TOWARDS COEVOLUTIONARY GENETIC PROGRAMMING WITH PARETO ARCHIVING UNDER STREAMING DATA

by

Aaron Atwater

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

at

Dalhousie University
Halifax, Nova Scotia
August 2013

Dedicated to NIMSLab, past and present

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abstract

Classification under streaming data constraints implies that training must be performed continuously, can only access individual exemplars for a short time after they arrive, must adapt to dynamic behaviour over time, and must be able to retrieve a current classifier at any time. A coevolutionary genetic programming framework is adapted to operate in non-stationary streaming data environments. Methods to generate synthetic datasets for benchmarking streaming classification algorithms are introduced, and the proposed framework is evaluated against them. The use of Pareto archiving is evaluated as a mechanism for retaining access to a limited number of useful exemplars throughout training, and several fitness sharing heuristics for archiving are evaluated. Fitness sharing alone is found to be most effective under streams with continuous (incremental) changes, while the addition of an aging heuristic is preferred when the stream has stepwise changes. Tapped delay lines are explored as a method for explicitly incorporating sequence context in cyclical data streams, and their use in combination with the aging heuristic suggests a promising route forward.

Acknowledgements

First and foremost I must thank my supervisor, Dr. Malcolm Heywood, for his guidance and mentorship throughout the past two years. Without his support – and seemingly infinite patience – this thesis would probably not exist. I would also like to thank Dr. Nur Zincir-Heywood, my undergraduate honours supervisor, for her support, and both the Drs. Heywood for providing the great working environment that is the NIMS Lab.

Thanks are also due to my readers Dr. Stan Matwin and Dr. Andy McIntyre for being members of my committee and for their thoughtful questions and insights on my thesis.

Thank you to my parents for their help, support, and confidence throughout the years, and also to my dear friend Amber for helping to keep me sane.

This work was supported in part by funding from the National Sciences and Engineering Research Council of Canada.

Chapter 1

Introduction

Streaming data applications represent a requirement for online as opposed to offline evolution. Unlike the classification task as traditionally assumed, a streaming data constraint implies that the learning algorithm cannot (directly) adopt a batch mode of operation. Specifically, **batch** or offline frameworks for classification assume that any exemplar from the training partition can be accessed at uniform cost. Two forms of fitness evaluation fall under the batch model: either evaluate over all training instances at each generation, or over a subset of training instances as sampled without constraint from the entire training repository at each generation. Conversely, **online** learning as applied to streaming data implies that only when training data explicitly lies within a current 'window' or 'block' of sequential records can it be utilized for training purposes. The content of the block is updated incrementally – as in a sliding window [1] – or under the assumption of non-overlapping updates in which the entire block content is updated [46]. In either case a finite number of records can only ever be accessed, implying that as many records are replaced as gained at each 'location' of the window / block. Learning algorithms applied under an online context are therefore faced with an additional issue: which if any of the training instances should be carried over between blocks of streaming data? Assuming that *all* training instances can be archived once encountered is not feasible in general, i.e., there is usually a requirement to in some way 'keep up' with the rate at which new data appears in the stream. Finally, we note that aside from the windowing restriction, streaming data is generally non-stationary (note that this is also referred to as concept drift, and the two terms are used interchangeably throughout this work). This means that the underlying (unknown) process responsible for 'creating' the data changes over time. This is not generally the case in offline data sets as it would result in contradictions between individual exemplars. Conversely, from a streaming data perspective the same set of measurements might be associated with different outcomes depending on their

'position' (or timestamp) within the stream.

Streaming applications appear frequently in financial environments (e.g., [10]), computer network applications (e.g., [41]) or 'big data' applications (e.g., [3]). The interface between data stream and evolutionary algorithm typically takes the form of a sliding window, an assumption that will be retained throughout this thesis. At each training epoch (generation) only data within the window can be accessed and used for fitness (performance) evaluation. In this work, evolution under a form of competitive coevolution will be assumed. Adopting such an approach enables us to address the task of identifying *what* to learn from at the same time as attempting to construct suitable classification models cf. genetic programming (GP) individuals. Pareto archiving (Section 3.2) will represent the specific formulation for competitive coevolution adopted. The implication of this is that a formal framework exists for identifying and retaining the (hopefully) few instances from the stream that are most useful in promoting the identification of the best GP individuals.

Within the above streaming data context the interest of this study lies in measuring the potential contribution made by competitive coevolutionary Pareto archiving. Such a scheme provides the capability to identify training instances that should be retained (archived) for longer than the current sample of instances provided by a sliding window. Fitness evaluation is only ever conducted over a subset of training instances as defined by the sliding window content and Pareto archive. This is somewhat different to previous research, as we do not explicitly assume that the current content of a sliding window is sufficient to support the identification of appropriate models (e.g., as in [11]) or that coevolving the size of the sliding window is sufficient for retaining useful training instances (e.g., [44]).

A further interest of this research is to characterize the suitability of heuristics used to maintain finite archiving constraints which are necessary to minimize computational overheads associated with Pareto archiving. Such heuristics have a direct impact on the capability of the system to operate within non-stationary environments. In particular, we are interested in quantifying the importance attributed to heuristics for diversity versus age. With this in mind, two artificial data generators will be employed to create multi-class classification benchmarks with stepped versus continuous variation of the underlying data generating process (Section 4.2), and subsequently

two methods of extending these processes to represent periodic or recurring variation (Section 4.3).

Chapter 2 will summarize related research in dynamic environments, as well as the approach to Pareto archiving and team based task decomposition in GP. Chapter 3 will discuss the components of the GP framework used in this work that support the adaption of evolutionary computation to streaming data environments, and define the modifications made to permit the framework to interact with a non-stationary stream of data. Chapter 4 outlines both the stationary and non-stationary datasets used or generated for benchmarking our proposed algorithmic modifications. Experimental methodology and results are presented in Chapter 5. The paper concludes with a discussion and an identification of future research in Section 6.

Chapter 2

Background

This chapter reviews previous work that relates to the major building blocks of the experiments conducted herein. Specifically, a short review is provided of the foundational work leading up to the principles used in the underlying GP framework – competitive coevolution, pareto archiving, symbiosis – as well as the basis for the deployment of GP in dynamic environments. The reviews are not intended to be exhaustive, but instead provide the necessary 'heads up' to the approach specifically adopted in the research described by this thesis.

2.1 Competitive Coevolution and Pareto Archiving

Pareto archiving was conceived as a framework by which a coevolutionary 'arms race' may be conducted between members of a 'learner' population and a population of 'tests'.[1] Thus, nontrivial tasks consist of many more task configurations than can possibly encountered for the purposes of composing the content of a training partition. The insight of competitive coevolution is to instead focus the 'test' population on finding the subset of training configurations that are most informative for developing the progress of the learners. Initial attempts at doing achieving this tended to result in rewarding the individuals of the test population for 'beating' the learners [19] cf., identifying the most difficult training scenarios given the current capability of the learners. However, it was shown that adopting such a framework generally results in disengagement [7]. There are at least two factors behind this: the learners are very weak thus take much longer to identify plausible solutions, and; rewarding tests for beating learners results in a loss of a meaningful learning 'gradient' from which to direct credit assignment.

To date, two generic approaches for addressing this have been proposed: Pareto

[1] In this work learner will be synonymous with a GP individual and test will be synonymous with an exemplar (from the stream).

archiving e.g., [14, 31, 9] and host parasite models incorporating parasite virulence e.g., [7, 6]. In this work we focus on the former Pareto archiving framework, in part due to the considerable experience gained in utilizing the scheme with various forms of GP, albeit as previously deployed as a mechanism for decoupling fitness evaluation from the cardinality of the original dataset under an offline batch interface to the data set [23, 24, 25, 28, 29, 12]. The objective of this thesis is to broaden the knowledge of the properties of Pareto archiving as applied to the specific characteristics of the streaming data task. There are therefore two specific issues of interest:

- Does Pareto archiving provide a useful memory mechanism for retaining specific GP individuals beyond the temporal horizon subscribed by the exemplars associated with the current sliding window interface to the stream;
- Assuming that it is also important to maintain the throughput of a classifier applied to streaming data, what biases might be most appropriate for maintaining a finite archive size under different forms of non-stationary process?

Providing some insight in to these questions represents the underlying goal of this thesis.

2.2 Genetic programming in dynamic environments

There have been several arguments made advocating for the ability of evolutionary methods to operate in dynamic environments. The case for maintaining a set (population) of many diverse models for detecting behavioural change within dynamic environments has been made explicit in [30]. Similarly, maintaining a diverse set of solutions at all times throughout the duration of an incoming stream of data makes it more likely that a 'good enough' solution may exist within this set at any given moment to allow for the continued feasibility and engagement of the population as a whole [30]. A recent survey article on open issues in genetic programming re-emphasized the general appropriateness and relevance of GP to dynamic environments (Section 2.3 in [33]). However, the study also observed that although there has been some success in applying GP solutions to specific application domains which share some of the properties of dynamic environments (e.g., [11]), there has been minimal analysis of the behaviour of how GP operates in such environments in general.

The task considered in this work represents a base case in a wide range of potential issues associated with machine learning as applied to the analysis of data streams. There are a number of traditional machine learning outcomes that are desirable in the context of streaming data: change detection, online data description, frequent pattern set identification, clustering, classification [1]; of these, this work is focused on the case of classification.

Within the specific context of creating streaming classifiers, many potential disadvantages remain in the algorithms proposed to date [1]. For example, previous algorithms are often limited to binary tasks or require on the order of millions of training instances in order to build a single classifier (whereas we may be interested in obtaining a classifier before significant portions of the stream have elapsed, before any behavioural change has begun to occur). Moreover, the accuracy of the online stream classifier is often lower than that of the offline batch equivalent. It is also desirable for a stream classifier to address the issue of *concept drift*, reflecting the fact that the underlying process responsible for the creation of incoming data is generally non-stationary. There are currently two algorithm-independent approaches to dealing with this phenomenon:[2] they either deploy a change detection metric to the original stream data (e.g., [1]), or – particularly in the case of ensemble style classifiers – detect when some change appears in the generated classifier's behaviour (e.g., [46]).

More generally, we note that unlike static or stationary tasks, dynamic or non-stationary tasks require convergence (of the GP learner population) to be pursued for only a finite amount of time. In the case of an underlying task being dynamic, convergence to a single solution potentially compromises the ability to react to change. It has been proposed by [10] that previous approaches investigated for applying evolutionary computation to dynamic environments fall into one of five forms: memory, diversity, multi-populations, problem decomposition and evolvability. The remainder of this chapter provides a short review of the contributions made and, where necessary, highlights their relation to this work.

Memory is defined as the capability to return to a previously evolved solution. This implicitly assumes that the task is in some way periodic in order for the memory

[2]Approaches specific to a particular machine learning representation have been proposed. For example, Bayesian or support vector approaches for addressing concept drift are frequently discussed [35, 42].

mechanism to be useful. Moreover, there must be a tradeoff between the amount of resources available for the memory mechanism and the amount (of resources) made available for supporting other properties, such as diversity. In this work we are interested in the utility of Pareto archiving as a candidate memory mechanism, with resource tradeoffs being explicitly controlled by the user via enforcement of finite archive sizes. Pareto archiving provides a much more formal basis for retaining both learners (GP individuals) and points (data 'distinguishing' between non-dominated learners). We also investigate the use of tapped delay lines, defined in Section 3.4, as a method of allowing individual learners to explicitly acknowledge previous states and return to them of its own volition. Indeed, such tapped delay lines provide a mechanism for constructing **temporal features** and are therefore potentially useful for facilitating change detection. Many instances of evolutionary algorithms applied to temporal sequence learning tasks therefore either explicitly incorporate pre-configured tapped delay lines (e.g., [41]) or explicitly support the evolution of properties of temporal features such as moving averages (e.g., [44, 27]).

Diversity as in mechanisms for supporting multiple species within a population. The most typically assumed mechanism is to vary the rate of mutation or introduce a fixed number of entirely new individuals at each generation [17]. Niching (speciation) has been considered to resist favouring the same individuals [8]. Likewise, solution age has been proposed to bias the retention of the 'middle aged' as opposed to the old or new [16]. [30] introduced 'sentinel' individuals as those which are uniformly distributed through the representation space. Such individuals act as a diverse source of genotypic information for seeding the population on a continuous basis. This approach however, is unique to the case of representations taking the form of a 'point' in an ordered space, as in a genetic algorithm, whereas the representation assumed in this work, GP as a whole, does not naturally support such an ordering. The benchmarking study conducted in Sections 5.2 and 5.3 will revisit and compare the advantages and disadvantages of assuming speciation and age heuristics in conjunction with Pareto archiving.

Multi-populations associate different populations with different aspects of the search space. The basic assumption in this case is that genetic drift associated with the smaller subpopulations will encourage the investigation of new areas of the search

space. Finding a suitable heuristic for controlling the exploration–exploitation trade-off controlling the interaction between the multiple populations remains an ongoing research issue. The research discussed in this these will therefore assume that niching (which is to say, diversity through fitness sharing) within a single population will be sufficient to maintain learners associated with the "different aspects of the search space".

Problem decomposition is taken to have been articulated first by Simon's watchmaker parable [40] in which the capacity to configure quickly is attributed to having appropriate 'modules' available. From the perspective of GP, task decomposition might be most synonymous with support for modularity, run–time libraries and teaming. Studies have shown the utility of each relative to static task domains, and the teaming metaphor used by the underlying GP framework in this work has been previously evaluated within static environments in [26].

Evolvability implies that a representation is assumed which is capable of supporting the identification of fitter parents in changing environments. Geno- to phenotypic mappings have been proposed in this respect e.g., [37]. In this work we equate evolvability with the degree to which modularity is supported, and a symbiotic co-evolutionary framework for teaming in GP specifically, as summarized below.

2.3 Symbiotic Bid-based GP

The specific form of GP assumed in this thesis takes the form of the symbiotic bid-based GP framework (SBB) [25, 12]. Such a framework provides several significant advantages over the monolithic approaches to GP popularized by Koza's tree structured GP, as follows:

- Teaming: Rather than assume that a single program is optimal for comprising a solution, a candidate solution is actually a 'team' of cooperating programs. The number of programs cooperating is not pre-specified and evolves over the course of the evolutionary cycle (training). To achieve this a symbiotic two population framework is assumed. A program population consists of the programs. Each program is executed (w.r.t. the current exemplar) and the output considered to represent a *bid*. Each program also has a single *scalar* outcome

– say, as assigned from the set of available class labels. Conversely, the team population consists of individuals that merely index some subset of the current content of the program population. Individuals from the team population assume a variable length representation, thus no assumptions are made regarding what constitutes optimal team membership. Fitness (performance) is only ever evaluated for the team population, by executing all the programs associated with a particular team. Only the program with the largest bid 'wins' the right to suggest its corresponding scalar outcome (class label) for each exemplar, thus a form of cooperative task decomposition. Given that there is no explicit measure of fitness at the program population, programs only 'die' when they receive no indexes from the team population, resulting in a variable size program population. Moreover, the relationship between the two populations is denoted symbiotic coevolution, as opposed to merely cooperative coevolution (in which case fitness of each cooperating population is known). In this work the combination of program and team population is considered to represent the 'learner' or 'GP individual'. Thus, unlike monolithic GP representations, it is now trivial to address the multi-class classification problem.

- Fitness sharing: Left uninhibited, the fittest member of any population will get to reproduce indefinitely, effectively filling the population with different versions of itself. This can result in the cherry picker effect, where a task is described by exemplars that are 'easy' to label cases (the cherries) and difficult to label cases results in a GP population 'converging' on a solution to the cherries before the population can evolve to address the difficult to label exemplars. Fitness sharing is a simple discounting scheme adopted in which the value that a correct classification is worth is discounted by the number of other GP individuals in the population that also correctly label that exemplar [38]. This property will be particularly important for the streaming data scenario.

- Representation: Multiple GP representations have been proposed since the original tree structured representation of Koza [22]. Specific examples including grammars [32], linear [4], and graphs [43]. Although any representation could

be assumed for individuals in the program population, the linear representation will be assumed here. Such a representation provides various advantages over the original tree structure representation, not least that it is much easier to identify redundant code *during* fitness evaluation, thus reducing the cost of fitness evaluation itself [4].

Extensive benchmarking studies have demonstrated that adopting the above properties in combination provides an extremely flexible framework for task decomposition and therefore learning from the solutions post training e.g., [26].

Chapter 3

Streaming Data Interfaces

This chapter introduces several concepts important to the experimental constructions used in Chapter 5. Section 3.1 explains the symbiotic coevolution principle employed by SBB, and Section 3.2 elaborates further on the Pareto archiving implementation and fitness sharing heuristics it uses. Section 3.3 then explains the sliding window concept by which the Pareto archive is permitted to interface with a data stream, and Section 3.4 provides the definition of the tapped delay lines used to provide explicit memory for the experimental configurations in Section 5.3.

3.1 Symbiotic Coevolution

Pareto archiving is employed as the 'memory mechanism' by which the uniqueness of GP individuals (learners) is formally established. Potentially, Pareto archiving provides a scheme for identifying a subset of learners as being non-dominated relative to the rest of the population. To do so, training exemplars represent 'objectives' and are rewarded for distinguishing between the capability of different learners. Such a model is of interest in a streaming context, as a basis is then provided for identifying a small number of data instances for retention beyond the immediate content of the sliding window i.e., interface to the stream (Section 3.3). Hereafter, the specific form assumed for GP will be the Symbiotic bid-based (SBB) framework for cooperatively evolving GP teams [25, 12]. Such a framework supports task decomposition and multi-class classification from a single evolutionary cycle and is therefore potentially capable of supporting multiple factors significant to dynamic environments (as reviewed in Chapter 2).

Symbiotic bid-based GP as applied to classification utilizes two explicit populations: program and team (or symbiont and host), which are coevolved through symbiosis [25, 12]. The program, or symbiont, population makes use of a bid-based GP representation [24] which defines each GP individual in terms of a tuple $\langle c, b \rangle$,

where c declares a scalar class label selected from the set of labels associated with the task domain ($c \in C$), and b is the program that evolves a context for deploying its class label. The team (host) population identifies combinations of symbionts that attempt to coexist in a coevolutionary relationship. Members of the team population assume a variable length representation, wherein the number of symbionts per team are adapted as part of the evolutionary cycle. Indeed, [3] noted that hosts were first composed from symbionts representing the most frequent classes and only later were symbionts added representing the less frequent classes. Fitness evaluation is only ever performed at the 'level' of the hosts; thus, hosts represent the 'learner' in the above discussion of Pareto dominance (Section 3.2).

Evaluation of a host, $l_i \in L^t$, is repeated for each training instance as identified by the point population at generation t, $p_k \in P^t$. For all symbionts a member of the host, $s_j \in l_i$, execute their programs w.r.t. point p_k. The host identifies the symbiont with maximum output (the winning bid) or $s^* = arg_{s_j \in l_i} max[s_j.b]$. The winning symbiont gains the right to suggest the class label for the current point, or

$$G(l_i, p_k) = \begin{cases} 1 & if \text{ IF } s^*.c = p_k.t \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where $p_k.t$ is the target label for point p_k. Naturally, $G(l_i, p_k)$ is the reward function referred to in Section 3.2.

Algorithm 1 summarizes the breeder model of evolution defining the stepwise application of SBB to a data stream [3]. Initialization provides initial sliding window content, S^t from the data stream, τ, and then samples from S^t until $P - P_{gap}$ points have been selected (Section 3.3). Initialization of the host and symbiont population follows the original SBB algorithm [25, 12]. On entering the main loop the remaining P_{gap} and L_{gap} points and teams are initialized. The point population assumes the same stochastic sampled process as used during the initialization step. Additional L_{gap} hosts are added through application of the SBB variation operators. Variation operators include crossover between hosts and symbiont mutation. The latter implies symbionts are first cloned, with the content varied through appropriate GP mutation operators [25, 12].

The 'Evaluate' function (line 11) resolves values for $G(l_i, p_k)$. Once all hosts have

Algorithm 1 Overview of the SBB training algorithm as applied to streaming data τ. S^t denotes the set of training instances associated with the sliding window at generation t; τ denotes the streaming data; $|P|$ and P^t are the size and content of the point population respectively; $|L|$ and L^t are the size and content of the host population respectively;.

```
1: procedure TRAIN
2:     t = 0                                              ▷ Initialization.
3:     S^t = INITSTREAM(τ(t))
4:     P^t = INITPOINTS(P, S^t)
5:     (L^t, L^t) = INITTEAMS(M)
6:     while t ≤ t_max do
7:         P^t = GENPOINTS(P^t, S^t)                      ▷ Add 'gap' size points
8:         (L^t) = GENTEAMS(L^t)                          ▷ ... and hosts
9:         for all l_i ∈ L^t do
10:            for all p_k ∈ P^t do
11:                EVALUATE(l_i, p_k)                     ▷ Evaluate fitness
12:            end for
13:        end for
14:        S^{t+1} = SHIFTSTREAM(τ(t + 1))                ▷ Resample
15:        P^{t+1} = SELPOINTS(P^t)
16:        (M^{t+1}) = SELTEAMS(L^t)
17:        t = t + 1
18:    end while
19:    return BEST(L^t, P^t)
20: end procedure
```

been evaluated on all points then the location of the sliding window is incremented relative to the data stream S^t (line 14). Point replacement is performed under the Pareto archiving–fitness sharing heuristic of Section 3.2 (line 15). Likewise, a fixed number of hosts, L_{gap}, are removed at each generation (line 16) resulting in members of the host population being either non-dominated (Pareto archive), $\mathcal{F}(L^t)$, or dominated, $\mathcal{D}(L^t)$. Thus, hosts are identified for replacement. Should symbionts *no longer* receive any host indexes then this is taken to imply that they were only associated with the worst L_{gap} hosts and they therefore 'die'. The symbiont population size is therefore free to float.

Finally, the role of function 'Best' (line 19) is to return the champion host for evaluation against the test partition. Given the streaming context – or continuous evolution against the stream – all hosts currently in the Pareto archive are evaluated against the current content of the point population. The assumption being that if the Pareto archiving strategy is effective, then the points remaining in the point population should be the most appropriate for prioritizing the champion host. The metric assumed for champion identification is the average class-wise detection rate or:

$$avg.DR = \frac{1}{|C|} \sum_{c \in C} DR_c(l_i) \tag{3.2}$$

where C is the set of class labels associated with this task and $DR_c(l_i)$ is the detection rate (that is, # of true positives for class C $\div$ # of representatives of class C) of host l_i relative to class c. Readers are referred to [25, 12] for additional SBB details e.g., instruction set and variation operators.

3.2 Pareto Archiving

A two stage process is assumed consisting of Pareto dominance ranking and fitness sharing,[1] the latter representing a heuristic for promoting diversity when enforcing finite archiving constraints [38].

Pareto dominance ranking: The members of the point population, p_k are taken to represent 'objectives'. Fitness is only ever evaluated against the content of

[1]For a tutorial on Pareto archiving as applied to GP see [23].

the point population. This decouples fitness evaluation from the cardinality of a data set. From a streaming data perspective P_{gap} points are replaced at each generation with a sample of new points taken from the current sliding window location [3]. Such objectives / points are used to distinguish between the capability of different GP learners under a pairwise test for Pareto dominance, or

$$\begin{aligned} &\forall p_k \in P : G(l_i, p_k) \geq G(l_j, p_k) \\ &\text{AND } \exists p_k \in P : G(l_i, p_k) > G(l_j, p_k) \end{aligned} \tag{3.3}$$

where P is the point population; l_i and l_j are two individuals (cf., learners) from the GP population currently under evaluation, and; $G(\cdot, \cdot)$ is the task specific reward function returning 0 on an incorrect classification and 1 on a correct classification (see Eqn (3.1) above).

Eqn (3.3) implies that learner l_i dominates learner l_j *iff* it performs as well on every point and better on at least one point. For a total of $|L|$ learners there are a total of $|L|^2 - |L|$ learner comparisons [9].[2] Moreover, for point p_k a comparison between learner l_i and l_j has the form of a distinction vector:

$$d_k[L \cdot i + j] = \begin{cases} 1 & if G(l_i, p_k) > G(l_j, p_k) \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Thus, when Eqn (3.4) returns a value of 1 then a *distinction* is said to have been made [14]. Points are penalized for defeating all learners or when it is defeated by all learners i.e., no distinctions result. Such points would therefore tend to be prioritized for removing from the point population. The fittest learners / GP individuals are naturally those promoted by the Pareto dominance expression – Eqn (3.3). One drawback of the Pareto approach is that as the number of objectives (points) increases, then comparatively weak overlapping behaviours may satisfy the Pareto archiving criteria [14, 31, 28]. SBB therefore adopts the following fitness sharing heuristic with the motivation of maintaining diversity in the points learners solve. This point will be revisited in Section 5.2 when we consider alternative heuristics that might be more appropriate to a data streaming context.

[2]This is distinct from the number of fitness evaluations per generation, with fitness evaluation being a more costly process than establishing the comparison vector.

Fitness sharing: The reward function $G(l_i, p_k)$ establishes a vector of distinctions (Eqn. (3.4)) for each point. The point population can now be divided into two sets: those in the non-dominated front (the archive), $\mathcal{F}(P^t)$, versus those that are not, $\mathcal{D}(P^t)$. Naturally, the decision to limit the number of archives to two is motivated by a desire to balance the computational cost of archive construction against maintaining monotonic progress to an 'ideal' training trajectory [9].

Up to P_{gap} points are replaced at each generation from the current stream window using the process of Section 3.3 [3]. Points are targeted for replacement under two basic conditions [25, 12]:

- If $|\mathcal{F}(P^t)| \leq |P| - P_{gap}$ THEN: stochastically select points for replacement from $\mathcal{D}(P^t)$ alone.
- If $|\mathcal{F}(P^t)| > |P| - P_{gap}$ THEN: all the dominated points are replaced, plus some subset $P_{gap} - |\mathcal{D}(P^t)|$ of the Pareto archive. A fitness sharing heuristic is assumed for weighting members of the Pareto archive such that the more unique the distinction the greater the weight. Thus, relative to distinction vector d_k of point p_k, fitness sharing is defined as:

$$\sum_i \frac{d_k[i]}{1 + N_i} \tag{3.5}$$

  where i indexes all distinction vector entries (Eqn (3.4)), and N_i counts the number of points in $\mathcal{F}(P^t)$ that make the same distinction.

3.3 Sliding Windows

The streaming data assumption places constraints on how training data can be accessed. Specifically, we assume a sliding window representation (Chapter 2). From the perspective of a population of GP classifiers under evolution this means that new training instances become available *incrementally* at each generation. A point population, P, is used to define the specific sample of training instances over which fitness evaluation is performed at generation, t. Likewise, at each generation some subset of the point population is replaced by training instances currently available from the

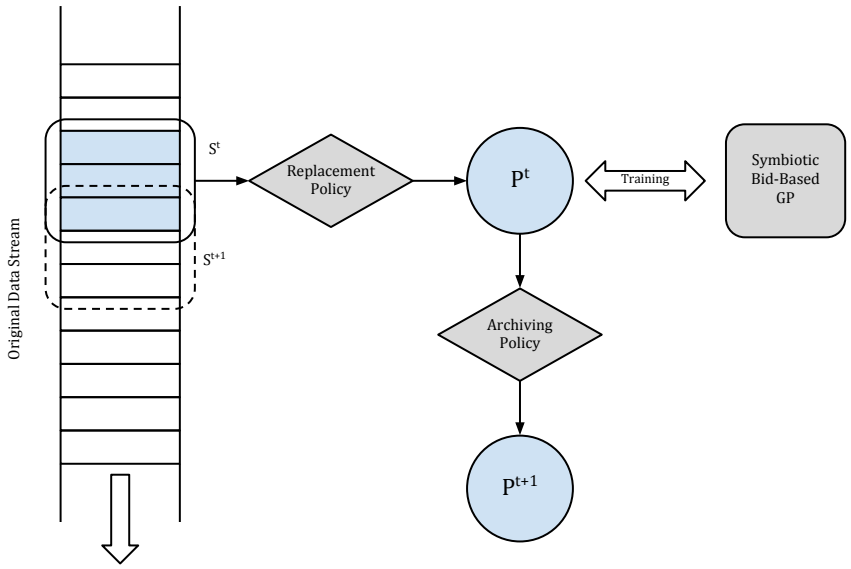


Figure 3.1: Relationship between streaming data, τ, sliding window, S^t, and point population, P. Only the content of the point population is used for fitness evaluation. A total of P_{gap} records are replaced from the point population at each generation. Pareto archiving defines up to $|P| - P_{gap}$ records retained between generations.

stream's sliding window, S^t. A breeder style replacement policy will be assumed, wherein P_{gap} is the number of points replaced. Such an architecture is summarized by Figure 3.1.

The sampling of training instances from the content of the current sliding window will assume the following process:

1. Define the class label to be sampled with uniform probability (thus we assume that we know how many classes exist);

2. Sample uniformly from the training instances that match that class (again as limited to the content of the sliding window, S^t).

Such a scheme reflects the fact that fitness evaluation at any training epoch is performed relative to the content of the point population, P, *not* the sliding window, S^t. The above two step process is repeated P_{gap} times at each generation, t, thus sampling the content of the sliding window. In addition, we assume that the initial interval over which the point population samples corresponds to 10% of the data from the stream. This corresponds to waiting for a *fixed period of time* to obtain content in

the window i.e., streams with a higher bandwidth would have a larger initial sample. The above two step process is used to seed the initial point population. After $\frac{t_{max}}{10}$ generations the S^t content is manipulated by a window protocol where several are discussed in Section 5.1. This follows recent streaming benchmarking practice e.g., [1].

Concept drift is introduced in the data sets used herein, in the incarnation of multiple distinct rulesets which partition the attribute space into separate classes being employed at different time points throughout the stream. Two distinct rulesets may or may not be conceptually related, and data representing both these cases is presented in Section 4.2. The challenge posed to an online learning system in such an environment is to not only correctly determine the correct class label for exemplars in the current environment, but also to quickly adapt to changes in the underlying concepts – something that cannot be explicitly trained upon in the sense of traditional supervised learning tasks.

3.4 Tapped Delay Lines

Tapped delay lines (TDLs), as implemented in this work, are analogous to devices used in audio signal processing [20] – e.g., finite impulse response (FIR) filters – and have been employed for embedding temporal relations between exemplars for GP in the past [41]. In streaming data contexts wherein the underlying behaviour that defines the stream is not only changing over time, but changing in a discernible pattern that repeats itself in some fashion, it seems intuitively desirable for a GP population to have some 'memory' of previous behaviour available to it. Such a construct might allow for, upon the re-emergence of a given behaviour, the training algorithm being able to fall back to previous models without needing to restart training entirely on an already known behaviour. Another case might be for the models (GP individuals in this case) to themselves recognize emerging patterns in the data and account for them explicitly. TDLs are intended to be a step toward the latter. Moreover, we note that the form of memory attributed to a TDL is distinct from / complementary to that associated with Pareto archiving. Specifically, Pareto archiving selects exemplars for retention beyond the current content of the sliding window interface of the stream cf., exemplars forming distinctions. Those exemplars retained under

this scheme however, have no particular temporal relationship with each other, and classification is still performed on an exemplar-by-exemplar basis. The TDL retains the relationship between a sequence of consecutive exemplars relative to a particular point, t, in the stream. Assuming a TDL implies that relative to some prior definition for TDL parameters (i.e., tap interval and quantity), the sequential information implicit in the TDL will provide a better basis for making a decision regarding the label associated with the latest exemplar, $x(t)$. Combining Pareto archiving with a TDL representation would imply that each entry of the Pareto archive represents TDL content relative to specific instances of t. Indeed, we will later benchmark Pareto archiving with and without the TDL 'representation'.

In this work, a tapped delay line is implemented as follows:

1. At the beginning of the data stream, until some parameter $\sigma \leq w$ data points are received, the sliding window (defined in Section 3.3) advances as normal.

2. For the next $\sigma \cdot \sigma_n$ data points, the data point is *prepended* to the data point that arrived σ data points earlier, which will be contained in the sliding window due to the parameterization constraints.

3. Once an aggregate data point in the sliding window is composed of $\sigma_n + 1$ individual data points, each new data point being prepended causes the oldest data point to be deleted. Thus an aggregate data point will never be composed of more than $\sigma_n + 1$ individual data points. In short, a TDL defines a first-in, first-out data structure that samples the previous $\sigma_n + 1$ data points (or a tap interval of unity).

A visual representation of this process is presented in Figure 3.2.

$lg(\sigma_n)$ bits are added to the instruction set of [25] and, upon program evaluation, interpreted as an index into the aggregate data point that gets passed to a GP individual. Thusly, at evaluation time, separate instructions coded by the individual to reference components of data under evaluation are interpreted by first using the 'tap index' to index into the *aggregate* data point, then using the 'attribute index' to index into the *individual* data point contained in that tap location. The mutation and crossover operators are applied to these bits by SBB as they are to the rest of

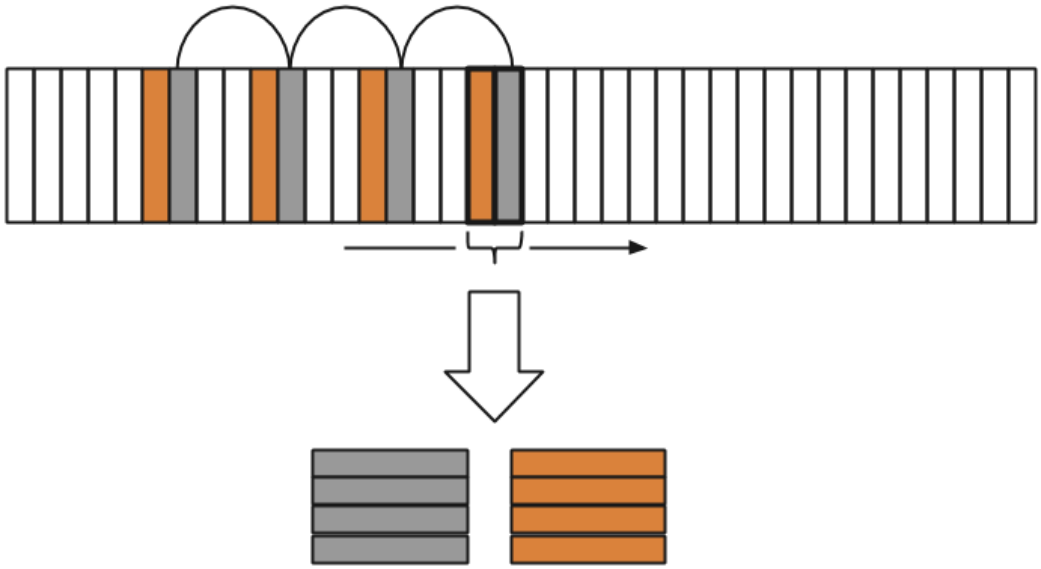


Figure 3.2: Visual representation of the tapped delay line mechanism of accumulating data from the stream. Colours highlight the sequence of σ_n points ("taps"), each separated by σ other points, that are accumulated to form a single 'aggregate' data point. (In this example only, $\sigma = \sigma_n = 4$)

the genotypic structure. That is to say, GP is an embedded paradigm in which attribute selection is a natural function of credit assignment [12].[3] Hence, at the end of evolution, GP individuals typically only index a subset of the attributes. In this case this would imply that only specific 'taps' from the TDL as well as specific attributes are indexed.

This results in only a $\frac{1}{\sigma_n}$ chance that a randomly generated (newly created) GP instruction will reference the current data point and not one of the $\sigma_n - 1$ other, previous data points when an aggregate data point is passed to a GP individual for evaluation. Although it is expected that the evolutionary cycle will eventually overcome this shortcoming, approaches such as weighting this indexing model could be used to reduce the number of generations required for this convergence to occur. Indeed, rather than employ a TDL, more general methods such as evolving the parameters of 'temporal features' such as moving averages have been shown to be particularly effective under various sequence learning tasks [44, 27]. These approaches are not considered in this work.

[3]Decision tree induction care of the C4.5 algorithm or MaxEntropy classifiers also provide this property.

Chapter 4

Synthesizing Streaming Benchmark Tasks

In this chapter we introduce a number of different task domains under which the proposed framework for classifying streaming data is to be evaluated. Section 4.1 will cover batch datasets used to benchmark the impact of introducing sliding window constraints to the GP algorithm. Section 4.2 will explain the methods used to synthesize data with underlying temporal behaviour change. Finally, Section 4.3 will discuss the methods used to emulate periodic behaviour reoccurence as might occur in real-world cyclical concept drift classification tasks.

4.1 Stationary Datasets

The first task domain utilizes datasets intended for classic non-streaming machine learning algorithms; thus, an **offline** 'batch' interface to the dataset is assumed. This implies that each exemplar is assumed to be entirely self contained. From the perspective of supervised learning, the goal of the machine learning algorithm is to map from the attribute space to a discrete label space (classification) or real-valued function space (regression). Hereafter, without loss of generality we will assume the classification task alone. Within such a task context it is assumed the data records can be stochastically reordered and the dataset can therefore be stratified and partitioned into independent training, validation and test partitions. As established in Section 1, the motivation for this is the assumption that the process creating the data is the same, or *stationary*. From a purely performance perspective, it is therefore only relative to the test partition that it is important to evaluate the operation of a candidate classifier i.e., the capacity to learn the underlying properties of the stationary process as characterized from the training / validation partitions, to the test partition or *generalization*. With this in mind, we will introduce an **online** version of the stationary classification task. In short, the proposed approach will train on the training partition of stationary classification tasks while enforcing a streaming style

dataset interface. Thus, arbitrary revisiting of previously encountered data will not be possible, and the sliding window interface implies that the training partition is visited only once. The motivation for this task is to establish a baseline performance for the GP algorithm when it switches operational modes from batch to streaming, and to evaluate the usefulness of Pareto archiving as a mechanism for retaining useful data records without the confounding effect of temporal behaviour change affecting the results.

To this end, three imbalanced datasets from the UCI Machine Learning Repository[1] are employed; namely **Census**, **Shuttle**, and **Thyroid**. These datasets are summarized in Table 4.1. From the perspective of offline 'batch' frameworks for classification, the underlying performance of the test partition of these datasets is well known, having been the subject of multiple benchmarking studies [25, 29, 45]. Indeed, the three data sets represent particularly interesting cases due to the imbalanced nature of the class distribution. Naturally, the streaming context also implies that the distribution of individual class representatives within the stream can have a significant impact on classifier performance. The pathological case of class imbalance, coupled with all instances of each class appearing in the same location temporally, would result in very biased models if extracted at interim moments in the training process. Thus, in keeping with recent machine learning practices for performing streaming data benchmarking (e.g., [1]), we stochastically reorder the training stream such that classes are distributed throughout the stream in keeping with their underlying frequency of occurrence (Table 4.1). Depending on the capacity of the sliding window, w, this implies that each class may or may not be represented within the contents of the window at each time step.

4.2 Synthesizing Streaming Datasets

The second task domain presents datasets which contain temporal behaviour change – the underlying process 'creating' the data is non-stationary (Section 1). This immediately implies that the dataset is 'ordered'. One immediate implication of this is that it is not possible to divide the data set into independent partitions as in the case of data created by a stationary process (Section 4.1). Instead, we will assume a

[1] http://archive.ics.uci.edu/ml/

Table 4.1: Characterization of benchmarking datasets. Attribute counts appear next to the respective dataset names; Values in parenthesis within the table denote test partition instance counts.

Class	Census (41)	Thyroid (21)	Shuttle (9)
1	187,141 (93,576)	93 (73)	34,108 (11,478)
2	12,382 (6,186)	191 (177)	37 (13)
3	–	3,488 (3,178)	132 (39)
4	–	–	6,748 (2,155)
5	–	–	2,458 (809)
6	–	–	6 (4)
7	–	–	11 (2)

non-stationary process that introduces either: 'sudden' transitions between multiple processes of data creation, or a 'slow' continuously changing **concept shift**. With these basic goals in mind, methods of generating synthetic data are used to solve several challenges in streaming classification tasks:

- as data streams are potentially infinite in length, a way of obtaining arbitrary amounts of data is required [13]
- existing datasets which are known (or suspected) to contain temporal concept drift tend not to be able to explicitly articulate what form the behavioural change takes or precisely when it occurs (e.g., [46])
- and; large volume datasets are desirable for testing the performance of streaming learning algorithms, but it is generally untenable to label such large datasets for use by any supervised learning algorithms without presupposing the existence of an efficient classifier.

In this work, two methods of generating synthetic data for benchmarking streaming classification algorithms are employed:

1. **Non-stationary data modelled as sudden transitions between stationary concepts:** Dataset Generator[2], a tool for generating benchmark data for data mining applications, is used to generate exemplars representing three separate concepts, C1, C2, and C3. Each of these concepts is represented by a ruleset

[2]Gabor Melli. The datgen Dataset Generator. http://www.datasetgenerator.com/

such as that shown in Figure 4.1. By themselves, these concepts each represent a single stationary learning task similar to those presented in Section 4.1. To combine the stationary datasets into a single dataset representing a streaming learning task, they are then mixed together using the method prescribed in [46]: a stream of 7,000,000 total exemplars is divided into "chunks" containing 500,000 exemplars each, and containing representation from each of the three concepts in a tuple written in the form ⟨%C1, %C2, %C3⟩. The entire stream can then be written as fourteen chunks expressed, in order, as follows: $\langle 100, 0, 0\rangle$ $\langle 100, 0, 0\rangle$ $\langle 100, 0, 0\rangle$ $\langle 90, 10, 0\rangle$ $\langle 80, 10, 0\rangle$ ... $\langle 10, 90, 0\rangle$ $\langle 0, 100, 0\rangle$ $\langle 0, 0, 100\rangle$. A visual representation of this pattern is shown in Figure 4.2. The resulting data set will be denoted **datgen**.

The data is generated using the parameterization set forth in [46]. Five real-valued attributes are specified that are relevant to the class attribute, and a sixth attribute that is irrelevant. Datgen then creates a decision tree-style list of rules that partitions the attribute space into 5 classes based on randomly generated threshold values, and assigns a class label to each of the tree branches. Exemplars are then created by randomly determining attribute values and evaluating them against the generated decision tree in order to assign a class label to the instance.

2. **Non-stationary data modelled as continuously varying process:** Two hyperplanes of the form $\Sigma_{i=1}^{d} a_i x_i = a_0$ are generated, as in [13], where the values of a_i represents the current state of the stream (initialized randomly), the values of x_i represent the attribute values of a given exemplar, and $d = 10$ represents the chosen dimensionality for the task domain. A stream of 150,000 exemplars is created, and concept drift is parameterized and simulated as follows: Every $N = 1,000$ exemplars, each entry in a vector $s = \{-1, +1\}^d$ (also initialized randomly) is given a 20% chance of inverting its value. The first 5 of 10 non-class attributes in a are then moved at a rate of $t = 10\%$ over the course of the next N exemplars, compounded every generation. This function is summarized in Equation 4.1:

```
Synthetic Data Set    Underlying Synthetic Rule Base
#   A  B  C  Class    (activation%) class <- class description
 1: 87 58 45 c3       (70.0%) c1 <- A=[1,9] & B=[68,76] & C=[8,16]
 2:  5 68 13 c1       (10.0%) c2 <- A=[72,80] & B=[82,90] & C=[32,40]
 3:  9 72 13 c1       (20.0%) c3 <- C=[45,53]
 4:  5 72 16 c1
 5: 11 68 53 c3
 6:  4 73  9 c1
 7:  5 71 12 c1
 8:  4 70  9 c1
 9:  5 71 16 c1
10: 76 84 34 c2
```

Figure 4.1: Example of output with default parameters of the **Dataset Generator** tool.

$$a_i = a_i + s_i \cdot \frac{a_i \cdot t}{N} \tag{4.1}$$

Exemplars are created by generating uniform random values for all non-class attributes. To determine the class label, the values a_0 for each hyperplane are first normalized by summing the rest of the entries in the vector and multiplying by a factor of 1/3 and 2/3 respectively, in order to keep the class distribution from becoming entirely one-sided. The class label is then equal to the number of hyperplanes for which $\Sigma_{i=1}^{d} a_i x_i < a_0$. The resulting dataset will be denoted **planar**.

This provides two basic scenarios for comparing the performance of the Pareto archiving heuristics presented in Section 5.2 under different styles of concept drift. The first (**datgen**) is a discrete case in which two different concepts cannot be interpolated between, and instead only the frequency of their appearance in the surrounding stream contents changes over time. The second (**planar**) is a continuous case in which two different concepts are related to each other by some randomly drifting function, and a given exemplar may exist on some fuzzy boundary between the two as the stream's make-up slowly drifts from being composed of one concept to another.

In keeping with established practice in generating synthetic data sets for classification, new exemplars are created by randomly choosing values for each attribute and

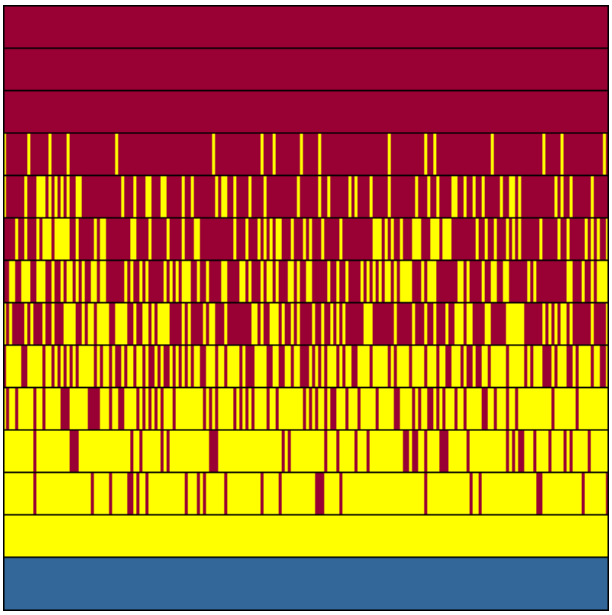


Figure 4.2: Visual representation of the pattern in which data from different rulesets is combined to create the **datgen** dataset. Here colours represent concepts C1-C3, while each horizontal bar represents a 'chunk' in the stream.

Table 4.2: Characterization of benchmarking datasets. Attribute counts appear next to the respective dataset names.

Class	datgen (7)	planar (11)
1	2,615,747	14,055
2	1,801,055	120,167
3	1,643,327	15,778
4	635,629	–
5	304,242	–

then applying the generated ruleset to determine the exemplar's class attribute. This tends to lead to a natural class imbalance dependent on how much of the attribute space falls under the jurisdiction of each generated rule. The datasets are summarized in Table 4.2.

4.3 Cyclical Datasets

Finally, the datasets presented in the previous section are expanded to represent cyclical concept drift task domains, in which quantifiable behavioural change that occurs within the data stream occurs more than once. Two cases for these cycles are

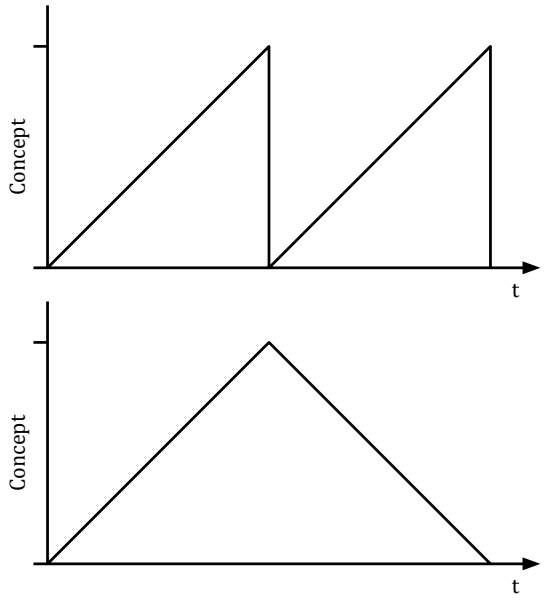


Figure 4.3: Visual representation of patterns used to represent cyclical concept drift. Above shows the **repeat** case in which the stream immediately switches to its initial start and replays the 'drift' pattern over upon reaching the halfway point. Below is the **mirror** case in which the pattern is replayed in reverse order, representing a gradual return from the end-state back to the start-state.

considered (illustrated in Figure 4.3):

- the **repeat** case in which a stream goes from being defined by its initial behaviour to some other behaviour, and then immediately resets to its initial behaviour and plays out the same behavioural change again, and
- the **mirror** case, in which the behavioural change is applied in reverse order after occuring the first time; this means that at the end of one full period, the stream has returned to its initial behavioural state.

In the case of the **datgen** dataset defined in Section 4.2, the final chunk (denoted $\langle 0, 0, 100 \rangle$) is moved to the end of the stream and the other chunks have the **repeat** and **mirror** operations applied to them. The resulting two datasets are thusly defined:

- **datgen-repeat**: $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 90, 10, 0 \rangle$ ... $\langle 0, 100, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 90, 10, 0 \rangle$ ... $\langle 0, 100, 0 \rangle$ $\langle 0, 0, 100 \rangle$
- **datgen-mirror**: $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 90, 10, 0 \rangle$ ... $\langle 0, 100, 0 \rangle$ $\langle 10, 90, 0 \rangle$ ... $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 100, 0, 0 \rangle$ $\langle 0, 0, 100 \rangle$

In the case of the **planar** dataset defined in Section 4.2, the random changes applied to the vectors a and s are recorded as they occur. To create the **planar-repeat** dataset, the two vectors are reset to their initial states at the end of the first half of data generation, and the recorded changes are played out again for the second half of data generation. To create the **planar-mirror** dataset, the recorded changes are simply applied in reverse during the second half of data generation. This implies that the non-stationary process creating the data is mirrored (although the data itself is not explicitly duplicated).

Chapter 5

Empirical Evaluation

This section is divided into three parts, corresponding to the task domains introduced in Chapter 4. First, the use of Pareto archiving is evaluated as a mechanism of providing a limited 'memory' of stream contents that retains useful exemplars over time w.r.t. non-stationary streams, as introduced in Section 3.2. This represents a base-case in which the interest is in assessing what is 'lost' when the constraints associated with the sliding window approach from Section 3.3 are adopted. Specifically, other researchers have already established the expected performance outcomes for this task under the offline batch approach to classification e.g., [25]. Any variation must therefore be attributed to adopting the streaming constraints. With this in mind, Section 5.1 reports on the 'base-case' evaluation under non-stationary conditions. The two non-stationary artificially created datasets as defined in Section 4.3 are then benchmarked in Section 5.2, both with and without the cyclic properties discussed in Section 4.3. Finally, Section 5.3 incorporates the use of tapped delay lines (Section 3.4) for providing the GP individuals with a form of memory that gives direct access to several previous exemplars taken from the stream.

5.1 Stationary Data Evaluation

5.1.1 Methodology

Four protocols are considered for defining the relationship between training data (τ), sliding window (S^t), and point population (P):

1. Pareto archiving without limits on how the training data (τ) is accessed – this establishes the performance baseline for Pareto archiving under a batch offline access policy ($S^t = \tau$) and finite Point population (P). The capacity of the Point population remains constant over all experiments. Training instances can be revisited any number of times during evolution, subject to a common

limit on the maximum number of generations performed during training (t_{max}). Hereafter this case is denoted **bat**.

2. Pareto archiving under a sequential access limit to training exemplars, but with revisiting to any *previously encountered* training instance possible – This scenario is equivalent to a sliding window, S^t, in which the lower bound remains unchanged (references the first training instance of τ) but the upper bound increases with training epoch, or $|\tau| \times \frac{t}{t_{max}}$, where $|\tau|$ is the size of the entire training partition, t is the current generation, and t_{max} is the last generation, common to all experiments. Thus at the last generation any exemplar from the entire training partition is eligible for appearing in the Point population (P). Hereafter this case is referred to as **agP**.

3. Pareto archiving under sequential access to training exemplars, and no capacity for revisiting previously encountered exemplars unless they were retained in point population – as per Figure 3.1, the sliding window, S^t, increments its location in proportion to the training epoch, t. Thus, under the stochastic sampling process of Section 3.3 the location of the sliding window is bound by the interval $[|\tau|(1-w), ...|\tau|]\frac{t}{t_{max}}$; where w is the percent capacity of the sliding window (Appendix A). As established in Section 3.3 a total of P_{gap} training instances are sampled from the sliding window at each generation. Hereafter this case is denoted **w*xx*P** where 'xx' takes the value for w.

4. No Pareto archiving under sequential access to training exemplars – this establishes the contribution of Pareto archiving. Instead, P_{gap} points are selected for replacement under the stochastic sampling process of Section 3.3. The parameterization of the sliding window follows points 2 and 3. Hereafter these cases are denoted **agN** and **w*xx*N** respectively.

It is important to note that τ would be unknown in practice, which means the value of t_{max} cannot be determined, i.e. training is performed on a continuous basis. Moreover, in cases 3 and 4, the initial construction outlined in Section 3.3 provides S^t with random access to the first 10% of the stream for the first $\frac{t_{max}}{10}$ generations. However, this work will assume a finite data set size whose size is known at initialization,

which simulates cutting the training process off at an arbitrary point in time; the above constraints are designed only to enforce a single pass through the data set for a common fixed number of generations and common Point population size i.e., the computational cost as measured in terms of the number of evaluations for each interfacing scenario is exactly the same. Thus, under such a common evaluation limit, we can investigate the impact of different class distributions, sliding window sizes and stream bandwidths. Post training evaluation metrics will take the form of avg. Detection rate (Eqn (3.2)) as calculated across the unseen test partition.

5.1.2 Parameterization

Relative to the three sliding window scenarios identified in Section 5.1.1, the following parameterizations are assumed:

Maximum number of generations, t_{max}: This defines the number of generations conducted while making a single pass through the data set (Section 5.1.1), and is held constant across all datasets. The impact of this is that new training instances become eligible at the rate of ≈ 6.65 per generation under Census; ≈ 1.45 per generation under Shuttle; and ≈ 0.13 per generation under Thyroid (i.e., Total training instance count $\div$ t_{max}).

Window size, w, for the sliding window: Three separate parameterizations are considered: 10, 5 and 1 percent of the training partition. Naturally, the smaller the window the less opportunity the point population has to sample them before they are shuffled out the sliding window.

SBB parameters: are in some cases increased relative to the original study [25]. In particular, the elitist nature of replacement (w.r.t. point population content) supports higher rates of mutation for adding (p_a) and deleting (p_d) symbionts during reproduction of the host (team) individuals (Table A.1). Likewise, the point and host population sizes have also increased, this time on account of the more significant role they play in maintaining a record of training instances and candidate solutions. That said, all SBB parameters assume common values and do not vary as a function of the data interface. Other studies have

benchmarked different forms of GP on the same datasets, both with (batch offline) Pareto archiving [29] or with fitness evaluation conducted over the entire training partition at each generation [45].

In each case a total of 50 runs are performed per experiment.

5.1.3 Results

Results have previously been reported on the three datasets under a common definition for the test partition in which the *entire training partition* was employed for fitness evaluation [45]. This is taken to establish an 'empirical upper bound' on what might be expected by way of the avg. DR for each dataset's independent test partition: 80.5%, 95.4% and 98.3% for Census, Thyroid and Shuttle respectively. In the following we perform:

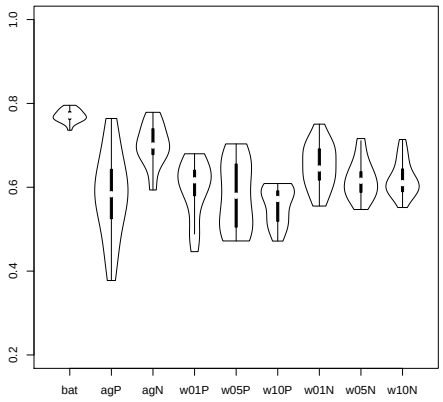
- a static analysis of the resulting performance post training relative to the 'unseen' test partition (Section 5.1.4), and then;
- investigate the dynamic properties resulting from the streaming context (Section 5.1.5).

The static analysis naturally characterizes to what extent assuming different forms of (sliding window) interface impacts on the ultimate performance of the GP classifier post training. The dynamic analysis enables us to look at the impact on the Point population Pareto archive. This will be decisive in providing an explanation for some of the outcomes from the static analysis.

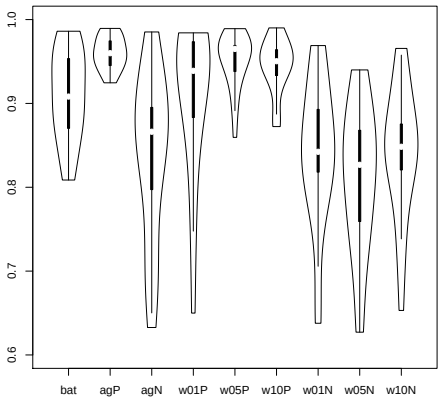
5.1.4 Static Evaluation

As outlined in Section 5.1.1 the static analysis considers a total of 5 data interfacing scenarios:

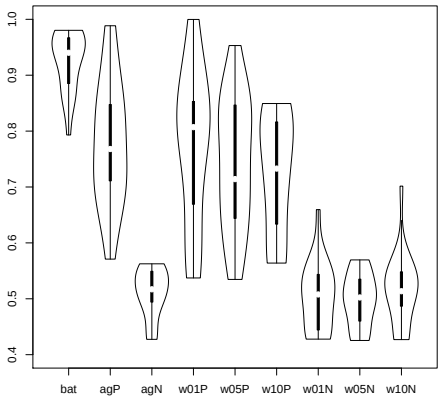
1. Pareto archiving, but no streaming constraint or the **bat** configuration;
2. Pareto archiving, with streaming constraint, but the opportunity for arbitrary revisiting once encountered, or the **agP** configuration;



(a) Census



(b) Thyroid



(c) Shuttle

Figure 5.1: Average Detection Rate metric under test data. See section 5.1.1 for column labels. *A note on interpreting violin plots: each plot contains an ordinary boxplot, with additional information illustrating the probability distribution of the result set drawn at either side.*

3. As per point 2, but with Pareto archiving replaced by uniform selection, or the **agN** configuration.

4. Pareto archiving, with streaming constraint and finite sliding window, or the **w*xx*P** configuration with *xx* denoting window sizes of: $1\%, 5\%, 10\%$, hence three distributions.

5. As per point 2, but with Pareto archiving replaced by uniform selection, or the **w*xx*N** configuration from section 5.1.1 with *xx* denoting window sizes of: $1\%, 5\%, 10\%$, hence three distributions;

The binary **Census data set** appears to represent the most difficult scenario (Figure 5.1(a)), a factor that is also reflected in the case of fitness evaluation performed across the entire training partition [45]; or an avg. DR of 80%. The non-Pareto archiving cases (columns 3, 7–9) return better results than the corresponding Pareto archived cases (columns 2, 4–6). Thus, taken pairwise the case of the aggregated stream window (agN) drops by approx. 10% whereas the finite window cases (w*xx*N) drop by approx. 5%. Pareto archiving under batch access to the training partition (column 1) appears to be largely unaffected. Hence, the relatively poor showing of Pareto archiving under Census is most definitely an artifact of the streaming context. We will return to the source for this performance reduction in the review of the dynamic properties of assuming a streaming context in Section 5.1.5.

The three class **Thyroid data set** is the smallest data set considered here (Figure 5.1(b)). This implies that given the fixed generation limit (common to all data sets) then evolution has more time to sample the content from any particular streaming window location, S^t i.e., S^t content is replaced at a slower rate. Pareto archiving (columns 2, 4–6) resulted in significant improvements w.r.t. the corresponding non-Pareto cases (columns 3, 7–9) in all but one parameterization. Moreover, the performance of Pareto archiving with streaming constraints enforced typically performs better than with batch access rights (column 1) on this data set i.e., even under a 1% window size (column 4) the quartile performance is at least as good as the batch case.

Performance on the seven class **Shuttle data set** is summarized by Figure 5.1(c). A very clear separation again appears between non-Pareto archived (columns 3, 7–9)

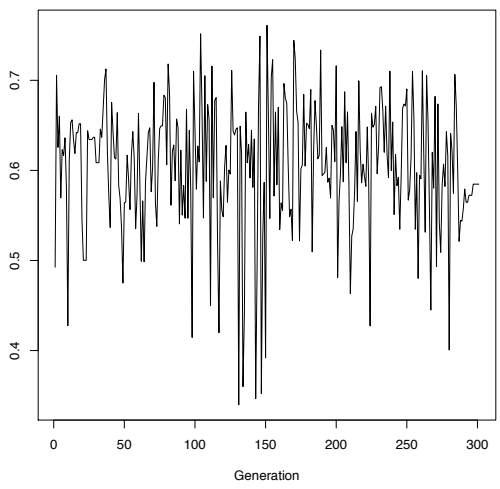
and Pareto archived (columns 2, 4–6) performance. Thus, significant differences appear between aggregate stream access (column 2 versus 3) and each sliding window variant (columns 4, 5, 6 versus 7, 8, 9). Moreover, the best stream access case corresponds to that of the most restrictive case, or a sliding window of 1%. However, no streaming access scenario is able to reliably approach the batch Pareto case. Moreover, the median performance of all the non-Pareto results indicate that the equivalent of 3 to classes are not being labelled at all. Conversely, median performance of the Pareto enabled models return Detection rates between 70 to 80%, suggesting that a minimum of 5 to 6 classes are detected (1 or 2 classes are missed).

Finally, we note that independent of the data set, there is a strong correlation between performance under the arbitrary revisiting stream constraint (ag$x$)versus strict sliding window constraint for S^t. Thus comparing agP to wxxP or agN to wxxN under a Wilcoxon non-parametric hypothesis test to each pair of stream configurations (agP versus agN; wxxP versus wxxN) implies that the non-Pareto case only proves a statistically significant improvement over Pareto archiving for the specific case of agP–agN and w10P–w10N under Census; in all other cases Pareto archiving results in significantly better avg. DR (p-values < 0.005).

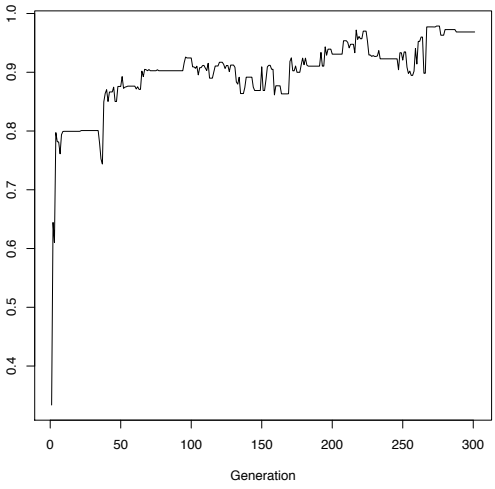
5.1.5 Dynamic Evaluation

The preceding section concentrated on evaluating performance relative to the independent test partition post training, i.e. once the entire stream had passed. In doing so it was possible to assess the impact of increasing constraints placed on the form of sampling performed on the training partition both with and without Pareto archiving. From a streaming perspective a clear preference is evident for Pareto archiving under the Thyroid and Shuttle data sets, whereas Pareto archiving appears to negatively impact performance under Census. In this section we investigate how performance on the independent test partition varies *during* training. This is particularly significant under the sliding window context. Hence, for clarity, in the following we concentrate on the specific case of w05P, where the trends reported are common across wxxP.

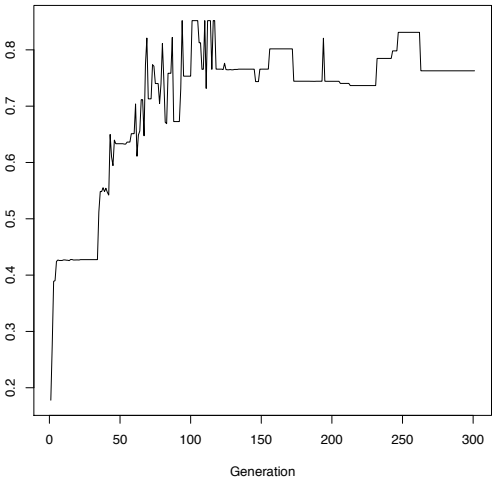
Figure 5.2 details how performance varies (w.r.t. a specific training run) on the test partition as SBB encounters the training data under the sliding window scenario of w05P. Test data is never used for any aspect of evolution, only as an independent



(a) Census

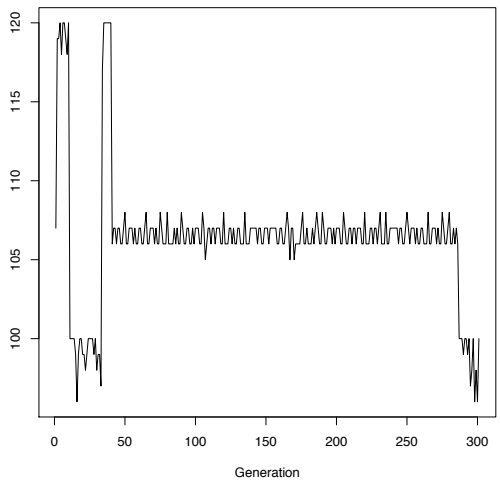


(b) Thyroid

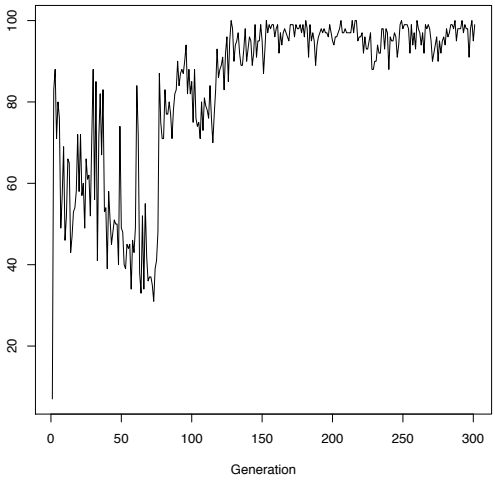


(c) Shuttle

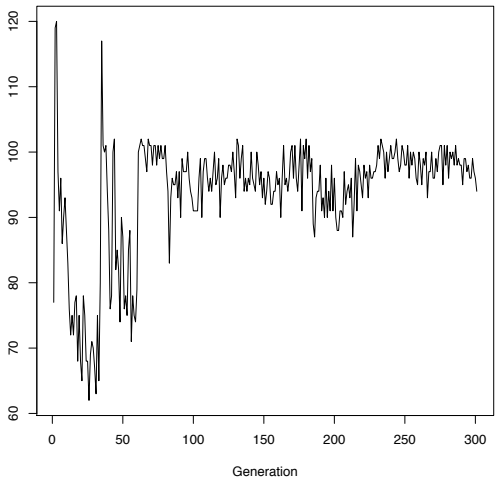
Figure 5.2: Average Detection Rate of test partition *during* training on the stream using Pareto archiving. Sliding window of 5% in all cases (w05P). For clarity generation axis is $\times 100$.



(a) Census



(b) Thyroid



(c) Shuttle

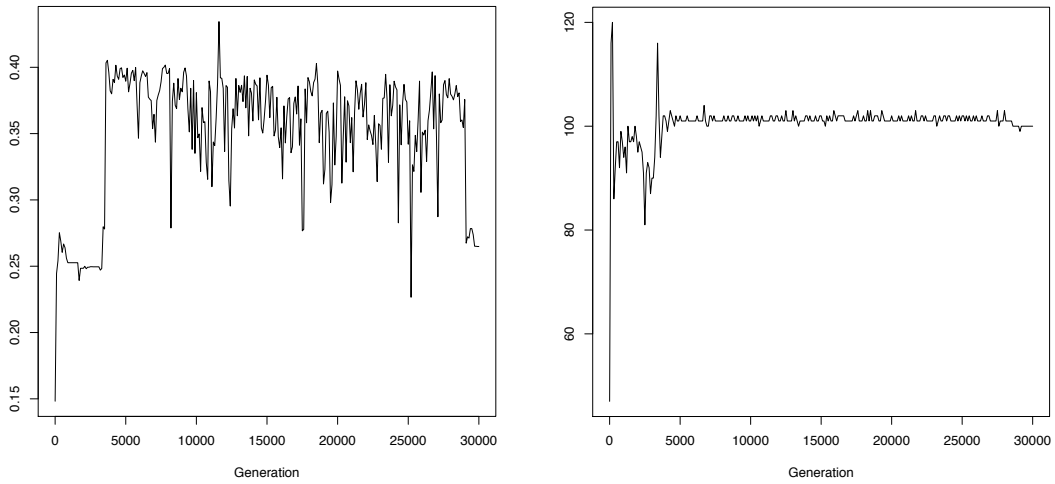
Figure 5.3: Size of Pareto archive *during* training on the stream using Pareto archiving. Sliding window of 5% in all cases (w05P). For clarity generation axis is $\times 100$.

assessment of the degree of regression or improvement at that particular training epoch. Given the computational overhead, a test evaluation is performed once every 100 generations. The contrast between training under Census versus Thyroid and Shuttle is again readily apparent. Specifically, Thyroid and Shuttle continue to improve after the initial period of training against 10% of the data (corresponding to the 0 to 3000 interval on the horizontal axis). Thus, both Thyroid and Shuttle have a distinct step-wise profile in which performance is essentially improving.

Figure 5.3 summarizes the corresponding utilization of the Pareto archive in the point population, P. Recall that P_{gap} points are replaced at each generation. Thus as long as the size of the Pareto archive remains smaller than $|P| - P_{gap}$ the likelihood of losing potentially good classifiers will be minimized. In the parameterization summarized by Table A.1 the Pareto archive should not exceed 100 individuals for this condition to hold. It is apparent that for both Thyroid and Shuttle this condition holds, but in the case of Census the Pareto archive actually matches the size of the Point population before the process of replacing a fixed number of points per generation (P_{gap}) forces the Pareto archive membership to 'sit' at about 105 training exemplars. Doubling the size of the point population had no measurable effect.

Given a 20% error rate, even when employing all the exemplars at each generation for fitness evaluation [45], and a training partition of nearly 200,000 exemplars then a worst case archive in the order of thousands of points might be anticipated. Moreover, we also question what such a rate of error implies. Specifically, this appears to imply that typically 20% of the data is mislabeled or, put another way, the attributes for 20% of the data are not sufficient for distinguishing between the two classes (class noise).

In order to investigate this hypothesis we introduce a process of re-labelling the Shuttle training data sets with 5, 10 and 20% probability. Thus, a data instance selected with uniform probability is relabelled to that of one of the other classes (also chosen with uniform probability). Figure 5.4 illustrates how the avg. DR of the test partition and Point archive varies through training for a 5% class-noise rate under the Shuttle data set. It is clear that at best the equivalent of just under 2 classes are learnt during the initial 10% training period (generation < 3000; avg. DR $\approx \frac{2}{7}$), and at best the equivalent of one more class appears thereafter (generation > 5000; avg.



(a) Test during stream (b) Point archive during stream

Figure 5.4: Dynamic properties of Shuttle data set with 5% noise under w05P sliding window.

DR $\approx \frac{3}{7}$). For completeness, Figure 5.5 provides a comparison of the post training performance on the test partition (over 20 runs) for cases of 5, 10 and 20% class noise. In short, the noise free median performance (column 5, Figure 5.1(c)) has dropped from an avg. DR of 70% to just above 30% (5% class noise rate) and thereafter decaying as the amplitude of class noise increases.

5.2 Non-Stationary Data Evaluation

5.2.1 Methodology

Section 5.1 noted that Pareto archiving is sensitive to contradictory labels – that is to say, the Point archive would rapidly accumulate exemplars from the stream that were forming 'distinctions'. This is to be expected, as such contradictions would likely support at least two GP classifiers as being non-dominated. Attempting to address this by just letting archive sizes grow with the number of distinctions is not desirable under streaming contexts as this has obvious implications for real-time operation. Indeed, Pareto archiving has a $P_{size}^2 - P_{size}$ computational cost, where P_{size} is the Point archive size [9].

Relative to the Pareto archiving framework of Section 3.2, we note that at present a

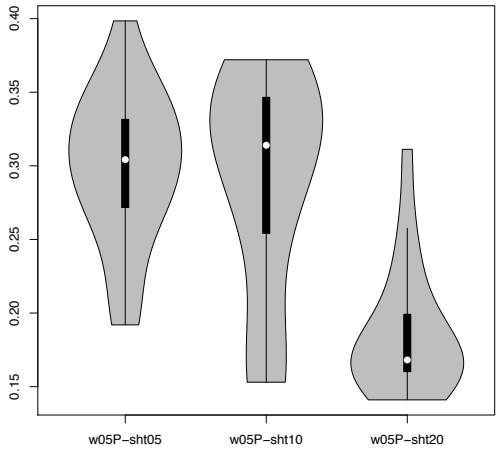


Figure 5.5: Average Detection Rate of Shuttle data set on test partition.

fitness sharing heuristic is used to promote the maintenance of diversity once the number of non-dominated points encounters the constraint imposed by a finite archive. This may or may not represent an appropriate heuristic for prioritizing point replacement under a streaming context. That is to say, once a point provides a distinction it could potentially lie in the point population indefinitely, such as in the case of an outlier or mislabelled data. However, under streaming applications the underlying processes determining stream content are frequently non-stationary. With this in mind, the fitness sharing heuristic of Equation 3.5 will be reconsidered with the goal of evaluating its role in allowing the GP populations to interact more effectively with the current contents of the stream. To that end, the following three configurations will be explored:

1. Fitness sharing in its original, unmodified format. This case will serve as the base case for attempted optimizations to the fitness sharing heuristic. Hereafter this case is denoted **org**.

2. Fitness sharing scores multiplied by the normalized inverse of point archive ages; that is, $score = score \times (1 - age/age_{max})$. Naturally, this heuristic introduces an age bias, with the motivation being to make older points more likely to be replaced. Hereafter this case is denoted **age**.

3. Points with a fitness sharing score of 0 are considered to be ranked highest; after this, points are taken in the order of highest score first. This case is intended to

address the issue of new points which **no** GP individuals are able to correctly classify being unable to enter the point archive. This prevents material that is currently entirely unclassifiable from being presented to the team population as a desirable problem to be solved. Hereafter this case is denoted **zero**.

In addition, online learning applications require an answer in the "here and now" of a data stream. Specifically, this implies that it is not possible to define a partition of the data into independent training / validation / test partitions. The classifiers are undergoing a continuous process of development as they 'react' to the non-stationary properties of the data stream. With this in mind, the following methodology will be adopted to characterize the performance of a champion GP individual from the population at any point, t, through the stream:

1. Training is limited to a fixed number of generations t_{max} over the duration of the stream.

2. Testing is performed at periodic intervals during training by extracting the current champion GP individual and evaluating it separately across the previous interval, and the next (as-yet-unseen) interval.[1] Such a comparison potentially establishes the generalization capability of the champion relative to the immediate past and future classification requirements. Expecting 'good' generalization beyond these limits is considered increasingly questionable, given the non-stationary property of the stream. The evaluation metrics used will be the accuracy (total number correct versus the total number of exemplars in the interval) and average detection rate (Eqn (3.2)) as calculated across the interval. The relative variation between accuracy and average detection rate provide a characterization for how sensitive the resulting classifier is to class imbalance.

5.2.2 Parameterization

Relative to the three fitness sharing heuristics identified in Section 5.2.1, the following parameterizations are assumed:

[1]The concept on an 'interval' specific to the process generating the data (Section 4.2). Under the *datgen* or non-stationary process with sudden changes, an interval corresponds to a 'chunk' of 500,000 exemplars associated with the same data generation process. Under the continuously varying non-stationary data generation model (*planar*), an interval is 1,000 consecutive exemplars, associated with the 'direction' of vector s.

Maximum number of generations, t_{max}: As in previous experiments. The impact of this is that new training instances become eligible at a rate of ≈ 233.33 per generation under **datgen** and ≈ 5 per generation under **planar** (Total training instance count $\div$ t_{max}).

Window size, *w*, for the sliding window: The size of the sliding window is fixed at 5% for all further experiments. This is chosen as a representative case that compromises between low memory overhead and greater access to training data.

SBB parameters: are unchanged from Section 5.1.2, and are summarized in Appendix A.

In each case a total of 50 runs are performed per experiment.

5.2.3 Evaluation

As outlined in Section 5.2.1, a total of three scenarios are considered across the **datgen** and **planar** data sets:

1. Unmodified fitness sharing scores or the **org** configuration;
2. Fitness sharing scores multiplied by inverse normalized exemplar age, or the **age** configuration;
3. Fitness sharing scores left unmodified, except scores of 0 are changed to ∞, or the **zero** configuration.

The average detection rate metrics, as calculated across periodic intervals in the stream, are shown in Figure 5.6 for the datgen data set, and in Figure 5.7 for the planar dataset.

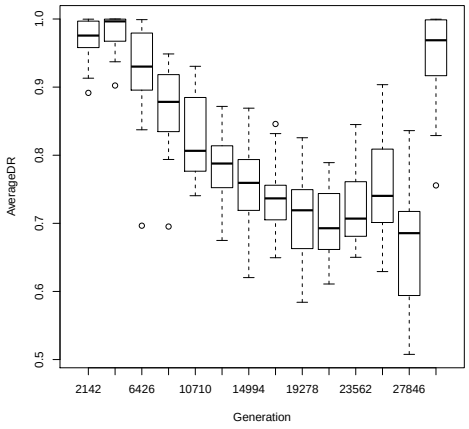
In the base case **org** for the **datgen** stream, performance hovers near unity for the first two intervals while the underlying concept C1 does not change, then begins to steadily decline as data points from the second concept C2 are gradually introduced. The decline in performance continues until the tenth chunk, at which point more than two-thirds of the stream is composed of representatives of C2, and only then does the algorithm begin adapting to the change in the underlying environment.

Interestingly, the next-to-last chunk of data composed entirely of C2 exemplars caused the performance of the algorithm to drop the lowest, but the team population is able to easily adjust to a brand new concept C3 when it abruptly takes over the stream (last chunk of the sequence). Such behaviour would seem to imply a weakness in the current GP model for adapting to change spread out over a significant period of time, at least in the case of discrete concept representation. Conversely, sudden change appears to result in a more effective retraining and replacement of GP classifiers.

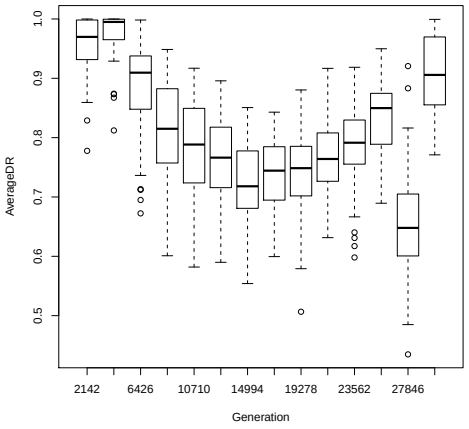
The other experiments **age** and **zero** in the **datgen** environment exhibit similar behaviour but appear to be able to start compensating for the gradual change in concept earlier in the stream. Thus, the quartile statistic is better at remaining above an average detection rate of 70% than under **org**. Of the three, the **age** case appears to adapt the quickest. This configuration, as described in Section 5.2.1, was intended to punish points that have spent more time in the archive – to the extent that the oldest point is deterministically removed at each generation.

In all cases for the **planar** stream, performance in terms of average detection rate appears to suffer initially from a pathological case of class imbalance. This is corroborated by the high accuracy metrics reported for that portion of the stream (Figure 5.8), and can be verified by looking at the class distribution of the point archive from an example run (not shown). In this case, the original diversity based heuristic appears to resist the incremental variation in the stream content better than either the explicitly age or 'disengaged point' biases (**age** and **zero** respectively). This is particularly true under the balanced detection rate metric of Figure 5.7. Note the different scales under the accuracy metric in Figure 5.8. In this case there is much more variation in the spread of the original heuristic (e.g., 1st quartile extends down to 50%) whereas the **age** and **zero** heuristics demonstrate less variance. However, in either metric, median performance never descends below 70% under **org** whereas both the alternative heuristics fail to do so.

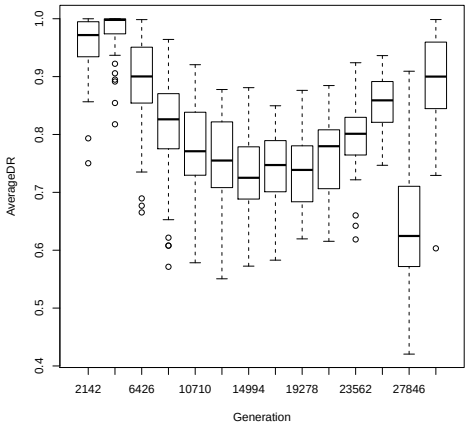
The magnitude of the effect of the different fitness sharing heuristics on the archive policy's age bias can be clearly seen in Figure 5.9. The **org** heuristic constantly churns the contents of the point archive, whereas both **age** and **zero** provide an additional level of granularity to the score values that allow the archive to distinguish between points enough to single some out for more long-term archiving.



(a) org

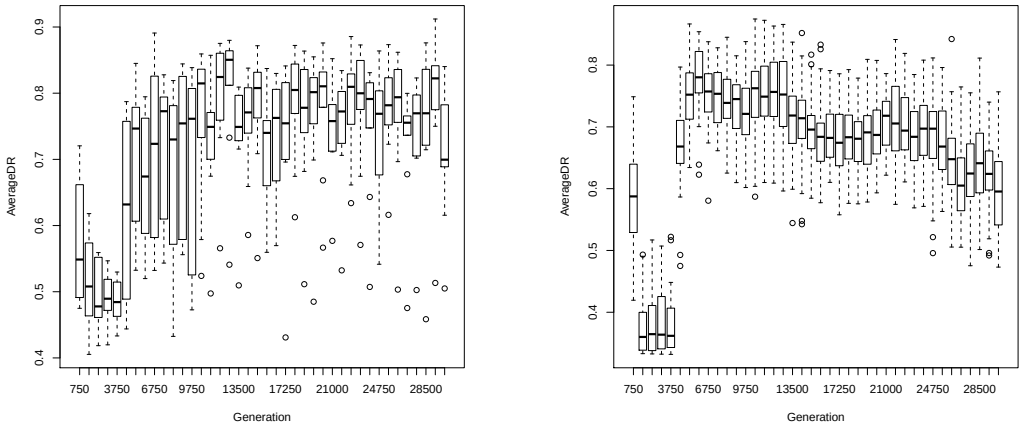


(b) age



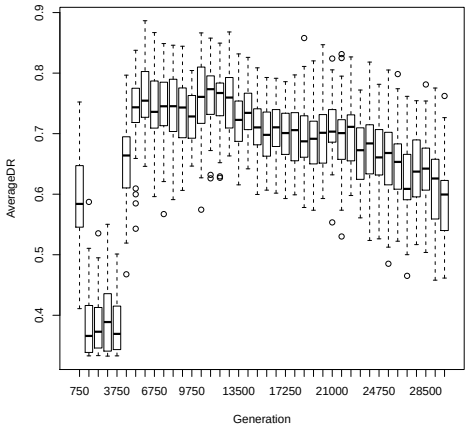
(c) zero

Figure 5.6: Average Detection Rate across previous chunks during training on the stream, using the *datgen* data set.



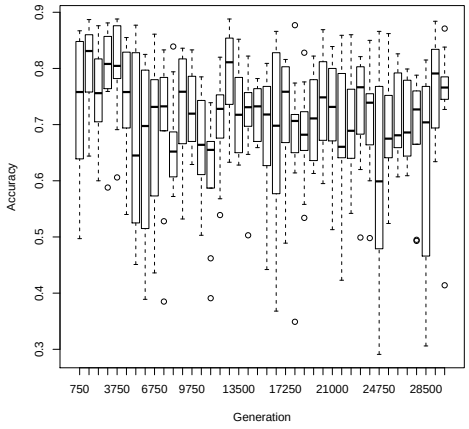
(a) org

(b) age

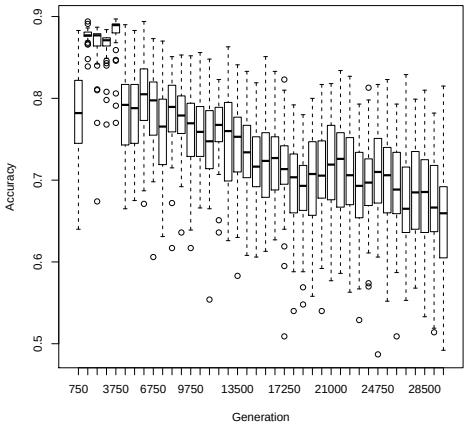


(c) zero

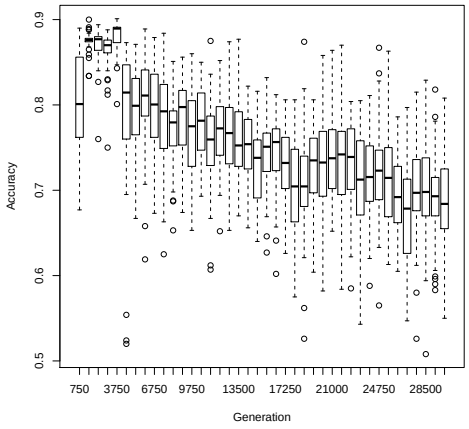
Figure 5.7: Average Detection Rate across previous chunks during training on the stream, using the *planar* data set.



(a) org



(b) age



(c) zero

Figure 5.8: Accuracy across previous chunks during training on the stream, using the *planar* data set.

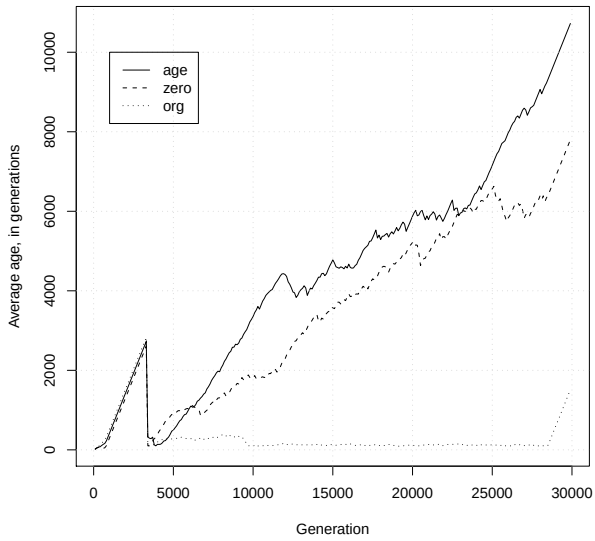


Figure 5.9: Average age of individuals in the point archive from single runs of experiments in the *planar* environment.

5.3 Tapped Delay Lines and Cyclical Data

5.3.1 Methodology

As noted in Chapter 2, behavioural change is not always an isolated or one-way occurrence and can frequently occur in a cyclical fashion. Moreover, if a classifier can base its labelling decision for exemplar t on a sequence of the most recent history of exemplars, then change detection might be more directly facilitated. In such a circumstance, the deployment of a Point archive as a mechanism for providing 'memory' of the stream becomes an even more literal metaphor – it would be desirable for a population of GP individuals to have access to some record representing the change in stream behaviour over time, in order to potentially 'recognize' previously seen patterns as they re-emerge in the stream and thus adapt to them in a more efficient (ideally instantaneous) manner. To that end, the experimental configurations of the previous Section 5.2.1 (that is, **org**, **age**, and **zero**) are evaluated across the cyclical datasets defined in Section 4.3. In addition, an exploration of providing the GP individuals with an explicit record of the stream (as opposed to the implicit record of the stream represented by the Pareto archive) is performed by running all

the aforementioned configurations with the addition of the tap delay line mechanism outlined in Section 3.4.

To summarize, this section will explore the following experimental configurations:

- **org**, **age**, and **zero** are evaluated against the **datgen-repeat** dataset with tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **datgen-repeat** dataset without tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **datgen-mirror** dataset with tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **datgen-mirror** dataset without tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **planar-repeat** dataset with tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **planar-repeat** dataset without tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **planar-mirror** dataset with tapped delay lines
- **org**, **age**, and **zero** are evaluated against the **planar-mirror** dataset without tapped delay lines

5.3.2 Parameterization

Relative to the scenarios identified in Section 5.3.1, the following parameterizations are assumed:

Maximum number of generations, t_{max}: This is set so that the number of training instances that become available at each generation is exactly the same as in Section 5.2.2. This is a simple doubling in the case of the **planar** dataset, but for the **datgen** dataset the value of t_{max} is multiplied by $\frac{27}{14}$ to account for the fourteenth chunk (of fourteen) not being replicated.

Window size, *w*, for the sliding window: The size of the sliding window remains fixed at 5%, as explained in Section 5.2.2.

Tap delay length, σ: The number of data points that are allowed to elapse before new data points begin to be appended to previous ones, as discussed in Section 3.4. This is set to be equivalent to the width of one sliding window, i.e., $\sigma = w$.

Tap delay count, σ_n: The number of taps that can be prepended to a point before new taps begin to overwrite the oldest ones, as discussed in Section 3.4. This is set to $\sigma_n = 7$, as the current number of operators and registers defined in this SBB implementation resulted in 3 bits being left for use before the opcodes spilled over the boundary of a 16 bit word.

Put another way, GP now views the input at any classification decision as a matrix of addressable read only memory as opposed to a vector of addressable read only memory. The cells actually addressed from the address space is an evolved parameter specific to the GP individual, cf. embedded as opposed to filter or wrapper classifier deployment [18].

SBB parameters: remain unchanged from Section 5.1.2, and are summarized in Appendix A.

In each case a total of 50 runs are performed per experiment.

5.3.3 Results

As outlined in Section 5.3.1, a total of six scenarios are considered across the **{datgen,planar}-{repeat,mirror}** datasets, and the average detection rate is presented for all cases: the **org** heuristic 1) without (Figure 5.10) and 2) with (Figure 5.11) tapped delay lines, the **age** heuristic 3) without (Figure 5.12) and 4) with (Figure 5.13) tapped delay lines, and the **zero** heuristic 5) without (Figure 5.14) and 6) with (Figure 5.15) tapped delay lines.

It should be noted that the first half of all configurations in which tapped delay lines are *not* employed are equivalent to the experiments performed in Section 5.2.3, and so performance in these cases will only be discussed for the second half of the training time relative to the first half.

5.3.4 Evaluation of datgen without Tapped Delay Lines

In the **org** case of unmodified fitness sharing, the **datgen-repeat** performance shows the ability to easily jump back to its near-unity performance as soon as the stream resets to its initial configuration. Both the **age** and **zero** heuristics have shown already that they have are able to adapt to the changes in the stream better than the **org** heuristic, but this appears to result in them subsequently being unable to re-adapt to previous stream behaviours as quickly. In the case of **datgen-mirror**, all three heuristics exhibit similar gradual performance degradation and similar recovery curves as the underlying stream makeup reconverges on a single concept.

5.3.5 Evaluation of datgen with Tapped Delay Lines

In all configurations on all **datgen** datasets, performance with the addition of tapped delay lines hovers near the minimum (equivalent to the stochastic or single-class classifier). Particularly interesting, however, is the behaviour of the **age** heuristic on both extension patterns of the dataset. Both **age** and **zero** exhibit a sudden increase in detection rate as soon as the **-repeat** event occurs, but the performance of the **age** heuristic appears able to continue improving throughout the latter half of the stream in both the **-repeat** and **-mirror** cases. In addition, **age** is the only heuristic that remains capable of occasionally obtaining the perfect performance on the final C3 chunk of the stream that the population without access to tapped delay lines was able to achieve so frequently.

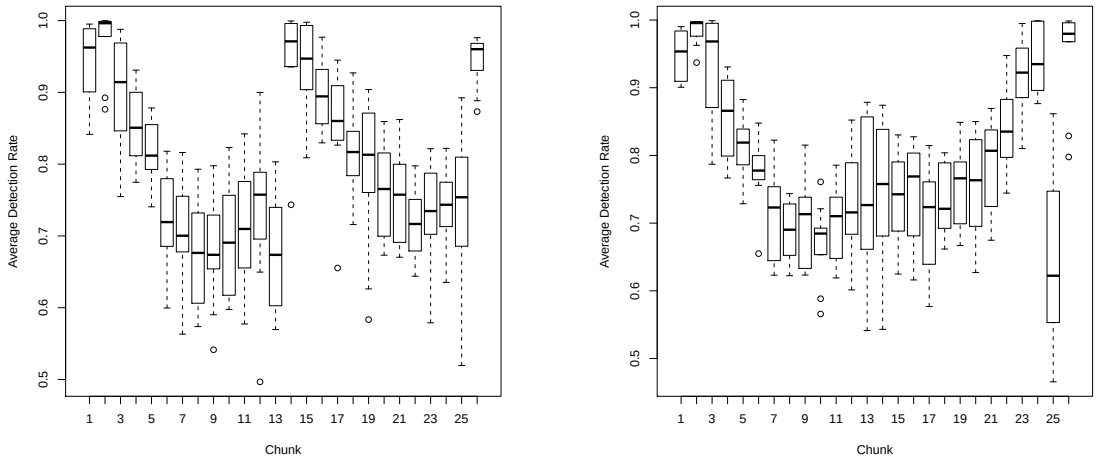
5.3.6 Evaluation of planar without Tapped Delay Lines

In the **org** case on the **planar-repeat** dataset, the return to initial stream behaviour causes the host population to experience a sharp decline in performance that is not shared by either the **age** or **zero** cases. Similar to the first time the behaviours were encountered, the **org** heuristic's performance varies widely and takes the majority of the length of the behaviour switch before it begins to reliably improve. Both the **age** and **zero** cases are again similar, with performance barely reaching the 70% avg. DR mid-switch and eventually petering out at the 50% rate (note that this is *not* due to pathological class imbalance as **planar** does not have a binary class distribution

– see Table 4.2). In the case of **planar-mirror**, all three fitness sharing heuristics appear to be able to maintain (but not improve on) the approximate detection rate they were able to attain before concepts began to reoccur.

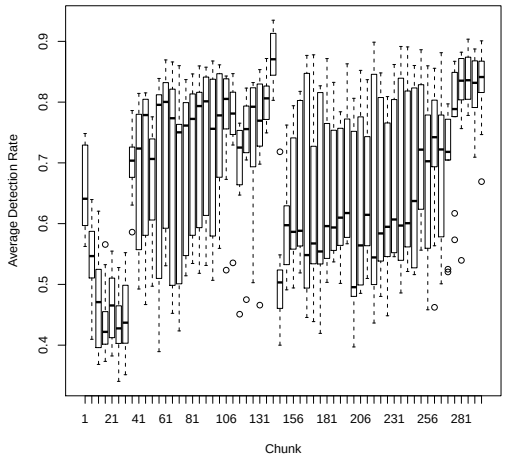
5.3.7 Evaluation of planar with Tapped Delay Lines

When tapped delay lines are provided to the teams under the **planar** dataset extensions, the performance again hovers near minimum for the **org** and **zero** cases. The **age** heuristic, however, is once again able to achieve significant performance improvement over the other two configurations, and indeed appears regularly able to classify two of the three classes instead of the single class detection that the other two maintain. Of particular interest here is that the **age** heuristic is occasionally able to reach unity in more than a quarter of all cases even in the first half of the stream, before the tapped delay line information has the chance to convey previously seen stream behaviour. Although the median performance in this case barely reaches the levels of the other configurations, the maximum performance is unparalleled by the other heuristics both with and without taps, making the combination of tapped delay lines with the **age** heuristic a promising candidate in scenarios where multiple runs can be performed on a data stream simultaneously.

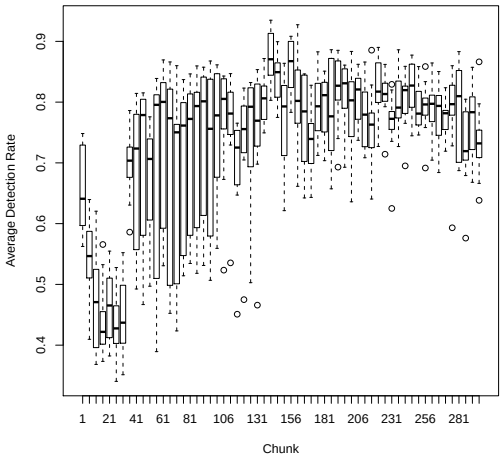


(a) datgen-repeat

(b) datgen-mirror

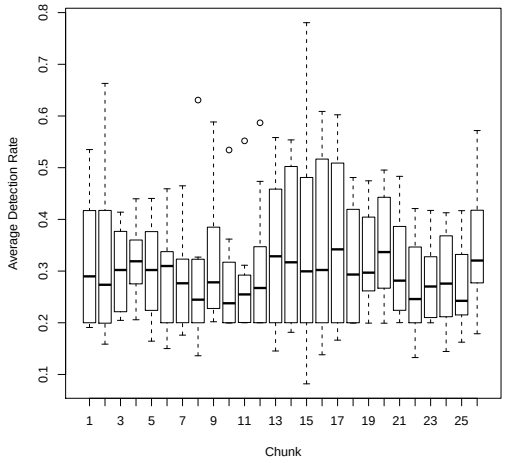


(c) planar-repeat

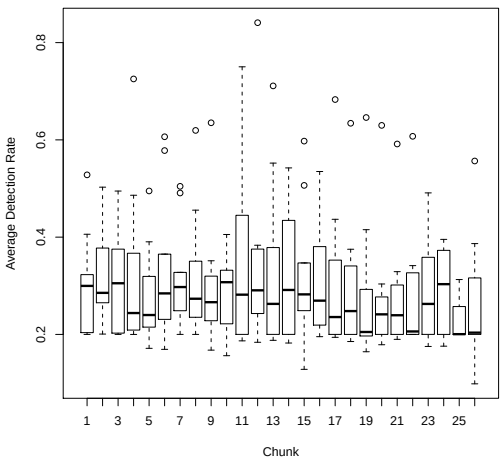


(d) planar-mirror

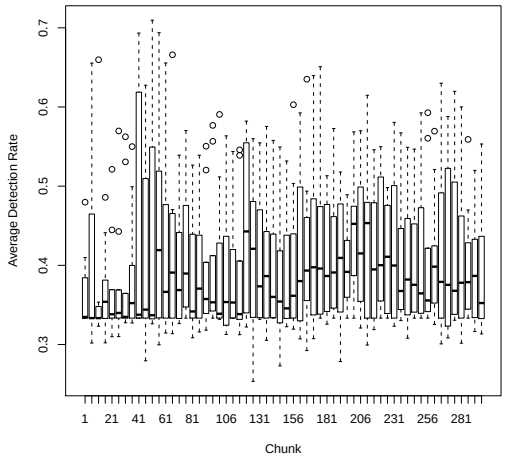
Figure 5.10: Average detection rate across previous chunks using **org** configuration without TDLs. Subcaption indicates data set.



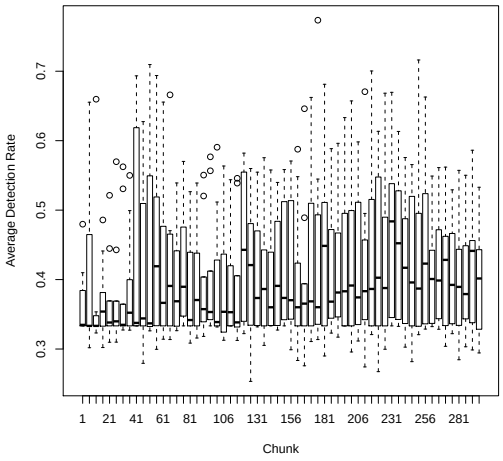
(a) datgen-repeat



(b) datgen-mirror

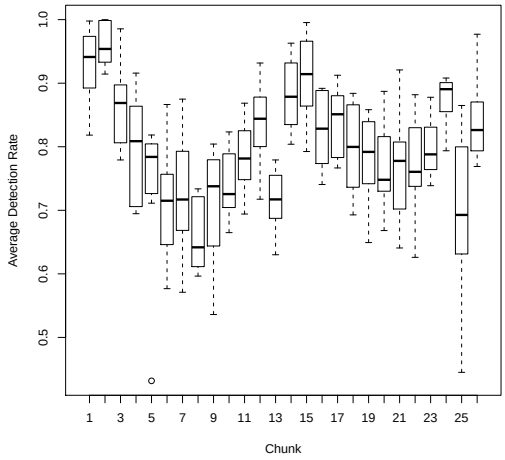


(c) planar-repeat

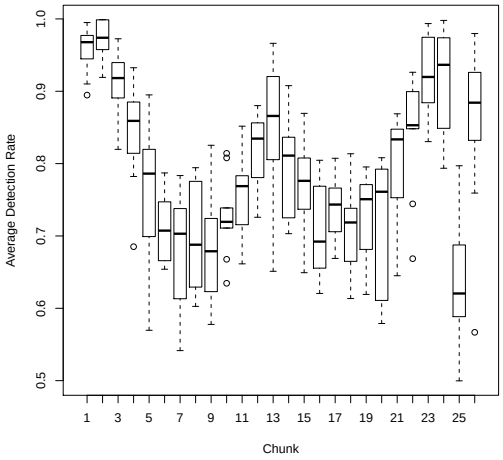


(d) planar-mirror

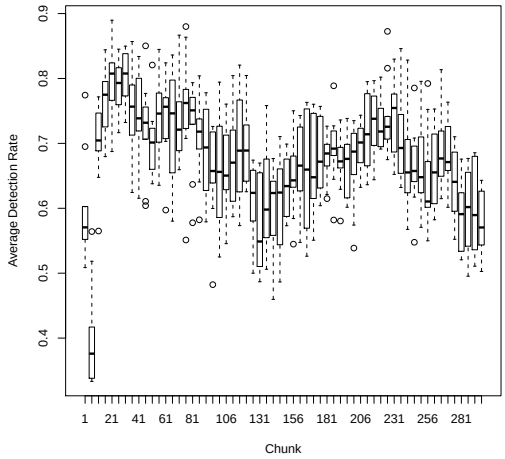
Figure 5.11: Average detection rate across previous chunks using **org** configuration with TDLs. Subcaption indicates data set.



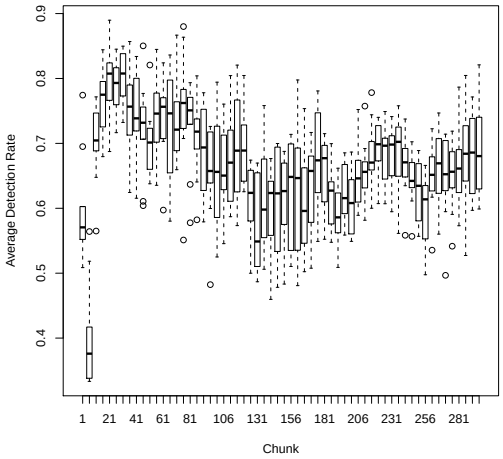
(a) datgen-repeat



(b) datgen-mirror

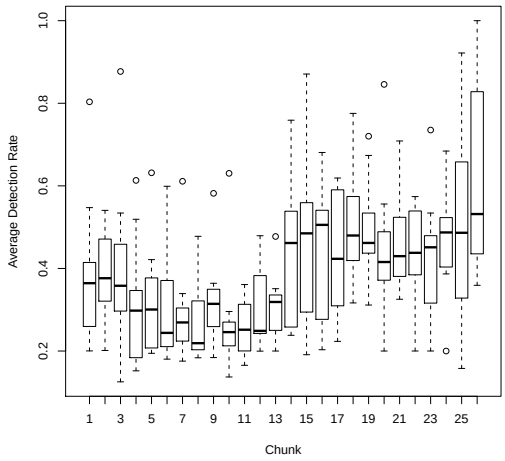


(c) planar-repeat

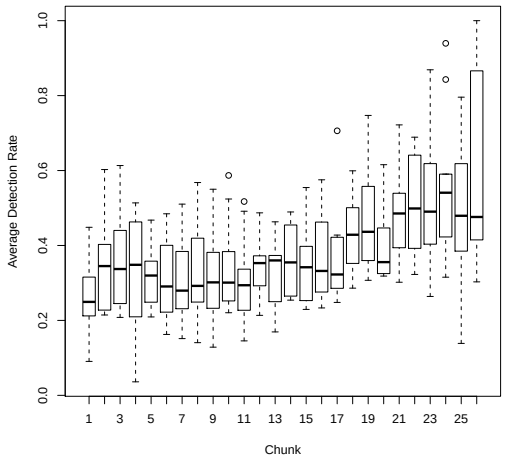


(d) planar-mirror

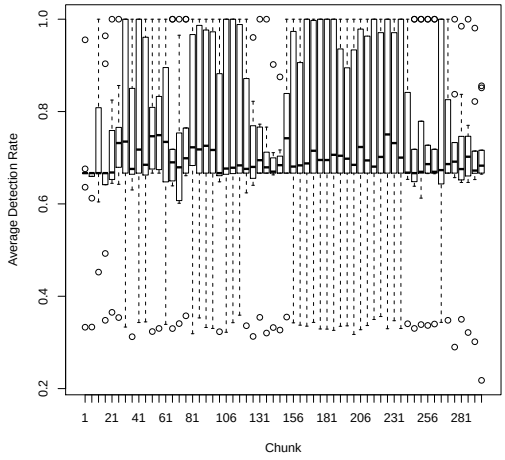
Figure 5.12: Average detection rate across previous chunks using **age** configuration without TDLs. Subcaption indicates data set.



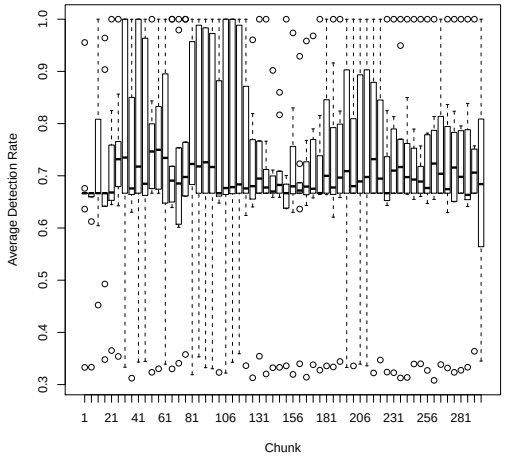
(a) datgen-repeat



(b) datgen-mirror

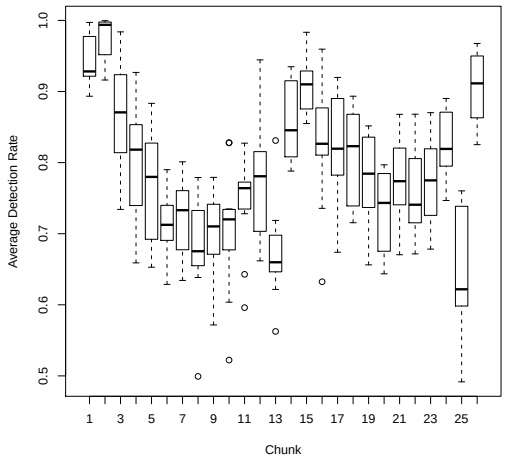


(c) planar-repeat

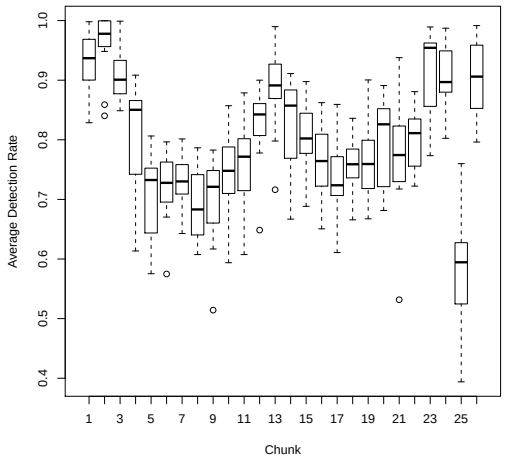


(d) planar-mirror

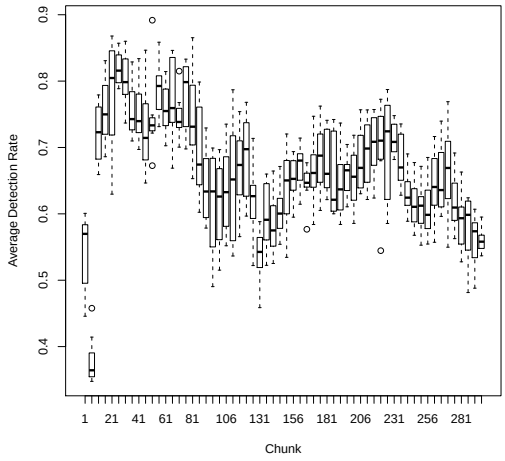
Figure 5.13: Average detection rate across previous chunks using **age** configuration with TDLs. Subcaption indicates data set.



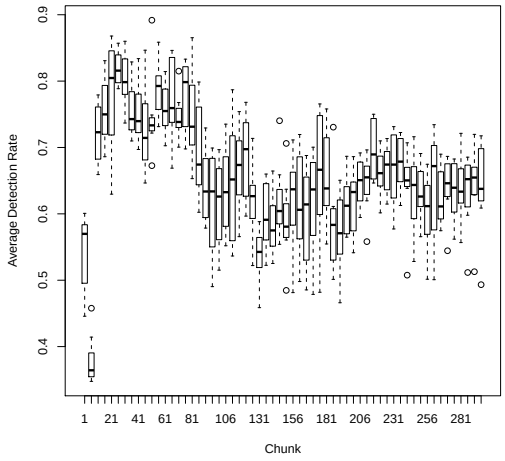
(a) datgen-repeat



(b) datgen-mirror

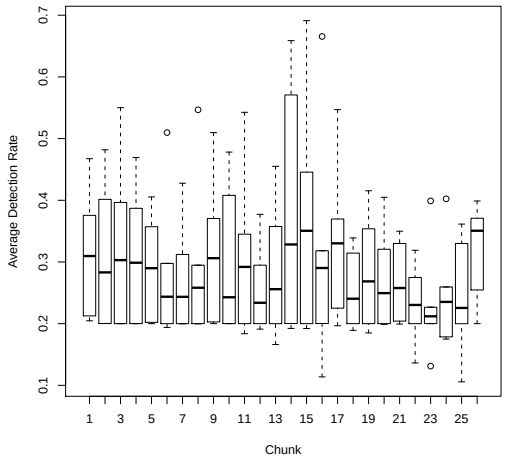


(c) planar-repeat

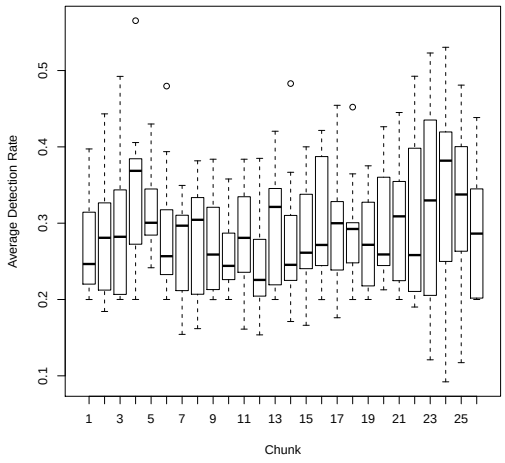


(d) planar-mirror

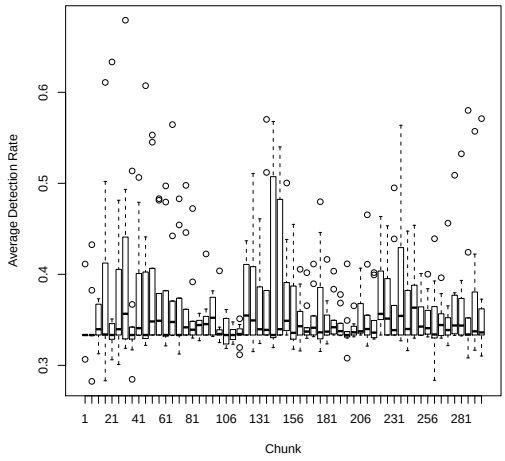
Figure 5.14: Average detection rate across previous chunks using **zero** configuration without TDLs. Subcaption indicates data set.



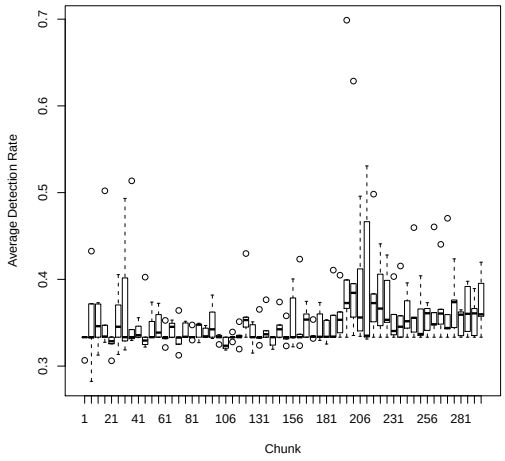
(a) datgen-repeat



(b) datgen-mirror



(c) planar-repeat



(d) planar-mirror

Figure 5.15: Average detection rate across previous chunks using **zero** configuration with TDLs. Subcaption indicates data set.

Chapter 6

Conclusions, Contributions, and Future Work

6.1 Conclusions

An initial study is conducted to assess the potential contribution of Pareto archiving under GP for a base case in which a regular classification task is limited to a single pass through the training partition under constraints consistent with streaming data i.e., the case of a stationary task (Section 5.1). From an application context the non-stationary scenario is of interest as it potentially represents a 'big data' scenario in which it is too expensive to perform multiple passes through the training data. A framework is considered in which a finite archive is enforced, under both Pareto and stochastic policies for archive maintenance. Such an archive is able to retain content independent from that of the sliding window, subject to a fixed size constraint. Three data sets are benchmarked under a common parameterization. Furthermore, a study is made of the impact of class noise, a property that all real world data sets will possess to some degree. Significant sensitivity appears to exist, with label error immediately resulting in a larger Pareto archive requirement, or at least when average detection rate represents the performance measure and the class distributions are imbalanced.[1] This in itself might provide the basis for dynamically sizing the Point population to avoid introducing replacement errors and therefore introducing coevolutionary pathologies into the GP population [14, 31, 9, 6]. Moreover, we note that at least two sources of noise can appear: labelling error (as investigated in Section 5.1), and attribute or sensor error.

With respect to the non-stationary data stream we note that the SBB algorithm provides various properties that are potentially advantageous under such a dynamic

[1]For example, in the case of the Shuttle data set, the experiment with 5% class noise rate resulted in a 30% avg. DR which achieves an accuracy of 75% in this case.

setting. For example, task decomposition is supported through speciation and bidding, whereas evolvability is addressed through the use of independent representations and variation operators for teams and programs. Pareto archiving provides a formal framework for retaining specific learners (GP teams) and the corresponding data points, or a memory mechanism. In order to test the capability of SBB under specific forms of non-stationary streaming data, two multi-class benchmarks are introduced. Specifically, the 'datgen' task assumes stepped changes to the underlying process whereas 'planar' adopts a continuously varying process. The resulting benchmarks indicate that diversity is more effective under the continuously varying process (Section 5.2). Conversely, introducing an additional age bias that deterministically retires the oldest non-dominated points is more effective in the non-stationary task with stepped changes.

As noted in Chapter 1, concept drift within data streams is not always a forward-marching process and can sometimes embed patterns of repetition or periodicity. The exact nature repetition can take many forms, and this work introduces two explicitly quantifiable patterns for extending non-cyclical datasets: 'repeat' and 'mirror'. A method of allowing classifiers to discern these temporal patterns is presented in the form of tapped delay lines, and the results seem to indicate that in combination with age-based heuristics it may be possible to achieve significant performance gains over the more basic, traditional sliding window formulation (Section 5.3).

6.2 Contributions and publications resulting from this work

1. A precisely defined methodology for generating arbitrary volume synthetic data sets suitable for use in benchmarking streaming classification algorithms is given, and a collection of data sets based on the methodology are created and made available for use. The data sets created can contain concept drift in a manner that is precisely quantified and explicitly tuneable. The datgen data set presents an environment in which discrete behaviours occur interchangeably at varying frequencies throughout the data stream, whereas the planar data set contains an environment in which data points 'smoothly' drift from being defined by one particular behaviour to being defined by another. In addition, two methods (-repeat and -mirror) are defined for introducing cyclical concept

drift to pre-existing non-cyclical data sets.

2. The adaptation of the GP framework used in this thesis to interface with temporally dependent data streams is generalizable to any traditional GP framework which is able to adopt the coevolutionary paradigm. The archiving and replacement policies (see Figure 3.1) used to control the Pareto archive can be implemented separately from the chosen GP framework, as the archive itself presents as a normal fixed-size training set (albeit with mutable content from generation to generation).

3. As performance evaluation is performed across the contents of an archive with fixed parameterized maximum size, the computational complexity of the training process is independent of the actual length / volume of the data stream.

4. Modifications to traditional fitness sharing heuristics to account for the temporal dependencies observed in streaming data environments are presented and their behaviour under discrete vs. continuous concept drift is quantified (see Section 6.1.

5. A mechanism for providing explicit memory of previous stream behaviour to individual classifiers at evaluation-time is given in the form of tapped delay lines, and their impact on classifier performance in various streaming environments with several archiving heuristics is explored.

Publications

At the time of this writing, portions of the work presented in this thesis have resulted in two distinct papers published in peer-reviewed conferences. The first of these [3] focuses on the work presented in Sections 4.1 and 5.1 in which the impact of Pareto archiving and the sliding window restriction on GP classifiers are explored, and can be found in the Proceedings of the ACM Genetic and Evolutionary Computation (ACM GECCO) 2012. The second paper [2] presents the streaming benchmark datasets defined in Section 4.2, and the fitness sharing heuristics and performance results of combinations thereof discussed in Section 5.2. This paper was published in the Proceedings of ACM GECCO 2013.

6.3 Future work

Future work could begin with a refined approach to the deployment of tapped delay lines. As noted in Section 3.4, the current incarnation of TDLs places no emphasis on using the actual attributes of a data point currently under examination over the historical data presented alongside it. This has a significant detrimental effect on the ability of the base case of the algorithm to classify non-cyclical data. Investigating an explicit bias or separation of past and present data in the GP representation should hopefully prove beneficial in counteracting this effect. Previous work has also suggested merit in evolving the parameters of the TDLs alongside the population to allow it to adapt to different environmental characteristics [41, 44, 27].

We are also interested in applying the GP framework developed in this work to active learning environments, in which data points can be accessed independent of their corresponding labels, and a cost is associated with requesting labels with the learning algorithm subjected to some limited labeling budget. It might be noted that with the parameterizations used (as enumerated in Appendix A) against the large size of the **datgen** dataset, the GP algorithm only ever accesses 10% of the data chosen stochastically. This is equivalent to employing a stochastic label request strategy with a fixed labeling budget of 10% and ignoring unlabeled data; the results obtained in Chapter 5 suggests our approach may be promising in this regard.

Other avenues of interest include decoupling the rate of the evolutionary cycle from the rate of data arrival from the stream. This work considered only data arriving in a linear fashion, whereas there are numerous contexts in which data arrives in a random or "bursty" manner. In these cases, more generations of training could be permitted during 'quiet' times of the stream, while the impact of proportionally high data arrival rates would need to be studied to evaluate the magnitude of the impact caused by fewer training generations. In such a context, change detection (see Chapter 1) would become even more important in order to bypass training generations when they are unnecessary. It may even be possible to introduce an articulated form of concept recognition to the algorithm, whereas the current framework uses a single archive and requires the archiving policy to serendipitously form its own representation of multiple concepts over time.

Finally, a significant next step to this work will be the application of the framework to real-world datasets. The scenarios the algorithms and mechanisms have been developed for are increasingly common in real world applications, and the interface of the Pareto archive to a sliding window means that the classification system can be deployed in real-time to arbitrary volume data streams with a fixed computational cost per time unit. Potential application domains include such varied areas as computer network data, utility and electrical grid usage, sensor network monitoring, and customer analytics.

Bibliography

[1] H. Abdulsalam, D. B. Skillicorn, and P. Martin. Classification using streaming random forests. *IEEE Transactions on Knowledge and Data Engineering*, 23(1):22–36, 2011.

[2] A. Atwater and M. I. Heywood. Benchmarking pareto archiving heuristics in the presence of concept drift: Diversity versus age. In *ACM Genetic and Evolutionary Computation Conference*, pages 885–892, 2013.

[3] A. Atwater, M. I. Heywood, and A. N. Zincir-Heywood. GP under streaming data constraints: A case for Pareto archiving? In *ACM Genetic and Evolutionary Computation Conference*, pages 703–710, 2012.

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

[5] P. Bruneau, F. Picarougne, and M. Gelgon. Incremental semi-supervised clustering in a data stream with a flock of agents. In *IEEE Congress on Evolutionary Computation*, pages 3067–3074, 2009.

[6] J. Cartlidge and D. Ait-Boudaoud. Autonomous virulence adaptation improves coevolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 15(2):215–229, 2011.

[7] J. Cartlidge and S. Bullock. Combating coevolutionary disengagement by reducing parasite virulence. *Evolutionary Computation*, 12(2):193–222, 2004.

[8] W. Cedeno and V. R. Vemuri. On the use of niching for dynamic landscapes. In *IEEE Congress on Evolutionary Computation*, pages 361–366, 1997.

[9] E. D. de Jong. A monolithic archive for pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[10] I. Dempsey, M. O'Neill, and B. A. *Foundations in Grammatical Evolution for Dynamic Environments*, volume 194 of *Studies in Computational Intelligence*. Springer, 2009.

[11] I. Dempsey, M. O'Neill, and A. Brabazon. Adaptive trading with grammatical evolution. In *IEEE CEC*, pages 2587–2592, 2006.

[12] J. A. Doucette, A. R. McIntyre, P. Lichodzijewski, and M. I. Heywood. Symbolic coevolutionary genetic programming: A benchmarking study under large attribute spaces. *Genetic Programming and Evolvable Machines*, 13:71–101, 2012.

[13] W. Fan, Y. Huang, H. Wang, and P. S. Yu. Active mining of data streams. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 457–461, 2004.

[14] S. G. Ficici and J. Pollack. Pareto optimality in coevolutionary learning. In *European Conference on Advances in Artificial Life*, pages 316–325, 2001.

[15] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. In *Parallel Problem Solving from Nature*, pages 312–321, 1994.

[16] A. Ghosh, S. Tstutsui, and H. Tanaka. Function optimization in non-stationary environment using steady state genetic algorithms with aging of individuals. In *IEEE Congress on Evolutionary Computation*, pages 666–671, 1998.

[17] J. J. Greffenstette. Genetic algorithms for changing environments. In *Proceedings of Parallel Problem Solving from Nature*, pages 137–144, 1992.

[18] I. Guyon and A. Elisseeff. An introduction to feature extraction. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors, *Feature extraction, Foundations and applications*, volume 207 of *StudFuzz*, chapter 1. Springer, 2006.

[19] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[20] J. O. S. III. *Physical Audio Signal Processing*. W3K Publishing, 2010.

[21] M. Kotanchek, G. Smits, and E. Vladislavleva. Exploiting trustable models via pareto GP for targeted data collection. In *Genetic Programming Theory and Practice VI*, pages 145–162. Springer, 2009.

[22] J. R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT, 1990.

[23] M. Lemczyk. Pareto-cevolutionary genetic programming classifier. Master's thesis, Faculty of Computer Science, Dalhousie University, 2006. http://web.cs.dal.ca/˜mheywood/Thesis/MCS.html.

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

[25] P. Lichodzijewski and M. I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *ACM Genetic and Evolutionary Computation Conference*, pages 363–370, 2008.

[26] P. Lichodzijewski and M. I. Heywood. Symbiosis, complexification and simplicity under gp. In *ACM Genetic and Evolutionary Computation Conference*, pages 853–860, 2010.

[27] A. Loginov and M. I. Heywood. On the impact of streaming interface heuristics on GP trading agents: An FX benchmarking study. In *ACM Genetic and Evolutionary Computation Conference*, pages 1341–1348, 2013.

[28] A. R. McIntyre and M. I. Heywood. Cooperative problem decomposition in pareto competitive classifier models of coevolution. In *European Conference on Genetic Programming*, pages 289–300, 2008.

[29] A. R. McIntyre and M. I. Heywood. Classification as clustering: A Pareto cooperative-competitive GP approach. *Evolutionary Computation*, 19(1):137–166, 2011.

[30] R. W. Morrison. *Designing evolutionary algorithms for dynamic environments*. Springer, 2004.

[31] J. Noble and R. A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. In *ACM Genetic and Evolutionary Computation Conference*, pages 493–500, 2001.

[32] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary automatic programming in an arbitrary language*. Springer, 2003.

[33] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3):339–363, 2010.

[34] A. Orriols-Puig, J. Casillas, and E. Bernado-Mansilla. First approach toward on-line evolution of association rules with learning classifier systems. In *ACM Genetic and Evolutionary Computation Conference*, pages 2031–2038, 2008.

[35] J. Quiñonero-Candela and M. Sugiyama and A. Schwaighofer and N. D. Lawrence. *Dataset shift in machine learning*. MIT, 2009.

[36] V. Ramos and A. Abraham. Swarms on continuous data. In *IEEE Congress on Evolutionary Computation*, pages 1370–1375, 2003.

[37] J. Reisinger, K. O. Stanley, and R. Miikkulainen. Towards and empirical measure of evolvability. In *ACM Genetic and Evolutionary Computation Conference*, pages 257–264, 2005.

[38] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evol. Comput.*, 5(1):1–29, 1997.

[39] D. Saad, editor. *On-line learning in neural networks*. Cambridge University Press, 1998.

[40] H. A. Simon. *The sciences of the artificial*. MIT Press, 2nd edition, 1996.

[41] S. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training genetic programming on half a million patters: An example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, 2005.

[42] M. Sugiyama and M. Kawanabe. *Machine learning in non-stationary environments*. MIT, 2012.

[43] A. Teller and M. Veloso. Pado: A new learning architecture for object recognition. In K. Ikeuchi and M. Veloso, editors, *Symbolic visual learning*. Oxford University Press, 1995.

[44] N. Wagner, Z. Michalewicz, M. Khouja, and R. R. McGregor. Time series forecasting for dynamic environments: The DyFor genetic program model. *IEEE Transactions on Evolutionary Computation*, 11(4):433–452, 2007.

[45] S. X. Wu and W. Banzhaf. Rethinking multilevel selection in genetic programming. In *ACM Genetic and Evolutionary Computation Conference*, pages 1403–1410, 2011.

[46] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 40(6):1607–1621, 2010.

Appendix A

Parameterizations

This section presents parameterizations that are not directly relevant to the construction of the experimental methodologies discussed in Chapter 5, and – where appropriate – discusses the motivation for their values. Section A.1 contains the parameters used for the GP framework this thesis builds off of, and Section A.2 discusses the motivation behind the configuration of the tapped delay lines explored in Section 5.3.

A.1 SBB Parameterizations

There are several parameters supported by the underlying SBB framework used for the tasks in this work. Except when otherwise specified for particular experimental configurations in Chapter 5, the parameterizations used for SBB are shown in Table A.1. These settings are taken directly from previous work on SBB, and no parameter tuning was performed to adjust the algorithm to particular datasets contained herein.

A.2 TDL Parameterizations

There are two primary parameters used to characterize the tapped delay lines as they occur in this work (defined in Section 3.4). A visual representation of these parameters can be found in Figure 3.2.

Tap delay length, σ: The number of data points that are allowed to elapse before new data points begin to be appended to previous ones. In all experimental configurations involving TDLs this is set to be equivalent to the width of one sliding window, i.e., $\sigma = w$. This is done so that the case of aggregate data points duplicating data seen elsewhere does not have a confounding effect on analysis of the results (when $\sigma < w$), while also not being so large that the total amount of saved data $\sigma \cdot \sigma_n$ exceeds the length of the data stream.

Table A.1: SBB parameterization.

	Description	Value
P_{size}	Point population size.	120
M_{size}	Team population size.	120
t_{max}	Default number of generations.	30 000
	. . . on **planar-{repeat,mirror}**	60 000
	. . . on **datgen-{repeat,mirror}**	57 857
p_d	Probability of learner deletion.	0.7
p_a	Probability of learner addition.	0.7
μ_a	Probability of action mutation.	0.1
ω	Maximum team size.	20
P_{gap}	Point generation gap.	20
M_{gap}	Team generation gap.	60

Tap delay count, σ_n: The number of taps that can be prepended to a point before new taps begin to overwrite the oldest ones. This is set to $\sigma_n = 7$ in all experimental configurations, due to the serendipitous fact that the current number of operators and registers defined in the underlying GP representation resulted in 3 bits being left for use before the opcodes spilled over the boundary of a 16 bit word.