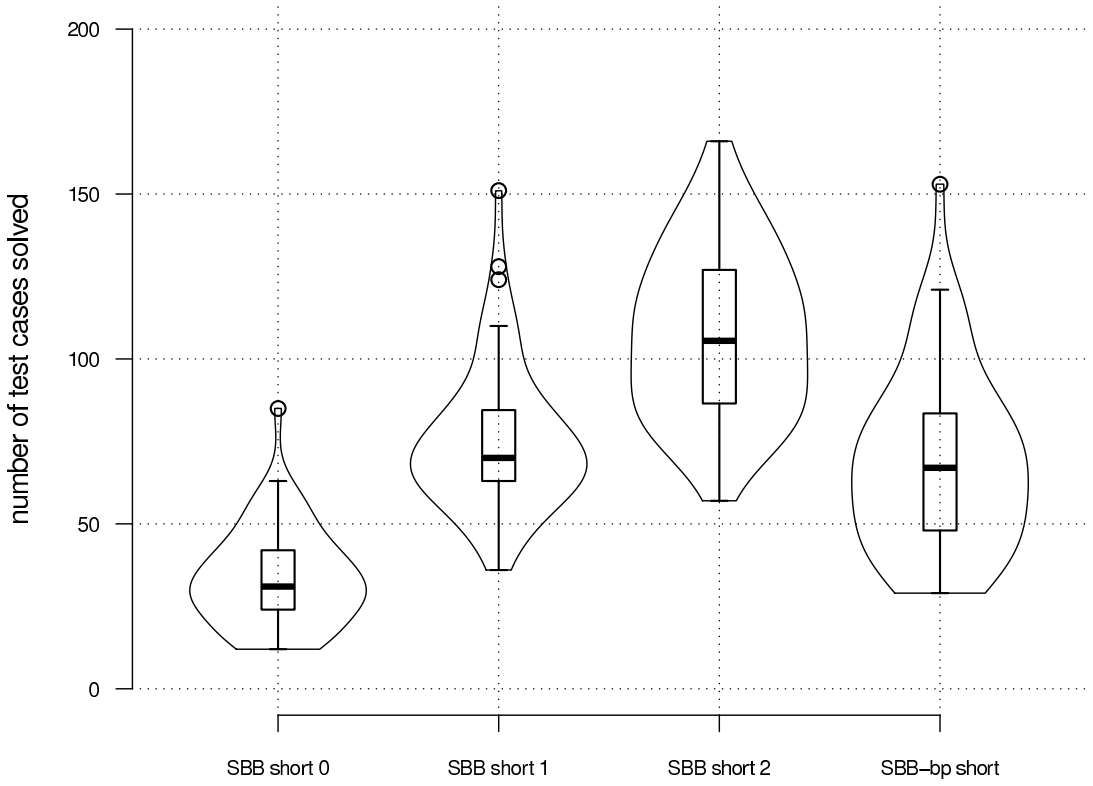


Figure 8.11: Number of random test cases solved by the best host assuming limited computational resources. The plot is analogous to Figure 8.7a but compares a three-level SBB configuration, SBB short, with a big program configuration using similar computational resources, SBB-bp short. The SBB short results are further broken down by level indicating the performance of the best host in the final population at each level.

with and without layering, similar levels of performance can be reached given sufficient computational effort. However, they also suggest that layering is more efficient and can yield stronger performance under limited resources. Here, the resource limit was applied artificially, but in practice, the limit may be enforced due to other constraints. Further work is required to analyze the tradeoff between constructing more layers or training each layer for a longer period of time.

8.5 Summary of Results

A two-level configuration of the SBB algorithm was applied to the Rubik's Cube problem and compared with two single-level configurations, a big host configuration and a big program configuration. Solving the Rubik's Cube was found to be a difficult task, with cube instances three twists away from the solved state already posing a challenge. Of the three SBB configurations that were compared, the two-level configuration and the big program configuration were found to be the most successful in terms of the number of test cases solved, however, in general, higher performance was expected. Furthermore, the performance of the SBB approach was clearly better than that of a random solver suggesting that there is sufficient structure in the domain to support learning, though it is also clear that the current results represent a starting point in this challenging problem domain.

A distinguishing quality of the two-level solutions was that they tended to be more efficient in terms of the number of moves used and that they resulted in better, or more diverse, population-wide performance. In addition, layering was found to be particularly beneficial under limited computational resources. One potential drawback of the two-level solutions was their overall complexity, though in terms of runtime efficiency they were found to be competitive with the other two approaches. As such, the hierarchical structuring of solution subcomponents provides a way to add complexity without increasing computational overhead, which, under difficult problem domains, is viewed as a very desirable property. This also emphasizes that when considering the complexity of solutions, the way in which that complexity is realized should be taken into account, i.e., not all complexity is the same.

What makes the Rubik's Cube domain, as formulated in this chapter, so difficult is that no obvious cost function is available. As such, the fitness function used here does not distinguish between the vast majority of cube states, providing little information about the quality of candidate solutions with respect to the entire state space. The lack of an appropriate fitness function may limit the quality of the solutions that can

be evolved, effectively imposing a low ceiling on the performance levels regardless of the computation effort applied. Resolving this issue is therefore suggested as a starting direction for any future work. For example, performance may improve with the use of an explicitly multi-objective fitness, e.g., with one objective per face. Another possibility is a lexicographic formulation that augments the function currently used to provide more information in situations that yield the worst-case distances, e.g., by considering the number of correctly placed cubies. Alternatively, one could use the stages in Thistlethwaite's strategy to define subgoals, though naturally this assumes some degree of domain-specific knowledge.

Chapter 9

Conclusion

The conclusion of this thesis begins with a summary of the core ideas and motivations in Section 9.1. Having identified a number of broad research objectives in the introductory chapter, these are then retrospectively discussed in Section 9.2. The final two sections of the conclusion then summarize the key contributions made in this thesis as well as a number of possible directions for future work.

9.1 Summary of Core Ideas and Motivations

In the domain of biological evolution, most tend to recognize the central tenets behind neo-Darwinism, i.e., the principle of evolution by which populations are believed to diversify over time. However, there appears to be little consensus about the exact nature of the mechanisms driving evolutionary change, and in particular, the role that symbiosis may play in the emergence of complex life forms. In contrast, the same type of discussion has not taken place in the EC community where the norm has been to rely on sexual reproduction with little consideration given to alternate mechanisms of inheritance such as symbiosis. At the same time, there has been an ongoing debate about the lack of stability in the crossover operator, though the focus appears to have been directed on improving existing frameworks as opposed to investigating new mechanisms for the stable and variable transfer of genetic material. Others have rejected crossover altogether relying solely on mutation for variation, though by itself mutation is not viewed as an inheritance mechanism, i.e., it is not capable of combining genes from multiple sources but rather acts as a background process.

Motivated by the ongoing discussions taking place in the field of biological evolution, the core idea behind this thesis is therefore the use of symbiosis as an alternative to sexual recombination in an EC setting. It is not claimed that this is the first work which considers this substitution, but given the limited attention that symbiosis has received in the EC community (and in the GP community in particular), the work

presented in this thesis is nonetheless considered to be novel[1]. Much like the products of crossover, the partnerships identified under a symbiotic metaphor may not always be stable, thus, the shift from recombination to symbiosis can on the surface be regarded as having limited practical significance. However, given the form of symbiosis as adopted in this thesis, the metaphor provides an elegant means of problem decomposition including a biologically-inspired approach to incremental complexification, both of which are considered vital in the successful development of solutions to difficult learning problems.

Specifically, the three-staged process of symbiosis-as-an-operator adopted in this work not only presents an alternative to sexual recombination, but it also represents an approach to both lateral and vertical problem decomposition. Often observed in complex systems, problem decomposition is considered to be an important problem solving strategy as it is viewed as synonymous with solution transparency, learnability, and efficiency. In the proposed approach, lateral problem decomposition is realized during the species coexistence and compartmentalization stages whereby individuals from distinct lineages may form partnerships onto which natural selection is applied as a group operator. Key to the compartmentalization of symbionts is the bid-based approach to context learning, separating the 'what to do' from the 'when to do it', and providing a metaphor for communication and consequently cooperation between the symbionts. Vertical problem decomposition is realized during the joint replication stage as units at lower levels of organization are incorporated into units at higher levels, i.e., during hierarchical model construction. Again, the same bid-based framework provides a simple way of incorporating behaviours developed at lower levels in the hosts at higher levels. A critical requirement in this hierarchical evolutionary process is the diversity of the lower-level building components since there is little to be gained in combining like units under the bid-based paradigm.

The bid-based approach to context learning and evolution through symbiosis (as opposed to sexual recombination) are therefore viewed as two central principles underlying the SBB algorithm. Together, they represent a self-contained framework; however, it is asserted that in order to apply this framework to truly difficult learning tasks, i.e., tasks where problem decomposition is likely to play a key role, the issue of scalability needs to be addressed. Alternatively, an approach capable of problem decomposition is of little use if the learning process cannot be conducted in an efficient manner. To this end, a third critical component in the proposed approach

[1]Specific contributions made are detailed in Section 9.3.

is the dynamic evaluation of candidate solutions through coevolution, thus allowing training overhead to remain constant regardless of the number of training scenarios. In addition, two other benefits are associated with the use of a dynamic cost function. First, a dynamic evaluation set supports directional natural selection and may therefore help to prevent premature convergence, where this is particularly important in the hierarchical construction process which requires diversity at lower levels. Second, a dynamic evaluation set may more effectively identify the relevant set of niches, particularly when the instance space is large, which in turn may complement the speciation process that is encouraged in the population of candidate solutions.

The above is a digest of the core ideas and motivations behind the SBB approach as presented in this work. The following section focuses on the research objectives as stated at the outset of this thesis, Section 1.6, summarizing the key empirical results that were obtained with the goal of providing tangible support for a number of the points touched upon above.

9.2 Research Objectives: Post-Mortem

A number of research objectives were stated in the introductory chapter of this thesis, Section 1.6. Here, these objectives are recounted, and some retrospective remarks relating how these objectives were satisfied in this thesis are presented. In the process, the discussion highlights what are viewed to be the most significant empirical results in Chapters 4 through 8.

> **Objective 1:** *To present a novel EC algorithm that incorporates symbiosis as a primary mechanism driving evolution and which supports problem decomposition.*

The SBB framework, detailed in Chapter 3, was described in terms of three underlying principles: a bid-based approach to context learning, evolution of solutions by means of symbiosis, and dynamic evaluation through coevolution. The bid-based action selection process provides the basis for inter-symbiont communication, casting GP in a non-traditional role, and thus supports the three-staged process of symbiosis-as-an-operator as adopted in this work. In contrast to traditional approaches in EC which rely on sexual recombination or mutation as the primary mechanisms driving evolution, the SBB framework relies primarily on symbiosis with mutation as a background operator. Under the symbiotic metaphor that is assumed, both lateral and vertical problem decomposition emerge naturally. To allow the approach to scale to

problems with large state spaces, which is a characteristic of domains where problem decomposition is likely to be important, the framework also included a dynamic evaluation function. The proposed approach is considered to be novel because, by adopting an abstraction of the biological process of symbiosis as the primary mechanism of inheritance, it gracefully combines both lateral and vertical problem decomposition, i.e., both context learning and layered learning; specific contributions relative to previous work are detailed in Section 9.3.

> **Objective 2:** *To demonstrate that the proposed approach is effective versus comparable approaches with a reduced capacity for problem decomposition.*

The SBB framework, in which symbionts represent explicit solution subcomponents that together may decompose the problem, was compared with a monolithic approach on a set of binary classification problems in Chapter 4. The monolithic approach was formulated in such a way that any benefits associated with an increased capacity for problem decomposition could be isolated. Based on the evaluation, it was determined that the symbiotic approach resulted in improved classification performance compared to the monolithic approach and that it also required less computational overhead during training. Initially, it was suggested that the symbiotic approach may yield solutions that are less complex, however, this claim was not supported by the results.

In the comparison of Chapter 4, reduced capacity for problem decomposition was associated with monolithic solutions that did not consist of explicit subcomponents, i.e., symbionts, and otherwise the two formulations that were compared consisted of just a single level. In Chapter 7, a two-level SBB configuration was compared against a single-level SBB configuration under the same fitness evaluation limit – as such, any benefits associated with the capacity for vertical problem decomposition could be determined. In terms of the number of test cases solved under the truck reversal domain, the two-level solutions were found to perform much better, a result attributed to the diverse behaviours evolved in the first level. In terms of overall solution complexity, the single-level solutions were simpler. However, it was suggested that because not all solution subcomponents are engaged in the two-level solutions at any one time step the associated increase in runtime cost is not proportional to the increase in complexity. It was also argued that the increased complexity in the two-level solutions was warranted given that the current state-of-the-art failed to produce competitive results, and that the way in which solution complexity is added

should also be considered, i.e., in the case of the SBB algorithm the increase in complexity was a result of layered learning and not a side-effect of evolution. For example, naively increasing the host size limit under a single-level configuration did not yield an improvement in performance, suggesting that a structured approach to complexification is key.

In the Rubik's Cube experiments of Chapter 8, a two-level SBB configuration was compared with a big host and big program configuration each consisting of a single level and assuming equal overhead in terms of the maximum number of instructions executed. With respect to the number of cube configurations solved, performance was lower than expected in all cases and no single configuration stood out. However, it was demonstrated that despite producing solutions that were more complex, the training overhead under the two-level configuration was comparable to the training overhead under the two single-level configurations. This emphasized that the way in which complexity is added is an important consideration, and in the case of the hierarchical SBB formulation, complexity can be increased in an efficient manner. A final result in the chapter suggested that the benefits of layering may be more apparent when limited computational resources are available, or alternatively, when one does not have the resources to continue training until a plateau in performance is reached.

> **Objective 3:** *To analyze the nature of the problem decomposition resulting from the application of the proposed approach.*

Under classification, the solutions evolved under a single-level SBB configuration were characterized in Chapter 4. It was found that, in terms of the number of effective instructions and unique features accessed, the symbionts appeared to be very simple even with no baseline level of complexity available, i.e., the monolithic approach was not amenable to such analysis. The simplicity of the solution subcomponents – the symbionts – suggested a high degree of problem decomposition since the ability of each subcomponent was limited. Furthermore, each symbiont was found to access a small number of features regardless of the dimensionality of the attribute space. In terms of the way in which individual symbionts associated themselves with different exemplars by way of the bidding process, it was found that typically there was only one symbiont that won a large portion of instances. Symbionts that used no features, i.e., bid a constant value, were also observed, suggesting the emergence of default hierarchies. Considering performance across the final host population, it was found that typically all instances in the training data were classified correctly by at least one

non-degenerate host, where this diverse behaviour is attributed to the fitness sharing mechanisms incorporated into the SBB framework. In Chapter 5, the SBB solutions were compared with AdaBoost ensembles with respect to the overlap in features accessed by the symbionts/trees, and here, it was found that relatively speaking the overlap in features accessed by the symbionts was found to be lower – this despite the SBB hosts using fewer features on most of the datasets. This result further emphasized the capacity for problem decomposition under the proposed approach, in this case, with respect to the attributes used by different subcomponents.

Under temporal sequence learning, SBB solutions on the truck reversal domain were analyzed in Chapter 7. Here, the focus was more on the ability of the second-level hosts to leverage the different behaviours learned at the first level, given that the first-level behaviours could potentially represent solutions to different subproblems. It was found that indeed the second-level hosts were able to successfully combine first-level hosts resulting in much improved performance. Furthermore, they were able to do so in novel ways solving test cases that were not solved at the first level. In addition, a detailed analysis of a two-level SBB solution illustrated how different first-level behaviours were exploited in the course of a single episode.

> **Objective 4:** *To compare the proposed approach against current state-of-the-art algorithms on a set of challenging real-world problems.*

One approach to the development of a new learning algorithm is to evaluate its performance on artificial problems under which specific behaviours can be isolated. However, the results on the artificial task may not carry over to real-world scenarios. In this work, the aim was to evaluate the proposed approach on real-world problems directly, thus focusing on performance, while at the same time observing solution characteristics of interest, and in particular, complexities and any problem decomposition that may emerge. Even though what one considers to be a real-world problem may be debatable, it is maintained that the problems used in this thesis are representative of the more difficult tasks encountered in benchmarking studies under classification and reinforcement learning. This, in turn, justifies the use of an approach that relies on problem decomposition despite the associated potential for increased overhead.

In Chapter 5, the SBB algorithm was compared with SVM classifiers as well as AdaBoost ensembles. The classification performance comparison highlighted the effectiveness of the two baseline algorithms, and especially AdaBoost, which, though perhaps not as cutting-edge, still yielded strong results. In most cases, the SBB classification performance lagged behind that of the other two algorithms, suggesting

a requirement for more consistency under the approach, e.g., through parameter tuning. On the other hand, the SBB solution complexities were typically found to be lower, particularly when considering the number of features used, making evident the complexity-performance tradeoff. An underlying theme in the chapter was that solutions should be assessed in terms of more than one criterion and that basing evaluation on a lone factor, e.g., classification accuracy, may fail to provide the whole picture. In particular, it was suggested that there is a growing need for using, e.g., GP, to model interesting relationships in data as opposed to focusing on classification performance alone.

Under temporal sequence learning, the SBB algorithm was compared against NEAT. The comparison was restricted to the truck reversal domain where random sampling was found to be effective and NEAT could therefore be easily applied, i.e., a more involved point sampling process was assumed under the Rubik's Cube domain making any integration with NEAT more involved. Despite using NEAT parameters specific to the truck reversal domain, and generic parameters under the SBB algorithm, the performance of the latter was found to be *much* better. This suggested that the SBB algorithm may be particularly well suited for temporal sequence learning, though further study is clearly required. Based on the results, it was also conjectured that the NEAT speciation mechanism may be effective at establishing context for crossover thus focusing on individual as opposed to population-wide performance. In contrast, by supporting diversity through implicit and explicit fitness sharing, the SBB algorithm was found to produce diverse first-level host populations that provide the basis for strong behaviour at the second level. Furthermore, the poor performance of NEAT helped to establish the relative difficulty of the truck reversal formulation used in Chapters 6 and 7.

> **Objective 5:** *To explore the design of a dynamic evaluation function and investigate the conditions under which specific design choices are appropriate.*

This objective refers to the interaction between the host and point populations that allows the SBB approach to scale to problems with many training cases. In Chapter 6, the design of this algorithm component was considered, and in particular, the focus was on the policies for managing the point population. Both binary and real-valued cost functions were considered under the assumption that the latter provide more information about the quality of the individuals in the host population.

The experiments of Chapter 6 uncovered a number of insights regarding different policies dictating the way in which points are added and removed. First, under real-valued rewards, it became apparent that the naive but more efficient strategies such as RSS and SSS are just as effective as the point fitness function in the SBB algorithm. However, under binary rewards, the naive sampling heuristics were found to be ineffective relative to the proposed point fitness function. This difference in behaviour was attributed to the increased amount of information available under the real-valued scheme, making it likely that a random set of points would provide a learning gradient; the same is typically not true under binary reward schemes necessitating additional effort to find and maintain an informative set of points. This also precluded the use of RSS on the Rubik's Cube problem where the fitness function provided information about only a small proportion of the possible cube configurations. Chapter 6 also presented a comparison with a formulation of the proposed point fitness function without a genotypic reward component, as well as the more traditional distinction-based point fitness commonly found in Pareto-coevolution. In the former case, including genotypic diversity was found to result in equal or better performance, where this can be attributed to improved coverage of the state space by members of the point population. In the latter case, no statistically significant difference was found between the performance under the distinction-based fitness and the proposed fitness functions, with both formulations factoring in the genotypic diversity reward. However, a preference for the proposed function was suggested on the basis of computational efficiency.

9.3 Key Contributions

The following are considered to be the key contributions made in the process of developing, evaluating, and presenting the SBB framework for problem decomposition:

1. Many different definitions of symbiosis – the living together of different organisms [8] – are observed in EC literature, though often the definitions are narrow as they are typically based on the specific cost-benefit characterization of the relationship [31], i.e., symbiosis is viewed as a state. A notable exception is the work on compositional versus accretive mechanisms of genetic inheritance [189] where a defining characteristic of symbiosis is that it is a process that combines genetic material from different lineages. In this thesis, an even broader definition of symbiosis is proposed for use in EC which is adopted directly from

evolutionary biology [121]. Specifically, the three-staged definition of symbiosis-as-an-operator assumed here is compatible with any cost-benefit characterization, and can also be viewed as representative of a compositional mechanism of inheritance. This formulation of symbiosis also provides an alternative to the crossover operator often employed in EC which may fail to adequately model homologous sexual reproduction in nature, i.e., which is variable but unstable.

2. Based on the definition of symbiosis assumed in this thesis a framework for both lateral and vertical problem decomposition is proposed. Whereas lateral problem decomposition is realized during the species coexistence and compartmentalization stages, vertical problem decomposition is realized during the joint replication stage. A desirable attribute of the proposed approach is that both forms of problem decomposition emerge from the same biologically-inspired definition of symbiosis.

   As suggested in Chapter 2, various approaches to lateral and vertical problem decomposition have been proposed. However, it is argued here that many of these approaches are capable of either lateral or vertical problem decomposition but not both, and furthermore, that compared to the proposed approach they often require considerably more knowledge to be specified *a priori*. For example, GP teaming has traditionally been used as a form of lateral problem decomposition requiring the *a priori* specification of the number of cooperating individuals, e.g., [19, 173], which, in the proposed approach, is determined automatically. Using layered learning [180], on the other hand, both lateral and vertical problem decomposition can be achieved, though in doing so intimate domain knowledge is required. In addition, approaches to lateral problem decomposition often follow what was referred to in Section 2.2.3 as an ensemble learning methodology, i.e., they result in a set of highly overlapping behaviours. In contrast, like the proposed approach, bid-based approaches such as Hayek [10] and Michigan-style classifier systems [72, 196] aim to identify unique contexts for specific behaviours, though they fail to structure solutions hierarchically. Finally, though the proposed approach mirrors the hierarchy observed in SANE [132], the latter fails to recognize that the two-population architecture can be applied iteratively to produce multiple levels.

3. Having established a symbiotic metaphor that supports both lateral and vertical problem decomposition, this thesis also proposed the bid-based approach to

context learning as a mechanism for enabling communication among the symbionts in a host. Under this approach, the issues of 'what to do' and 'when to do it' are explicitly separated with the aim of evolving a set of non-overlapping behaviours – this is in contrast to the norm under approaches following the ensemble learning methodology. This separation also recognizes that the most suitable action associated with a given bidding behaviour may change depending on the context, i.e., the other symbionts in the host, potentially facilitating the learning process since the same behaviour does not have to be learned from the beginning if it is to be combined with a different action. In addition to forming the basis for cooperation among symbionts at same level, the bid-based metaphor also supports the hierarchical model construction process with no modifications to the underlying algorithm, i.e., only the action set is redefined at each successive level; as was already suggested, the hierarchical organization of solution subcomponents is what differentiates the proposed approach from other bid-based formulations, e.g., [10, 72, 196].

In this thesis, the decision was made to evolve the bidding behaviours using GP, though other representations, e.g., neural networks, may also be effective. This cast GP in a non-traditional role, that is, using programs to map input to a bid value as opposed to mapping input directly to an output. The choice to use GP was based in part on historical reasons and in part because it represents a flexible and robust approach to evolving real-valued functions.

4. The investigation of Chapter 6 established conditions under which naive point sampling heuristics such as RSS, employed in order to reduce computational overhead, may be suitable. Specifically, a distinction was made between binary reward schemes, indicating either success or failure, and real-valued reward schemes that provide considerably more information by allocating partial credit. It was then suggested that in the case of binary rewards, the naive sampling heuristics are unlikely to be effective and that greater effort must be expanded in order to find and maintain an informative set of points, e.g., by explicitly coevolving a set of points. These results were consistent with other observations made recently under the proposed approach [113]. In the process, a new point fitness function was proposed that provides an alternative to the distinction-based function traditionally used under Pareto-coevolution [46, 139]; though no significant difference in performance was observed under the two functions, it was argued that the proposed function is more efficient.

5. Increased solution complexity is often associated with a reduced capacity for generalization since it can lead to the memorization of the training data including any outliers, i.e., it may result in the models over-fitting the training data. However, the results of Chapter 7 demonstrated that this need not be the case and that increased complexity can lead to improved generalization if the solutions are structured hierarchically. Furthermore, it was suggested that the computational overhead under the proposed hierarchical models is proportional not to the overall complexity, in terms of the number of programs across the solution, but to the number of levels in the hierarchy. This is because not all subcomponents are engaged at every time step, a property stemming directly from the use of the bid-based action selection process and the ensuing problem decomposition. Thus, it has been shown that hierarchical model construction represents a more efficient approach to increasing complexity than using a comparable single-level approach. Alternatively, as suggested by the running time comparisons in Chapter 8, hierarchical model construction can result in increased complexity given the same computational resource limit. In contrast, methods that focus on monolithic solutions, e.g., canonical GP, typically fail to address the issues of scalability and complexity.

   This thesis also argued that when evaluating complexity two factors need to be considered, namely, whether or not additional complexity is warranted and the way in which complexity grows. For example, it was suggested that in contrast to classification, where one would expect simple solutions to suffice in many situations, under the typically more challenging domain of temporal sequence learning increased complexity may be warranted. With respect to the way in which complexity is added, it could be the case that growth in complexity is a side-effect of evolution with little or no benefit in terms of the end-goal. On the other hand, if additional complexity is added in a structured manner, as in the proposed approach, then improved performance can result with modest increases in computational overhead.

Though perhaps not contributions to the greater research community *per se*, the following remarks are still regarded as noteworthy. First, the philosophy in evaluating the proposed approach was to consider a number of factors, where this reflects recent calls for more comprehensive evaluation in ML [83, 167]. For example, solutions in the classification domain were not compared solely based on accuracy, instead, the evaluation included the mcdr measure as well as measures of solution complexity.

Second, the proposed approach was evaluated on a set of real-world problems from the outset, as opposed to, e.g., studies on compositional mechanisms of inheritance which focus on artificial tasks [189]. As such, this thesis assumed a more practical approach to evaluation. Finally, it is noted that the proposed approach is generic in that it can be applied directly to classification and temporal sequence learning, and perhaps with slight modifications, to other problem domains such as regression. This is in contrast to some of the algorithms used in the comparisons, notably, the SVM and AdaBoost frameworks which are tailored towards classification. It may thus very well be the case that slight tweaks to the proposed algorithm guided by the target domain may lead to improved performance. Related to this is the apparent robustness of the proposed approach, i.e., the core parameter settings used across all the experiments was the same. In contrast, an explicit search was performed to find reasonable problem-specific parameter settings for the SVM and NEAT algorithms.

9.4 Future Work

Previous formulations leading up to the version of the SBB algorithm as presented in this thesis have assumed various forms, e.g., [110, 111, 112], so if the past is any indication of the future this suggests that there are many design decisions in the current framework which may be worth investigating further. The following are viewed as the most relevant directions for future work, in terms of modifications to the proposed algorithm as well as other topics of research:

1. The most recent development in the formulation of the proposed approach has been the inclusion of the hierarchical construction process that iteratively redefines the action set in terms of behaviours learned at lower levels. Accordingly, this is the algorithm component that has received the least attention so far, and thus presents the most opportunities for improvement. For example, currently all hosts from one level form the action set at the next level, whereas it may be more efficient to eliminate weak or redundant hosts from consideration before proceeding to the next level. Under the multi-level formulation, it may also be worthwhile to allow hosts at the topmost level to reference hosts at all lower levels as well as the problem-specific actions themselves thus resulting in a more flexible action set. Under temporal sequence learning, instead of evaluating performance with respect to a single goal state, it may be useful to evaluate hosts at lower levels with respect to arbitrary start and stop conditions with the goal

of evolving temporal abstractions, i.e., meta-actions applicable over multiple consecutive time steps [7]. Finally, in this thesis, the same parameters were used at all levels, whereas is may be more appropriate to adjust the parameters according the height of the current level. For example, it may be the case that at lower levels it is better to evolve a larger population consisting of smaller individuals, where this is consistent with the goal of evolving a diverse set of simple behaviours upon which to base subsequent levels.

2. The success of the proposed approach hinges on the emergence of appropriate bidding behaviours in the symbiont population, and so this is viewed as another algorithm component where further study may lead to significant improvements. Specifically, it was established in Chapter 4 that, even with no explicit pressure to do so, the bids may evolve so that there typically exists a discernible margin between the highest bid and the second-highest bid – still, there is room for improvement. For example, a symbiont associated with an action that is a poor choice under the current conditions can still submit a high bid and not adversely affect the host's behaviour as long as it is outbid by a symbiont associated with the optimal action. Placing the over-bidding symbiont in a different context (host) is then likely to have a disruptive effect hindering the search for effective symbiont combinations. One approach to improving bidding behaviour may be to borrow from the XCS classifier system formulation [196] and require a symbiont's bid to accurately reflect the degree to which its action suits the current conditions.

3. Allowing the actions to assume real values may lead to improved performance in domains where the action space is naturally continuous, e.g., the truck reversal problem. In addition, it may also make the approach applicable to regression tasks.

4. Previous work on modular interdependency has focused on optimization problems expertly designed to clearly illustrate the concept, e.g., the hierarchical-if-and-only-if function [189, 191]. The symbionts in the proposed approach can be viewed as subsystems that are structurally modular but which, through the bid-based action selection process, exhibit non-trivial behavioural interdependencies. Although one would like to show that candidate solutions under the proposed approach are a demonstration of modular interdependency under model building, it is not easy to do so. For example, under supervised learning,

one could argue that it is possible to narrow down the number of configurations of a given symbiont that are optimal, regardless of the other symbionts in the same host, to one, i.e., the configuration where the bid is the maximum when the symbiont's action matches the desired output otherwise it is the minimum[2]. Combining multiple symbionts of this form (making sure that all actions are collectively represented) would then result in optimal overall behaviour since the highest bid would always correspond to the desired output, implying that the system is separable. Under temporal sequence learning, on the other hand, it is not clear how to define what an optimal configuration of a symbiont is without the context provided by other symbionts, suggesting that perhaps in this case the system is non-decomposable. The fact that, under the original definition of modular interdependency [189], the same system appears to be both separable and non-decomposable suggests that the definition of modular interdependency needs to be extended under the model building scenario.

5. Finally, the results of Chapter 7, involving the comparison of the proposed approach with NEAT, suggest that the SBB framework may be effective on continuous temporal sequence learning tasks. Recent unpublished results on the Acrobot handstand problem [174] are consistent with this observation, but a more comprehensive evaluation on a cross-section of continuous and discrete problems would help in determining the most appropriate applications of the proposed approach.

[2]It is noted that the definition of modular interdependency [189] is specified in terms of the number of configurations of a given module that satisfy some criterion, with no reference to the way in which one arrives at those configurations.

Bibliography

[1] C. W. Anderson and W. T. Miller. A challenging set of control problems. *Neural Networks for Control*, pages 475–508, 1990.

[2] P. J. Angeline and J. B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 236–241, 1992.

[3] A. Asuncion and D. J. Newman. UCI Machine Learning Repository, 2007.

[4] R. Axelrod. The evolution of strategies in the Iterated Prisoner's Dilemma. In *Genetic Algorithms and Simulated Annealing*, pages 32–41. London: Pitman Publishing, 1987.

[5] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. San Francisco, CA: Morgan Kaufmann, 1998.

[6] 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 International Conference on Artificial Evolution*, 2009.

[7] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[8] A. De Bary. *Die Erscheinung der Symbiose*. Strasburg: Verlag Trubner, 1879.

[9] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[10] E. B. Baum. Toward a model of mind as a laissez-faire economy of idiots extended abstract. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 28–36, 1996.

[11] E. B. Baum. Toward a model of intelligence as an economy of agents. *Machine Learning*, 35:155–185, 1999.

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

[13] E. B. Baum and I. Durdanovic. An evolutionary post production system. *Advances in Learning Classifier Systems*, pages 3–21, 2000.

[14] E. B. Baum and I. Durdanovic. Toward code evolution by artificial economies. *Evolution as Computation*, pages 314–332, 2002.

[15] F. H. Bennett, J. R. Koza, J. Shipman, and O. Stiffelman. Building a parallel computer system for $18,000 that performs a half peta-flop per day. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1484–1490, 1999.

[16] H.-G. Beyer. An alternative explanation for the manner in which Genetic Algorithms operate. *BioSystems*, 41:1–15, 1997.

[17] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 92–100, 1998.

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

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

[20] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. New York, NY: Springer, 2007.

[21] L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

[22] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[23] J. Cartlidge and S. Bullock. Combating coevolutionary disengagement by reducing parasite virulence. *Evolutionary Computation*, 12(3):193–222, 2004.

[24] C.-C. Chang and C.-J. Lin. *LIBSVM: A library for Support Vector Machines (Version 2.9)*, 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[25] K. Chellapilla. Evolving nonlinear controllers for backing up a truck-and-trailer using Evolutionary Programming. In *Proceedings of the Seventh International Conference on Evolutionary Programming*, pages 417–426, 1998.

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

[27] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[28] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[29] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the International Conference on Genetic Algorithms and their Applications*, pages 183–187, 1985.

[30] R. Curry, P. Lichodzijewski, and M. I. Heywood. Scaling Genetic Programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 37(4):1065–1073, 2007.

[31] J. M. Daida, C. S. Grasso, S. A. Stanhope, and S. J. Ross. Symbionticism and complex adaptive systems I: Implications of having symbiosis occur in nature. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 177–186, 1996.

[32] C. Darwin. *On the Origin of Species by Means of Natural Selection*. London, England: John Murray, 1859.

[33] R. Dawkins and J. R. Krebs. Arms races between and within species. In *Royal Society of London, Series B, 205*, pages 489–511, 1979.

[34] E. D. De Jong and J. B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[35] A. Dessi, A. Giani, and A. Starita. An analysis of automatic subroutine discovery in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 996–1001, 1999.

[36] W. F. Doolittle. Uprooting the tree of life. *Scientific American*, (282):90–95, 2000.

[37] M. Dorigo. New perspectives about default hierarchies formation in Learning Classifier Systems. In *Proceedings of the Congress of the Italian Association for Artificial Intelligence on Trends in Artificial Intelligence*, pages 218–227, 1991.

[38] J. Doucette, P. Lichodzijewski, and M. I. Heywood. Evolving coevolutionary classifiers under large attributed spaces. In R. Riolo, U.-M. O'Reilly, and T. McConaghy, editors, *Genetic Programming Theory and Practice VII*, pages 37–54. 2010.

[39] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2007.

[40] N. El-Sourani, S. Hauke, and M. Borschbach. An evolutionary approach for solving the Rubik's Cube incorporating exact methods. In *Applications of Evolutionary Computation*, volume 6024/2010 of *Lecture Notes in Computer Science*, pages 80–89. 2010.

[41] P. G. Espejo, S. Ventura, and F. Herrera. A survey on the application of Genetic Programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(2):121–144, 2010.

[42] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems 2*, pages 524–532, 1990.

[43] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training Support Vector Machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[44] R. Feldt. Generating multiple diverse software versions with Genetic Programming. In *Proceedings of the EUROMICRO Conference, Workshop on Dependable Computing Systems*, pages 387–396, 1998.

[45] S. G. Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Brandeis University, 2004.

[46] S. G. Ficici and J. B. Pollack. Pareto optimality in coevolutionary learning. In *Proceedings of the Sixth European Conference on Advances in Artificial Life*, pages 316–325, 2001.

[47] S. G. Ficici and J. B. Pollack. A game-theoretic memory mechanism for coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 286–297, 2003.

[48] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. Hoboken, NJ: John Wiley & Sons, 2006.

[49] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Chichester, UK: Wiley, 1966.

[50] G. Folino, C. Pizzuti, and G. Spezzano. GP ensembles for large-scale data classification. *IEEE Transactions on Evoluationary Computation*, 10(5):604–616, 2006.

[51] C. M. Fonseca and P. J. Fleming. An overview of Evolutionary Algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[52] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

[53] C. Gagne, M. Sebag, M. Schoenauer, and M. Tomassini. Ensemble learning for free with Evolutionary Algorithms? In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1782–1789, 2007.

[54] J. Gama and P. Brazdil. Cascade generalization. *Machine Learning*, 41(3):315–343, 2000.

[55] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in Genetic Programming. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 312–321, 1994.

[56] S. Geva, J. Sitte, and G. Willshire. A one neuron truck backer-upper. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 850–856, 1992.

[57] D. E. Goldberg and J. Richardson. Genetic Algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 41–49, 1987.

[58] S. M. Gustafson and W. H. Hsu. Layered learning in Genetic Programming for a cooperative robot soccer problem. In *Proceedings of the European Conference on Genetic Programming*, pages 291–301, 2001.

[59] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[60] I. Guyon, S. R. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. In *NIPS*, 2004.

[61] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[62] S. Harmeling, G. Dornhege, D. Tax, F. Meinecke, and K.-R. Muller. From outliers to prototypes: Ordering data. *Neurocomputing*, 69, 2006.

[63] E. H. Hastings, R. K. Guha, and K. O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.

[64] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 2009.

[65] T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright. Evolving a team. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 23–30, 1995.

[66] M. Herdy and G. Patone. Evolution Strategy in action, 10 ES-demonstrations. Technical Report TR-94-05, International Conference on Evolutionary Computation, 1994.

[67] M. I. Heywood and P. Lichodzijewski. Symbiogenesis as a mechanism for building complex adaptive systems: A review. In *Proceedings of the European Workshop on Evolutionary Algorithms and Complex Systems*, pages 51–60, 2010.

[68] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[69] J. L. Hintze and R. D. Nelson. Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.

[70] J. H. Holland. Genetic Algorithms and the optimal allocation of trials. *SIAM Journal of Computing*, 2(2):88–105, 1973.

[71] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* Cambridge, MA: MIT Press, 1986.

[72] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. *Pattern Directed Inference Systems*, 1978.

[73] J. Horn, D. E. Goldberg, and K. Deb. Implicit niching in a Learning Classifier System: Nature's way. *Evolutionary Computation*, 2(1):37–66, 1994.

[74] G. S. Hornby. ALPS: The age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 815–822, 2006.

[75] W. H. Hsu and S. M. Gustafson. Genetic Programming and multi-agent layered learning by reinforcements. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 764–771, 2002.

[76] J. Hu, E. Goodman, K. Seo, Z. Fan, and R. Rosenberg. The hierarchical fair competition (HFC) framework for sustainable Evolutionary Algorithms. *Evolutionary Computation*, 13(2):241–277, 2005.

[77] P. Husbands and F. Mill. Simulated co-evolution as the mechanism for emergent planning and scheduling. In *Proceedings of the International Conference on Genetic Algorithms*, pages 264–270, 1991.

[78] H. Iba. Bagging, boosting and bloating in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1053–1060, 1999.

[79] K. Imamura, T. Soule, R. B. Heckendorn, and J. A. Foster. Behavioral diversity and a probabilistically optimal GP ensemble. *Genetic Programming and Evolvable Machines*, 4(3):235–253, 2003.

[80] H. Ishibuchi, N. Tsukamoto, and Y. Nojima. Evolutionary many-objective optimization: A short review. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2419–2426, 2008.

[81] D. Jackson. Hierarchical Genetic Programming based on test input subsets. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1612–1619, 2007.

[82] D. Jackson and A. P. Gibbons. Layered learning in boolean GP problems. In *Proceedings of the European Conference on Genetic Programming*, pages 148–159, 2007.

[83] N. Japkowicz. Why question machine learning evaluation methods? (An illustrative review of the shortcomings of current methods). Technical Report WS-06-06, AAAI-2006 Workshop on Evaluation Methods for Machine Learning, 2006.

[84] 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.

[85] E. D. De Jong. The MaxSolve algorithm for coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 483–489, 2005.

[86] E. D. De Jong. A monotonic archive for Pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[87] K. A. De Jong. *Evolutionary Computation: A Unified Approach.* Cambridge, MA: MIT Press, 2006.

[88] K. A. De Jong and W. M. Spears. Learning concept classification rules using Genetic Algorithms. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 651–656, 1991.

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

[90] D. Kanevskiy and K. Vorontso. Cooperative coevolutionary ensemble learning. In *Proceedings of the International Conference on Multiple Classifier Systems*, pages 469–478, 2007.

[91] K. E. Kinnear. Alternatives in automatic function definition: A comparison of performance. *Advances in Genetic Programming*, pages 119–141, 1994.

[92] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agarwal. Application of Genetic Programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation*, 4(3):242–258, 2000.

[93] R. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 700–705, 1997.

[94] J. R. Koza. A genetic approach to finding a controller to back up a tractor-trailer truck. In *Proceedings of the 1992 American Control Conference*, pages 2307–2311, 1992.

[95] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[96] J. R. Koza. Architecture-altering operations for evolving the architecture of a multi-part program in Genetic Programming. Technical report, Stanford University, 1994.

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

[98] K. Krawiec, B. Kukawka, and T. Maciejewski. Evolving cascades of voting feature detectors for vehicle detection in satellite imagery. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[99] D. Kunkle and G. Cooperman. Twenty-six moves suffice for Rubik's Cube. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, pages 235–242, 2007.

[100] C. G. Kurland, B. Canback, and O. G. Berg. Horizontal gene transfer: A critical view. *Proceedings of the National Academy of Sciences of the United States of America*, 100(17):9658–9662, 2003.

[101] U. Kutschera and K. J. Niklas. The modern theory of biological evolution: An expanded synthesis. *Naturwissenschaften*, 91(6):255–276, 2004.

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

[103] U. Kutschera and K. J. Niklas. Charles Darwin's Origin of Species, directional selection, and the evolutionary sciences today. *Naturwissenschaften*, 96:1247–1263, 2009.

[104] I. Kwee, M. Hutter, and J. Schmidhuber. Market-based reinforcement learning in partially observable worlds. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 865–873, 2001.

[105] J.-B. Lamarck. La philosophie zoologique. In *Collection 10-18, Union Generale d'Editions, Paris, 1968*. 1809.

[106] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Berlin: Springer-Verlag, 2002.

[107] P. L. Lanzi and R. L. Riolo. Recent trends in Learning Classifier Systems research. *Advances in Evolutionary Computing: Theory and Applications*, pages 955–988, 2003.

[108] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 2010. To appear.

[109] M. Lemczyk and M. I. Heywood. Training binary GP classifiers efficiently: A Pareto-coevolutionary approach. In *Proceedings of the European Conference on Genetic Programming*, pages 229–240, 2007.

[110] P. Lichodzijewski and M. I. Heywood. GP classifier problem decomposition using first-price and second-price auctions. In *Proceedings of the European Conference on Genetic Programming*, pages 137–147, 2007.

[111] P. Lichodzijewski and M. I. Heywood. Coevolutionary bid-based Genetic Programming for problem decomposition in classification. *Genetic Programming and Evolvable Machines*, 9(4):331–365, 2008.

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

[113] P. Lichodzijewski and M. I. Heywood. Binary versus real-valued reward functions under coevolutionary reinforcement learning. In *Proceedings of the 2009 International Conference on Artificial Evolution*, 2009.

[114] P. Lichodzijewski, M. I. Heywood, and A. N. Zincir-Heywood. Cascaded GP models for data mining. In *Proceedings of the IEEE Congress on Evoluationary Computation*, volume 2, pages 2258–2264, 2004.

[115] P. Lichodzijewski, M. I. Heywood, and A. N. Zincir-Heywood. CasGP: Building cascaded hierarchical models using niching. In *Proceedings of the IEEE Congress on Evoluationary Computation*, volume 2, pages 1180–1187, 2005.

[116] T. Lim, W. Loh, and Y. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40:203–228, 2000.

[117] E. Littmann and H. Ritter. Cascade network architectures. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 398–404, 1992.

[118] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404, 1999.

[119] S. Luke and L. Spector. Evolving teamwork and coordination with Genetic Programming. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 1500–156, 1996.

[120] L. Margulis. *Symbiogenesis and Symbionticism*, chapter 1, pages 1–14. Symbiosis as a Source of Evolutionary Innovation. MIT Press, 1991.

[121] J. Maynard Smith. *A Darwinian View of Symbiosis*, chapter 3, pages 26–39. Symbiosis as a Source of Evolutionary Innovation. MIT Press, 1991.

[122] T. McConaghy and G. G. E. Gielen. Template-free symbolic performance modeling of analog circuits via canonical-form functions and Genetic Programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(8):1162–1175, 2009.

[123] A. R. McIntyre. *Novelty Detection + Coevolution = Automatic Problem Decomposition: A Framework for Scalable Genetic Programming Classifiers*. PhD thesis, Dalhousie University, 2007.

[124] A. R. McIntyre and M. I. Heywood. On multi-class classification by way of niching. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 581–592, 2004.

[125] A. R. McIntyre and M. I. Heywood. Cooperative problem decomposition in Pareto competitive classifier models of coevolution. In *Proceedings of the European Conference on Genetic Programming*, pages 289–300, 2008.

[126] C. Mereschkowsky. Uber natur und ursprung der chromatophoren im pflanzenreiche. 1905.

[127] B. L. Miller and M. J. Shaw. Genetic Algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 786–791, 1996.

[128] M. Mitchell. *Complexity: A Guided Tour*. New York, NY: Oxford University Press, 2009.

[129] T. Mitchell. *Machine Learning*. Boston, MA: WCB/McGraw-Hill, 1997.

[130] D. J. Montana. Strongly typed Genetic Programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[131] J. H. Moore, N. Barney, C.-T. Tsai, F.-T. Chiang, J. Gui, and B. C. White. Symbolic modeling of epistasis. *Human Heredity*, 63(2):120–133, 2007.

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

[133] D. E. Moriarty and R. Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[134] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary Algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[135] P. Nardon and A.-M. Grenier. *Serial Endosymbiosis Theory and Weevil Evolution: The Role of Symbiosis*, chapter 12, pages 153–167. Symbiosis as a Source of Evolutionary Innovation. MIT Press, 1991.

[136] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 357–363, 1989.

[137] D. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3):18–23, 1990.

[138] S. G. Nitschke. Neuro-evolution methods for designing emergent specialization. In *Proceedings of the European Conference on Advances in Artificial Life*, pages 1120–1130, 2007.

[139] J. Noble and R. A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 493–500, 2001.

[140] P. Nordin and W. Banzhaf. Complexity compression and evolution. In *Proceedings of the International Conference on Genetic Algorithms*, pages 310–317, 1995.

[141] P. Nordin and W. Banzhaf. Evolving turing-complete programs for a register machine with self-modifying code. In *Proceedings of the International Conference on Genetic Algorithms*, pages 318–325, 1995.

[142] P. Nordin, F. Francone, and W. Banzhaf. *Explicitly defined introns and destructive crossover in Genetic Programming*, chapter 6, pages 111–134. Advances in Genetic Programming 2. MIT Press, 1996.

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

[144] G. Paris, D. Robiliard, and C. Ronlupt. Applying boosting techniques to Genetic Programming. In *Proceedings of the International Conference on Artificial Evolution*, pages 267–278, 2001.

[145] T. K. Paul, Y. Hasegawa, and H. Iba. Classification of gene expression data by majority voting Genetic Programming classifier. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2521–2528, 2006.

[146] M. D. Platel, M. Chami, M. Clergue, and P. Collard. Teams of genetic predictors for inverse problem solving. In *Proceedings of the European Conference on Genetic Programming*, pages 341–350, 2005.

[147] E. Popovici, A. Bucci, P. Wiegand, and E. D. De Jong. Coevolutionary principles. In *Handbook of Neural Computing*. 2010.

[148] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 249–257, 1994.

[149] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[150] K. De Queiroz. Species concepts and species delimitation. *Systematic Biology*, 56(6):879–886, 2007.

[151] R. Quinlan. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[152] I. Rechenberg. *Evolutionsstrategie - Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Stuttgart, Germany: Frommann-Holzboog, 1973.

[153] A. Riid and E. Rustern. Fuzzy logic in control: Truck backer-upper problem revisited. In *Proceedings of the International Conference on Fuzzy Systems*, pages 513–516, 2001.

[154] R. L. Riolo. Bucket brigade performance: II. Default hierarchies. In *Proceedings of the International Conference on Genetic Algorithms*, pages 196–201, 1987.

[155] J. P. Rosca and D. H. Ballard. Learning by adapting representations in Genetic Programming. In *Proceedings of the IEEE Congress on Computational Intelligence*, pages 407–412, 1994.

[156] J. P. Rosca and D. H. Ballard. Discovery of subroutines in Genetic Programming. *Advances in Genetic Programming: Volume 2*, pages 177–201, 1996.

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

[158] J. Rubini, R. B. Heckendorn, and T. Soule. Evolution of team composition in multi-agent systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1067–1074, 2009.

[159] L. Sagan. On the origin of mitosing cells. *Journal of Theoretical Biology*, 14(3):255–274, 1967.

[160] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406, 1999.

[161] M. Schoenauer and E. Ronald. Neuro-genetic truck backer-upper controller. In *Proceedings of the IEEE Congress on Computational Intelligence*, volume 2, pages 720–723, 1994.

[162] C. Seiffert, T. M. Khoshgoftaar, J. van Hulse, and A. Napolitano. Resampling or reweighting: A comparison of boosting implementations. In *Proceedings of the Twentieth IEEE International Conference on Tools with Artificial Intelligence*, pages 445–451, 2008.

[163] H. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962.

[164] D. Singmaster. *Notes on Rubik's 'Magic Cube'*. Penguin Books, 1981.

[165] B. F. Smets and T. Barkay. Horizontal gene transfer: Perspectives at a crossroads of scientific disciplines. *Nature Reviews Microbiology*, 3:675–678, 2005.

[166] S. F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 422–425, 1983.

[167] M. Sokolova, N. Japkowicz, and S. Szpakowicz. Beyond accuracy, F-score and ROC: A family of discriminant measures for performance evaluation. In *Proceedings of the ACS Australian Joint Conference on Artificial Intelligence*, pages 1015–1021, 2006.

[168] D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training Genetic Programming on half a million patterns: An example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, 2005.

[169] T. Soule. Voting teams: A cooperative approach to non-typical problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 916–922, 1999.

[170] T. Soule. Heterogeneity and specialization in evolving teams. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 778–785, 2000.

[171] T. Soule. Cooperative evolution on the intertwined spirals problem. In *Proceedings of the European Conference on Genetic Programming*, pages 434–442, 2003.

[172] T. Soule, R. B. Heckendorn, B. Dyre, and R. Lew. Ensemble classifiers: AdaBoost and Orthogonal Evolution of Teams. In R. Riolo, T. McConaghy, and E. Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, pages 55–70. 2010.

[173] T. Soule and P. Komireddy. Orthogonal Evolution of Teams: A class of algorithms for evolving teams with inversely correlated errors. In R. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, pages 79–95, 2007.

[174] W. M. Spong. Swing up control of the acrobot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2356–2361, 1994.

[175] K. Stanley. *NEAT C++*, 2001. `http://nn.cs.utexas.edu`.

[176] K. Stanley. *The Neuroevolution of Augmenting Topologies (NEAT) Users Page*, 2010. `http://www.cs.ucf.edu/~kstanley/neat.html`.

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

[178] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[179] P. Stone. Learning and multi-agent reasoning for autonomous agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 13–30, 2007.

[180] P. Stone and M. M. Veloso. Layered learning. In *Proceedings of the European Conference on Machine Learning*, pages 369–381, 2000.

[181] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1999.

[182] M. E. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, 2006.

[183] A. Teller. *Evolving programmers: The co-evolution of intelligent recombination operators*, chapter 3, pages 45–68. Advances in Genetic Programming. MIT Press, 1996.

[184] A. Teller and M. Veloso. PADO: A new learning architecture for object recognition. In *Symbolic Visual Learning*, pages 81–116. 1996.

[185] K. Tumer and D. Wolpert. A survey of collectives. *Collectives and the Design of Complex Systems*, pages 1–42, 2004.

[186] E. J. Vladislavleva, G. F. Smits, and D. den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.

[187] A. R. Wallace. *Darwinism: An Exposition of the Theory of Natural Selection with Some of its Application*. London, UK: MacMillan, 1889.

[188] I. E. Wallin. *Symbionticism and the Origin of Species*. Baltimore, US: Williams and Wilkins, 1927.

[189] R. A. Watson. *Compositional Evolution: Interdisciplinary Investigations in Evolvability, Modularity, and Symbiosis*. PhD thesis, Brandeis University, 2002.

[190] R. A. Watson and J. B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 702–709, 2001.

[191] R. A. Watson and J. B. Pollack. Modular interdependency in complex dynamical systems. *Artificial Life*, 11(4):445–458, 2005.

[192] A. Weismann. *Das Keimplasma: Eine Theorie der Vererbung*. Germany: Gustav Fischer-Verlag, Jena, 1892.

[193] G. M. Weiss and F. Provost. Learning when training data are costly: The effects of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[194] S. Whiteson and P. Stone. Concurrent layered learning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 193–200, 2003.

[195] R. P. Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, 1996.

[196] S. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[197] S. W. Wilson. Generalization in the XCS classifier system. In *Proceedings of the Third Annual Conference on Genetic Programming*, pages 665–674, 1998.

[198] S. M. Winkler, M. Affenzeller, and S. Wagner. Using enhanced Genetic Programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10(2):111–140, 2009.