Coevolutionary Bid-Based Genetic Programming for Problem Decomposition in Classification*

Peter Lichodzijewski Malcolm I. Heywood

July 1, 2008

Abstract

In this work a cooperative, bid-based, model for problem decomposition is proposed with application to discrete action domains such as classification. This represents a significant departure from models where each individual constructs a direct input-outcome map, for example, from the set of exemplars to the set of class labels as is typical under the classification domain. In contrast, the proposed model focuses on learning a bidding strategy based on the exemplar feature vectors; each individual is associated with a single discrete action and the individual with the maximum bid 'wins' the right to suggest its action. Thus, the number of individuals associated with each action is a function of the intra-action bidding behaviour. Credit assignment is designed to reward correct but unique bidding strategies relative to the target actions. An advantage of the model over other teaming methods is its ability to *automatically* determine the number of and interaction between cooperative team members.

The resulting model shares several traits with learning classifier systems and as such both approaches are benchmarked on nine large classification problems. Moreover, both of the evolutionary models are compared against the deterministic Support Vector Machine classification algorithm. Performance assessment considers the computational, classification, and complexity characteristics of the resulting solutions. The bid-based model is found to provide simple yet effective solutions that are robust to wide variations in the class representation. Support Vector Machines and classifier systems tend to perform better under balanced datasets albeit resulting in black-box solutions.

1 Introduction

Divide-and-conquer appeals to the intuition that solving a problem through decomposition will facilitate the search process and make the ensuing solutions more transparent. However, despite the utility of a population-based search mechanism, canonical Genetic Programming (GP) provides a solution in the form of a single super-individual [27]. That is to say,

*Originally published in *Genetic Programming and Evolvable Machines*, http://www.springer.com/computer/artificial/journal/10710.

schema theorems in their many forms predict that populations will evolve towards a single candidate solution [28]. Models that attempt to support a divide-and-conquer approach to learning under the GP paradigm generally fall into one of two forms: the hierarchical 'module acquisition' methodology in which code reuse within candidate solutions is the goal [1, 27] or some form of team behaviour [7, 45]. We are interested in the latter, that is to say, situations where multiple individuals from the same population are evolved to 'cooperate' or decompose the original problem domain into a set of interacting behaviours that collectively solve the problem as originally posed.

The problem domain of interest is classification where a supervised learning context with binary feedback (correct or incorrect) is assumed. A central tenet unique to the 'team' based methodology – as opposed to the canonical GP approach to classification – is the mechanism for establishing problem decomposition. Earlier GP approaches for building teams [7, 45, 47] have relied on the explicit specification of the individuals appearing in a team. An individual in the population was therefore viewed as a team consisting of several member classifiers. This makes the process of credit assignment between team members during evolution straightforward but at the natural disadvantage of assuming sufficient information to make the decision regarding team memberships.

In this work a different view is taken to the goal of learning under a discrete supervised domain. Instead of requiring individuals to learn a mapping between (input) exemplar and a (discrete) target action, we require individuals to evolve a relation between (input) exemplar and a bid (a real-valued scalar over the unit interval). Each individual has a single scalar action assigned at initialization from the set of actions defined by the domain (e.g., possible class labels), and the individual with largest bid wins the right to specify its predefined action[1]. Thus intra-action behaviours are explicitly supported without an *a priori* specification of the number of individuals of each action comprising the team.

This work also considers the issue of GP scalability on very large datasets. To this end, the bid-based cooperative mechanism is integrated with a Pareto-based competitive coevolutionary paradigm [14, 16, 39]. Pareto-based coevolution models the interaction between learners (bidding individuals in this case) and test points (training exemplars) as a competitive game. The proposed approach exploits this interaction so that it is not necessary to evaluate learner fitness over the entire training dataset, but rather, over a fixed subset of exemplars that are able to discriminate between the learners. In this way, the computational overhead required during learning is decoupled from the size of the training dataset. Such a model has been demonstrated to significantly reduce the computational overhead which has traditionally limited GP to small datasets [33, 35, 36].

The proposed approach, denoted Bid GP (BGP), shares similarities with Learning Classifier Systems (LCS) [20]. In particular, both approaches decompose a problem by selecting a subset of the population to act at a given time step. However, whereas classifier systems typically evolve condition-action-strength rules the proposed approach replaces condition and strength with a single bid procedure. This results in deterministic individual behaviour because, in contrast to strength, bid values are not adapted during an individual's lifetime. Furthermore, in BGP the subset selected to act always consists of a single individual facilitating credit assignment. Although the proposed approach does borrow from XCS [51] in

[1]Hereafter, we will use 'class' and 'action' interchangeably.

using accuracy as a component of individual fitness, the way in which this accuracy affects the individual's survival, and in general the proposed training algorithm, is very different.

In the following we first provide a review of related work, Section 2, and then develop the BGP model of teaming, Section 3. Section 4 conducts an extensive empirical evaluation of BGP under the classification domain (i.e., a problem domain with a discrete set of behaviours) using nine large datasets that include unbalanced class distributions as well as a varying number of classes. Given the similar teaming properties of classifier systems (bid-based action selection, no *a priori* specification of the cooperating individuals), we benchmark BGP against a real-valued XCS classifier system in addition to the second-order Support Vector Machine (SVM) implementation LIBSVM [12]. We find that relative to both XCS and SVM, the ensuing BGP solutions are much more transparent as the XCS model typically utilizes the entire population in the 'team' while the support vectors in the SVM models number in the hundreds and tens of thousands. Moreover, BGP decompositions appear to be correlated with the complexity of the data and are effective at resisting class imbalances as a whole.

2 Related Work

The approach proposed in this work specifically addresses the concept of automated problem decomposition through a scalable coevolutionary bid-based model of credit assignment. As such, we review literature appropriate to both problem decomposition and scaling GP to larger classification problem domains. In particular, Section 2.1 considers specific attempts to provide problem decomposition within GP, whereas Section 2.2 reviews the learning classifier system paradigm in which problem decomposition is an inherent property of the model. Section 2.3 summarizes recent approaches for scaling GP to larger datasets, and Section 2.4 provides background to the competitive coevolutionary paradigm utilized in this work. Recent results from the more general cooperative agent-based domain are considered in Section 2.5, and a summary of the various tradeoffs between different paradigms is provided in Section 2.6.

2.1 Problem Decomposition in Genetic Programming

As indicated in the introduction, this work focuses on problem decomposition by means of multiple GP-based individuals learning to assign different individuals to different parts of the dataset / environmental conditions.

Ensemble approaches compose solutions from multiple individuals where each individual represents a complete solution. Problem decomposition emerges when ensemble members make non-overlapping errors; a single member may thus make the wrong decision provided that it is masked by other members that make the right decision. As such, a voting scheme is required to resolve cases where these base classifiers do not agree. Boosting and bagging [4] are two established machine learning ensemble methods that have been used with GP [17, 21]. They modify the training dataset in order to achieve variation in the ensemble members.

Other approaches rely on the stochastic nature of GP to generate diversity within the

ensemble. Island-based approaches such as N-Version GP [23] compose ensembles by selecting individuals from independently evolved populations. Team approaches [7, 45], on the other hand, evolve groups of associated classifiers evaluating each team as a unit. Whereas the teaming approach were found to produce individual members that cooperated well but performed poorly in isolation [45], island approaches evolved individuals that were relatively fit but whose errors tended to be correlated [23]. As such, little benefit was gained from combining individuals into teams in the island approach. This result suggests that combining arbitrarily selected individuals that were independently evolved is not sufficient to consistently provide cooperative behaviour and that the algorithm needs to recognize this and explicitly promote cooperation (e.g., by recognizing teams of individuals). Orthogonal Evolution of Teams [47] has been proposed to evolve fit individuals that cooperate well by applying pressure both to teams as a whole and to their individual members.

Other approaches borrow from Evolutionary Multi-Objective Optimization to explicitly encourage problem decomposition. The competitive Pareto-Coevolutionary GP Classifier (PGPC) [33] recognizes that a single individual may not be able to perform well on all problem instances. As such, it considers the performance of each individual on each problem instance separately and searches for a solution in terms of a non-dominated set of individuals. As with most ensemble methods, it requires a post-training voting scheme to provide a single class label on unseen data. Finally, whereas the typical approach to classification is to construct models that discriminate between classes, the multi-objective approach of [36, 37] encourages classifiers to act as novelty detectors. Such a model is explicitly cooperative resulting in individuals being explicitly rewarded for finding non-overlapping behaviours that solve the larger (classification) problem. Although this approach does not use a voting scheme, a clustering step in the inner loop of fitness evaluation increases the computational overhead.

2.2 Learning Classifier Systems

LCS [20] were proposed as an approach to learning by interacting with an environment. Specifically, the system was designed to interact with the environment through a set of detectors and effectors so as to maximize reward in the long run. In the *Michigan* approach, a population of classifiers is evolved where each classifier is a condition-action rule with an associated strength value that estimates the reward expected when, given that the condition is satisfied, the classifier's action is taken. Each individual in this population represents a solution sub-component that can only be applied in the context of all the other individuals. In the original LCS formulation, the conditions were composed of strings over the alphabet $\{0, 1, \#\}$.

Since their initial introduction, there has been a lot of research done in the field of LCS resulting in many alternate forms of the approach [29]. Here, the focus will be on Wilson's Michigan-style XCS [51]. This formulation appears to be particularly significant because of its use of an accuracy-based fitness measure and a niche-based GA. As such, the approach was shown to produce classifiers that are both accurate and general (i.e., use as many wildcard symbols in the conditions as is possible). A concise description of the XCS data structures and algorithm is provided in [10]. Versions of XCS that can support real-valued conditions have been suggested [46, 53], and although XCS can be applied to classification tasks directly

[3, 53], improved performance can be achieved by assuming a supervised context [5].

The way in which the BGP team members interact is analogous to the way in which Michigan-style classifiers interact: the BGP approach selects the action associated with the highest bid and Michigan-style classifiers select the action associated with the highest prediction. In this way, the bid component in the BGP model and the condition and prediction of Michigan-style classifiers serve the same function. This is the reason for including XCS in the later empirical evaluation, Section 4.

2.3 Genetic Programming on Large Classification Problems

The number of fitness evaluations that have to be performed in a typical run of GP is a product of the population size, the number of iterations, the number of runs, and the size of the training dataset. For typical parameter settings and datasets of even moderate size the amount of time required to train becomes prohibitive. Settings for the first three parameters are often required to fall within a certain range to guarantee solutions of quality and therefore they cannot be significantly reduced. The last factor is problem specific and limits canonical GP to classification problems with relatively few training instances. This limitation was recognized in early work where a dataset consisting of 3772 training instances was considered too large to be manageable [18]. Today, even with considerable hardware advances, problem domains exist (e.g., intrusion detection) where GP cannot be directly applied due to excessive training times.

One way to address the problem of the high computational overhead is to use specialized hardware [17, 22, 25]. In contrast, the approach pursued in this work attempts to solve the problem by reducing the number of fitness evaluations. In particular, instead of using all exemplars, the idea is to evaluate the population on a *subset* of the entire training dataset on each iteration of the algorithm [18]. In one class of such algorithms, active learning, the performance of individuals on specific training instances can be used to bias the selection of these instances into the subset.

Different algorithms have been proposed that fall within this active learning paradigm. By measuring exemplar difficulty, Dynamic Subset Selection (DSS) [18] biases selection in favour of exemplars that tend to be misclassified by the population. To make sure that all instances are eventually seen by the system, selection based on age is occasionally performed. In order to extend this paradigm to datasets that cannot be entirely cached in memory, hierarchical versions of DSS have also been proposed [13, 44]. These algorithms first partition the dataset into blocks that fit entirely in memory then limit subset selection to blocks that have already been cached in order to take advantage of faster read speeds. The balanced block version of the algorithm [13] further refines selection to handle highly unbalanced class distributions.

In [30], information about the problem structure is collected in an undirected weighted graph and used to guide subset selection. The approach was shown to be effective, however, it still requires significant overhead to maintain and use the graph data structure. In particular, an $\Omega(N^2 log N^2)$ sorting step is required on a dataset of size N to rank fitness cases for selection.

Coevolution has also been used to reduce the number of fitness evaluations by evolving the training cases alongside GP individuals. The active learning approach [2] associates a subset with each individual and grows this subset during training. Intuitively, one would expect a

larger subset to be more effective at distinguishing between progressively fitter individuals. More recently, Pareto-coevolution has been used for subset selection [33, 36]. Here, training instances are selected based on the distinctions that they make (as described in Section 2.4) in order to collectively provide an accurate evaluation of the coevolved individuals.

2.4 Ideal Evaluation and Pareto-Coevolution

The concepts of Pareto-dominance and distinctions [16, 39] were used in the Delphi system [14] to approximate an ideal evaluation function using a reduced number of tests. This coevolutionary framework evolves a population of learners against a population of tests. The approach centers around an interaction function $G(l_i, t_k)$ which returns values from an ordered set and indicates the outcome of applying learner l_i to test t_k (revealing information about the underlying problem objectives).

If $\mathbf{v_1}$ and $\mathbf{v_2}$ are two objective vectors then the Pareto-dominance relation $dom(\mathbf{v_1}, \mathbf{v_2})$ which indicates that $\mathbf{v_1}$ dominates $\mathbf{v_2}$ is defined as:

$$dom(\mathbf{v_1}, \mathbf{v_2}) \Leftrightarrow \forall q : \mathbf{v_1}[q] \geq \mathbf{v_2}[q] \wedge \exists q : \mathbf{v_1}[q] > \mathbf{v_2}[q]. \tag{1}$$

Delphi defines an objective or outcome vector for the ith learner l_i over all the tests as

$$\mathbf{o_{l_i}}[k] = G(l_i, t_k) \tag{2}$$

where t_k is the kth test. The Pareto-dominance relation applied to these outcome vectors, referred to as the coevolutionary evaluation function, is then used to determine which learners are discarded from the population.

If the number of learners is n, the approach also constructs an n^2-dimensional distinction vector $\mathbf{d_{t_k}}$ for each test t_k as

$$\mathbf{d_{t_k}}[n \cdot i + j] = \begin{cases} 1 & \text{if } G(l_i, t_k) > G(l_j, t_k) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $1 \leq i, j, \leq n$ index the learners. Hence, a distinction is made if the outcome on t_k for learner l_i is *strictly* greater than the outcome for learner l_j. On the test side, these distinction vectors are used to determine which tests are discarded from the test population, again using Pareto-dominance.

An ideal evaluation function, which determines whether learner l_i is superior to learner l_j, is defined using the Pareto-dominance relation on the (typically unknown) underlying objectives. With the above setup and a test population of size $n^2 - n$, the coevolutionary evaluation function is capable of representing this ideal evaluation function *exactly* because $n^2 - n$ is an upper bound on the number of distinctions that can be made between the n members of the learner population. Such a Complete Evaluation Set of tests may not be necessary for successful coevolution and the size of the test population relative to the size of the learner population may be reduced further. This serves as motivation for this work where the goal is to reduce the GP training overhead.

2.5 Collective and Multi-Agent Models

Collective problem solving naturally implies problem decomposition and as such has received a lot of interest from the wider research community. One recent example of this is the COIN framework [48] where the goal is to provide learnable local utility functions that when maximized also maximize the global utility function G. One such family of functions derived using the COIN framework are difference utility functions. If ζ is the joint action of all agents, the difference utility of agent η is

$$DU_\eta = G(\zeta) - \Gamma(f(\zeta)) \tag{4}$$

where $\Gamma(f)$ is a utility independent of η's action. Intuitively, the goal of the second term is to eliminate noise caused by the actions of the other agents improving the 'learnability' of the overall utility. Specific instances of the difference utility function include the Aristocrat Utility in which Γ is designed to reflect η's expected behaviour, and the Wonderful Life Utility in which Γ reflects ζ but without η's contribution [48]. Due to the iterative nature of the BGP training algorithm, where a complete solution does not exist until after the last training iteration, difference utilities cannot be directly calculated as there is no way to determine $G(\zeta)$ and $\Gamma(f(\zeta))$. Resolving this remains a challenge for future research.

Multi-agent models of problem solving also provide a natural metaphor for problem decomposition [41]. This body of work covers a wide range of properties, however, a specific case of particular relevance to the proposed approach is the case of coevolutionary multi-agent behaviours [42]. The coevolutionary multi-agent model assumes the basic cooperative coevolutionary framework of [43] in which an independent population exists for each action. The number of actions, and hence the number of individuals cooperating in a team, is thus predefined. A central design decision rests in establishing who should interact with an individual at each 'evaluation', thus, fitness is a function of the combination of payoffs from one or more evaluations. Game theory may be employed for establishing the design for such interactions, however, computational considerations limit the number of combinations that may be considered, adding noise to the evaluation. Moreover, pathologies exist, such as disengagement and focusing (also a problem with competitive coevolution) and relative over-generalization. The latter corresponds to a tendency for credit assignment to reward individuals that assume a 'jack-of-all-trades' policy; this may or may not represent an 'optimal' model of collaboration for the problem domain in question. Solutions to date require an estimator for the 'optimal collaborator' where this is naturally problem dependent.

2.6 Summary

Relative to the research summarized in Section 2.1, we note several similarities. Early metaphors for GP teaming worked well within the confines of the team but perform poorly on an individual basis [7, 45], an example of relative over-generalization. Conversely, it is apparent that the recent Orthogonal Evolution of Teams model [47] may address the relative over-generalization issue through the utilization of search operators that take a structured approach to 'mixing' between evaluation or team member selection. Conversely, in the proposed bid-based model, all the diversity is established from a single population, thus we rely on the appropriate formulation of a fitness function that is able to reward (correct) unique

behaviours in isolation. If successful in this goal, the bid based model will avoid the limitation of a one-model-per-action paradigm – implicit in the coevolutionary multi-agent model – while also avoiding the pathological tendency of previous GP teaming metaphors towards producing either weak but cooperative behaviours, or strong but correlated behaviours. In addition, the bid-based model is to work in conjunction with the competitive coevolutionary component in order to scale to problem domains described in terms of a large number of exemplars (states). Such a constraint precluded the earlier 'wealth' based formulation for bid-based GP, and currently has not been explicitly demonstrated, to the authors knowledge, under the COIN framework.

3 Bid GP

BGP assumes a discrete problem domain limiting the set of actions represented by the population to a finite, discrete, set of behaviours. We assume the classification domain for illustrative purposes in the following description as well as in the empirical evaluation, Section 4.

As in the Delphi system, a test population and a learner population are competitively coevolved. The test population indexes a subset of the entire training dataset[2]. The learner population consists of a set of bidders whose behaviour is evolved using GP resulting in the cooperative coevolutionary behaviour. Specifically, each learner defines a *bid* and an *action*. The bid is represented as a program while the discrete action is defined *a priori* from the set of possible class labels. Together, the goal of the learners is to correctly classify the tests. The goal of the tests is to accurately distinguish between the learners.

In order to deploy the model post training each learner in the final solution executes its bid program on the input feature vector and submits an associated bid. The highest bidder is selected as the winner and the action of this winner is then output as the label for the that exemplar. As the final solution may contain multiple learners with the same action, the interaction between the bidding behaviours of these learners results in a cooperative intra-action policy in addition to the inter-action cooperation of learners of different actions.

Various formulations of the bid process have been proposed [34]. However, in order to provide a simple interface to the competitive coevolutionary paradigm, thus scaling to larger datasets, we base credit assignment on a first-past-the-post model [35]. In particular, the proposed model has no concept of 'wealth' relying instead on a normalized outcome vector that penalizes degenerate behaviours. In the following two subsections, an overview of the system is given followed by a detailed description of the specific algorithm steps.

3.1 Coevolutionary Bid-Based Learning Algorithm: Overview

Unbalanced class distributions may adversely affect training by favouring learners based only on their action. For this reason, the learner population is partitioned so that selection and search operators are applied to learners with the same action. In addition, since distinctions between learners of different actions are not meaningful in this setup, a separate coevolutionary test population is maintained for each learner partition during training. As such,

[2] As opposed to the test data employed for assessing the generalization performance of the trained classifier.

1. $S = \emptyset$
2. for each $a \in L$
   (a) initialize(L_{pop}, a)
   (b) initialize(T_{pop})
   (c) while not max generation
      i. L_{pop} = generate(L_{pop})
      ii. T_{pop} = generate(T_{pop})
      iii. for each $t_k \in T_{pop}$
         A. bid(L_{pop}, t_k)
      iv. normalize(L_{pop})
      v. L_{pop} = select(L_{pop})
      vi. T_{pop} = select(T_{pop})
   (d) L_{pop} = reduce(L_{pop})
   (e) $S = S \cup L_{pop}$

Figure 1: Generic coevolutionary training algorithm. S is the final solution output by the system, L_{pop} and T_{pop} are the learner and test populations, and L is the set of exemplar class labels.

for a given action, a learner partition and its associated test population are independent of any other partition and population. To simplify implementation the learner populations can therefore be trained in series[3] and this is the approach taken here, Figure 1.

The algorithm returns a solution set of learners S which is initialized to be empty in Step 1. Training then iterates on a class by class basis, Step 2. The learner population L_{pop} is initialized with random bid programs and action matching the current class a under consideration, Step 2a. The test population T_{pop} is initialized to contain non-duplicated indices into the training dataset, Step 2b. The learner and test populations are then evolved in a series of generations, Step 2c. Here, in contrast to the learners which already exist at the beginning of each generation, new learners are those members explicitly generated in Step 2(c)i (analogous terminology is used for tests). Once training of the learners for the current action a is complete, some individuals in the population may be deemed redundant and eliminated by means of a reduction procedure, Step 2d. The learners in the reduced population are added to the solution set S in the final step, Step 2e.

At the start of each generation L_{pop} and T_{pop} are augmented with the new learners and tests, Steps 2(c)i and 2(c)ii respectively, after which each learner in L_{pop} is applied to each test in T_{pop} in a series of bidding rounds, Step 2(c)iiiA. During a bidding round each learner l_i has its outcome on test t_k calculated and recorded in an associated outcome vector $\mathbf{o_{l_i}}$. Once the bidding rounds are completed, the outcome vectors are normalized in Step 2(c)iv.

[3]Compared to a parallel implementation, training overhead increases minimally as Step 2(c)ii in Figure 1 is executed $|L|$ times more often where L is the set of class labels.

Using these normalized outcome vectors, the algorithm determines if a new learner should remain in the population, Step 2(c)v. A new learner can remain in the population only by displacing an old learner introduced in an earlier generation: this will happen if the outcome vector of the new learner dominates the outcome vector of the old learner, Eq. 1. Similarly these outcome vectors are used to calculate the distinctions which determine whether any of the new tests should remain in the population, Step 2(c)vi.

A normalization step was performed to reduce the contribution of the outcome values for learners if they showed a tendency towards a degenerate bidding behaviour. For example, on tests not matching its action, a learner could achieve maximum reward by always bidding as low as possible. Since Pareto-dominance is used for selection, these learners would otherwise persist in the population; a more discriminating (and more desirable) learner may not have evolved an effective policy for bidding on the out-of-class tests and thus never dominate such a degenerate that maximizes 'reward' by always bidding the minimum. The normalization step is one way to prevent this[4].

3.2 Coevolutionary Bid-Based Learning Algorithm: Detailed Description

Specific algorithm steps are detailed sequentially with regard to the main loop, Step 2, of the learning algorithm.

Learner initialization, Step 2a. Each learner in the initial population L_{pop} is created by generating a random bid program and associating it with the current class a under consideration.

Test initialization, Step 2b. The test population T_{pop} is initialized to contain non-duplicate indices into the training dataset. A post-condition of the test initialization step is that each of the possible training labels is represented by at least one exemplar, therefore, for each possible training label an exemplar is first selected with uniform probability and added to T_{pop}. Thereafter, indices into the training dataset are selected with uniform probability, irrespective of label, and added to T_{pop}.

Learner generation, Step 2(c)i. Each learner $l_i \in L_{pop}$ is considered in turn. With probability ϱ a new learner is created from l_i through a series of search operators (Section 3.3), otherwise, a random learner is created. The new learners are added to the population doubling its size.

Test generation, Step 2(c)ii. Test points are generated without assuming any particular bias. New tests are therefore generated by uniform sampling of the training dataset and eliminating duplicates until the size of T_{pop} is doubled. Such a model is assumed as test points represent indexes into the training data, thus, there is no context on which variation operators such as crossover and mutation can be based.

[4]An alternative approach might take the form of an auction-based model for credit assignment. Such models are based on the concept of 'wealth' [34]. However, it becomes increasingly difficult to establish robust mechanisms for deriving the 'wealth' property when the subset of exemplars from which wealth-based performance metrics are derived is continuously varying – as is the case of the competitive coevolutionary paradigm central to scaling the BGP model to large problem domains.

Bid, Step 2(c)iiiA. The input test t_k is presented to each learner l_i which executes its bid program on t_k and outputs a bid $b \in (0, 1)$. Bids are restricted to the unit interval by applying a sigmoid to the raw real-valued results of program execution, Section 3.3. The outcome of applying l_i to t_k is then calculated as

$$G(l_i, t_k) = \begin{cases} b & \text{if action of } l_i \text{ matches label of } t_k \\ 1 - b & \text{otherwise} \end{cases} \tag{5}$$

and appended to an initially empty outcome vector $\mathbf{o_{l_i}}$ Following the bidding round, each learner l_i has an associated outcome vector, Eq. 2.

Outcome normalization, Step 2(c)iv. Given a learner l_i let M_i denote the set of tests in T_{pop} whose class matches the action of l_i and let $\bar{M}_i = T_{pop} - M_i$, the set containing all the other (non-matching) tests. Scaling factors μ_i and $\bar{\mu}_i$ are then calculated with respect to the outcome vector $\mathbf{o_{l_i}}$ as

$$\mu_i = \frac{1 + \Sigma_{t_k \in M_i} \mathbf{o_{l_i}}[k]}{1 + |M_i|} \tag{6}$$

and

$$\bar{\mu}_i = \frac{1 + \Sigma_{t_k \in \bar{M}_i} \mathbf{o_{l_i}}[k]}{1 + |\bar{M}_i|} \tag{7}$$

thus considering in-class and out-of-class exemplars respectively. Since each outcome falls in $(0, 1)$ both μ_i and $\bar{\mu}_i$ are limited to the unit interval. The outcome vector $\mathbf{o_{l_i}}$ is then normalized as

$$\mathbf{o_{l_i}}[k] \leftarrow \begin{cases} \bar{\mu}_i \cdot \mathbf{o_{l_i}}[k] & \text{if } t_k \text{ in } M_i \\ \mu_i \cdot \mathbf{o_{l_i}}[k] & \text{otherwise.} \end{cases} \tag{8}$$

This normalization is performed to prevent individuals from indiscriminately bidding always the minimum or always the maximum where this is significant on multi-class or unbalanced datasets. For example, if a learner l_i always bids the minimum it will achieve maximum outcomes on all tests in $\bar{M}_i$ and as such will be difficult to dominate. However, such a learner clearly exhibits degenerate behaviour and should be eliminated from the population. Since this learner will also achieve minimum outcomes on tests in M_i, the normalization factor μ_i will be very low. In this way, the outcomes on tests in $\bar{M}_i$ will be reduced from their initially maximum values facilitating the removal of the individual.

Learner selection, Step 2(c)v. The learner population L_{pop} is composed of two sets of learners: the set of new learners C created in the current generation, Step 2(c)i, and the set of learners P already present at the start of the generation. Each new learner $l_i \in C$ is then compared to each learner $l_j \in P$. If $\mathbf{o_{l_i}}$ is found to dominate $\mathbf{o_{l_j}}$ according to Eq. 1 then l_j is removed from P, l_i is marked for selection, and the next learner in C is considered.

As learner selection proceeds and new learners in C displace learners in P, P may shrink and it may become more difficult for new learners in C to dominate learners in P. Thus, the order in which the learners in C are considered is randomized, and for each learner in C that is considered, the order in which the learners in P are considered is randomized. Once all the learners in C are processed, L_{pop} in the next epoch is formed by combining the remaining learners in P with the learners in C marked for selection.

The above process also ensures that a new learner is selected only if its outcome vector is unique relative to the contents of the current population. It reduces the size of L_{pop} to what it was after initialization, Step 2a.

Test selection, Step 2(c)vi. With two exceptions, test selection is *exactly* the same as learner selection. That is, new tests are compared to tests that were carried over from the previous generation to see if any of the new tests dominate an old test. The two exceptions are as follows. First, the decision of whether or not to replace a test carried over from a previous generation with a new test is made based on Pareto-dominance with respect to the tests' distinction vectors. Thus, tests are favoured if they provide more information about relative learner performance. Second, a test is not removed from the population if it is the last one of its class. This was done to ensure that an exemplar was available to represent each class in the test population at all times, thus complementing the test population initialization heuristic (this was not relevant in learner selection because all learners were of the same class).

Reduction, Step 2d. Reduction is done after the last generation for the current action a, Step 2, is complete. The outcome vectors for the learners are therefore updated to reflect the remaining tests in T_{pop}. Based on these updated outcome vectors any dominated or duplicate learners are then discarded from L_{pop}. Given that T_{pop} is coevolved to distinguish between the learners, this step is designed to identify and remove redundant individuals.

Given a test set, setting outcomes as in Eq. 5 it is possible for all learners, including learners of different classes, to have vectors of unity outcomes. If the learners were not partitioned based on their action, the selection and reduction mechanisms would consider all but one of these learners to be redundant. This assertion would be incorrect because, at minimum, at least one learner of each action should be present in the final solution. This is another reason for performing the partitioning as in the proposed approach, Step 2.

Whenever outcomes were compared, as in Eq. 1, the equality operator was deliberately made imprecise. Two outcomes would be considered equal if their absolute difference was less than δ for relatively high values of δ such as 0.1. This was done to reduce the number of ways in which outcome vectors could vary and prevent situations where two learners were both selected despite their behaviour being virtually the same. Such a scheme is similar to the concept of ϵ-dominance in Evolutionary Multi-Objective Optimization [31]. However, the definition of ϵ-dominance in [31] cannot be applied directly to the competitive coevolutionary model: in particular, for two learners l_i and l_j and test t_k, that definition could result in unity for both $\mathbf{d}_{\mathbf{t_k}}[n \cdot i + j]$ and $\mathbf{d}_{\mathbf{t_k}}[n \cdot j + i]$ suggesting a preference for l_i over l_j and at the same time a preference for l_j over l_i.

3.3 Linear Genetic Programming

Linear GP [8] was used to evolve the learners' bid procedures. Each program consisted of a sequence of binary instructions representing one- and two-operand operations applied to inputs and a set of null-initialized registers. Following program execution, the real-value GP output y was extracted from a predefined register. To obtain bid values, b, in the unit interval, the Sigmoid function $f(y) = (1 + e^{-y})^{-1}$ was applied. Such an operator provides a smooth transition between the two asymptotic limits of zero and one. Individuals are therefore

encouraged to 'focus' their bidding behaviour on one of three regions – zero, transition, one – where this is the same for all individuals. Without this, individuals would have to identify the common bidding region first and then evolve relevant bid strategies within these regions.

Four stochastic search operators were applied to the bid program of an existing learner to generate a candidate (training Step 2(c)i): (1) *delete* removed an arbitrary instruction, (2) *add* inserted a random instruction at an arbitrary location, (3) *mutate* flipped an arbitrarily selected bit in the program, and (4) *swap* exchanged the location of two arbitrary instructions. These operators were applied independently with a predefined probability. In all cases, a uniform distribution was used to select bits/instructions and to set bit values during the generation of random individuals.

Whenever a learner was generated (training Steps 2a and 2(c)i), its bid program size was selected from a predefined range with uniform probability. The delete and add operators were included to allow varying program complexity within this fixed-length representation. The swap operator was added to remedy situations where the correct instructions were present but in the wrong order, and the mutate operator was included to alter a single field of an existing instruction.

4 Evaluation and Results

BGP was evaluated on nine large datasets of varying complexity. A performance comparison was made against the XCS model on account of its use of a bid-based mechanism for problem decomposition as well as its ability to evolve this decomposition without an *a priori* assignment of team memberships. XCS was shown to evolve general and accurate classifiers [52] making it a leading LCS formulation and especially suitable for classification [5, 40]. The two bid-based algorithms, BGP and XCS, were compared using a baseline defined by the LIBSVM [12] support vector machine implementation (version 2.85).

Section 4.1 presents a description of the XCS implementation, the datasets, the parameterization used in the experiments as well as a methodology established for comparing deterministic and stochastic models of classifications. Section 4.2 present the empirical evaluation from the perspective of classification performance, computational requirements, and an assessment of the degree to which problem decomposition is facilitated.

4.1 Experimental Setup

The following empirical evaluation will compare BGP with the XCSR and SVM models of classification under multiple datasets and performance criteria. In the following we establish the specific configuration of the XCSR classifier, Section 4.1.1, characterize the benchmark datasets, Section 4.1.2, and establish performance criteria for the post-training test set evaluation, Section 4.1.3. Parameterization for each of the models is then presented in Section 4.1.4. The results of the benchmarking study characterize classification, problem decomposition, model complexity, classwise classifier decomposition, competitive coevolutionary behaviour, and computational requirements in Sections 4.2.1 through 4.2.6 respectively.

4.1.1 XCS Implementation

The proposed approach was compared against a real-valued variant of the XCS+TS 1.2 classifier system implementation [11] which is based on the description of XCS in [10]. This implementation was selected because it represents a reliable code base and it is well documented and openly available. In addition to the original XCS algorithm [51], XCS+TS 1.2 makes available several optional enhancements. In the following, the XCS system used is specified emphasizing the particular design choices that were made including any modifications to the original XCS+TS 1.2 code. Basic knowledge of XCS is assumed but additional information can be found in [10, 11, 51].

A population $[P]$ of at most N micro-classifiers[5] is evolved where the action in each condition-action rule is selected from the set of possible class labels. Each classifier cl has the following set of learning parameters: prediction p_{cl} and prediction error ϵ_{cl} (both in the same units as the payoff), fitness f_{cl}, experience exp_{cl}, GA time stamp ts_{cl}, and action set size estimate as_{cl}. The classifier population is initialized to be empty relying instead on covering and the GA for classifier generation. To train the system, a sequence of 'explore trials' is performed where each explore trial consists of the following steps:

1. A training pattern σ is randomly selected with uniform probability.

2. A match set $[M]$ is calculated as the set of classifiers in $[P]$ whose conditions match σ. Covering is performed if not every class is represented in $[M]$. For each class not present in $[M]$, a matching classifier cl with the corresponding action is created. Parameters exp_{cl} and as_{cl} are set to 1, parameters p_{cl}, ϵ_{cl}, and f_{cl} are set to initial values p_I, ϵ_I, and f_I respectively, and ts_{cl} is set to the current time step. Classifier cl is then added to $[M]$. If the population size limit N is reached, deletion is performed using the fitness and action set size bias method [26].

3. A pure explore strategy is followed whereby an action act is selected from those present in $[M]$ with uniform probability. Since $[M]$ represents all classes all actions are possible. The classifiers in $[M]$ advocating act are recorded in the action set $[A]$.

4. The action act is executed resulting in a payoff P. Here, P depends on whether the class corresponding to act matches the true class of σ.

5. The learning parameters for each classifier cl in $[A]$ are updated in the following order: exp_{cl}, ϵ_{cl}, p_{cl}, as_{cl}, and f_{cl}. The experience exp_{cl} counts the number of times cl participates in $[A]$ so it is simply incremented. The update to f_{cl} is done as in [51] using the Widrow-Hoff delta rule except here the intermediate calculation of the absolute accuracy is performed as

$$\kappa_{cl} = \begin{cases} 1 & \text{if } \epsilon_{cl} \leq E \\ \alpha(E/\epsilon_{cl})^{\nu} & \text{otherwise} \end{cases} \tag{9}$$

[5]For efficiency, micro-classifiers representing identical condition-action rules are recorded as a single macro-classifier. Each macro-classifier has an associated numerosity which is used to weigh calculations accordingly. Here, the algorithm is described in terms of micro-classifiers.

where E is the error threshold (as a function of ϵ_0 but in the same units as the payoff) below which cl is considered maximally accurate and α and ν parameterize the shape of the error function. Parameters ϵ_{cl} and p_{cl} are also updated as in [51] using Widrow-Hoff, as is the action set size estimate as_{cl}. All Widrow-Hoff updates are done using the same learning rate β.

6. If, based on the mean value of the time stamps ts_{cl} of each classifier cl in $[A]$, the GA has acted on the classifiers in $[A]$ more than θ_{GA} trials ago, the GA is applied to the current action set. First, the time stamps in $[A]$ are set to the current time step (i.e., trial number in single-step problems). Two parents are then selected from $[A]$ using fitness-based tournament selection [9] using two independent tournaments of size $\tau \times |[A]|$. One offspring is generated from each parent as an exact copy with the exception of the experience and action set size estimates which are both reset to 1. The two offspring are crossed-over with probability χ then mutated with probability μ. If crossover does occur, prediction, prediction error, and fitness values in the offspring are set to the mean of the corresponding parent values. GA subsumption [52] is performed whereby if a parent's condition is more general than that of its offspring a copy of the parent is inserted instead of the offspring provided that the parent is sufficiently experienced and accurate. Insertions into $[P]$ may increase the size of the population beyond N and in that case deletion using fitness and action set size bias is again performed.

In classifier systems, exploit trials are used for performance monitoring (i.e., the GA is turned off and the learning parameters are not adapted) and here were turned off during training. To measure system performance on the test data post training, an exploit trial is performed on each test exemplar by deterministically selecting the action predicted to yield the highest payoff and assigning the associated class to the exemplar (the prediction array is calculated using fitness-weighed averaging).

XCS+TS classifier conditions were originally represented as strings constructed over the alphabet $\{0, 1, \#\}$ and therefore applicable only to problems with binary inputs. To construct classifier conditions that can support real-valued inputs, the Unordered Bound Representation [46] was used. This representation was selected because, compared to other representations such as the Center-Spread representation [53], it introduces the least bias in the intervals that are produced by the system. Given d-dimensional real-valued input patterns, the Unordered Bound Representation constructs a classifier condition by associating each input feature with an interval predicate. Specifically, the ith input feature is associated with the ith interval predicate $[l_i, u_i)$ in the classifier condition. The interval itself is encoded as the ordered pair (p_i, q_i) and the genotype to phenotype mapping defined as $l_i = min(p_i, q_i)$ and $u_i = max(p_i, q_i)$. Thus, for all $l_i \neq u_i$, each phenotype can be encoded as one of two genotypes. For a classifier condition to match an exemplar, every interval in the condition must contain its associated feature value.

The real-valued implementation of XCS+TS, XCSR, represented each classifier condition as the sequence of ordered pairs $((p_1, q_1), (p_2, q_2), \ldots (p_d, q_d))$. The elements of each ordered pair were encoded as floating point data types (i.e., a one-of-m representation was not used as suggested in [46]). Exemplar feature and intervals in the range $[0, 1)$ were assumed therefore 'don't care' predicates were encoded as the maximally general interval. Two-point crossover

was used whereby points p_1 and p_2 were selected from $\{1, 2, \ldots, d\}$ with uniform probability thus defining a contiguous range of ordered pairs which was then swapped between two individuals. Mutation was performed with probability μ on each ordered pair by adding to or subtracting (with equal likelihood) from each element an independently derived value $U[0, V_m)$. Here, $U[a, b)$ denotes a number selected with uniform probability from the range $[a, b)$. Finally, an input feature x_i was covered by generating the interval $l_i = p_i = x_i - U[0, s_0)$ and $u_i = q_i = x_i + U[0, s_0)$ and encoding it as either the tuple (p_i, q_i) or (q_i, p_i) with equal probability.

4.1.2 Datasets

The datasets used in the evaluation are summarized in Table 1. The Census Income (CEN) and KDD Cup '99 (KDD) datasets were obtained from the UCI KDD Archive [19]. The remaining datasets were obtained from the UCI Machine Learning Repository [38] and include the ANN Thyroid Disease (THY), Forest Covertype (COV), Pen-Based Recognition (PEN), Statlog Shuttle (SHU), Poker (POK), Optical Recognition (OPT), and Statlog Landsat (SAT) datasets. This selection was made to contain problems with varying numbers of classes and features, unbalanced class distributions, and large numbers of training exemplars. Thus, there are four datasets with fewer than 5000 training exemplars, two with 11000 to 25000, and two with over 100000. Feature counts vary from 9 to 64 whereas class distributions vary from equal to extremely unbalanced with cases of minority class counts representing less than 0.1% of the data. Moreover, POK, COV, and KDD utilize test sets with different distributions under training and test conditions. On all datasets, the original training and test partitions included in the distribution were used. On KDD, several choices are possible and here the '10% subset' dataset was selected for training and testing was done on the 'corrected' dataset (as is the norm on this domain).

Preprocessing was done to enumerate nominal attributes for all three approaches under consideration. Since XCSR assumes input values in the range $[0, 1)$, feature values in the training patterns were linearly scaled to fall into this range with respect to the minimum and maximum values in each dimension. On the test partition, this scaling was also performed, however, the minimum and maximum values as determined from the training partition were used[6]. This could result in test patterns whose values fell outside the maximally general interval $[0, 1)$ and therefore would not be matched by any classifier. The normalization of data for use with the SVM algorithm was the same as for XCSR whereas the BGP model retained the original definition of numerical features.

4.1.3 Test Set Classification Performance Criteria

Classifier performance is typically quantified using accuracy defined as the fraction of all instances that are labeled correctly. However, such a metric is very misleading under unbalanced datasets or even balanced multi-class datasets [24, 50]. To this end, we make use of

[6]Otherwise the test data would have been used to build the model compromising the independence of the test partition.

Table 1: Summary of the datasets used in the evaluation. Shown are and the class distributions on the training and test partitions. The value in parentheses following each dataset label indicates the number of features. A '-' denotes the class is not present in a dataset.

	Classwise pattern counts										
	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	All
THY (21)											
Train	93	191	3488	-	-	-	-	-	-	-	3772
Test	73	177	3178	-	-	-	-	-	-	-	3428
CEN (41)											
Train	187141	12382	-	-	-	-	-	-	-	-	199523
Test	93576	6186	-	-	-	-	-	-	-	-	99762
KDD (41)											
Train	97277	1126	391458	4107	52	-	-	-	-	-	494020
Test	60593	16347	229853	4166	70	-	-	-	-	-	311029
COV (54)											
Train	1620	1620	1620	1620	1620	1620	1620	-	-	-	11340
Test	210220	281681	34134	1127	7873	15747	18890	-	-	-	569672
PEN (16)											
Train	780	779	780	719	780	720	720	778	719	719	7494
Test	363	364	364	336	364	335	336	364	336	336	3498
SHU (9)											
Train	34108	37	132	6748	2458	6	11	-	-	-	43500
Test	11478	13	39	2155	809	4	2	-	-	-	14500
POK (10)											
Train	12493	10599	1206	513	93	54	36	6	5	5	25010
Test	501209	422498	47622	21121	3885	1996	1424	230	12	3	1000000
OPT (64)											
Train	376	389	380	389	387	376	377	387	380	382	3823
Test	178	182	177	183	181	182	181	179	174	180	1797
SAT (36)											
Train	1072	479	961	415	470	1038	-	-	-	-	4435
Test	461	224	397	211	237	470	-	-	-	-	2000

Table 2: BGP parameter settings.

Parameter	Value
minimum program size	1
maximum program size	48
delete/add/mutate/swap prob.	0.5
number of registers	8
function set	$\{cos, exp, log, +, \times, -, \div, \%\}$
L_{pop} size	25
T_{pop} size	25
ϱ	0.9
number of generations	50000
δ	0.1

classwise detection rates defined as

$$DET_c = \frac{TP_c}{FN_c + TP_c} \tag{10}$$

where DET_c is the detection rate for class c, and TP_c and FN_c refer to the true-positive and false-negative counts with respect to class c. We summarize the classwise detection rates by defining a score for a classifier i over all classes as

$$SCORE_i = \frac{1}{|C|} \sum_{c \in C} DET_c(i) \tag{11}$$

where C is the set of class labels and $DET_c(i)$ returns the detection rate of classifier i on class c. This classwise analysis of performance was performed because on unbalanced datasets[7] a model can achieve high raw accuracy by ignoring minority classes. In applications where its the minority classes that are of interest, such degenerate solutions are of limited use.

4.1.4 Parameterization

The parameters used in evaluating the proposed BGP approach are shown in Table 2. Here, the population sizes refer to the counts before generation, Steps 2(c)i and 2(c)ii of the training algorithm. Given that 50 learners are evaluated on 50 tests in each of the 50000 epochs for each class, each run requires $|C| \times 1.25 \times 10^8$ program evaluations where C is the set of class labels.

With the exception of the maximum population size N, the error threshold ϵ_0 (as a fraction of the maximum payoff), and the number of explore trials $nrExp$, all the XCSR parameters remained fixed throughout the experiments. For these three parameters, however, values were determined on a dataset-by-dataset basis. Initially, a set of exploratory runs using $\epsilon_0 = 0.001$ and population sizes 1000 and 10000 was made. The results were inspected

[7]All datasets with more than two classes will be 'unbalanced' since each class can at best (i.e., when the number of exemplars belonging to each class is the same) account for $\frac{1}{\#ofclasses}$ of all exemplars.

Table 3: Parameter values used in evaluating XCSR.

Dataset	N	$nrExp$	ϵ_0
THY	1000	375000	0.001
CEN	1000	250000	0.001
KDD	10000	62500	0.001
COV	10000	87500	0.001
PEN	10000	125000	0.001
SHU	1000	875000	0.001
POK	10000	125000	0.001
OPT	10000	125000	0.001
SAT	10000	75000	0.001

to determine the more suitable population size (by considering overall accuracy as well as classwise detection), and using this setting for N and $\epsilon_0 = 0.0001$, another set of runs was completed. This was done to determine if the system would do better at detecting classes that accounted for less than 0.1% of the dataset. Throughout, $nrExp$ was set so that the total number of times a classifier condition was evaluated equaled the number of times a program was evaluated using the BGP approach (i.e., larger populations were allocated fewer trials so that the computational effort remained roughly the same). The best settings for N and ϵ_0 that were found using this procedure, as well as the associated number of explore trials, are summarized in Table 3. The results that follow were reported for runs using these settings.

The remaining XCSR parameters were set as follows: $\beta = 0.05$, $\alpha = 0.1$, $\nu = 5$, $\theta_{GA} = 50$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 50$, $\delta = 0.1$, $\theta_{sub} = 50$, $P_{\#} = 0.33$, $p_I = 10$, $\epsilon_I = 0$, $f_I = 0.01$, $p_{exp} = 1.0$, $\theta_{mna} = |C|$, $doGASubsumption = true$, $doActionSetSubsumption = false$. Detailed parameter descriptions can be found in [10]. Some of the parameters, and in particular β and θ_{GA}, were set according to the suggestions in [40] to account for unbalanced class distributions. Tournament selection was performed using $\tau = 0.4$. The system received payoffs of 0 and 1000 for incorrect and correct classification respectively and the error threshold E in Eq. 9 was set to $1000\epsilon_0$. Finally, the Unordered Bound Parameters s_0 and V_m were set to 1.0 and 0.1 respectively.

Version 2.85 of LIBSVM was used in the experiments. This release of LIBSVM supports the use of second-order information to provide a more robust convergence model in addition to the widely utilized Sequential Minimal Optimization model [15]. In common with most SVM models, the principal learning parameters take the form of penalty function 'C' and declaration of an appropriate kernel function. In the latter case, both radial (Gaussian) and sigmoid kernels were considered, with the performance of the radial model dominating that of the sigmoid, thus, in the following, we only report results under the radial kernel. Three values for C were tried for each dataset resulting in three possible SVM classifiers: the one yielding the highest score value on the training data was selected for comparison (i.e., post-training, the metric of Eq. 11 on the training partition was used to select the most appropriate SVM model).

For each dataset, thirty runs using different initial conditions were performed using BGP

and XCSR. Since the SVM training algorithm does not depend on the initialization, only three LIBSVM runs, one for each value of C, were performed on each problem. Given this disjunction between distributions of results under BGP and XCSR versus a single performance point under the SVM model, we assume the following basis for comparison. The performance point from the SVM model is used to normalize the distribution of performance points from the BGP and XCSR models. Specifically, a normalized BGP/XCSR score is obtained by dividing the raw BGP/XCSR score by the SVM score for the given dataset. If a normalized score is greater (less) than 1, it means the associated initialization was better (worse) than the SVM score. XCSR and BGP results are therefore characterized in terms of a distribution of results (one for each of the thirty initializations per model) relative to the SVM baseline on each dataset.

4.2 Results

4.2.1 Classification Performance

Figure 2 compares the distribution of BGP and XCSR score values with respect to the SVM baseline, Section 4.1.4. Figure 2 thus summarizes the spread in the BGP and XCSR scores and at the same time compares them with the scores obtained using the SVM algorithm. Table 4 lists the SVM test scores that were used to normalize the BGP and XCSR scores, hence providing the baseline for the comparison. Based on Figure 2, the following observations can be made:

1. Both BGP and XCSR manage to outperform the SVM model on some of the datasets. XCSR substantially outperforms the SVM model on THY where all but two of the XCSR initializations yield higher scores. BGP substantially outperforms the SVM model on SHU but also does well on THY and CEN yielding third quartile values that beat the SVM score. On KDD, BGP manages to better the SVM baseline in four of the thirty initializations.

2. With respect to the training partition, THY, CEN, KDD, SHU, and POK are considered to be unbalanced datasets while COV, PEN, OPT, and SAT are balanced. Compared to XCSR, BGP appears to do better on the unbalanced datasets. The one exception appears to be THY where XCSR typically is able to better BGP even though the highest BGP scores do match the highest XCSR scores. On POK, both BGP and XCSR do poorly compared to the SVM baseline but the BGP scores still represent an improvement over the XCSR scores. With the exception of OPT, XCSR tends to do better on the balanced datasets.

3. The observation that BGP performs relatively well on unbalanced problems is further supported by the observation that, compared to the SVM baseline, it also tends to do well on this subset of datasets.

4. When BGP outperforms XCSR, it tends to do so by a considerable margin such as on CEN, SHU, OPT, and to a lesser degree KDD. The margins observed when XCSR outperforms BGP are not as high, with the best BGP results falling within or above the XCSR interquartile range.

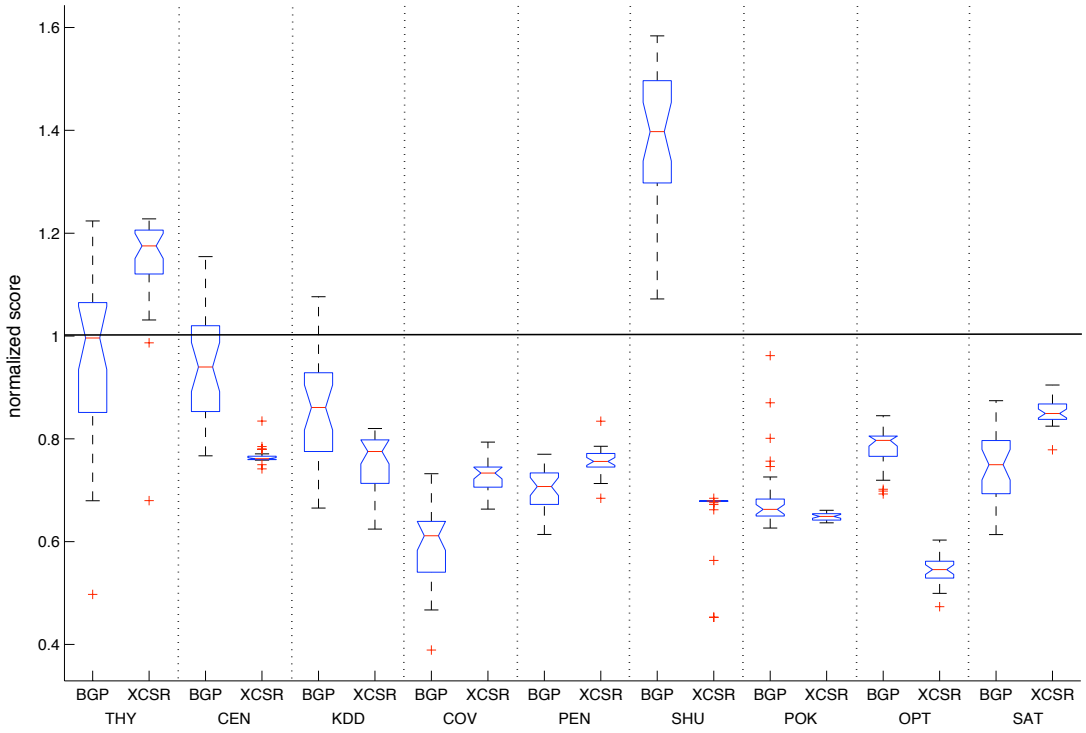

Figure 2: Overall comparison between the BGP and XCSR scores as normalized with respect to the SVM baseline. The top and bottom of each box represent the first and third quartile values over the thirty initializations, while the horizontal line through the middle of each box represents the median. The whiskers extend from the top/bottom of each box to the farthest value that is within 1.5 times the distance between the first and third quartile values. Any other scores represent extreme outliers and are denoted by a '+'. On a given dataset, each BGP and XCSR score is normalized by the single SVM score on that dataset so normalized scores greater than 1 represent values exceeding the SVM score.

5. XCSR is susceptible to producing degenerate results. On CEN, the median detection rate of XCSR on class 1 is 0.006 implying that the XCSR solutions tend to indiscriminatingly label all instances as belonging to class 0. On POK, XCSR labels all instances as belonging to either class 0 or 1 and on SHU the median detection rates on classes 1, 2, 5, and 5 are all 0.000.

6. Compared to XCSR, the BGP scores are subject to greater variance. On all the datasets, the difference between first and third quartile values is always greater for the BGP approach. This implies that a greater degree of exploration is present under the BGP model, where we will return to this property in future work.

In summary, of the three approaches, no single one clearly dominates the other two with respect to the score values. However, BGP tends to do better when the class distributions

Table 4: SVM score results. For each dataset, the values for the complexity parameter C yielding the highest training score is shown along with the associated test score. The test score shown for each dataset is used in normalizing the BGP and XCSR test scores on that dataset in Figure 2.

	THY	CEN	KDD	COV	PEN	SHU	POK	OPT	SAT
C	100	10	100	100	100	100	100	100	100
Test	0.806	0.658	0.611	0.735	0.979	0.631	0.157	0.971	0.851

in the training dataset are skewed, a reflection of the utility of the normalization applied to the outcome vectors, Eq. 8.

4.2.2 Problem Decomposition

Given a solution set S of learners returned by the BGP algorithm, Figure 1, a 'win count distribution' is used to illustrate how the learners in S decompose the set of problem instances. In particular, by way of the bidding process, each learner in S is associated with a subset of exemplars, i.e., the cases for which it is the highest bidder. Since each exemplar is associated with a single learner, the union of these subsets, one for each learner, forms a partition on the exemplar space. The win count distribution then refers to the cardinalities of these subsets. They are analyzed because they provide information about the solutions that are evolved, which in turn may provide insight into properties underlying the problem domain.

An overview of the BGP win count distributions on the test partition of each dataset is shown in Figure 3. The distributions are presented by histograms showing counts (on the y-axis) of the number of learners that win a particular fraction of the test dataset (on the x-axis, normalized by the size of the test dataset). For example, on PEN, the distribution is strongly skewed to the right as most learners tend to win a small fraction of the test dataset and no learners win more that about 50% of the dataset. It should be noted that for each dataset these histograms show counts summed over all initializations.

Based on Figure 3, two types of behaviours are observed. On THY, CEN, KDD, and SHU, small peaks towards the right-hand side of the histograms are seen indicating that a small fraction of learners in the final solution provide the label for a large number of exemplars. On the other five datasets, the distributions are seen to be skewed to the right indicating that the win counts for the learners in a solution tend to be more similar with each learner suggesting a label for a small fraction of examples. This behaviour for the members of the same solution can be deduced even though the histograms consider win counts summed over all thirty initializations: for example, as is observed on THY, if a learner is seen to win 90% of the bidding rounds the other learners in the same solution can win at most 10% of the bidding rounds.

The pattern of behaviour we identify in this is that THY, CEN, KDD, and SHU all have a single 'major' class, with all other classes denoting a minor contribution to the training exemplar counts, Table 1. Conversely, COV, PEN, OPT, and SAT are based on training datasets with a much more equal distribution of exemplars per class. The only

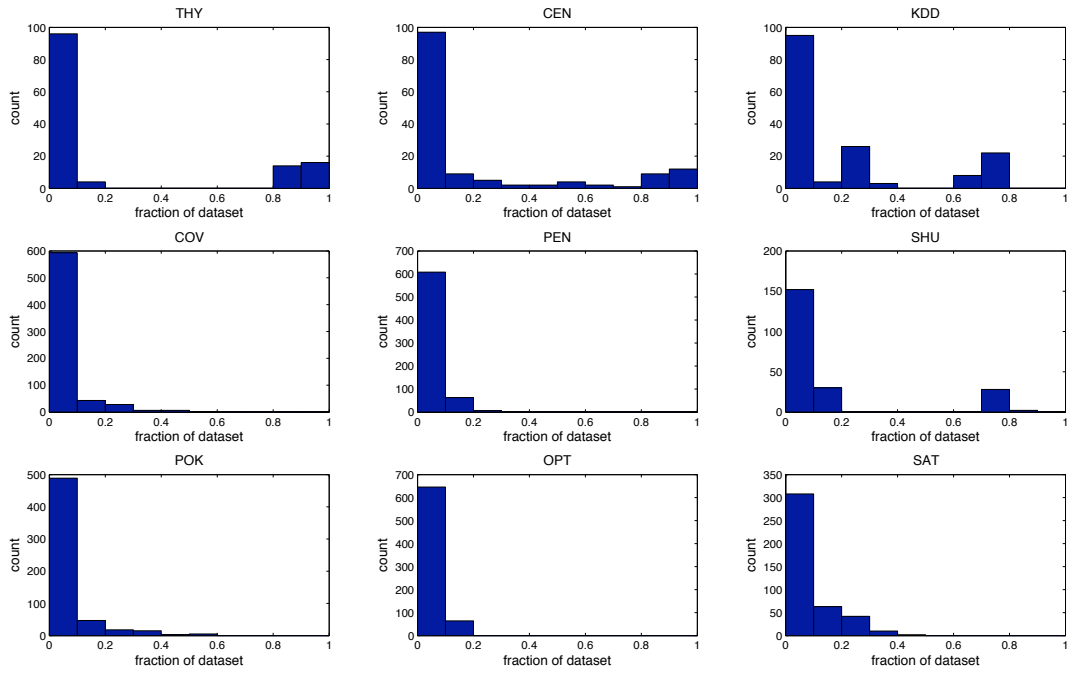

Figure 3: Overview of the win count distributions on each dataset over all initializations. For each initialization, the final solution (S) was applied to the test partition and the number of times each learner won a bidding round recorded. These counts were linearly scaled to fall between 0 (representing no rounds) and 1 (representing all rounds). The normalized counts across all initializations were then used to construct the histograms in this figure: the x-axis denotes the range $[0, 1]$ partitioned into 10 equal-sized bins corresponding to fractions of the test dataset and the y-axis denotes the number of learners winning the number of rounds associated with each bin (i.e., the sum of counts across to the bins equals the total number of learners in all thirty solutions).

exception to this is POK in which two of the nine classes dominate the training exemplar distribution. Based on these observations, one could conclude that BGP seeks out the most general individuals producing win counts that are proportional to the distribution of the underlying dataset (i.e., individuals representing classes with more instances have proportionally higher win counts). In addition, it is on THY, CEN, KDD, and SHU where the BGP model provides the strongest classification performance relative to the SVM baseline, Figure 2.

In the case of datasets with the more uniform win count distributions, OPT and PEN stand apart from the other two as their associated histograms show the most uniform distribution of win counts. With respect to the class distributions in the underlying dataset, Table 1, OPT and PEN also stand apart because they represent well balanced training datasets. Although the COV training partition is also balanced, its win count distribution is not as uniform (especially taking into account the scale on the y-axis) because unlike OPT and PEN the distribution of the COV training exemplars does not match that of the test exemplars (and Figure 3 is based on the test partition). This further supports the assertion that BGP favours learners that win many exemplars because with balanced datasets it is not possible for a learner to behave correctly and win 'most' of the dataset.

4.2.3 Model Complexity Results

The number of learners used in the final solution for the BGP approach is shown in Figure 4. The datasets for which the fewest individual were used are THY, CEN, KDD, and SHU, the same problems for which the win counts distributions, Figure 3, are highly skewed and for which BGP returns the strongest classification results. In fact, on these four datasets it is not uncommon to see solutions consisting of one learner per class (particularly on KDD and SHU). This is consistent with the observations made in Section 4.2.2 suggesting that BGP seeks out maximally general individuals. For example, the THY individuals in the right-hand side peak in Figure 3 likely correspond to learners matching class 2; since class 2 accounts for roughly 93% of the dataset and the detection rate on class 2 (not shown) is in the mid- to high-nineties, each of these learners can be expected to win close to 90% of the THY instances. Also, given that the number of learners that can potentially participate in the solution is twenty-five *per class*, the reduction procedure based on the Pareto-dominance with respect of the outcome vector is shown to be an effective way to reduce the complexity of the solution.

There appears to be a significant difference between the number of learners in the THY, CEN, KDD, and SHU solutions and the number of learners in the COV, PEN, POK, OPT, and SAT solutions. Relative to the SVM baseline classification performance, it is interesting to note that the four datasets on which BGP excels in terms of classification correspond to the problems where simpler BGP solutions are observed. Thus, from the BGP model perspective, on datasets found to be more difficult, more learning resource is assigned. It is clear however that this is a learning bias, with the SVM model clearly finding CEN, KDD, and SHU more difficult than COV, PEN, and OPT, Table 4.

For comparison, the number of classifiers and support vectors in the solutions returned by XCSR and the SVM baseline is shown in Table 5. The XCSR solutions are expressed in terms of the number of macro-classifiers. The population size values indicate that for

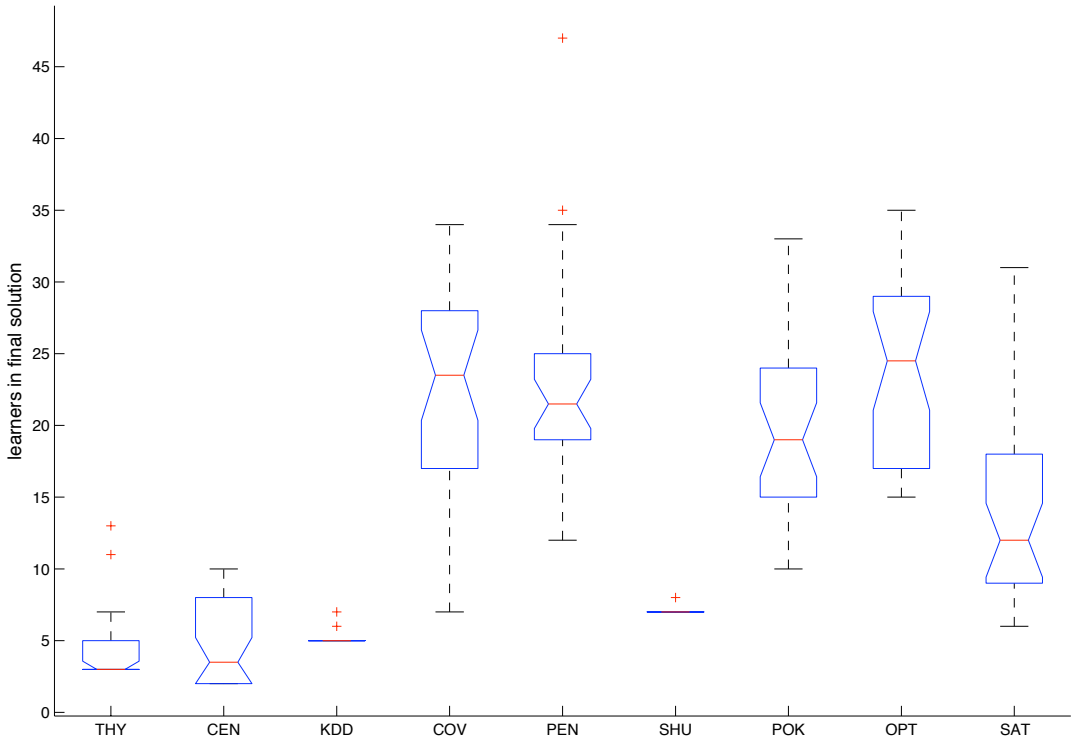

Figure 4: Number of learners in the final solution for each problem using the BGP approach. Shown is the spread across the thirty initializations where the elements of the boxplot are as described in Figure 2.

Table 5: XCSR and SVM solution complexities. For XCSR, the mean number of macro-classifiers over all thirty initializations is shown along with the associated standard deviation in parentheses. For the SVM baseline, the number of support vectors is detailed for the C value as in Table 4 (and the Guassian kernel).

	XCSR	LIBSVM
THY	881.2 (14.3)	479
CEN	965.5 (6.0)	23856
KDD	9825.1 (15.4)	2048
COV	9890.7 (9.6)	7073
PEN	9131.8 (36.8)	576
SHU	644.8 (39.4)	3966
POK	8364.8 (51.4)	22625
OPT	9864.9 (13.1)	663
SAT	9753.5 (17.0)	1294

the most part the classifier population contains many unique conditions (i.e., numerosities tend to be low). On all problems, the classifier population size limit N was always reached (not shown). The SVM solution complexities are shown in terms of the number of support vectors.

Comparing the complexity of the BGP solutions to the complexities of the XCSR and SVM solutions is not straightforward because each is expressed in different terms (learners, macro-classifiers, and support vectors). However, even if learners are viewed as the most complex 'unit of measurement' each containing at most 48 instructions including introns[8], the BGP solutions are clearly much more succinct. In many cases, the counts differ by several orders of magnitude.

Finally, we note that unlike the BGP model, for the other two approaches there is no correlation between model simplicity (complexity) and classification performance. Both XCSR and SVM models return good and bad results under both simple and complex solutions alike.

4.2.4 BGP Solution Breakdown by Class

To provide a more complete picture of the solutions evolved using the BGP approach the classwise solution breakdown is determined, Figure 5. To express such a breakdown, the number of learners of each class in the final solution was recorded and the mean value over all initializations is reported. The problems with the most balanced training partitions, COV, PEN, and OPT, result in breakdowns that tend to be more unbalanced (i.e., the classes are less equally represented in the final solution). On the other hand, the problems with the most skewed training partitions, THY, CEN, KDD, and SHU, yield more balanced breakdowns (as well as better classification results and lower solution complexity). The exception to this is POK where it is observed that the mean counts for all but classes 0 through 3 are virtually one – this is not surprising given that the total number of training

[8]The removal of introns, which were found to account for between 60% to 90% of instructions in linear GP [6], was not performed .

exemplars of classes 4 through 9 is negligible. Such a distribution in classifier allocation indicates that the unbalanced datasets contain a significant number of redundant exemplars. Thus, as is also evident from Figure 4, the behaviour embodied in the major classes can be expressed effectively using a small number of classifiers. This indeed matches the scenario on KDD data where nearly 79% of the dataset corresponds to the denial of service style of attack; a behavioural characteristic that is not particularly inconspicuous.

A key factor in this property is probably the competitive coevolutionary element of the BGP framework. In scenarios such as THY, CEN, KDD, and SHU the competitive model appears to be particularly effective at weeding out exemplars that do not contribute to classifier development at each generation. The role of the competitive model is investigated in more detail for the THY dataset in Section 4.2.5.

4.2.5 Competitive Coevolution and Test Point Discovery

With the exception of THY, BGP was shown to achieve better performance than XCSR on the datasets with unbalanced class distributions in the training partition, Figure 2. THY was also found to be the exception in previous work [35] where it was suggested that the problem was due to the replacement policy. In particular, the Pareto-dominance relation, which is used to select learners, does not distinguish between learners of different 'usefulness'. For example, a learner that solves just one objective that no other learner solves is considered equivalent to a learner that solves many objectives that no other learner solves since the two form a non-dominated set. In this way, the learner population may fail to generalize (in the sense that the total number of solved objectives is small) leading to a coevolutionary dynamic referred to as *focusing* [49]. A side-effect of focusing are low turnover rates as the over-specialized learners tend to persist in the population.

Figure 6 shows the cumulative counts for the number of times a learner carried over from the previous generation is dominated by a new learner (i.e., the new replaces the old in the population) on THY and CEN. The CEN results were included for comparison because like THY this dataset has an unbalanced class distribution yet BGP performs well on it. From Figure 6, it is apparent that compared to CEN the THY turnover rates are lower[9]. The problem is not focusing, however, as this would imply more individuals in the final solution than are observed, Figure 4. Specifically, if a learner in the population over-specializes it is not likely to dominate other learners (otherwise it would do well not on a few but on many objectives). As such, other learners are likely to remain in the solution after reduction, Step 2d of the learning algorithm. This is in contrast to the learner counts in Figure 4 where most solutions on THY contain one learner per action.

The precise reason for the relative breakdown of the proposed algorithm on THY is undoubtedly multifaceted. Pareto-coevolution has been used to coevolve poker hands [39], cellular automata and initial condition densities [16], and real-valued vectors [14]. In these problem domains, there is structure which can be exploited by the search operators that generate offspring from parents. For example, real-valued vectors can be mutated to generate offspring in the neighbourhood of their parents. Within the BGP framework, this can be

[9]Given that 1250000 new learners are generated when training learners of each action for 50000 generations, Step 2 of the learning algorithm, on THY roughly one in 127, 116, and 173 new learners is accepted into the population when training for action 0, 1, and 2 respectively.

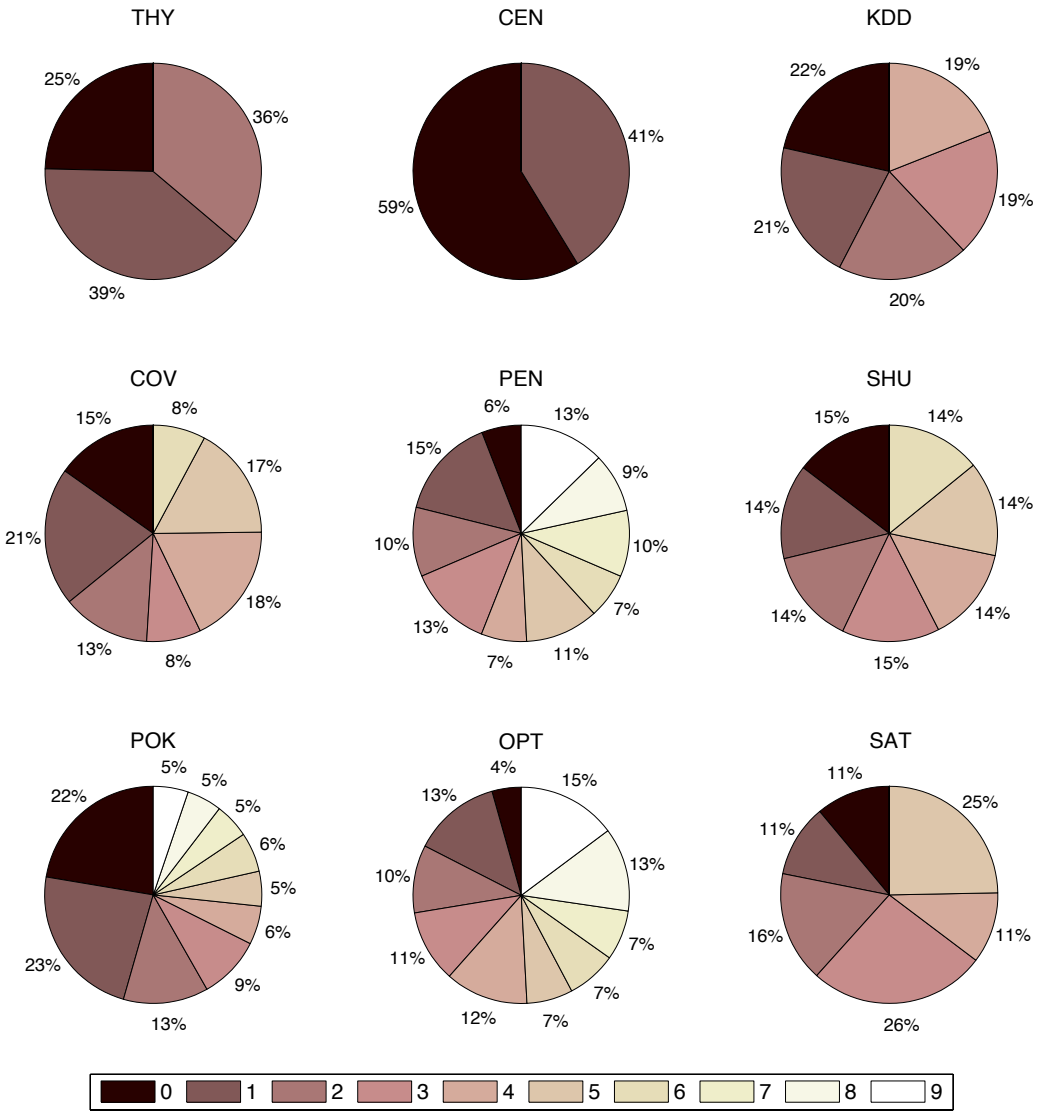

Figure 5: Solution breakdown by class. For each problem, the mean number of learners of each class participating in the final solution is calculated over all thirty initializations. These are shown in a pie chart format where classes 0 through 9 are allocated slices counter-clockwise from 12 o'clock.

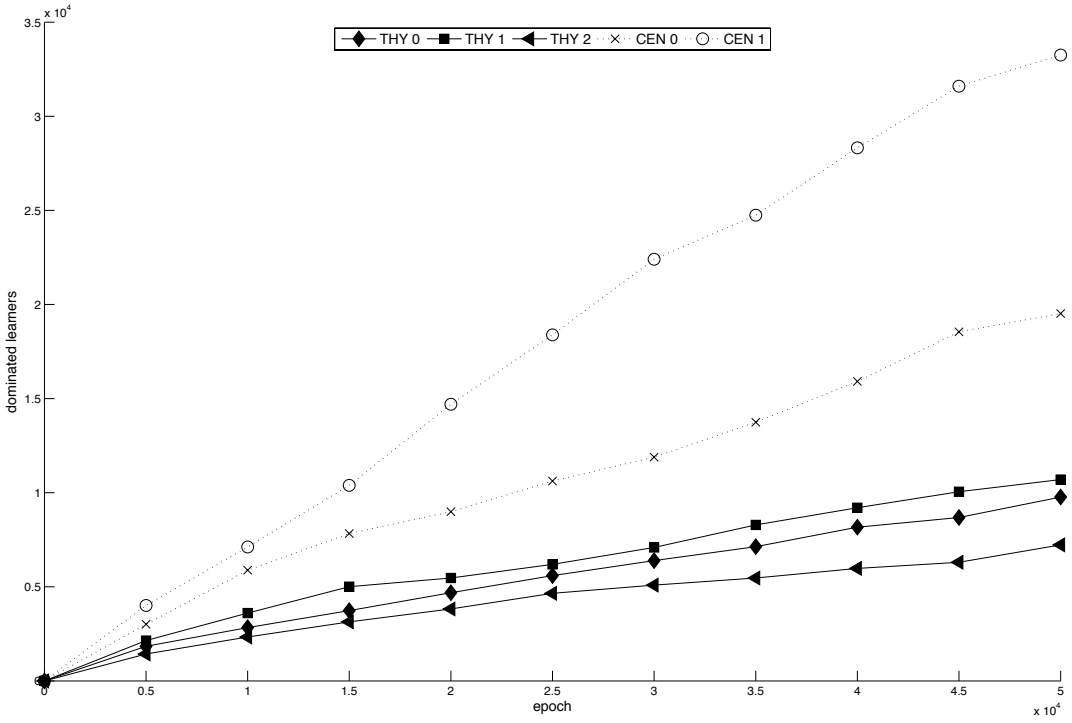

Figure 6: Classwise learner turnover rates on THY and CEN. Every 5000 epochs, denoted on the x-axis, the cumulative number of learners of each class carried over from the previous generation that were dominated by new learners is shown, y-axis. For each class and epoch, the plotted point represents the median value over all thirty initializations.

done on the learner side[10] but not on the test side. Here, tests are represented as indices into the dataset and as such mutation and crossover have no interpretation. The genetic material present in the test population thus cannot be used to guide the search for progressively better tests, i.e., offspring tests that make the same distinctions as their parents plus additional distinctions. The test population therefore does not truly evolve but rather it serves purely as a memory mechanism that saves tests which are encountered and found to be highly informative. Moreover, the total set of training exemplars is fixed. Thus, if the total set of training exemplars is disengaged from the ability of the initial set of learners, it will take longer for the learners to establish appropriate sets of behaviours.

The discovery of informative tests in this way is likely to be infrequent leading to low turnover rates in the learner population. Indeed, a positive correlation (> 0.5) was found between the number of dominated tests and the number of dominated learners when training each action on THY. However, just because the search for test points is inefficient does not mean that good informative tests will not be found. BGP is able to find good solutions occasionally, as indicated by the large spread in the BGP results shown in Figure 2, and even on THY the best BGP score matches the best XCSR score (and 50% of the BGP solutions match the SVM baseline). Finally, we note that an alternative heuristic for populating the point population is to enforce an additional constraint for equal representation of each class. Such a heuristic has proven useful in the wider machine learning literature [50].

4.2.6 BGP Training Times

The training times required to train BGP are shown in Table 6. They were collected using the C *getrusage*() function and account for both the user and system time. When total running times are considered the values for the different datasets show a lot of variation. When the approximate times required to train *each class* are considered, however, there is very little variation. This suggests that as expected, by decoupling the overhead required to evaluate programs from the size of the dataset, the overall training time is also decoupled from the size of the dataset. Although there is room for improvement (runs on KDD using PGPC took minutes, albeit in a binary context [32]) these run times are significantly faster that GP without coevolution (e.g., on the KDD dataset a single run on one class using canonical GP takes eleven hours to complete).

Training times using XCSR are not available, however, given that matching a classifier condition is generally faster than running a GP program training using XCSR is expected to be faster as well. However, if the system is to be deployed, the advantage that XCSR has in the matching speed for a single condition may be outweighed by the number of conditions that have to be evaluated. For example, on KDD, a solution evolved using BGP can be expected to require five program evaluations, Figure 4, while a solution evolved using XCSR can be expected to require close to 10000 conditions to be tested on every instance, Table 5.

Compared to the LIBSVM implementation of the SVM model, BGP training is expected to be more costly due to the multiple initializations that are required. However, SVM models are sensitive to the availability of suitable cache memory provision for the recording of intermediate results. Indeed, providing more efficient memory models is one of the major

[10] Although a small mutation in a program's genotype may yield a disproportionally large change in its phenotype.

Table 6: Running times in hours for the proposed approach. Shown are the first quartile, median, and third quartile values over thirty initializations. Values in parentheses indicate the total running time divided by the number of classes in each dataset.

Dataset	Q1	MED	Q3
THY	1.75 (0.58)	1.77 (0.59)	1.81 (0.60)
CEN	1.22 (0.61)	1.23 (0.61)	1.24 (0.62)
KDD	3.25 (0.65)	3.34 (0.67)	3.38 (0.68)
COV	4.11 (0.59)	4.16 (0.59)	4.19 (0.60)
PEN	6.18 (0.62)	6.23 (0.62)	6.28 (0.63)
SHU	4.21 (0.60)	4.27 (0.61)	4.33 (0.62)
POK	6.05 (0.60)	6.12 (0.61)	6.17 (0.62)
OPT	5.88 (0.59)	5.91 (0.59)	5.96 (0.60)
SAT	3.82 (0.64)	3.85 (0.64)	3.93 (0.65)

research directions for the SVM paradigm. In this case, the larger datasets (CEN, KDD) required support for 2 GB of RAM. Conversely, evolutionary methods require multiple runs, but maintain a constant memory model, as defined by the constant learner and test population sizes. Finally, from a deployment perspective, the BGP model is still more effective given the simplicity of the ensuing solutions.

5 Conclusion

This work presented a bid-based approach to classification in which learners evolve a bidding behaviour relating input exemplars to bids. This is in contrast to traditional approaches to GP classification where input exemplars are mapped directly to a class label. To allow the model to scale to large datasets, Pareto-based coevolution was employed whereby the GP-based learners were evolved alongside a population of training instances. The Pareto-based framework explicitly recognizes that a single learner may not be able to maximize every objective simultaneously and instead formulates the solution in the form of a Pareto front. As such, the learners attempt to decompose the problem by way of their bidding behaviour while the goal of the tests is to distinguish between the learners. Since the test population represents a small, fixed-size, subset of the entire training partition, the algorithm decouples the cost of fitness evaluation from the total number of training exemplars. Problem decomposition appears as a natural property of the system as individuals are associated with exemplars on which they provide the winning bid.

The BGP approach was compared to the XCSR classifier system implementation and to the LIBSVM support vector machine implementation of a second-order SVM model. Over the nine datasets, no single approach was able to always outperform the other two with respect to the score values. BGP, however, was found to do better on datasets with unbalanced training partitions. More generally, the model appears to be particularly effective at resisting complexity as measured relative to XCSR and the SVM baseline its solutions were much more succinct. This translated into more transparent solutions and, in all likelihood,

much faster response times during deployment.

A major advantage of BGP over other GP-based approaches to teaming is its ability determine the composition of the teams with respect to the numbers and type of team members. This was found to provide useful insight regarding the problem domain under consideration. For example, even though the training KDD partition was by far the largest the BGP solutions typically contained one learner per class; this reflects the fact that 79% of the KDD datasets represented denial of service attacks whose characteristics are unsophisticated yet repetitive. Likewise, solutions on the other other large datasets – CEN and SHU – also benefited from BGP's ability to find very succinct yet accurate solutions.

Future work will return to the problem of incorporating structure that could be utilized by the search operators in the test population. If tests that distinguish between learners cannot be found efficiently, selection of learners is not likely to be effective. In classification, structure relating parent tests to offspring could be discovered by including a preprocessing step such as clustering. Other domains may already be structured in such a way so as to allow the search operators to act more naturally (e.g., mutating a vector of angles and velocities in the pole balancing problem).

Finally, we are interested in alternate approaches to identifying the team members comprising the final solution. That is to say, currently the BGP model evolves a population of learners that, following reduction, are included in the final solution; in contrast, it might be more fruitful to utilize a symbiotic relation between a learner population and team identification population with the goal of simplifying the identification of relevant credit assignment paths and reducing the resulting variation in the solutions.

Acknowledgments

This work was conducted while Peter Lichodzijewski held a Precarn Graduate Scholarship and a Killam Postgraduate Scholarship. Malcolm. I. Heywood would like to thank the Natural Sciences and Engineering Research Council of Canada, The Mathematics of Information Technology and Complex Systems network, and the Canadian Foundation for Innovation for their financial support.

References

[1] P. J. Angeline and J. B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 236–241. Hillsdale, NJ: Lawrence Erlbaum, 1992.

[2] D.-Y. Cho B.-T. Zhang. Genetic Programming with active data selection. In B. McKay, X. Yao, C. S. Newton, J.-H. Kim, and T. Furuhashi, editors, *Simulated Evolution and Learning: Second Asia-Pacific Conference on Simulated Evolution and Learning*, pages 146–153. London, UK: Springer-Verlag, 1998.

[3] A. J. Bagnall and G. C. Cawley. Learning Classifier Systems for data mining: A comparison of XCS with other classifiers for the forest cover data set. In D. C. Wunsch,

M. Hasselmo, K. Venayagamoorthy, and D. Wang, editors, *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1802–1807. Oxford, UK: Elsevier Science, 2003.

[4] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[5] E. Bernado-Mansilla and J. M. Garrell-Guiu. Accuracy-based Learning Classifier Systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11:209–238, 2003.

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

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

[8] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. New York, NY: Springer, 2007.

[9] M. Butz, K. Sastry, and D. E. Goldberg. Tournament selection in XCS. In E. Cant-Paz, J.A. Foster, K. Deb, L.D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, J. Wegener, D. Dasgupta, M.A. Potter, and A.C. Schultz, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1857–1869. Berlin: Springer-Verlag, 2003.

[10] M. Butz and S. W. Wilson. An algorithmic description of XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 253–272. Berlin: Springer-Verlag, 2001.

[11] M. V. Butz. Documentation of XCS+TS C-Code 1.2. *IlliGAL Technical Report 200323*, 2003.

[12] C.-C. Chang and C.-J. Lin. *LIBSVM: A library for Support Vector Machines (Version 2.85)*, 2007. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[13] R. Curry, P. Lichodzijewski, and M. I. Heywood. Scaling Genetic Programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 37(4):1065–1073, 2007.

[14] E. D. De Jong and J. B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12:159–192, 2004.

[15] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training Support Vector Machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[16] S. G. Ficici and J. B. Pollack. Pareto optimality in coevolutionary learning. In J. Kelemen and P. Sosik, editors, *Proceedings of the 6th European Conference on Advances in Artificial Life*, pages 316–325. Berlin: Springer-Verlag, 2001.

[17] G. Folino, C. Pizzuti, and G. Spezzano. GP ensembles for large-scale data classification. *IEEE Transactions on Evoluationary Computation*, 10(5):604–616, 2006.

[18] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in Genetic Programming. In Y. Davidor, H.-P. Schwefel, and R. Mnner, editors, *Proceedings the Third Conference on Parallel Problem Solving from Nature*, pages 312–321. Berlin: Springer-Verlag, 1994.

[19] S. Hettich and S. D. Bay. The UCI KDD Archive [http://kdd/ics/uci/edu]. Irvine, CA: University of California, Dept. of Information and Comp. Science, 1999.

[20] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. *Pattern Directed Inference Systems*, 1978.

[21] H. Iba. Bagging, boosting and bloating in Genetic Programming. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1053–1060. San Francisco, CA: Morgan Kaufmann, 1999.

[22] F. H. Bennett III, J. R. Koza, J. Shipman, and O. Stiffelman. Building a parallel computer system for $18,000 that performs a half peta-flop per day. In Banzhaf W, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1484–1490. San Francisco, CA: Morgan Kaufmann, 1999.

[23] K. Imamura, T. Soule, R. B. Heckendorn, and J. A. Foster. Behavioral diversity and a probabilistically optimal GP ensemble. *Genetic Programming and Evolvable Machines*, 4(3):235–253, 2003.

[24] N. Japkowicz. Why question machine learning evaluation methods? (An illustrative review of the shortcomings of current methods). In C. Drummond, W. Elazmeh, and N. Japkowicz, editors, *AAAI-2006 Workshop on Evaluation Methods for Machine Learning, Technical Report WS-06-06*, pages 6–11. Menlo Park, CA: AAAI Press, 2006.

[25] H. Juillé and J. B. Pollack. Massively parallel Genetic Programming. *Advances in Genetic Programming*, 2:339–357, 1996.

[26] T. Kovacs. Deletion schemes for classifier systems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 329–336. San Francisco, CA: Morgan Kaufmann, 1999.

[27] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[28] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Berlin: Springer-Verlag, 2002.

[29] P. L. Lanzi and R. L. Riolo. Recent trends in Learning Classifier Systems research. *Advances in Evolutionary Computing: Theory and Applications*, pages 955–988, 2003.

[30] C. W. G. Lasarczyk, P. W. G. Dittrich, and W. W. G. Banzhaf. Dynamic subset selection based on a fitness case topology. *Evolutionary Computation*, 12(2):223–242, 2004.

[31] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[32] M. Lemczyk and M. I. Heywood. Pareto-coevolutionary Genetic Programming classifier. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, Franz R., C. Ryan, and D. Thierens, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 945–946. New York, NY: ACM Press, 2006.

[33] M. Lemczyk and M. I. Heywood. Training binary GP classifiers efficiently: A Pareto-coevolutionary approach. In M. Ebner, M. O'Neill, A. Ekrt, L. Vanneschi, and A. I. Esparcia-Alczar, editors, *Proceedings of the European Conference on Genetic Programming*, pages 229–240. Berlin: Springer-Verlag, 2007.

[34] P. Lichodzijewski and M. I. Heywood. GP classifier problem decomposition using first-price and second-price auctions. In M. Ebner, M. O'Neill, A. Ekrt, L. Vanneschi, and A. I. Esparcia-Alczar, editors, *Proceedings of the European Conference on Genetic Programming*, pages 137–147. Berlin: Springer-Verlag, 2007.

[35] P. Lichodzijewski and M. I. Heywood. Pareto-coevolutionary Genetic Programming for problem decomposition in multi-class classification. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 464–471. New York, NY: ACM Press, 2007.

[36] A. McIntyre and M. I. Heywood. Multi-objective competitive coevolution for efficient GP classifier problem decomposition. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, pages 1930–1937. IEEE Press, 2007.

[37] A. R. McIntyre and M. I. Heywood. MOGE: GP classification problem decomposition using multi-objective optimization. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, Franz

R., C. Ryan, and D. Thierens, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 863–870. New York, NY: ACM Press, 2006.

[38] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/∼mlearn/mlrepository.html]. Irvine, CA: University of California, Dept. of Information and Comp. Science, 1998.

[39] J. Noble and R. A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 493–500. San Francisco, CA: Morgan Kaufmann, 2001.

[40] A. Orriols-Puig and E. Bernado-Mansilla. Bounding XCS's parameters for unbalanced datasets. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, Franz R., C. Ryan, and D. Thierens, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1561–1568. New York, NY: ACM Press, 2006.

[41] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[42] L. Panait, S. Luke, and R. P. Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.

[43] M. Potter and K. de Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[44] D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training Genetic Programming on half a million patterns: An example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, 2005.

[45] T. Soule. Cooperative evolution on the intertwined spirals problem. In E. Cant-Paz, J.A. Foster, K. Deb, L.D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, J. Wegener, D. Dasgupta, M.A. Potter, and A.C. Schultz, editors, *Proceedings of the European Conference on Genetic Programming*, pages 434–442. Berlin: Springer-Verlag, 2003.

[46] C. Stone and L. Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.

[47] R. Thomason and T. Soule. Novel ways of improving cooperation and performance in ensemble classifiers. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1708–1715. New York, NY: ACM Press, 2007.

[48] K. Tumer and D. Wolpert. A survey of collectives. *Collectives and the Design of Complex Systems*, pages 1–42, 2004.

[49] R. A. Watson and J. B. Pollack. Coevolutionary dynamics in a minimal substrate. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 702–709. San Francisco, CA: Morgan Kaufmann, 2001.

[50] G. Weiss and F. Provost. The effect of class distribution on classifier learning. In *Technical Report ML-TR 43, Department of Computer Science, Rutgers University*, 2001.

[51] S. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[52] S. W. Wilson. Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. Fogel, M. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. San Francisco, CA: Morgan Kaufmann, 1998.

[53] S. W. Wilson. Get real! XCS with continuous-valued inputs. In L. Lanzi P, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications*, pages 209–222. Berlin: Springer-Verlag, 2000.