Pareto-coevolutionary Genetic Programming Classifier

by

Michal Lemczyk

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

at

Dalhousie University
Halifax, Nova Scotia
May 2006

Abstract

The conversion and extension of the Incremental Pareto-Coevolution Archive algorithm (IPCA) into the domain of Genetic Programming classifier evolution is presented. The coevolutionary aspect of the IPCA algorithm is utilized to simultaneously evolve a subset of the training data that provides distinctions between candidate classifiers.

To remain practical, a method for limiting the sizes of the IPCA archives is required. Several archive pruning basis are proposed and evaluated. Furthermore, methods for consolidating the resultant pareto-front of classifiers into one label per testing data point are presented and evaluated.

The algorithm is compared against; a traditional GP classifier using the entirety of the training data for evaluation, in addition to GP classifiers which perform Stochastic, Cycling, and Dynamic subset selection.

Results indicate that the PGPC algorithm often outperforms the subset selection algorithms in terms of classification performance, and often outperforms the traditional classifier while requiring roughly $\frac{1}{128}$ of the wall-clock time.

Chapter 1

Introduction

Binary classification problems within the context of a supervised learning paradigm provide the basis for a wide range of application areas under machine learning. However, in order to provide scalable as well as effective solutions, it must be possible to train classifiers efficiently. Although Genetic Programming (GP) has the potential to provide classifiers with many desirable properties, the computational overhead in doing so has typically been addressed through hardware related solutions alone [21],[5],[14].

This work concentrates on how the training process can be made more efficient by evaluating classifier fitness over some adaptive subset of the total training data. To date, the typical approach has been to utilize an active learning algorithm for this purpose, where the Dynamic Subset Selection (DSS) family represents one widely used approach [7], [17],[24]. An alternative approach to the problem, however, would be to formulate the problem as a competition between two populations, one representing the classifiers, the other the data.

Progress has recently been made using Genetic Algorithms based on a Pareto foundation of this coevolutionary approach, albeit within the context of player behaviours in gaming environments. In particular, this work is based on the "IPCA" algorithm, where this has been shown to address several potential problems with the competitive coevolutionary approach (relativism, focusing, disengagement, and intransitivity) [2].

The algorithm reported in this work, hereafter denoted the Pareto-coevolutionary Genetic Programming Classifier (PGPC) is novel in the fact that it extends a Genetic Algorithm "game-playing" context into the domain of GP classification. Furthermore, pruning is utilized to limit the sizes of the IPCA algorithm archives (the respective pareto-fronts) using various basis to allow for efficient execution. This differs from the method employed in the follow-up of the IPCA algorithm (LAPCA) [11], which used a NSGA [25] - inspired approach which stored the top N pareto-layers of the archive, keeping the pareto-front in its entirety. Additionally, PGPC differs from the methods utilized by various EMOO algorithms [15], [29] which tended to perform clustering on the pareto-front of solutions (learners) using the GA coordinates to limit the pareto-front size. In a GP environment, clustering of learners (programs) is difficult as compared to Cartesian point coordinates, so using the structure made available by the coevolution of learners and training data points, the interactions of the learners against the points is used as a means to perform pruning.

The PGPC algorithm will be evaluated against a traditional GP classifier using the entire training data for learner evaluation, a GP classifier that implements Stochastic sampling (a fixed size, randomly selected subset), a GP classifier that implements a cycling subset selection algorithm (a sliding window scheme), and a GP classifier that implements Dynamic Subset Selection as per [7] (age and difficulty consideration for each training point). Classification "score" (equal weight detection rate and 1 - false positive rate), and run-time will be used for evaluation and comparison.

Success will be defined as either meeting the traditional GP algorithm's classification performance using less time, or exceeding the comparison subset selection algorithms' performance using comparable time (a factor of the traditional GP algorithms time). It should be noted that classification performance and run-time are two objectives with a continuum of trade-offs existing between them. The aforementioned measures of success are chosen to simplify the continuum, as either classification performance is loosely fixed to the traditional algorithm and run-time is measured, or time is fixed as being less than the traditional GP algorithms, and classification performance is compared. The latter objective avoids degeneracies such as a very fast algorithm which is completely useless for classification purposes, or an extremely slow algorithm which happens to be very accurate.

1.1 Approach

The co-evolutionary approach of the IPCA algorithm will allow for the “binding” of the learner and training data point subset evolutions, keeping the point subset relevant to the current set of learners. The pareto-front of learners allows the system to explore the search space along different attractors present in the data, and hopefully provide a diverse set of optimal solutions. In regards to the pareto-front of points, each point is pareto-equivalent to the others in the front, and as such provides a “distinction” between the learners that is not duplicated in the archive, as per IPCA. Therefore the pareto-front of points itself is the subset of training data that provides a learning gradient for the learners.

In regards to evolution of the point subset, the point pareto-front will “move” with the learners. The point subset attempts to represent the most relevant subset of points for distinguishing between the current population of learners. The final pareto-front of points after pruning is not indicative of a global relevant subset, rather a currently relevant one. It is still possible to extract useful information out of the final subset as it constitutes a set of more non-trivial data points as opposed to ones easily defeated early on in the learner evolution. Furthermore, the point pareto-front may also be “tracked” during the evolution (output at each generation), to determine a globally relevant subset of points that provide a learning gradient for the entire evolution. Thus allowing informative analysis of the input problem.

The original IPCA algorithm performed no pruning on the learner and point archives. Empirically, the learner archive (pareto-front) remained small, and the point archive which contained the current set of relevant points in addition to the previously relevant set grew without bounds. To address this issue, the LAPCA algorithm stored the point archive in layers, with the pareto-dominant front as the first layer, and the remaining points forming subsequent ones [11].

In the case of the proposed PGPC algorithm’s environment, experiments showed that both the learner and point pareto-fronts grow dramatically on the training data sets, since it may be that each training point is an underlying objective and provides a distinction between learners (see Section 5.4). To retain efficiency, the previously relevant points may not be stored, nor can the pareto-fronts in their entirety. A pruning method must be adopted to limit the size of the archives.

Finally, in the context of a classification problem, a heuristic is required to define how the pareto-front of learners is consolidated to return one class per testing point. Since the pareto-front of learners may be diversified to correctly classify subsets of the data, a method to recognize and utilize any structure inherent in the front must be developed. In short, the co-evolutionary approach decomposes the problem, with different learners from the pareto-front solving different parts of the underlying problem. Various voting schemes will be investigated for determining how the set of learners provide class labels to unseen exemplars.

Moreover, a pareto-front of learners may still be valuable in its raw form as a tool in the analysis of the diverse solutions to the classification problem, where an expert in the domain may identify and analyze any pareto-solutions of interest.

In summary, in order to utilize the IPCA framework from GAs within a GP classification context, three issues need to be addressed:

- Transferring the IPCA algorithm into the domain of a GP environment. This includes the issues of individual generation, selection, and evaluation without the symmetry present in a GA algorithm between learners and points (the learners are represented as GP trees, while the points are indices into the training data set).
- Limiting the size of the archives via pruning. Namely on which basis are individuals accepted and removed from the archives.
- Solutions from training do not take the form of a single “super” classifier, but a set of classifiers, that is, the contents of the pareto-front. A methodology is necessary for determining how classification on unseen data is determined using the contents of the pareto-front, without the aid or use of a domain expert.

Chapter 2

Background and Related Work

2.1 Genetic Programming

The framework for performing an evolutionary search within the space of solution-programs as defined by Genetic Programming (GP) [16], is a suitable mechanism for attaining a classifier system. This is especially true in cases where domain-related knowledge is limited, as the genetically-inspired properties of the framework yield a stochastic search that includes both explorative and exploitative aspects, and therefore provides a "general" search.

GP achieves this by extending the concepts of Genetic Algorithms [8] from a co-ordinate based domain, into one composed of programs. Yielding a population of individuals (programs) which undergo selection based on fitness with regards to the target problem (i.e., classification) followed by breeding the next generation through various operators. Ultimately yielding a singular individual or a population which are "fit" with respect to their target problem and may be utilized for system testing on unseen data.

Benefits of the framework include; an intuitive basis founded upon biological concepts, allowing for the framework and operation to be easily explained to a broad-ranged audience (i.e., generations, populations of individuals, genetic encoding, mutation, and re-combination). The aforementioned capacity for both exploration and exploitation of the search space by multiple independent individuals (locations within the search space), allows for the framework to make progress towards multiple local optima simultaneously, and thus in effect not be bound to a singular optimum which may not be global.

Unfortunately, the previously described nature of the GP framework may prove to be detrimental and inefficient in certain cases where alternate search paradigms such as steepest descent would prove superior (i.e., unimodal function optimization). Furthermore, the stochastic aspects of the GP framework may combat the detrimental effects of deceptive information within the search space, however they may also lead to the re-visiting of previous states. Additional memory may be allocated to store all previous states and thereby identify instances of re-visiting , however the costs of doing so is typically prohibitive. Finally, parameter selection related to the operation of GP is at best an intuitive exercise. Although more generations and a larger population of individuals would seem to increase evolutionary performance with respect to the problem at hand, the costs in terms of both time and memory requirements would also increase, yielding the question of what values are adequate or appropriate? Additionally, the whole aspect of individual selection and breeding result in complex behaviour, as the resultant population diversity and progress towards the objective may not be directly related to specific parameters. Furthermore, the definition of the individuals themselves requires attention, as to how they are structured (e.g., syntax trees, or a linear sequence of register instructions), which operations may be utilized within the programs, initialization of the individuals, and limits upon their size and complexity need to be specified.

The most pertinent aspects of GP to this work concern the efficient evaluation of the population of individuals, and the mechanism for selecting the population member(s) to be utilized for testing of the system.

In regards to efficiency; dependent upon the implementation, at each generation all individuals must be evaluated on the training data (as per fitness proportionate selection). Yielding the generalized expression for the number of function evaluations performed by GP of: population size * number of generations * number of training exemplars * number of trials (under different random number seed values). Assuming a canonical GP parameterization (5000 individuals, 50 generations, 50 trials) on a semi-typical training exemplar data set (5000 exemplars), the number of function evaluations falls in the range of $6.25 * 10^{10}$. If a “large” data set is utilized; of roughly

500 000 training exemplars, as utilized in the experimentation of this work, then the number of function evaluations increases to the range of $6.26 * 10^{12}$. Assuming that the optimal set of GP parameters is unknown, multiple runs may be necessary to discover a set that proves adequate for the task at hand, further increasing the number of evaluations. Concerning the previous example; the employment of an efficient method for evaluating the population of individuals on the training data would decrease the computational complexity of the two most significant terms, as such returning the GP evolutionary process upon large data sets back into the realm of practicality.

In terms of system testing, a classical best-trained individual in the population scheme assumes that there is a singular optimal solution to the input problem, often based on a singular fitness value (accuracy), or in cases of multiple apparent objectives, they may be arbitrarily weighed into one value. This assumption may very well be false, in that it may be desirable to evolve a set of individuals spanning the trade-off "front" between various objectives. Appropriate objectives might include independent accuracy over various sub classes in the problem, in the simplest case balancing detection rate against false positive rate, or external factors such as individual size [12]. Moreover, at the very least ensemble methods have demonstrated that sets of (weak) learners, combined under an appropriate voting scheme, provide a very strong model of learning [19]. Rather than taking an explicit weak learning methodology, we are interested in encouraging problem decomposition during training as part of the learning activity, where this has the potential to provide more transparent solutions as well as accurate models.

2.1.1 Training Data Subset Selection

To address the issue of efficiency in evaluating a GP individual, whether it be each individual as in a fitness proportionate selection scheme, or a subset as in a tournament scheme, a reduction in the number of training data points utilized would be beneficial; all the while maximizing the accuracy of the fitness measure of the individuals.

If the assumption is made that each training point is equally informative, then it may be said that the number of points utilized in the evaluation of an individual would be linearly proportionate to the accuracy of the fitness measure of the individual, regardless of the subset selection mechanism.

Within real-world classification problems, this assumption may very well not hold. It may be the case that certain points duplicate the information of others, or are not significant with regards to the classification. For example, the informative value of points on the periphery of a cluster may be more important than the informative value of the points on the interior. Furthermore, outliers may have an even greater significance (or lesser, depending on the classification paradigm).

Therefore depending on the mechanisms employed by a training data subset selection algorithm, the impact of the subset upon the accuracy of an individuals fitness measure may be substantial.

Even if the variance of informative value of the various points is acknowledged and accounted for in a static sense, the state of the classifiers being evolved at any given point in time is an additional dimension in the value of a training data point. For example, two linearly separable clusters of points may be considered “easy” as opposed to two intertwined clusters, and it may be desirable to have the evolutionary system “learn” them first, and then proceed to the more difficult points. In other words, a learning gradient for the classifiers to follow may be desirable, and as such, different points would occupy different sections of the gradient. Therefore a subset’s contents with regards to the current gradient would affect an individual’s fitness score in addition to the selective pressure required by the GP framework to attain convergence to a suitable solution.

In summary, the problem of selecting an appropriate subset of training points to evolve the classifiers upon at a given point of evolution may be as difficult as solving the original classification problem. Since the subset selection task may very well be viewed as a classification problem; given the current state of the system, should this point be included in the subset or not?

The following works describe various algorithms for performing the subset selection task, with varying outcomes:

- The Stochastic Subset Selection algorithm (SSS) [22] randomly selects a fixed sized subset of training points at each evaluation of any individual. Within its GP implementation, it was found to increase evolution speed in addition to maintaining diversity in the population, so as to escape local optima. In other words, classification quality was attributed to the addition of stochastic sampling to the training data via the subset selection mechanism, where obvious similarities exist with bagging based sampling methods (the solution however still takes the form of a single "super" individual).

- The Dynamic Subset Selection algorithm (DSS) [7] reduces the number of evaluations of a GP algorithm by selecting a subset of training points based on the "age" and "difficulty" of each point. With age being the number of generations since the point was last included in the subset, and difficulty being the number of misclassifactions of the point during that time. A weighed value of the two yields a selection probability for entry to the next subset. Results indicated that the DSS method may perform better than one utilizing the entirety of the training data.

  The inclusion of age and difficulty acknowledges the variability of information value of the points in addition to their current location upon the learning gradient. That is to say, the learners may forget previously learnt exemplars (hence the age parameter), whereas a bias is assumed towards training points that are more difficult to classify. A notable point is the requirement of the algorithm to have the weighing of age and difficulty arbitrarily specified by the user. If this is selected incorrectly, say a too high of a weight on difficulty, then the learning gradient provided by the subset would suffer; as very few individuals would correctly classify the subset contents, and as such the resultant fitness values would be very similar, leading to a loss of selection pressure (i.e., disengagement). Conversely, if too low of a weight is placed on difficulty, then the bias towards correctly classifying difficult points is lost; it may be possible for far too many learners to correctly classify the subset, yielding similar fitness values, and as such lead again to a loss of selection pressure.

- The RSS-DSS algorithm [24] applies a hierarchical approach to classification on a large input data set for the purposes of anomaly detection in a network environment. The combination of a steady state tournament and I/O efficient blocks of training data arranged in a Random Subset Selection (Randomly chosen subset, common to all evaluations per generation) into a Dynamic Subset Selection scheme (performed on the subset of RSS selected blocks) resulted in considerable evaluation speed increases.

  The requirement on the user to specify the DSS age-difficulty weight still exists, in addition to parameters specifying the number of evaluations per hierarchical step (width/height of the hierarchy).

- The Topology Based Selection (TBS) method [17] constructs a topology upon the training cases, such that an occurrence of an individual correctly classifying a pair of cases, strengthens the edge value between those two cases, yielding a graph describing "similarity" between cases. A binary search over possible edge-value thresholds is performed as to return a desired sized subset of cases using a vertex cover-like algorithm over the thresholded graph. Comparisons against the DSS and SSS algorithm indicate that the TBS method evolves faster, possibly due to the avoidance of premature convergence.

  Although the computation of the topology of the training cases yields information analyzable by problem-domain experts, the additional computational overhead of computing and storing the topology may be prohibitive for large data sets. As stated in the referenced paper, for a training set of size V, the number of edge values is $O(V^2)$, and the computational complexity of sorting the edges as to perform a binary search for an appropriate threshold value is $O(V^2 \log V)$.

  The algorithm does require that relatively minor parameters be specified, namely the rate at which an edge value of the topology diminishes over time (as to prefer newer evaluations), and the maximum number of binary search iterations to perform to find a balanced threshold value.

It should be noted that all of the previously described methods for performing subset selection are in a sense arbitrary. In that the basis for subset selection is predefined (say age and difficulty), and the mechanisms for selection are fixed (say random selection, or random with probability influenced by some arbitrarily weighed criteria). No use of evolutionary concepts or approaches is made, and as such leave an area of exploration for the proposed algorithm. Moreover, the disengagement problem is not explicitly addressed, thus point subsets do not explicitly represent those most appropriate for distinguishing between the current performance of learners. Instead a priori defined metrics are assumed to be representative of biases appropriate for selecting relevant point subsets.

2.1.2 Training Data Class Balance

In the context of selecting a subset of training points for the purposes of evolving a classifier, the ratio of the respective classes within the subset or even the input set may have an impact upon the resultant classifier's performance.

Japkowicz and Stephen in [10] utilize artificial data sets with varying levels of size, complexity, and imbalance to evaluate Over-sampling (forcing a balanced training set through over-representing the smaller sized class), Under-sampling (forcing a balanced training set via the removal of instances of the over-represented class), and Cost-modifying (varying the fitness function by altering the cost of misclassifying instances of each class to reflect the input balance) on the C5.0 algorithm. Results indicate an improvement over the raw data set, with the improvement varying with the method utilized. With the least optimal being Under-sampling, and the optimal being Cost-modifying, as it achieves the same goals as over and under sampling without increasing the training set size (Oversampling), or reducing the amount of information available (Undersampling).

Their additional experiments consist of evaluating the sensitivity of other classification algorithms (Multi-Layer Perceptrons, and Support Vector Machines) to class imbalance and the effects of the various remedial algorithms. Results indicate oversampling being superior to undersampling for MLPs, and SVMs being completely insensitive to the class balance issue, with any class balance processing yielding detrimental results.

As such, it may be inferred that class balance within a training data subset employed in a GP environment may have an impact upon classifier performance. It should be noted that a limited size training data subset interferes with the definitions of over and undersampling, as a class balanced subset in fact is Under-sampling of the majority class (to attain a subset, one must remove data), and may also require Over-sampling of the minority class in the case where the minority class would not fill half of the subset.

The work of Weiss and Provost [27], extends onto the class balance concept by evaluating the relationship between training data class balance (distribution), and classifier performance of the C4.5 algorithm. Results indicate that to maximize accuracy, the natural input balance is generally reasonable, and in order to maximize the area under the Receiver Operating Characteristic (ROC) curve, a balanced distribution is reasonable, however the optimal balance seems to be problem dependent.

In summary, the aforementioned works illustrate the (possible) importance of class balance in the training of a learning system. As such, class considerations with regards to the training data will be made by the proposed algorithm, bearing in mind the optimal training balance may be problem dependent.

2.2 Co-evolution

2.2.1 Multi-objective Methods: Co-operative Co-evolution

In evaluating individuals, an arbitrary decision is often made on how to weigh in various aspects of “fitness” into a singular value as to decide upon an optimal solution through a common ranking of the candidates. The following works acknowledge the possibility of multiple objectives within a problem, and provide mechanisms to select a set or "front" of solutions spanning the trade-offs between the various objectives. In a sense, evolving multiple sub-populations of individuals, with each sub-population being specialized to a subset of the problem (section of the trade-off front), in other words co-evolving co-operating populations.

Many of the early algorithms within this field incorporated multi-objective considerations into a Genetic Algorithm (GA) framework [9], [25], and [4]. In order to ascertain the multi-objective fitness values of differing individuals (vectors as opposed to singular values), a mechanism for comparing two individuals was required, one that did not arbitrarily weigh the various objectives (vector entries) in any manner. Commonly, pareto-dominance (often referred to as just "dominance") was utilized. With pareto-domination between two individuals being defined as the dominating individual having a greater-than-or-equal value for each vector index (objective), with at least one value being strictly greater than, in other words:

$$dominate_{Obj}(a, b) \Leftrightarrow \forall i : Obj_i(a) \geq Obj_i(b) \wedge \exists i : Obj_i(a) > Obj_i(b)$$

Such a comparison or ranking scheme allowed for individuals specialized on different aspects of the trade-off front (pareto-front), to be fairly judged with no bias being introduced to any of the objectives. It may only be said that one individual is superior to another (fitter), only if it pareto-dominates it (it is superior across all the underlying objectives).

Stemming from these works, the need to maintain a separate elitist "archive" containing the current optimal pareto-front (non-dominated individuals) was identified [15], [29], and [28]. As storing the pareto-front within the GA population allowed for the potential loss of pareto-front members, which may have been highly specialized and difficult to re-breed. Furthermore, maintaining an external archive allowed for easy comparisons of generated candidate individuals against the currently known best solutions, in addition to simplifying the use of a niching bias to promote diversity along the front (a uniform distribution) through archive entry criteria, as well as utilizing the archive for individual generation purposes, as the GA population was allowed to perform more of an exploratory role, while the archive performed more exploitation of the currently known optimal front (localized exploration).

With regards to maintaining the separate archive, several methods for limiting the size of, or condensing the archive were proposed. They include archive entry criteria based methods, utilizing niching (rejecting pareto-equivalent individuals residing in “crowded” areas), as well as several clustering methods performed on the archive points (consolidating nearby points into one representing the cluster).

2.2.2 Competitive Co-evolution

A paradigm pertinent to this work and related to Multi-objective methods is competitive co-evolution of multiple populations of individuals. Where the evolutionary system is defined by the contents and interactions between the said populations. Often viewed within a “gaming” context, a competitive coevolutionary system allows for the simultaneous evolution of both game players and opponents, avoiding the requirement of an external set of opponents to be developed, which may be just as difficult as producing the desired player, especially in symmetric games.

Within such a paradigm, multi-objective concepts may be observed since the defeat of a singular opponent may be considered a single objective, and likewise a set of opponents, multiple objectives, with the trade-offs between a set constituting a front (as in co-operative co-evolution), however the objective of the players is often to defeat the opponent, hence competitive co-evolution.

The following works present various observations and algorithms related to competitive coevolution pertinent to this work:

- The work of Watson and Pollack [26] provides a suitable introduction to the co-evolutionary framework and subsequent identified potential issues which plague it. These include the loss of the learning gradient between the two populations (dis-engagement), undesirably over-focusing on a weakness of an opponent yielding degenerate solutions, and “relativism” which is the disconnect between the perceived performance of the populations against a global metric (i.e., two bad players playing against each other would have the same score as two good players playing against each other; the gradient between good and bad players is not present).

The referenced work continues on to describe an artificial environment (minimal substrate) to observe and analyze these behaviours in a controlled manner independent of any problem specific or induced behaviour. Observed results on the defined minimal substrate indicate the presence of these coevolutionary issues even within a simple numerical game environment, and their effects upon both subjective and objective fitness values.

To attain a suitable measure of objective performance, a coevolutionary system should address the aforementioned issues in order to make the most of the interactions between multiple populations towards the goals specified by designer.

- Noble and Watson [20] utilize pareto-concepts in a coevolutionary set-up to evolve "Texas Hold'em Poker" Genetic Algorithm strategies. As stated by them; the use of pareto or multi-objective optimization methods makes the concept of "players as dimensions" explicit, and as such may avoid issues of intransitive superiority (rock-paper-scissors), as a pareto approach would provide a diverse set of players, rather than a best-on-average.

  Poker strategies, encoded as a numeric sequence of parameters to playing "thresholds" (bluffing probability, hand contents which qualify as a "strong" hand, etc) are initialized and played multiple times per each generation. Analysis of which strategies won or lost chips to other strategies results in the identification of pareto-dominant strategies. The pareto-front was retained, and any remaining slots in the population were filled by the offspring of randomly chosen members of the pareto-front, using crossover and mutation. The resultant population of strategies was then re-evaluated again.

  Within their observations of their preliminary algorithm, they observed that their entire population of strategies was often contained within the pareto-front. To address this issue, they divided their population by half into the pareto-front (archive) and normal population. In the case that the pareto-front archive size was exceeded, a basis for pruning was selected, namely removing strategies which defeated a small number of opponents (but still remained pareto-equivalent); in other words, a greedy method.

Results of their method against a fixed set of manually constructed strategies indicated superior performance for the pareto-coevolutionary method.

- In regards to combining considerations to the previously mentioned issues related to coevolution, and a pareto-based approach to Genetic Algorithms, the DELPHI algorithm [13] describes a mechanism for evolving a set of "tests" alongside "learners", in which the tests constitute (evolve towards) a "Complete Evaluation Set" which provides an accurate measure of learner evaluation.

  Within the context of evaluating an evolving set of learners (aka: individuals, players, in the case of this work; classifiers), the notion of "underlying objectives" of a problem describes the minimal set of objectives, or features, or aspects of the problem which determine the outcome of an interaction. If these objectives are known, they may be manually specified as a basis for a learner to be evaluated upon (singular evolution). However, if these objectives are unknown (or in the case of this work, an unknown subset of a large input set), the evolution of a set of tests towards the set of underlying objectives would provide an accurate basis for learner evaluation. Furthermore, since the number of underlying objectives may be plural, a multi-objective approach to both learner and test evolution would allow for the exploration and identification of the set of underlying objectives and the corresponding learners.

  A notable aspect of the cited work is the use of "distinctions" as a fitness objective of the tests, as opposed to defeating the population of learners. Often coevolution is structured as to have two populations competing against one another (with the objective being victory), as to develop an "arms race" between the two. As previously observed, this may lead to situations where there is a disconnect between the two populations (loss of gradient), as one completely dominates the other. By using the ability of a test to provide distinctions (the outcomes against the test can differentiate between two learners), the tests evolve towards a more central location between the set of learners, as opposed to a higher, superior location. As such, the objectives of the tests in fact become to provide a learning gradient for the learners, and they will "move" with the changing set of learners to maintain their fit, gradient-producing position.

The use of a pareto-based approach in conjunction with a distinction based fitness objective for the tests, results in a system which avoids many of the issues faced by coevolution.

A summary of the DELPHI algorithm follows:

1. Randomly initialize the populations of learners and tests.
2. While the learner criterion has not been met (number of generations, etc)
   (a) Combine the learner population with one generated from it.
   (b) Combine the test population with one generated from it.
   (c) Compute the interactions between all learners and all tests.
   (d) Compute the distinctions make by each test with regards to the previous step.
   (e) Evaluate the tests based on their pareto-ranking on distinctions, and retain the most fit into the population.
   (f) Evaluate the learners based on their pareto-ranking on outcomes (victories), and retain the most fit into the population.
   (g) Loop.

Specifics with regards to evaluation and retention may be applied based on the problem and encoding. In the cited work, learners are selected only if they dominate some existing member (the dominated member is replaced), and tests are selected only if they dominate their parent, as to preserve possibly found underlying objectives (diversity).

Experimental results indicate progress on simple artificial problems containing multiple underlying objectives, superior to comparison methods employing fitness measures of average value against the other population, pareto-based non-dominational sorting, and variants of the described method.

As stated by the authors, issues and unexplored aspects of the work include the amount of exploration possible within one generation by each of the populations. Referencing the work done in the evolutionary multi-objective optimization field, the use of an archive and population per each of the learners and tests, would allow for exploration by both while still retaining the pareto-front. Furthermore, issues relating to the movement of individuals along the front (trade-offs between the multiple objectives) again my be resolved by looking at the EMOO field.

- Extending the DELPHI algorithm, the Incremental Pareto-Coevolution Archive (IPCA) algorithm [2] addresses many of the issues raised by the previous algorithm, and subsequently incorporates many of the developed EMOO methods to the pareto-coevolutionary system. Primarily, the use of a separate archive to store the pareto-fronts of the respective learner and test populations.

  Utilizing an archive to store the current pareto-front of individuals, any generated individuals may be compared against it, and as such forgetting previously learnt traits within the populations may be avoided since the archive may only be overwritten by superior individuals (in the case of this work and the IPCA algorithm, superiority is defined by pareto-dominance), yielding a guarantee of monotonic progress.

  The following definitions are pertinent to the algorithm, they are:

  - A learner L is "useful" with respect to a set of learners LS, and a set of tests TS, if it is not pareto-dominated by the outcomes of any learner in

LS over the set TS, and no learner in LS has equal outcomes over TS to the learner.

$$\begin{aligned} useful_{Learner}(L, LS, TS) \quad \Leftrightarrow \quad & \nexists L' \in LS : dominate_{TS}(L', L) \\ & \wedge \\ & \nexists L' \in LS : \forall T \in TS : \\ & outcome(L, T) = outcome(L', T) \end{aligned}$$

In other words, a useful learner is pareto-equivalent or dominant, and phenotypically unique to the pareto-front.

– A test T is "useful" if its addition to a set of tests TS identifies a generated learner from the set GL deemed not-useful with respect to a set of learners LS, and tests TS, as useful.

$$\begin{aligned} useful_{Test}(T, TS, GL, LS) \quad \Leftrightarrow \quad & \exists L \in GL : \overline{useful_{Learner}}(L, LS, TS) \\ & \wedge \\ & useful_{Learner}(L, LS, TS \cup T) \end{aligned}$$

In other words, the addition of a useful test to TS makes a pareto-dominated or not-unique learner pareto-equivalent or dominant and unique, thus providing a new distinction.

– In conjunction with the previous definitions, if a generated learner is undefeated by any test in the set TS, and a generated test defeats the learner, then the learner and test are deemed useful. It should be noted that the learner would be labeled useful according to the first definition, as it would pareto-dominate the set of learners LS, unless its outcomes are equivalent to another pareto-dominant learner in the set.

$$\begin{aligned} useful_{Test}(T, TS, LS) \quad \Leftrightarrow \quad & \exists L \in LS : \forall T' \in TS : \\ & L \text{ defeats } T' \wedge T \text{ defeats } L \end{aligned}$$

The algorithm follows:

1. Loop over the number of generations:

  (a) Remove any pareto-dominated learners in the learner archive.
  (b) Generate a new population of learners.
  (c) Generate a new population of tests.
  (d) Identify the set of useful generated tests, and add them to the test archive.
  (e) Identify any useful generated learners, and add them to the learner archive.

If learners and tests are generated with non-zero probability, then the acceptance criteria of the archives provide a guarantee of monotonic progress of the system.

It should be noted that the learner archive is trimmed to only maintain pareto-dominant individuals, however the test archive is not. This allows for the retention of previously identified tests which provided a distinction on a possible underlying objective, but no longer do so. Their retention allows the for the retainment of progress over that possible underlying objective by the pareto-front of learners, and as such their removal under some pareto-dominant selection criterion based on distinctions would allow for possible loss of selection pressure, in other words “forgetting”.

Experimental results of the IPCA algorithm against the DELPHI algorithm show comparable performance over a continuous search space, and consistent progress on a discretized search space where the separation of archive and population provides more freedom for exploration.

A noted issue of the IPCA algorithm is the management of an unbound test archive, and its practical implications, and the possibility of limiting the archive sizes at the expense of reliability.

- Extending the IPCA algorithm, the LAyered Pareto-Coevolution Archive (LAPCA) algorithm [11] applies an NSGA inspired approach to limiting the size of the archives, via a tunable number of layers.

As stated in the cited work, the IPCA algorithm maintains a single layer of pareto-equivalent learners, however the test archive may grow indefinitely. To address this issue, the LAPCA algorithm maintains a fixed number of n learner layers with the first being the pareto-equivalent set of learners, the second being the pareto-equivalent set if the individuals in the first layer are not considered, and so on. The tests are then structured into layers of sets separating the layers of learners, where the tests in a layer provide distinctions between the layers of learners.

Experimental results against the IPCA algorithms indicate superior progress for the LAPCA algorithm over a discretized search space. Furthermore, the IPCA test archive grows indefinitely as the number of generations progress, whereas the LAPCA algorithms test archive size tends to be limited depending on the number of specified layers.

Acknowledged by the author of the cited work, a limitation of the algorithm is the fact that the sizes of the layers (pareto-fronts) may still be excessive. Suggestions are made to transfer over concepts from the EMOO field of research to address this issue (e.g., clustering).

- Investigating the effects of a pareto-coevolutionary approach utilizing an archive to store informative tests, upon the problems of intransitivity and cycling, the work of de Jong [3] compares the performance of the IPCA and LAPCA algorithms against an IPCA variant without the archive over two intransitive number games.

  Results indicate that monotonic progress may be made upon an intransitive search space via the paradigm of viewing an intransitive relationship (e.g., Rock, Paper, Scissors), as a set of dominance relationships. Furthermore, the use of an archive to store informative tests avoids cycling through regression of the populations.

A final consideration of the cited work is the question of efficiency while maintaining reliable progress. Most of the monotonically-increasing pareto-coevolutionary approaches maintain a sizable archive to store previously found solutions or informative points as to avoid forgetting; although acceptable in academic contexts, real-world applications may find this to be prohibitive.

2.3 No Free Lunch Theorems

Within the context of search problems, or in the case of data classification; searching for the correct classification, the "No Free Lunch" theorem and associated implications [23] must be considered. Briefly stated, the No Free Lunch theorem describes the performance of a search algorithm on a search space as a sequence of steps or states, where under permutation of the search space, two different search algorithms would follow the same sequence of steps. Leading to the conclusion that over all search problems, the performance of all search algorithms including brute force enumeration would be equal.

As such, there is no universally optimal or efficient search algorithm as any set of search heuristics or strategies may be broken by a "worst case scenario". Therefore choosing an appropriate search strategy requires domain knowledge of the problem to exploit.

The NFL theorems have ramifications to the PGPC algorithm proposed by this thesis, in that over all data sets, it would exhibit equal performance to other algorithms. However, if real-world data problems exhibit a measure of similarity, the PGPC algorithm may turn out to be attuned to the problems, and as such exhibit efficient behaviour.

Chapter 3

Algorithm Description

3.1 Classifier Framework

Within the context of this work, the classification environment is formulated as follows: The learners are defined as GP trees, and the points as indices into the training data. The interaction between them utilizes the classical GP approach of an absolute switching function (wrapper) centered at $gpOut = 0.0$, see Equation 3.1.

$$\text{If } gpOut \leq 0.0 \text{ then return class 0, else return class 1.} \tag{3.1}$$

Where *gpOut* is the numerical output from the individual under evaluation given a data point as input.

3.2 Visualization Ground Work

Throughout the description of the PGPC algorithm, several figures illustrate the behaviour and scenarios justifying the steps of the algorithm. The following section provides the basis for these illustrations, and the steps undertaken to map a high dimensional (binary) pareto-front of points versus learners into one intuitively grasped in three dimensions as utilized in the algorithmic description examples.

In the proposed classification context, each training point yields a binary value (correctly or incorrectly classified). Grouped together, these training points yield a set of axis on which a learner's position corresponds to its classification performance over the point subset. A set of pareto-equivalent learners within this set of axis constitutes a pareto-front, with the maximal position being one which correctly classifies all training points. See Figure 3.1.

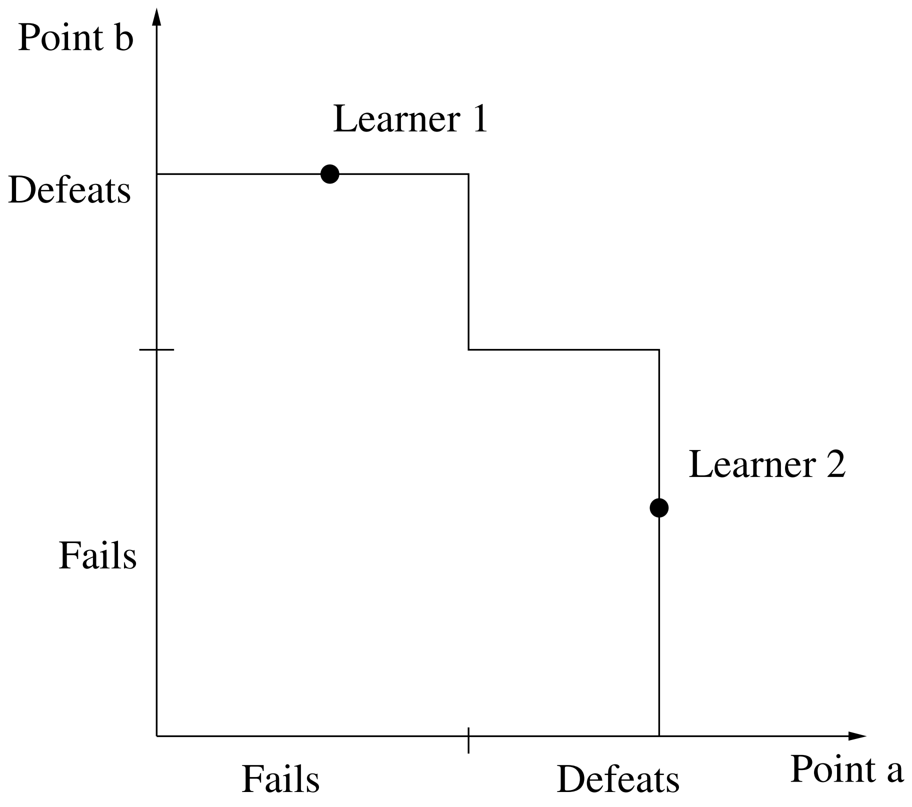


Figure 3.1: A two point, two learner binary pareto front example.

In order to attain high classification performance, it is desired that the hyper-hull of the pareto-front expands to the maximum size (perfect classification on any subset of point dimensions). However, to attain efficiency through training point subset selection, the number of dimensions utilized in the description of the pareto-front should be minimized, while still providing an adequate gradient for learner selection and evolution. An example consisting of two required and one redundant dimension is given in Figure 3.3, where the redundant dimension may be removed, providing an efficiency increase, at no loss to learner discrimination.

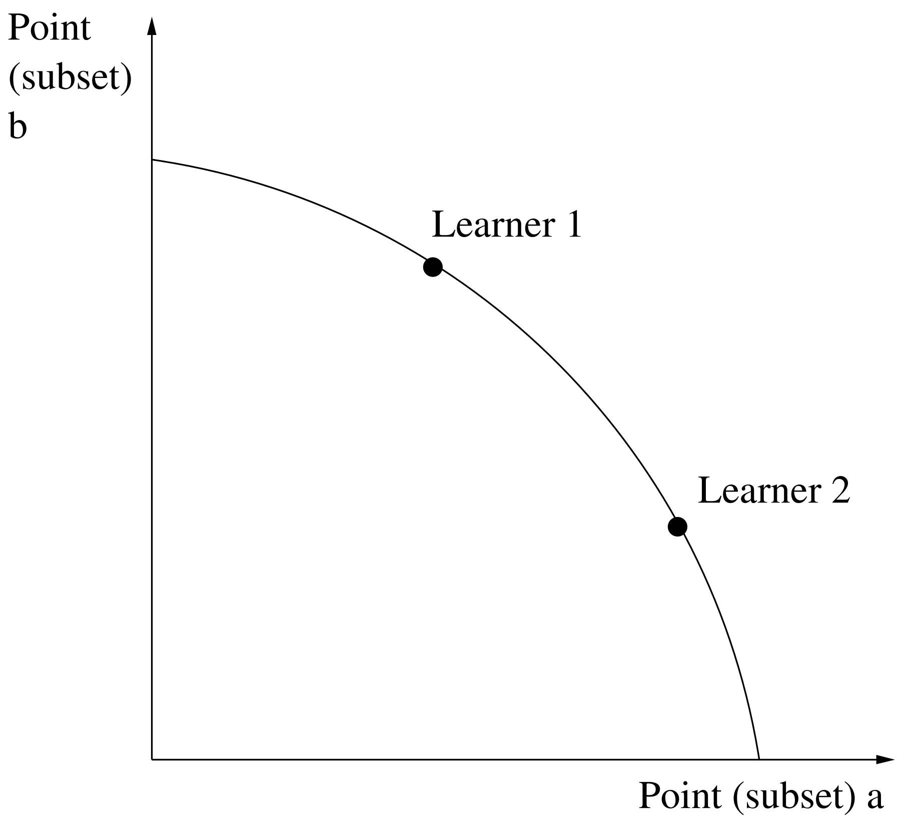


Figure 3.2: A two point (subset), two learner continuous pareto front example.

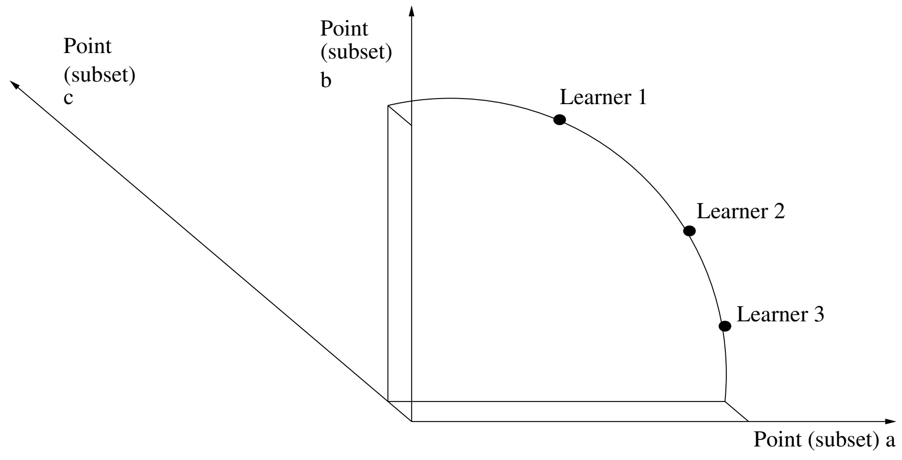


Figure 3.3: A three point (subset), continuous pareto front example, with one dimension being redundant (Point subset c).

3.3 Pareto-coevolutionary GP Classifier Algorithm

Firstly, the PGPC algorithm is based on the previously described IPCA algorithm, and as such, additional details on common sections may be referenced in Section 2.2.2, and in [2].

The following pseudo-code describing the PGPC algorithm will assume the existence of the functions defined in Algorithms 1, 2, and 3.

Algorithm 1 random(*start*, *end*)

```
return a uniformly selected random number in the range of start to end
```

Algorithm 2 outcome(*Learner*, *Point*)

```
{Return the results of the interaction between Learner and Point}
if Learner correctly classifies Point then
  return 1
else
  return 0
end if
```

Algorithm 3 computeArchiveOutcomes(*LearnerArchive*,*PointArchive*)

```
{Compute the outcomes of the learner archive against the point archive}
for $i = 0$ to $|LearnerArchive| - 1$ do
  for $j = 0$ to $|PointArchive| - 1$ do
    $archiveOutcomes[i][j] \Leftarrow outcome(LearnerArchive[i], PointArchive[j])$
  end for
end for
return archiveOutcomes
```

The PGPC algorithm utilizes four populations of individuals: (1) a fixed size learner population which supports the exploratory aspect of learner evolution. (2) a learner archive which contains the pareto-front of learners, bound by a maximum size value. (3) a fixed size point population (point population << training exemplar count). (4)

a point archive which describes the current subset of training points, relevant to the learner archive, bound by a maximum size value. Figure 3.4 summarizes the organization of data dependencies for each step of the algorithm.

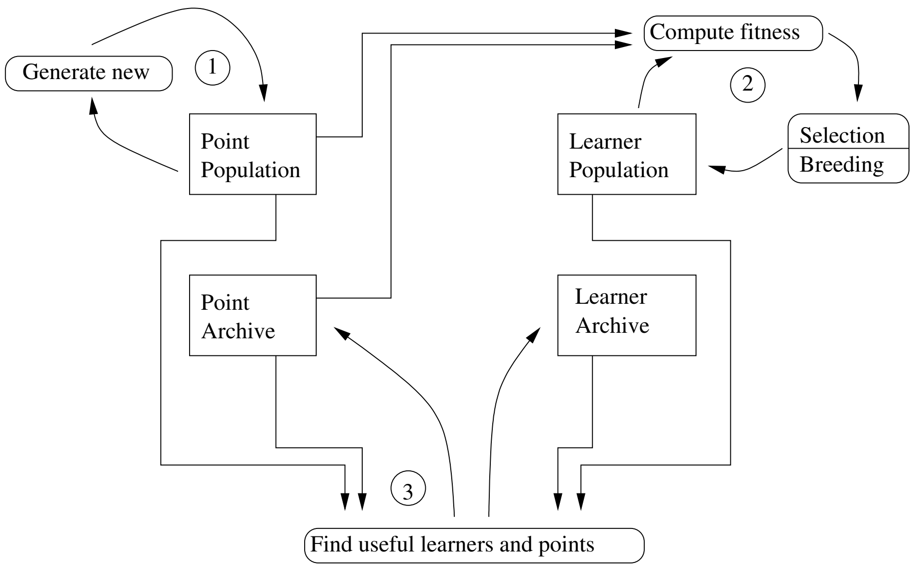


Figure 3.4: The PGPC algorithm.

The PGPC algorithm consists of the following steps:

1. **Generate points in the point population:** Since the points are indices within the training data, the original crossover operator utilized in the IPCA algorithm makes little sense. How would recombination of two indices into a list, which may be randomized, perform exploitation? Therefore, only mutation was utilized to generate the point population, with mutation being performed on each population member. With mutation taking the form of replacement with an alternate integer (index into the training data) within the valid range, selected with uniform probability.

   One parameter to consider is the class balance of the resultant population; since the input data set may be unbalanced, class-aware selection may be considered worthwhile [10] [27] (See Section 2.1.2). The implementation of the algorithm utilized two approaches to point generation; one that performs purely random (uniform) generation oblivious to class balance, denoted as “G-0”, and the other which randomly fills each half of the population with points belonging to the same class, denoted as “G-1”. See Algorithm 4.

Algorithm 4 Generate a new point population

```
if G-0 then
  {Uniform random generation}
  for $i = 0$ to $|PointPopulation| - 1$ do
    $PointPopulation[i] \Leftarrow random(0, |TrainingData|)$
  end for

else if G-1 then
  {Class balanced random generation}
  for $i = 0$ to $\frac{|PointPopulation|-1}{2}$ do
    repeat
      $PointPopulation[i] \Leftarrow random(0, |TrainingData|)$
    until $PointPopulation[i]_{class}$ equals 0
  end for
  for $i = \frac{|PointPopulation|-1}{2}$ to $|PointPopulation| - 1$ do
    repeat
      $PointPopulation[i] \Leftarrow random(0, |TrainingData|)$
    until $PointPopulation[i]_{class}$ equals 1
  end for
end if
```

2. **Generate learners in the learner population:** The canonical GP model of Koza is assumed [16], although the PGPC model could utilize any standard GP model for representation or definition of search and selection operators. Using fitness proportionate selection (with fitness being calculated on the point population and archive). Mutation and crossover operators are used to generate offspring. See Algorithm 5.

3. The following steps deal with the entry criteria for the point and learner archives:

(a) **Compute the set of useful points regarding the learner population and archive:** As per IPCA; if a newly generated learner is dominated by the learner archive or contains equal values, and the addition of a new point provides a distinction such that the generated learner is pareto-equivalent to the archive with no equal values, the point is inserted into the archive. This allows for the growth of the hyper-hull of the pareto-front via the addition of a dimension which provides a distinction between pareto-front learners as in Figure 3.5. To maintain the upper bound on the size of the point archive, pruning may have to be performed during the insertion of useful points. See Algorithms 6, and 7.

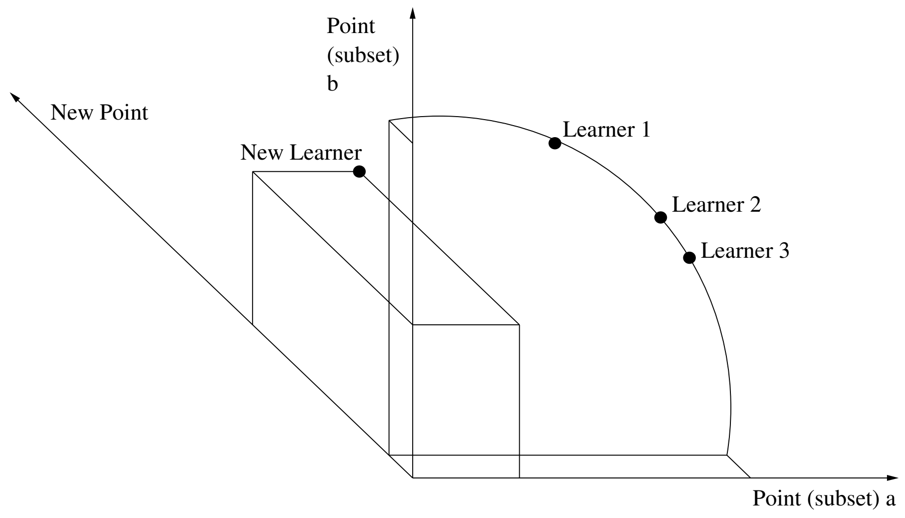


Figure 3.5: Addition of a new axis (New Point), makes a previously dominated learner (New Learner) join the pareto front of learners.

(b) **Compute the set of useful learners regarding the point population and archive:** As per IPCA; any generated learner that is pareto-equivalent to the archive with no equal values enters the archive (See Figure 3.6). Furthermore, if a generated learner is undefeated, and a generated point defeats it, they both enter their respective archives, as to provide the next step of the learning gradient for the learners to evolve towards (See Figure 3.7). See Algorithms 8, and 9.

Again, to maintain the upper bounds on the point and learner archives, pruning may have to be performed during the insertions.

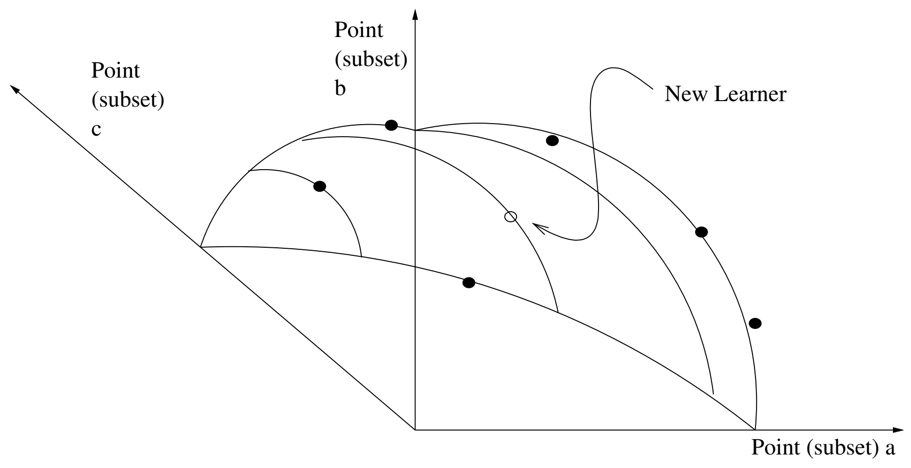


Figure 3.6: A useful (pareto-equivalent) learner (New Learner) is added to the pareto front.

4. **Remove duplicates in the learner and point archives and newly-dominated learners in the learner archive:** As per IPCA.

5. **Loop to generate learners.**

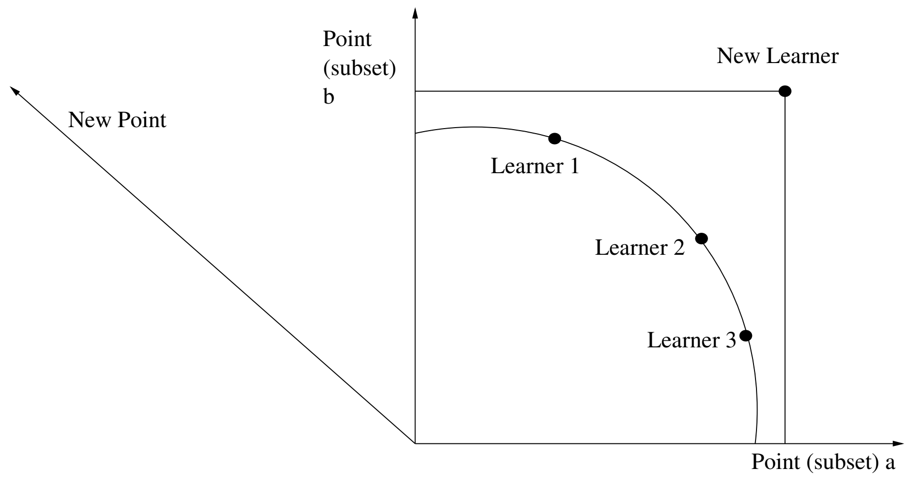


Figure 3.7: A generated axis (New Point), defeats an undefeated learner (New Learner) and is added as a next “step” for subsequent learners to evolve towards.

Algorithm 5 Generate a new learner population

```
for $i = 0$ to $|LearnerPopulation| - 1$ do
    $LearnerPopulation[i]_{fitness} \Leftarrow 0.0$

    {Evaluate learner fitness over the point population}
    for $j = 0$ to $|PointPopulation| - 1$ do
        $LearnerPopulation[i]_{fitness} \Leftarrow LearnerPopulation[i]_{fitness} + outcome(LearnerPopulation[i], PointPopulation[j])$
    end for

    {Evaluate learner fitness over the point archive}
    for $j = 0$ to $|PointArchive| - 1$ do
        $LearnerPopulation[i]_{fitness} \Leftarrow LearnerPopulation[i]_{fitness} + outcome(LearnerPopulation[i], PointArchive[j])$
    end for

    {Normalize the fitness value}
    $LearnerPopulation[i]_{fitness} \Leftarrow \frac{LearnerPopulation[i]_{fitness}}{|PointPopulation|+|PointArchive|}$
end for

{Perform GP operations}
$Parents \Leftarrow GPselection(LearnerPopulation)$ {Fitness proportionate selection}
$Offspring \Leftarrow GPbreeding(Parents)$ {Application of mutation and crossover operators}
$LearnerPopulation[Parents] \Leftarrow Offspring$ {Replacement of parents}
```

Algorithm 6 Identify the set of useful points

```
{Compute the archive outcomes}
$archiveOutcomes \Leftarrow computeArchiveOutcomes(LearnerArchive, PointArchive)$

{Over all generated learners}
for $i = 0$ to $|LearnerPopulation| - 1$ do
  {Compute outcomes against the Point archive}
  for $j = 0$ to $|PointArchive| - 1$ do
    $outcomes[j] \Leftarrow outcome(LearnerPopulation[i], PointArchive[j])$
  end for

  {Identify non-useful learners}
  $usefulLearner \Leftarrow true$
  for $j = 0$ to $|LearnerArchive| - 1$ do
    if ($archiveOutcomes[j]$ dominates $outcomes$) OR ($archiveOutcomes[j]$ equals
    $outcomes$) then
      $usefulLearner \Leftarrow false$
      $j \Leftarrow |LearnerArchive|$ {Break}
    end if
  end for

  {For non-useful learners}
  if $usefulLearner$ equals $false$ then
    {See if a generated point provides a distinction}
    for $j = 0$ to $|PointPopulation| - 1$ do
      evaluateCandidatePoint($archiveOutcomes, outcomes,$
      $LearnerPopulation[i], PointPopulation[j], LearnerArchive, PointArchive$)
    end for
  end if
end for
```

Algorithm 7 evaluateCandidatePoint(*archiveOutcomes*, *outcomes*, *Learner*, *Point*, *LearnerArchive*, *PointArchive*)

```
{Compute non-useful learner outcomes against the candidate point}
$outcomes[|PointArchive|] \Leftarrow outcome(Learner, Point)$

{Compute learner archive outcomes against the candidate point}
for $k = 0$ to $|LearnerArchive| - 1$ do
    $archiveOutcomes[k][|PointArchive|] \Leftarrow outcome(LearnerArchive[k],$
    $PointPopulation[j])$
end for

{Test for usefulness of the non-useful learner}
$usefulLearner \Leftarrow true$
for $j = 0$ to $|LearnerArchive|$ do
    if ($archiveOutcomes[j]$ dominates $outcomes$) OR
    ($archiveOutcomes[j]$ equals $outcomes$) then
        $usefulLearner \Leftarrow false$
        $j \Leftarrow |LearnerArchive|$ {Break}
    end if
end for

{Insert the candidate point into the point archive if the non-useful learner is now
useful}
if $usefulLearner$ equals $true$ then
    $PointArchive \Leftarrow insertIntoPointArchive(PointArchive, Point,$
    $LearnerArchive)$
end if
```

Algorithm 8 Identify the set of useful learners

```
{Compute the archive outcomes}
$archiveOutcomes \Leftarrow computeArchiveOutcomes(LearnerArchive, PointArchive)$

{Over all generated learners}
for $i = 0$ to $|LearnerPopulation| - 1$ do
  {Compute outcomes against the Point archive}
  for $j = 0$ to $|PointArchive| - 1$ do
    $outcomes[j] \Leftarrow outcome(LearnerPopulation[i], PointArchive[j])$
  end for

  {Identify useful learners}
  $usefulLearner \Leftarrow true$
  for $j = 0$ to $|LearnerArchive| - 1$ do
    if ($archiveOutcomes[j]$ dominates $outcomes$) OR ($archiveOutcomes[j]$ equals
    $outcomes$) then
      $usefulLearner \Leftarrow false$
      $j \Leftarrow |LearnerArchive|$ {Break}
    end if
  end for

  {Insert the useful learners into the learner archive}
  if $usefulLearner$ equals $true$ then
    $LearnerArchive \Leftarrow insertIntoLearnerArchive(LearnerArchive,$
    $LearnerPopulation[i], PointArchive)$
  end if

  {Test for the undefeated learner special case}
  testForUndefeatedCase($outcomes, LearnerPopulation[i], PointPopulation,$
  $PointArchive, LearnerArchive$)
end for
```

Algorithm 9 testForUndefeatedCase(*outcomes*, *Learner*, *PointPopulation*, *PointArchive*, *LearnerArchive*)

```
{Identify undefeated learners}
if $\sum_{k=0}^{|PointArchive|-1} outcomes[k]$ equals $|PointArchive|$ then
  {Identify generated points which defeat the undefeated learner}
  for $j = 0$ to $|PointPopulation| - 1$ do
    if $outcome(Learner, PointPopulation[j])$ equals 0 then
      {Insert the identified points into the point archive}
      $PointArchive \Leftarrow insertIntoPointArchive(PointArchive,$
      $PointPopulation[j], LearnerArchive)$
    end if
  end for
end if
```

3.3.1 Archive Pruning

To maintain the efficiency of the algorithm, a limit on the archive sizes may have to be placed. This limit may be thought of as a tunable parameter of efficiency vs accuracy, however the relationship between the two may depend on the input data set, as the number of underlying objectives which the point archive strives to evolve towards may vary.

Within the context of the pruning algorithm, the first (arbitrary) assertion will be that a newly generated learner or point should enter the archive at the possible cost of evicting an older member. This assertion will help avoid stagnation within the archive at the risk of forgetting. Alternate insertion basis may be considered and evaluated in the future. Within this framework, all that remains is to provide a basis for selection of an archive individual for removal.

The following pseudo-code describing the archive pruning operations will assume the existence of the functions defined in Algorithms 10, 11, and 12.

Algorithm 10 hammingDistance(A,B)

```
  return the Hamming distance between the two binary arrays $A$, and $B$.
```

Algorithm 11 euclideanDistance($PointA$,$PointB$)

```
  return the Euclidean distance between the two points $PointA$, and $PointB$.
```

Algorithm 12 computeDistinctions($PointArchive$,$LearnerArchive$)

```
{Compute the archive outcomes}
$archiveOutcomes \Leftarrow computeArchiveOutcomes(LearnerArchive, PointArchive)$

{Convert the scores into distinctions}
for $i = 0$ to $|PointArchive| - 1$ do
  $index \Leftarrow 0$
  for $j = 0$ to $|LearnerArchive| - 1$ do
    for $k = 0$ to $|LearnerArchive| - 1$ do
      if $archiveOutcomes[i][j] > archiveOutcomes[i][k]$ then
        $distinctions[i][index] \Leftarrow 1$
      else
        $distinctions[i][index] \Leftarrow 0$
      end if
      $index \Leftarrow index + 1$
    end for
  end for
end for
return $distinctions$
```

Learner Archive Pruning

In providing a basis for learner removal, two approaches were implemented with regards to each learner's performance against the point archive, they consist of either enhancing diversity or greedy replacement.

The diversity enhancing approach (denoted as "L0"), randomly removes one of the two "closest" individuals. With distance being measured phenotypically using the Hamming distance between the outcomes against the point archives (correct or incorrect classification for each archive point). This approach removes some of the "redundancy" in the archive (learners which may differ by their outcomes on only a few points), however it may inadvertently remove (one of two) high scoring individuals.

The greedy approach (denoted as "L1"), consists of removing the learner with the worst performance against the point archive (with the measure being the number of incorrectly classified instances). Within this view, a learner to be removed may have entered the archive by simply correctly classifying one training archive point while misclassifying the remainder, therefore removing the "worst" learner deletes some of the explorative diversity of the archive in favour of increased average accuracy. This basis was also utilized in [20] (see Section 2.2.2) to maintain their learner archive size.

For details, reference Algorithms 13, 14, and 15.

Algorithm 13 insertIntoLearnerArchive($LearnerArchive$, $Learner$, $PointArchive$)

```
if $|LearnerArchive| > MaxLearnerArchiveSize$ then
  {Remove an archive member}
  if L-0 then
    {Diversity enhancing}
    $LearnerArchive \Leftarrow l0pruning(LearnerArchive, PointArchive)$

  else if L-1 then
    {Greedy}
    $LearnerArchive \Leftarrow l1pruning(LearnerArchive, PointArchive)$
  end if
end if

{Insert the new Learner}
$LearnerArchive \Leftarrow LearnerArchive \cup Learner$
```

Algorithm 14 l0pruning($LearnerArchive$, $PointArchive$)

```
{Compute the archive outcomes}
$archiveOutcomes \Leftarrow computeArchiveOutcomes(LearnerArchive, PointArchive)$

{Find the two closest individuals}
$closestDistance \Leftarrow \infty$
**for** $i = 0$ to $|LearnerArchive| - 1$ **do**
  **for** $j = 0$ to $i - 1$ **do**
    {Compute Hamming distance between the outcomes}
    $distance \Leftarrow hammingDistance(archiveOutcomes[i], archiveOutcomes[j])$
    **if** $distance < closestDist$ **then**
      $closestDistance \Leftarrow distance$
      $closestA \Leftarrow i$
      $closestB \Leftarrow j$
    **end if**
  **end for**
**end for**

{Randomly remove one of the two closest individuals}
**if** $random(0, 1) < 0.5$ **then**
  $LearnerArchive \Leftarrow LearnerArchive \setminus LearnerArchive[closestA]$
**else**
  $LearnerArchive \Leftarrow LearnerArchive \setminus LearnerArchive[closestB]$
**end if**
```

Algorithm 15 l1pruning($LearnerArchive$, $PointArchive$)

```
{Compute the archive outcomes}
$archiveOutcomes \Leftarrow computeArchiveOutcomes(LearnerArchive, PointArchive)$

{Find the worst performing individual}
$worstPerformance \Leftarrow \infty$
**for** $i = 0$ to $|LearnerArchive| - 1$ **do**
  $performance \Leftarrow 0$
  **for** $k = 0$ to $|PointArchive| - 1$ **do**
    $performance \Leftarrow performance + archiveOutcomes[i][k]$
  **end for**
  **if** $performance < worstPerformance$ **then**
    $worstPerformance \Leftarrow performance$
    $worstLearner \Leftarrow i$
  **end if**
**end for**

{Remove the worst performing individual}
$LearnerArchive \Leftarrow LearnerArchive \setminus LearnerArchive[worstLearner]$
```

Point Archive Pruning

To provide a mechanism for archive point removal, several combinations of greedy, diversity enhancing, phenotypic, and genotypic basis were considered with and without class considerations.

Methods utilized include:

- “P0”: Delete one of the two closest points in the archive. With closest being defined as the Euclidean distance based on data point coordinates (diversity enhancing, genotypic), discounting the class information.
- “P1”: Delete one of the two closest points in the archive, based on the number of distinctions made by the points against the learner archive (diversity enhancing, phenotypic).
- “P2”: Delete the worst performing point in the archive based on the number of distinctions made (greedy, phenotypic).
- “P3”: Delete one of the two closest points in the archive having the same class. With closest being the Euclidean distance of the point co-ordinates (diversity enhancing, genotypic, class conscious).
- “P4”: Delete one of the two closest points (Euclidean distance) in the archive having the same class, with the class being the over-represented one in the archive (diversity enhancing, genotypic, class conscious and balance enforcing).

Reference Algorithms 16, 17, 18, 19, 20, and 21.

Algorithm 16 insertIntoPointArchive($PointArchive$, $Point$,$LearnerArchive$)

```
if $|PointArchive| > MaxPointArchiveSize$ then
  {Remove an archive member}
  if P-0 then
    {Diversity enhancing, genotypic}
    $PointArchive \Leftarrow p0pruning(PointArchive, LearnerArchive)$

  else if P-1 then
    {Diversity enhancing, phenotypic}
    $PointArchive \Leftarrow p1pruning(PointArchive, LearnerArchive)$

  else if P-2 then
    {Greedy, phenotypic}
    $PointArchive \Leftarrow p2pruning(PointArchive, LearnerArchive)$

  else if P-3 then
    {Diversity enhancing, genotypic, class conscious}
    $PointArchive \Leftarrow p3pruning(PointArchive, LearnerArchive)$

  else if P-4 then
    {Diversity enhancing, phenotypic, class conscious, balance enforcing}
    $PointArchive \Leftarrow p4pruning(PointArchive, LearnerArchive)$
  end if
end if

{Insert the new Point}
$PointArchive \Leftarrow PointArchive \cup Point$
```

Algorithm 17 p0pruning(*PointArchive*, *LearnerArchive*)

```
{Find the two closest points based on co-ordinates}
$closestDistance \Leftarrow \infty$
for $i = 0$ to $|PointArchive| - 1$ do
  for $j = 0$ to $i - 1$ do
    {Compute Euclidean distance between the points}
    $distance \Leftarrow euclideanDistance(PointArchive[i], PointArchive[j])$
    if $distance < closestDistance$ then
      $closestDistance \Leftarrow distance$
      $closestA \Leftarrow i$
      $closestB \Leftarrow j$
    end if
  end for
end for

{Randomly remove one of the two closest individuals}
if $random(0, 1) < 0.5$ then
  $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestA]$
else
  $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestB]$
end if
```

Algorithm 18 p1pruning(*PointArchive*, *LearnerArchive*)

```
{Find the two closest points based on distinctions against the Learner archive}
$closestDistance \Leftarrow \infty$

{Compute the distinctions made by the point archive}
$distinctions \Leftarrow computeDistinctions(PointArchive, LearnerArchive)$

{Find the two closest points}
for $i = 0$ to $|PointArchive| - 1$ do
  for $j = 0$ to $i - 1$ do
    {Compute Hamming distance between the distinctions}
    $distance \Leftarrow hammingDistance(distinctions[i], distinctions[j])$
    if $distance < closestDistance$ then
      $closestDistance \Leftarrow distance$
      $closestA \Leftarrow i$
      $closestB \Leftarrow j$
    end if
  end for
end for

{Randomly remove one of the two closest individuals}
if $random(0, 1) < 0.5$ then
  $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestA]$
else
  $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestB]$
end if
```

Algorithm 19 p2pruning(*PointArchive*, *LearnerArchive*)

```
{Remove the worst performing point based on distinctions against the Learner archive}

{Compute the distinctions made by the point archive}
$distinctions \Leftarrow computeDistinctions(PointArchive, LearnerArchive)$

{Find the point with the lowest number of distinctions}
$worstNumberDistinctions \Leftarrow \infty$
for $i = 0$ to $|PointArchive| - 1$ do
  $numDistinctions \Leftarrow 0$
  for $k = 0$ to $|LearnerArchive|^2 - 1$ do
    if $distinctions[i][k]$ equals 1 then
      $numDistinctions \Leftarrow numDistinctions + 1$
    end if
  end for
  if $numDistinctions < worstNumberDistinctions$ then
    $worstNumberDistinctions \Leftarrow numDistinctions$
    $worstPoint \Leftarrow i$
  end if
end for

{Remove the worst point}
$PointArchive \Leftarrow PointArchive \setminus PointArchive[worstPoint]$
```

Algorithm 20 p3pruning($PointArchive$, $LearnerArchive$)

```
{Find the two closest points based on co-ordinates, and class}
$closestDistance \Leftarrow \infty$
**for** $i = 0$ to $|PointArchive| - 1$ **do**
  **for** $j = 0$ to $i - 1$ **do**
    {Compute Euclidean distance between the points}
    $distance \Leftarrow euclideanDistance(PointArchive[i], PointArchive[j])$
    **if** $distance < closestDistance$ **then**
      **if** $PointArchive[i]_{class}$ equals $PointArchive[j]_{class}$ **then**
        $closestDistance \Leftarrow distance$
        $closestA \Leftarrow i$
        $closestB \Leftarrow j$
      **end if**
    **end if**
  **end for**
**end for**

{Randomly remove one of the two closest individuals}
**if** $random(0, 1) < 0.5$ **then**
  $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestA]$
**else**
  $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestB]$
**end if**
```

Algorithm 21 p4pruning(*PointArchive*, *LearnerArchive*)

```
{Find the two closest points based on co-ordinates, class, and class balance}
$closestDistance \Leftarrow \infty$
for $i = 0$ to $|PointArchive| - 1$ do
   for $j = 0$ to $i - 1$ do
      {Compute Euclidean distance between the points}
      $distance \Leftarrow euclideanDistance(PointArchive[i], PointArchive[j])$
      if $distance < closestDistance$ then
         if $PointArchive[i]_{class}$ equals $PointArchive[j]_{class}$ then
            if $PointArchive[i]_{class}$ is the minority in $PointArchive$ then
               $closestDistance \Leftarrow distance$
               $closestA \Leftarrow i$
               $closestB \Leftarrow j$
            end if
         end if
      end if
   end for
end for

{Randomly remove one of the two closest individuals}
if $random(0, 1) < 0.5$ then
   $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestA]$
else
   $PointArchive \Leftarrow PointArchive \setminus PointArchive[closestB]$
end if
```

Archive Pruning Dangers

The effect of removing a section of a pareto-front stored in an archive for increased efficiency is interpreted as removing some set of equally performing individuals (in the pareto view of equal), with the impact to performance being dictated by the contributions of the removed individuals. However, the act of removing previously found points, where such points provided distinctions between learners at some previous point in time, seems innocuous enough; given that the context of the current learner front is redundant as all learners defeat them. Conversely, it very well may be the case that the pruned point describes an underlying objective of the problem, and constitutes a part of the minimal set of dimensions upon which the pareto-front of learners resides upon. Figure 3.8 illustrates a scenario where the loss of a point no longer providing distinctions between the set of learners results in the loss of progress along that underlying learner objective as selection pressure is removed.

The pruning of archives, specifically the point archive, constitutes a danger since it allows for forgetting to occur along an underlying objective, thus breaking the guarantee of monotonic progress associated with the non-pruned IPCA algorithm. The frequency of instances of regression exhibited by the PGPC algorithm over various real-world data sets will be measured and evaluated to ascertain the practical costs of archive pruning in Section 5.3.

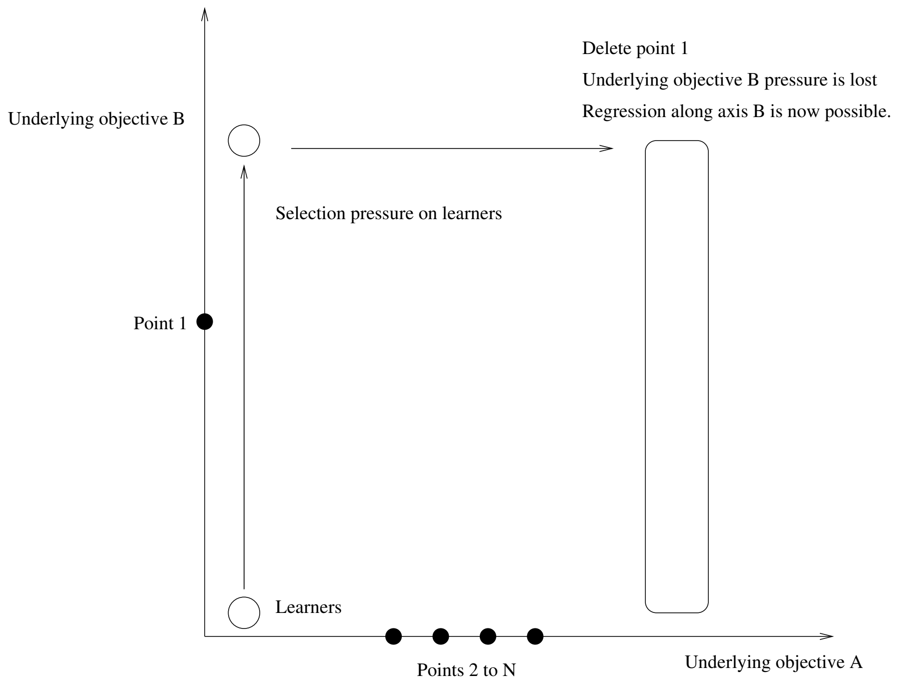

Delete point 1
Underlying objective B pressure is lost
Regression along axis B is now possible.
Underlying objective B
Selection pressure on learners
Point 1
Learners
Points 2 to N
Underlying objective A

Figure 3.8: The deletion of an "old" point archive member (age is defined by cardinality) may mean the loss of an underlying objective and gradient pressure.

3.3.2 Learner Pareto-front Evaluation: Post-processing

In order to attain a measure of classification performance on testing data at the conclusion of training, the learner pareto-front must be interpreted or mapped to provide one class prediction per testing point. To perform this mapping several selection/voting methods with varying complexities were considered, they include:

- Average archive value (voting scheme) (“AA”):
  A simple method of allotting each pareto-front learner one vote for its class prediction. The class with the majority of the votes is selected as the systems' prediction for the data point. This method only requires the evaluation of each learner in the learner archive per testing point. See Figure 3.9, and Algorithm 22.

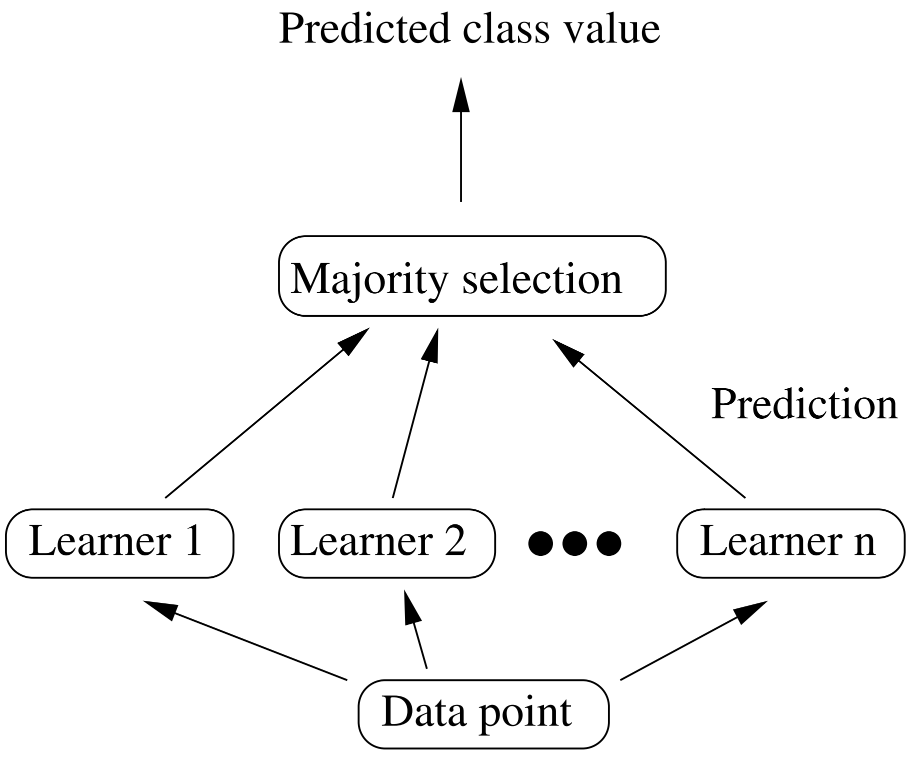


Figure 3.9: Average archive value (AA) post-processing method.

Algorithm 22 AA post processing of $LearnerArchive$ on an input $TestPoint$.

```
$voteCount \Leftarrow 0$
for $i = 0$ to $|LearnerArchive| - 1$ do
    $voteCount \Leftarrow voteCount + LearnerArchive[i]_{prediction}(TestPoint)$
end for
$voteCount \Leftarrow \frac{voteCount}{|LearnerArchive|}$
if $voteCount < 0.5$ then
    return $class0$
else
    return $class1$
end if
```

- Test to training point Clustering (nearest neighbour based) ("TC"):

  This method involves the genotypic mapping of each input testing point to its nearest training point. All the learners in the learner archive are evaluated on the training point, and the subset that correctly classifies the point is used to classify the test point using an average voting scheme. In the case of no learners correctly classifying the training point, the subset consists of all of the archive learners. This method may utilize a precomputed mapping of nearest neighbours from the test to training data to aid in execution speed. See Figure 3.10, and Algorithm 23.

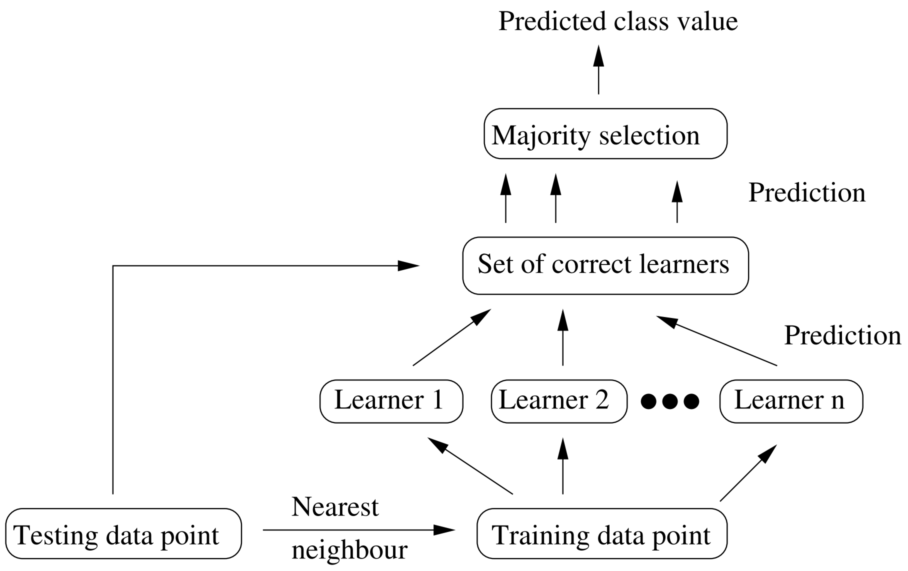


Figure 3.10: Test to training point Clustering (TC) post-processing method.

Algorithm 23 TC post processing of $LearnerArchive$ on an input $TestPoint$.

```
Require: $TestPoint$ to nearest $TrainingPoint \in LearnerArchive$ mapping
  {Find the set of learners that correctly classify $TrainingPoint$}
  $learnerSet = \emptyset$
  for $i = 0$ to $|LearnerArchive| - 1$ do
    if $outcome(LearnerArchive[i], TrainingPoint)$ equals 1 then
      $learnerSet = learnerSet \cup LearnerArchive[i]$
    end if
  end for
  if $|learnerSet|$ equals $\emptyset$ then
    $learnerSet = learnerArchive$
  end if

  {Use the $learnerSet$ to classify $TestPoint$}
  $voteCount \Leftarrow 0$
  for $i = 0$ to $|learnerSet| - 1$ do
    $voteCount \Leftarrow voteCount + learnerSet[i]_{prediction}(TestPoint)$
  end for
  $voteCount \Leftarrow \frac{voteCount}{|learnerSet|}$
  if $voteCount < 0.5$ then
    return $class0$
  else
    return $class1$
  end if
```

- Gaussians of the *gpOut* values for the two classes (GPout based)("G2"):
  Using the training data, two gaussians describing the Local Membership Function (LMF)(one for each class) of the *gpOut* values are computed for each learner in the archive. Then given the *gpOut* value of each learner on the input testing point, the membership value on each gaussian is computed, reference Figure 3.11. Using the formula:

  $$membership = e^{\frac{-1(gpOut-mean)^2}{2(stdDev)^2}}$$

  Test point classification is determined by the higher membership value, and confidence in the classification is measured as the difference between the two membership values. The classification of the most confident learner in the learner archive is chosen as representative of the system. See Figure 3.12, and Algorithms 24, 25 .

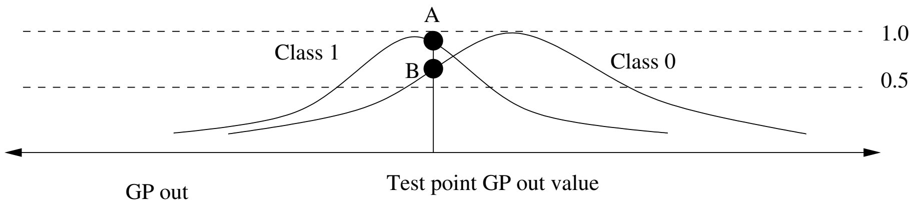


Figure 3.11: Membership of a *gpOut* value of a point on the two LMFs of a learner.

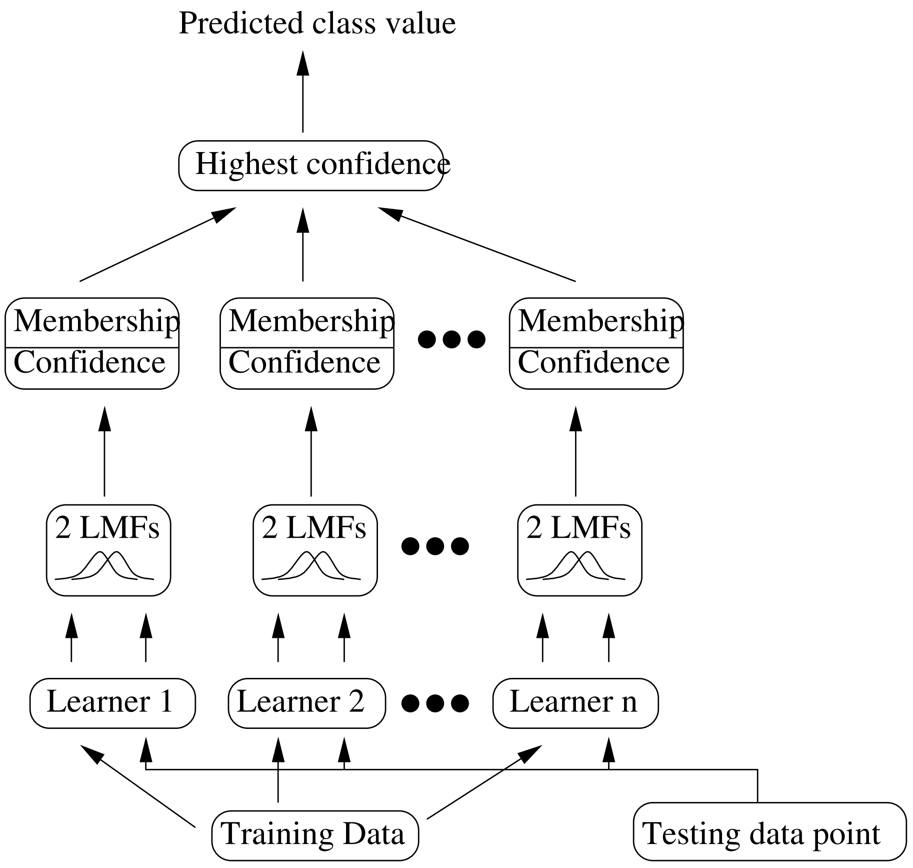


Figure 3.12: Gaussians of the *gpOut* values for the two classes (G2) post-processing method.

Algorithm 24 G2 post processing of *LearnerArchive*. Initial step.

```
{Compute the mean and standard deviation of the gaussians for each class for each learner}
for $i = 0$ to $|LearnerArchive| - 1$ do
    $trainingDataNum0 \Leftarrow 0$
    $trainingDataNum1 \Leftarrow 0$
    $mean0[i] \Leftarrow 0.0$
    $mean1[i] \Leftarrow 0.0$
    for $j = 0$ to $|Data_{training}| - 1$ do
        if $Data_{training}[j]_{class}$ equals 0 then
            $mean0[i] \Leftarrow mean0[i] + LearnerArchive[i]_{gpOut}(Data_{training}[j])$
            $trainingDataNum0 \Leftarrow trainingDataNum0 + 1$
        else
            $mean1[i] \Leftarrow mean1[i] + LearnerArchive[i]_{gpOut}(Data_{training}[j])$
            $trainingDataNum1 \Leftarrow trainingDataNum1 + 1$
        end if
    end for
    $mean0[i] \Leftarrow \frac{mean0[i]}{trainingDataNum0}$
    $mean1[i] \Leftarrow \frac{mean1[i]}{trainingDataNum1}$

    $stdDev0[i] \Leftarrow 0.0$
    $stdDev1[i] \Leftarrow 0.0$
    for $j = 0$ to $|Data_{training}| - 1$ do
        if $Data_{training}[j]_{class}$ equals 0 then
            $stdDev0[i] \Leftarrow stdDev0[i] + \frac{(LearnerArchive[i]_{gpOut}(Data_{training}[j]) - mean0[i])^2}{trainingDataNum0}$
        else
            $stdDev1[i] \Leftarrow stdDev1[i] + \frac{(LearnerArchive[i]_{gpOut}(Data_{training}[j]) - mean1[i])^2}{trainingDataNum1}$
        end if
    end for
    $stdDev0[i] \Leftarrow \sqrt{stdDev0[i]}$
    $stdDev1[i] \Leftarrow \sqrt{stdDev1[i]}$
end for
```

Algorithm 25 G2 post processing of $LearnerArchive$ on an input $TestPoint$. Per testing point step

```
Require: Completion of G2 initial step; valid values of $mean0$, $mean1$, $stdDev0$, and $stdDev1$.
  {Plot the gpOut value on each learner's gaussian}
  for $i = 0$ to $|LearnerArchive| - 1$ do
    $gpOut \Leftarrow LearnerArchive[i]_{gpOut}(TestPoint)$
    $membership0 \Leftarrow e^{\frac{-1(gpOut - mean0[i])^2}{2(stdDev0[i])^2}}$
    $membership1 \Leftarrow e^{\frac{-1(gpOut - mean1[i])^2}{2(stdDev1[i])^2}}$
  end for

  {Find the learner with the highest classification confidence}
  $highestConfidence \Leftarrow -1$
  for $i = 0$ to $|LearnerArchive| - 1$ do
    if $|membership0 - membership1| > highestConfidence$ then
      $highestConfidence \Leftarrow |membership0 - membership1|$
      if $membership0 > membership1$ then
        $highestConfidenceClass \Leftarrow 0$
      else
        $highestConfidenceClass \Leftarrow 1$
      end if
    end if
  end for
  return $highestConfidenceClass$
```

3.4 Computational Complexity

If the size of the training data is denoted as $Data_{training}$, the size of the testing as $Data_{testing}$, and the number of generations as gen, with the addition of the learner and point populations and archives being respectively denoted as L_{pop}, P_{pop}, L_{arch}, and P_{arch}, the following describes the computational and storage complexities of the PGPC algorithm with respect to the utilized archive pruning parameters. For details concerning derivation, reference Appendix A.1:

Time complexity:	
Learner pruning basis:	
If L = 0	$L_{basis} = O(L_{arch})$
If L = 1	$L_{basis} = O(1)$
Point pruning basis:	
If P = 0,3,4	$P_{basis} = O(P_{arch})$
If P = 1	$P_{basis} = O(L_{arch}^2 + P_{arch})$
If P = 2	$P_{basis} = O(L_{arch}^2)$
Time complexity =	$O(gen*$ $((L_{arch}^2 * P_{arch})+$ $(L_{pop} * P_{pop} * L_{arch} * P_{arch})+$ $(L_{pop} * P_{pop} * P_{arch} * P_{basis})+$ $(L_{pop} * P_{arch} * L_{arch} * L_{basis})+$ $P_{arch}^2))$

Storage complexity:	
Point pruning basis:	
If P = 0,3,4	$P_{basis} = O(1)$
If P = 1,2	$P_{basis} = O(L_{arch}^2 + P_{arch})$
Storage complexity =	$O(Data_{training} + P_{pop} + L_{pop}+$ $(L_{arch} * P_{arch}) + P_{basis})$

If a simplification is allowed, and the population and archive sizes of both of the learners and points are set to a common value (denoted $subset$), then the complexities collapse to:

Time complexity:	
Learner pruning basis:	
If L = 0	$L_{basis} = O(subset)$
If L = 1	$L_{basis} = O(1)$
Point pruning basis:	
If P = 0,3,4	$P_{basis} = O(subset)$
If P = 1,2	$P_{basis} = O(subset^2)$
Time complexity =	$O(gen * (subset^4 +$ $(subset^3 * P_{basis}) +$ $(subset^3 * L_{basis})))$

At worst $O(gen * subset^5)$, and at best $O(gen * subset^4)$. With the storage complexity being $O(Data_{training} + subset^2)$; independent of the pruning basis.

Furthermore, the computational complexities of each post processing method are (reference Appendix A.2 for details). Note, $Data_{entirety}$ is the sum of the training and testing data:

AA	$O(L_{arch} * Data_{testing})$
TC	$O((((Data_{dimension} * Data_{training}) +$ $Data_{testing}) * \log Data_{training}) +$ $(L_{arch} * Data_{testing}))$
G2	$O(L_{arch} * Data_{entirety})$

Yielding an algorithm independent of the input training data size for evolution, but depending on the post processing method, possibly utilizing all of the data in a singular instance (independent of the number of generations).

Chapter 4

Experimentation

To attain a measure of classification performance, the detection rate and 1 - false positive rate is averaged to produce a single classification "score". In other words, the average of the detection rate on both classes 0 and 1. Where detection rate is defined as the number of True Positives divided by the total number of Positives, and False Positive Rate is defined as the number of False Positives divided by the number of Negatives. See Equation 4.1.

$$Score = \frac{\frac{TruePositives}{Positives} + (1 - \frac{FalsePositives}{Negatives})}{2} \tag{4.1}$$

Such a combined scheme is employed in order to provide a more informative metric of performance under the typically unbalanced datasets characterizing the large real world applications used to evaluated this work. To measure "efficiency", the run-time of the algorithms on a common machine under common conditions will be recorded.

The comparison of score and run-time against a set of comparison algorithms will determine the relative classification performance and efficiency of the proposed PGPC algorithm with respect to common parameters such as the number of generations, terminal set, etc, and a common post processing method.

Since there exist multiple proposed methods for archive pruning and point generation, an instance of the algorithm is executed under a set of parameter combinations (Learner archive pruning method - Point archive pruning method - Point Generation method), denoted "L-P-G". Each combination will be evaluated, and a scheme will be presented to select an optimal combination indicative of the PGPC algorithm.

4.1 Data Sets

The classification data sets used in the experiments consist of[1]:

- heart-disease:Cleveland ("Heart"): Known to return test errors in the range of 15 to 35% [18].
- liver-disorders ("Liver"): Known to return test error rates in the range of 28 to 34% [18].
- kdd04:Physics ("KDD04P"): Known to return test error rates in the range of 26 to 30 %.
- Adult: Known to return test error rates in the range of 15 to 20 % [1].
- KDD99: Known to return test error rates in the range of 10 % [1].

Each data set is considered a two-class problem, with either an arbitrarily chosen training/test partition or one specified by the problem itself. The data set features are summarized in Table 4.1, and the class balance (ratio of the more represented class out of the training and testing sets), in Table 4.2.

Table 4.1: Basic features of the data sets.

Data set	Training Points	Testing Points	Dimension
Heart	227	76	13
Liver	258	87	6
KDD04P	10046	3349	78
Adult	33916	11306	14
KDD99	494020	311027	41

4.2 Experimental Hardware and Software

The relevant hardware of the test machine utilized for the run-time experiments consists of:

[1]Available at:
http://www.ics.uci.edu/~mlearn/MLSummary.html [Heart,Liver,Adult]
http://kodiak.cs.cornell.edu/kddcup [KDD04P]
http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [KDD99]

Table 4.2: Class balance of the data per training and testing subsets. Ratios are specified using the larger class .

Data set	Training	Testing
Heart	0.559956	0.514851
Liver	0.559420	0.640580
KDD04P	0.500212	0.506084
Adult	0.751316	0.754677
KDD99	0.803091	0.805191

- Pentium 4, 2.60GHz HT, 800MHz FSB
- 1GB DDR400 RAM
- 36GB SATA 10K-RPM Hard Drive
- With the GP implementation being based on the lilGP[2] framework, running on Fedora Core 3 Linux.

4.3 Statistical Measures

Due to the stochastic nature of the GP based algorithms, performance may be related to the sequence of random numbers utilized in the evolution. Therefore a total of 30 different random number seed values were utilized for each experiment, with the median being used to characterize algorithm performance.

4.4 Parameters

The lilGP parameters common to all of the GP algorithms may be found in Table 4.3 where this corresponds to Koza's canonical GP as appropriate for classification problems [16], and the parameters specific to the PGPC algorithm in Table 4.4. Note that no attempt was made to optimize these selections. The parameters specific to each of the comparison algorithms to be described in the subsequent section may be found in Table 4.5.

Although the learner archive is the set of learners constituting an "answer" to the

[2]Available at: http://garage.cps .msu.edu/software/lil-gp

classification problem, the learner population is still used for exploration and the evolution of the learners. Therefore the sum of both the archive and population sizes is used to provide a suitable learner set size for the comparison algorithms as to maintain a measure of equality. The same holds true for the point population and archive, since they are both used in the evaluation of the fitness of a learner, they constitute the number of points that the algorithm can "access", therefore the comparison algorithm's subset sample size is set to be the same.

Table 4.3: LilGP parameters common to all of the algorithms.

Number of generations	500
Population initialization method	Half-and-half
Population initialization depth	2-6
Individual maximum nodes	1000
Individual maximum depth	17
Learner breeding phase selection method	Fitness Proportionate
Learner breeding phase operators	Crossover, Mutation
Learner breeding phase frequency	0.8, 0.2
Individual function set	*, /, +, -, sin, cos, exp, sqrt

Table 4.4: Parameters specific to the PGPC algorithm.

Learner population size	25
Learner archive maximum size	25
Point population size	25
Point archive maximum size	25

Table 4.5: Parameters specific to each of the comparison algorithms.

	Regular	Cycling	Random	DSS
Learner population size	50	50	50	50
Learner evaluation sample size	Training data	50	50	50

4.5 Comparison Algorithms

The base line comparison algorithm is a canonical (traditional tree structured) GP classifier [16] (See Section 2.1) denoted as "Regular", consisting of only one learner population. At every generation, the fitness of each learner is computed using the entire training data set. The absolute switching function wrapper maps the *gpOut* value of the individual against the training data point class (reference Equation 3.1), and the number of correct mappings (classifications) is recorded and normalized into a fitness value (accuracy). The fitness values of the individuals are used to perform fitness proportionate selection for breeding the next generation of individuals. Upon completion of the evolution, the highest scoring individual (highest accuracy throughout the entire evolution, not just in the final population), is used to classify the testing data using the same switching function. This best-trained-individual method will be denoted as "BT" and will be used as a comparison for the AA post processing method. The TC and G2 methods will be simply applied to the entire learner population.

This comparison algorithm is indicative of the best possible performance of the PGPC classifier if fitness evaluation is conducted over the entire training data set. It may be possible to outperform this algorithm if the learner-point dynamics of PGPC perform a better search of the solution space, or are more conductive to the post processing methods used.

The three additional comparison algorithms use a limit on the number of training exemplars to provide a more accurate comparison for the PGPC classifier. These algorithms will allow for an evaluation of the IPCA-based dynamics and solution space search efficiency. They differ from the Regular algorithm only in the method in which training subsets are identified for fitness evaluation.

4.5.1 Cycling Subset Selection

At any evaluation of any individual, a global index into the training data is incremented (with wrap-around at the end of the training data set). Such that the training points subsequent to the index are utilized to compute the fitness. Upon completion of the evolution, the best case individual from the population is identified over all training exemplars, and the testing data classification is performed as in the Regular algorithm. See Figure 4.1.

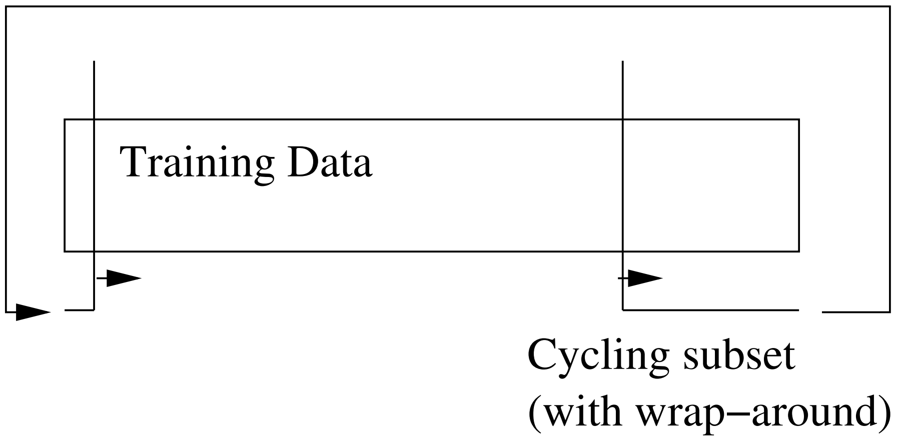


Figure 4.1: Cycling subset selection.

4.5.2 Random Subset Selection

This algorithm is based on the “Stochastic sampling” method described in [22] (see Section 2.1.1). Where at any evaluation of any individual, random training data points are uniformly chosen to populate a fixed size subset, and used to compute the fitness of the individual. Again, a single best individual is identified post training using all the training data, and the testing data classification is done as in the Regular algorithm. See Figure 4.2.

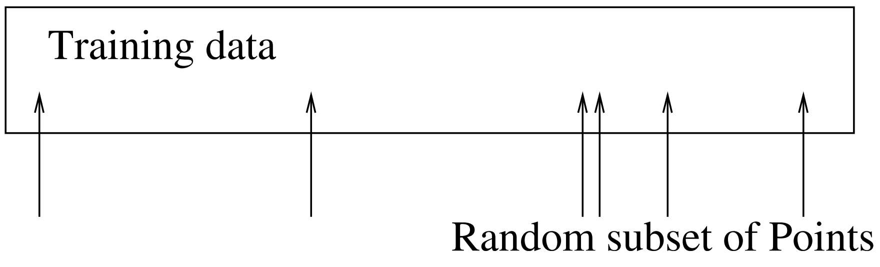


Figure 4.2: Random subset selection.

4.5.3 Dynamic Subset Selection

This algorithm implements the DSS algorithm described in [7], where each training point has an associated “age” and “difficulty” value. At each generation, an arbitrarily chosen balance of the age and difficulty values produces a “point weight” value for each training point p, in this implementation:

$$\text{PointWeight}_p = \text{Difficulty}_p^{1.0} + \text{Age}_p^{3.5}$$

mimicking the values utilized in the experiments of the referenced work. Each point weight is normalized over all point weight values, and multiplied by the target subset size, yielding a selection probability value.

Using the selection probability value, each point is randomly chosen for insertion into the subset for the current generation. It should be noted that the nature of the selection probability value may mean that the target subset size may slightly exceed or fall under the specified limit.

For each training point that was not selected for entry into the current generation subset, its age value is increased, and the difficulty value left unchanged. For each point which gained entry into the current generation subset, its age value is reset to zero, and upon evaluation of all learners in the current generation, the difficulty value is set to the number of occurrences where the point was classified incorrectly. As in the previously described algorithms, the fitness value of each learner is computed using the subset selected for the current generation, the entire training data set identifies the candidate solution post training, and the testing data classification is performed as in the Regular algorithm. See Figure 4.3.

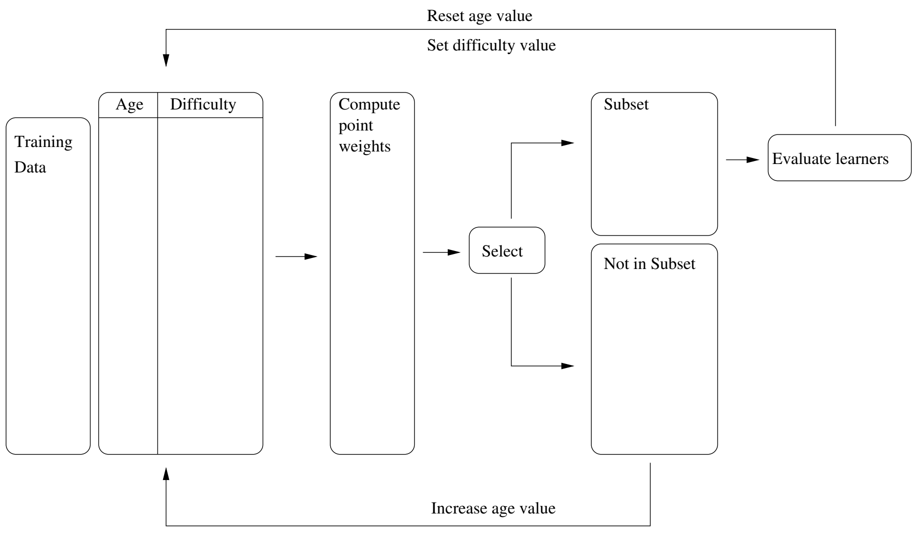


Figure 4.3: Dynamic subset selection.

Chapter 5

Results

5.1 Classification and Run-time Performance

In order to provide one instance of the PGPC algorithm to evaluate against the comparison algorithms, a scheme or mapping must be devised to select one parameter combination (learner archive pruning method, section 3.3.1 - point archive pruning method, section 3.3.1 - point generation method, section 3.3), to be utilized as representative of the PGPC algorithm. To this end, PGPC algorithm variants are denoted in terms of a tuple (L-P-G), see Table 5.1.

Table 5.1: Summary of the PGPC parameter combination components.

Learner archive pruning (L)	0	Diversity enhancing, based on outcomes against the point archive.
	1	Greedy, based on outcomes against the point archive.
Point archive pruning (P)	0	Diversity enhancing, based on co-ordinate values
	1	Diversity enhancing, based on distinctions against the learner archive.
	2	Greedy, based on distinctions against the learner archive.
	3	Diversity enhancing, based on co-ordinate values, preserves cluster boundaries
	4	Diversity enhancing, based on co-ordinate values, preserves cluster boundaries, enforces class balance.
Point generation method (G)	0	Uniform random generation.
	1	Class balanced, uniform random generation.

The classification performance results of each of the parameter combinations on each of the input data sets may be referenced in Appendix B. The basis and results of the chosen parameter selection scheme follow:

Table 5.2 describes the highest scoring parameter combination for each dataset-post processing method instance.

Table 5.2: Highest ranking parameter combination(s) (L-P-G) per each dataset-post processing instance.

Data set	AA	TC	G2
Heart	1-3-0	1-3-0	1-3-0
Liver	1-4-0	1-0-0,1-3-1	1-0-1
KDD04P	1-3-1	1-4-0	1-3-1
Adult	1-4-1	1-4-1	0-1-1
KDD99	1-2-1	1-4-0	1-0-0,1-3-0

Applying a ranking to the parameter combinations yields a mechanism for identifying combinations which may perform well on average across the multiple data sets, such a ranking may be referenced in Appendix C.

Table 5.3 illustrates the highest average ranking combination and associated average rank value, if the relative ranks of the parameter combinations were averaged for each post-processing method across all the data sets.

Table 5.3: Highest average ranking parameter combination (L-P-G) per post-processing method and associated average rank.

	AA	TC	G2
Best average combination	1-4-1	1-4-0,1-4-1	1-4-0
Best average rank	2.80	2.20	4.20

Acknowledging the fact that the class balance of the input data sets may affect the performance of the various parameter combinations, especially considering that many of them enforce a measure of class balance upon the point archive. The class balance of the input data was chosen as a decision factor in the partition of the input data sets into two groups, as to hopefully attain a higher average ranking scheme.

It should be noted that the threshold class balance value was not an issue in this work as the input data sets may be split into {Heart,Liver,KDD04P} (50-56% class balance), and {Adult,KDD99} (75-80% class balance). The author acknowledges the fact that for data sets lying in between these two groups, the recommended parameter combination may be unknown, and thus recommends evaluating both.

Table 5.4 shows the highest average ranking combination and associated average rank value per each of the two sub-groups of data sets. It should be observed that the average rank values are better for each of the combinations when compared to the class-unaware method, indicating that a partition based on class balance is beneficial.

Using the derived parameter combination selection scheme, Tables 5.5, 5.6, 5.7, 5.8, and 5.9, present the final comparisons grouped by post-processing method for each input data set. Detection rate, False positive (FP) rate, and Score range from zero to unity, and time is provided in seconds, furthermore a higher score value is indicative of superior classification performance, and a lower time value of superior efficiency. Note that the nearest neighbour ("NN") preprocessing step utilized in the TC method is appended to the comparisons as its run-time should be included in any estimate of real-world execution time.

The reader may also reference Appendices D, and E, for quartile results of the measured score and run-time values. In short, the observed variance of results was generally similar across all of the algorithms, and as such the median value may be considered indicative of general performance.

Table 5.4: Highest average ranking parameter combination (L-P-G) per post-processing method and associated average rank across the two sub-groups of data sets partitioned by class balance .

Data sets	AA	TC	G2
Heart, Liver, KDD04P combination:	1-4-0	1-4-0	1-3-1
Average rank value	2.33	2.00	2.33
Adult, KDD99 combination:	1-4-1	1-4-1	1-4-0
Average rank value:	1.50	1.50	3.00

Table 5.5: Results of the post processing instances of the various algorithms on the Heart dataset.

Algorithm	Detection Rate	FP Rate	Score	Time
PGPC - AA	0.589744	0.081081	0.752945	35.42
Regular - BT	0.538462	0.081081	0.729383	9.46
DSS - BT	0.666667	0.135135	0.7388739	1.77
Random - BT	0.487179	0.189189	0.562717	3.89
Cycling - BT	0.487179	0.189189	0.608455	3.56
PGPC - TC	0.564103	0.081081	0.715870	35.43
Regular - TC	0.641026	0.189189	0.712405	9.46
DSS - TC	0.641026	0.189189	0.725225	1.79
Random - TC	0.487179	0.297297	0.594941	3.90
Cycling - TC	0.435897	0.243243	0.594941	3.57
NN	0.487179	0.297297	0.594941	0.01
PGPC - G2	0.820513	0.189189	0.802148	39.34
Regular - G2	0.923077	0.297297	0.797990	9.49
DSS - G2	0.897436	0.216216	0.774428	1.84
Random - G2	0.743590	0.243243	0.750173	3.94
Cycling - G2	0.820513	0.243243	0.745322	3.65

Table 5.6: Results of the post processing instances of the various algorithms on the Liver dataset.

Algorithm	Detection Rate	FP Rate	Score	Time
PGPC - AA	0.709677	0.589286	0.560484	54.67
Regular - BT	0.322581	0.160714	0.584677	15.52
DSS - BT	0.193548	0.053571	0.560484	2.58
Random - BT	0.645161	0.642857	0.497984	1.88
Cycling - BT	0.709677	0.767857	0.499712	1.69
PGPC - TC	0.419355	0.285714	0.566820	54.68
Regular - TC	0.419355	0.285714	0.566820	15.52
DSS - TC	0.419355	0.285714	0.566820	2.59
Random - TC	0.419355	0.303571	0.565092	1.88
Cycling - TC	0.451613	0.285714	0.566820	1.70
NN	0.419355	0.285714	0.566820	0.02
PGPC - G2	0.225806	0.142857	0.543203	43.43
Regular - G2	0.161290	0.107143	0.519873	15.56
DSS - G2	0.129032	0.107143	0.543203	2.64
Random - G2	0.193548	0.125000	0.530818	1.91
Cycling - G2	0.193548	0.142857	0.538018	1.73

Table 5.7: Results of the post processing instances of the various algorithms on KDD04P dataset.

Algorithm	Detection Rate	FP Rate	Score	Time
PGPC - AA	0.520556	0.164012	0.678974	40.79
Regular - BT	0.775091	0.253097	0.760975	354.00
DSS - BT	0.773277	0.251327	0.760975	4.16
Random - BT	0.530230	0.521534	0.504348	3.16
Cycling - BT	0.530230	0.521534	0.504348	2.60
PGPC - TC	0.714631	0.284366	0.715788	40.90
Regular - TC	0.725514	0.310914	0.707514	354.32
DSS - TC	0.720677	0.305015	0.707587	4.50
Random - TC	0.701935	0.316814	0.692826	3.68
Cycling - TC	0.701330	0.315044	0.692826	3.13
NN	0.699516	0.313864	0.692826	142.88
PGPC - G2	0.769045	0.253097	0.756794	40.63
Regular - G2	0.773277	0.253097	0.758600	355.60
DSS - G2	0.773277	0.251327	0.755687	5.63
Random - G2	0.730955	0.301475	0.633192	5.95
Cycling - G2	0.699516	0.292625	0.674578	5.01

Table 5.8: Results of the post processing instances of the various algorithms on the Adult dataset.

Algorithm	Detection Rate	FP Rate	Score	Time
PGPC - AA	0.599625	0.105984	0.736611	41.38
Regular - BT	0.928976	0.625090	0.611569	1973.74
DSS - BT	0.969175	0.911319	0.526903	11.29
Random - BT	0.960267	0.911319	0.527521	3.63
Cycling - BT	0.898734	0.910598	0.520470	3.46
PGPC - TC	0.704993	0.251983	0.715597	41.68
Regular - TC	0.831810	0.555155	0.637742	1975.35
DSS - TC	0.844351	0.578226	0.632186	13.34
Random - TC	0.835795	0.568493	0.628435	5.56
Cycling - TC	0.834154	0.570656	0.636016	5.65
NN	0.808955	0.528839	0.640058	33.30
PGPC - G2	0.668425	0.169430	0.747856	41.57
Regular - G2	0.733474	0.282624	0.709502	1981.17
DSS - G2	0.731950	0.367700	0.707612	20.40
Random - G2	0.712729	0.308219	0.713315	11.57
Cycling - G2	0.715659	0.247296	0.723867	12.89

Table 5.9: Results of the post processing instances of the various algorithms on the KDD99 dataset.

Algorithm	Detection Rate	FP Rate	Score	Time
PGPC - AA	0.914581	0.045039	0.918419	56.97
Regular - BT	0.904626	0.030994	0.909291	40347.74
DSS - BT	0.957370	0.299198	0.827497	120.87
Random - BT	0.000076	0.000017	0.500000	4.20
Cycling - BT	0.075864	0.006717	0.501884	2.58
PGPC - TC	0.910544	0.006024	0.952254	57.20
Regular - TC	0.909474	0.005595	0.951360	40392.51
DSS - TC	0.908667	0.006024	0.951467	166.27
Random - TC	0.905553	0.004555	0.950136	20.96
Cycling - TC	0.905625	0.004258	0.950770	17.77
NN	0.908240	0.005529	0.959755	292078.53
PGPC - G2	0.852833	0.034906	0.863319	122.69
Regular - G2	0.772429	0.226697	0.764104	40574.29
DSS - G2	0.956268	0.297399	0.817037	343.84
Random - G2	0.954232	0.271884	0.716530	155.87
Cycling - G2	0.859273	0.247343	0.718183	140.45

Visual representations of the scores of the various algorithms may be found in Figures 5.1, 5.2, 5.3, 5.4, and 5.5.

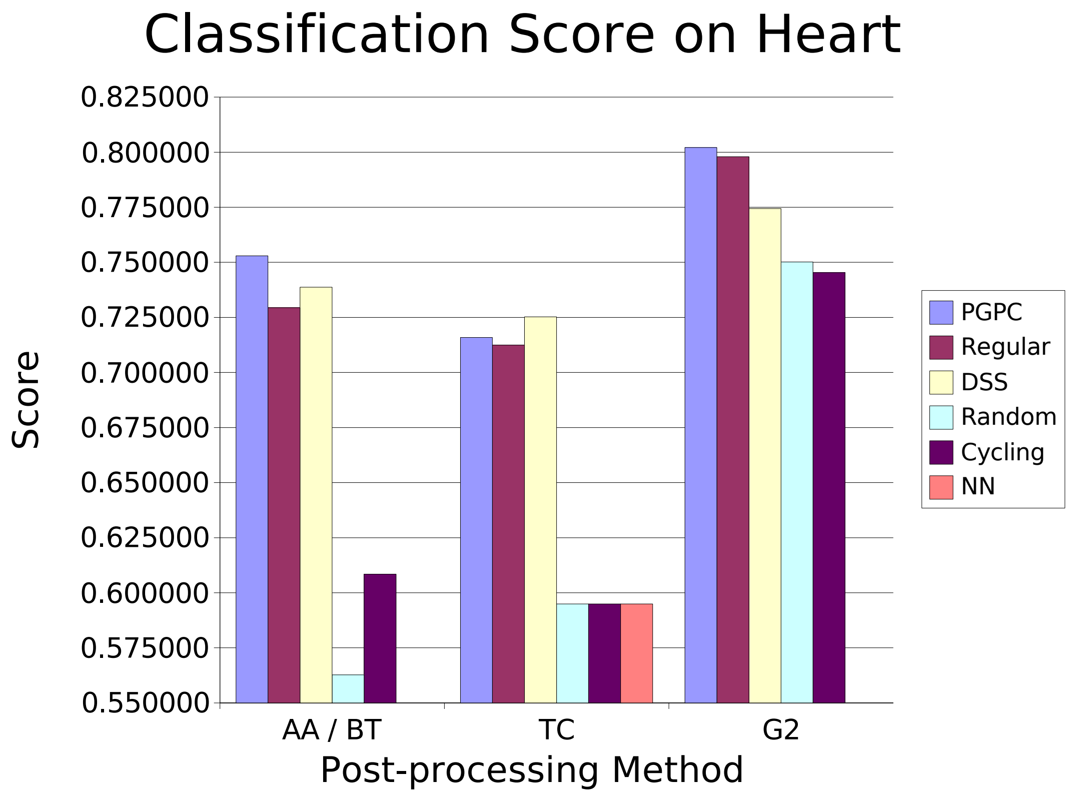


Figure 5.1: Median test classification score on Heart by the various algorithms under various post-processing methods.

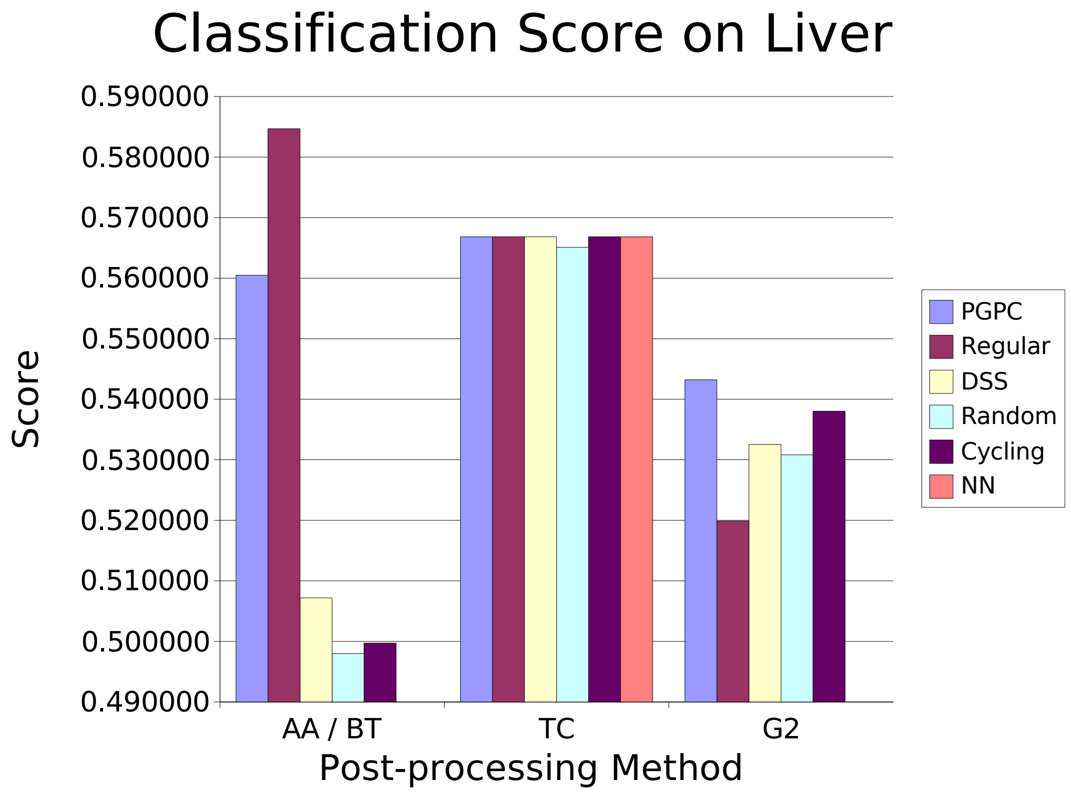


Figure 5.2: Median test classification score on Liver by the various algorithms under various post-processing methods.

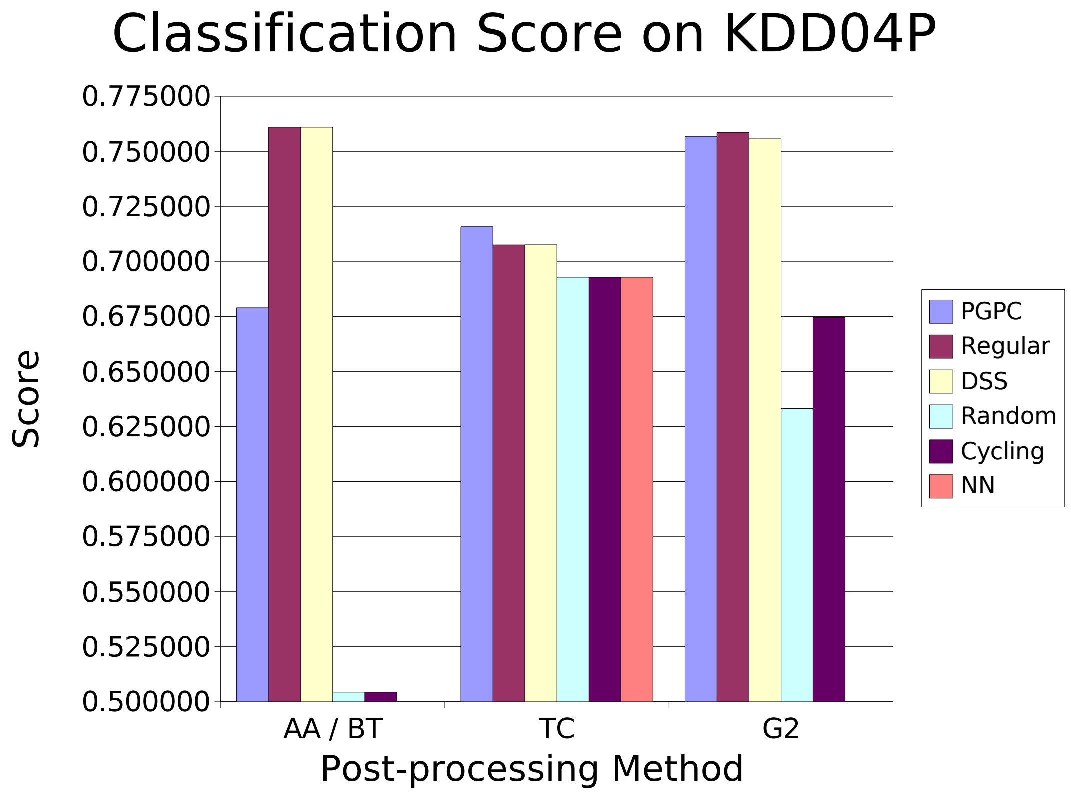


Figure 5.3: Median test classification score on KDD04P by the various algorithms under various post-processing methods.

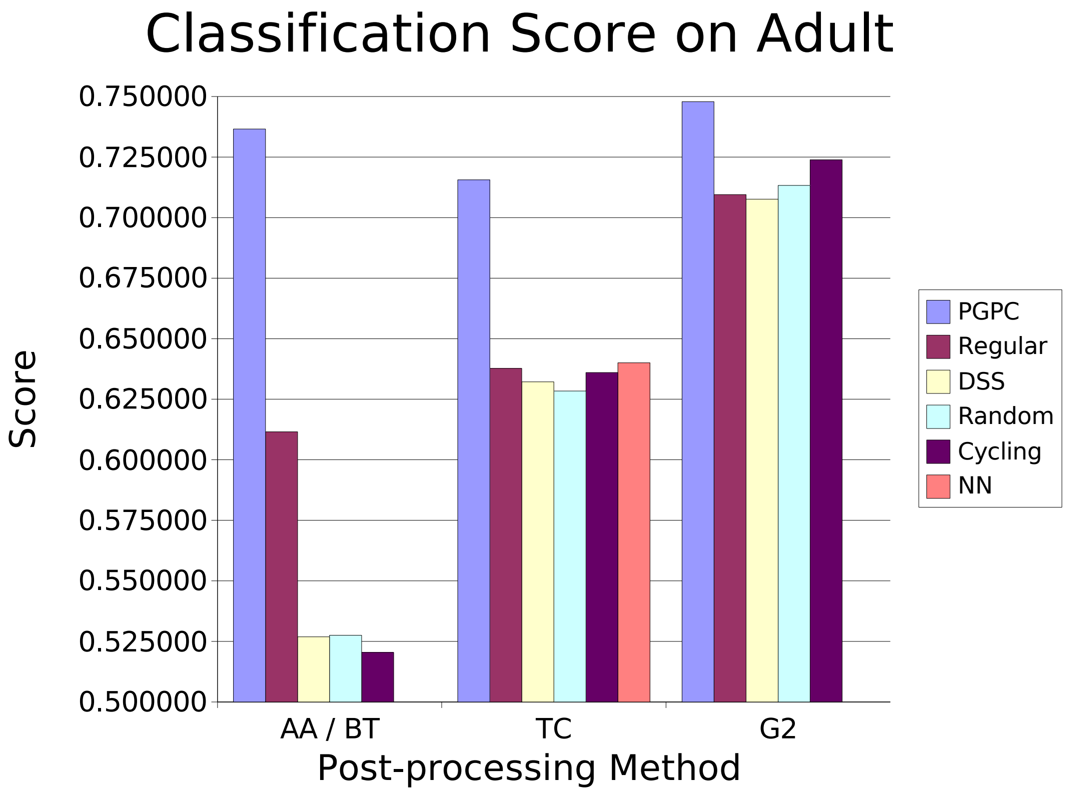


Figure 5.4: Median test classification score on Adult by the various algorithms under various post-processing methods.

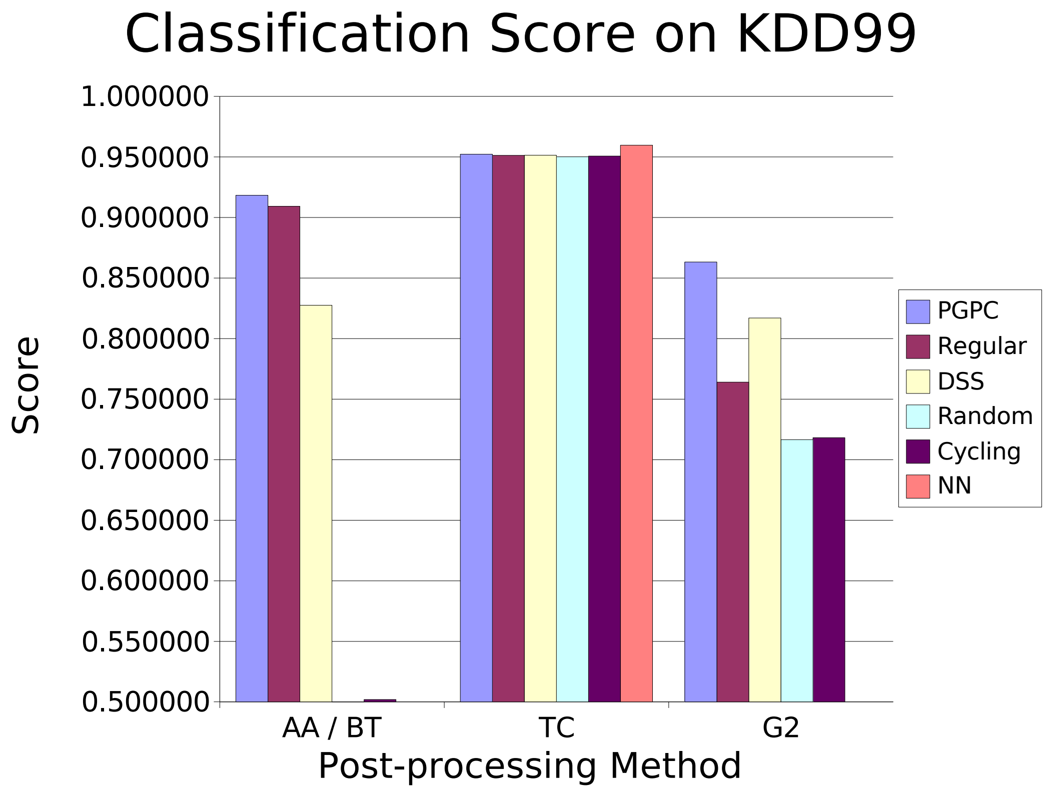


Figure 5.5: Median test classification score on KDD99 by the various algorithms under various post-processing methods.

5.2 Classification and Run-time Performance: Summary

A summary of the notable comparisons between the various algorithms may be found in Table 5.10. We highlight the highest scoring algorithm-post-processing combination per data set in addition to providing a ranking under which post processing methods certain algorithms out-scored others.

It should be noted that comparisons between the algorithms under different post-processing methods will not be made, as the effects upon the results vary, and no one method is clearly superior, they will be all considered independently. Furthermore, the varying run-time costs of the different post-processing methods make cross-method comparisons difficult. As such, the choice of post-processing method will be left to the algorithm user, and the PGPC algorithm will be hereafter evaluated and compared against the comparison algorithms under a common post-processing method.

Table 5.10: Summary of experiment results (score).

Data set	Overall best	PGPC beats Regular	PGPC beats DSS	PGPC beats Random	PGPC beats Cycling	DSS beats Regular
Heart	PGPC - G2	all	AA, G2	all	all	BT, TC
Liver	Regular - BT	TC-tie, G2	all, TC-tie	all	all, TC-tie	TC-tie, G2
KDD04P	Regular - BT, DSS - BT	TC	TC, G2	all	all	BT-tie, TC
Adult	PGPC - G2	all	all	all	all	none
KDD99	NN	all	all	all	all	TC, G2

Utilizing the derived parameter combination mapping, the PGPC algorithm scored higher than the comparison Cycling and Random selection algorithms for each post processing method on each data set (with the exception of Cycling-TC on Liver, which was a tie). This indicates that the proposed algorithm provided a mechanism for attaining a superior subset of training exemplars to perform the learner evolution upon, in addition to evolving a set of learners that is more "accurate" and conducive to the post processing methods.

Within the context of those results, the associated costs of evolving the subset of points does incur additional overhead in terms of run-time by an average factor of 13.62 against the simpler Random and Cycling methods. Note that the time factors computed for this section do not include the NN preprocessing step, only the average of the AA/BT,TC,and G2 evolution and evaluation times.

In regards to the performance against the Regular algorithm, the PGPC algorithm managed to attain higher classification scores on the Heart data set (using all post-processing methods), the Liver data set (using the TC and G2 methods), the KDD04P set (using the TC method), the Adult data set and the KDD99 set (using all methods).

In terms of execution efficiency with regards to the subset size, the PGPC algorithm exhibited an average speed increase of approximately 127.71 times against the Regular algorithm; whereas the much simpler Random and Cycling algorithms exhibited a factor of 1043.88 times.

Against the more advanced DSS algorithm, the PGPC algorithm exhibited similar performance as against the Regular algorithm. Indicating that even with the same amount of available information (subset size), the PGPC algorithm can out-perform the DSS algorithm.

It should be noted that the DSS algorithm often tied the Regular algorithm in terms of classification score while using a subset of points, thus performing better than the Random and Cycling algorithms, however it did not out-perform the Regular algorithm as often as PGPC.

In terms of execution efficiency, the DSS algorithm exhibited an average speed increase against the Regular algorithm by a factor of 91.71, slower than the Random and Cycling, and PGPC algorithms. This is most likely due to the DSS algorithm having to maintain age and difficulty values for each of the training points, whereas the other subset selection algorithms evaluated dealt strictly on a subset.

It may be observed that on the smaller data sets (Heart, Liver), the overhead of evolving the point subset was greater than the cost of evaluating each learner on each training point. Therefore, for those data sets, the execution time of the PGPC algorithm was in fact slower. As the number of training points increased (KDD04P, Adult, KDD99), the evaluation of every training point far outweighed the cost of evolving the subset of points, yielding a relative speed gain for the PGPC algorithm.

In summary, the PGPC algorithm proved to be more time efficient than the Regular algorithm (on the larger data sets), additionally it often defeated the Regular algorithm in terms of score, more so than the DSS algorithm, which in fact ran slower than the PGPC algorithm, satisfying the first stated objective of the algorithm; similar classification performance to the Regular algorithm whilst utilizing less run-time resources.

Furthermore, although not as time efficient as the Random and Cycling algorithms, the PGPC algorithm exhibited the most superior classification performance out of all of the comparison subset selection algorithms, thereby satisfying the second stated objective; superior classification performance when compared to the other subset selection algorithms.

With regards to the post-processing methods applied to the various algorithms over the input data sets, the results tend to vary with the data set. In the general case, the post-processing methods often managed to increase the PGPC AA score, and occasionally increased the BT scores of the comparison algorithms; with the increase being dependent upon the algorithm. This follows from the notion that the PGPC algorithm produces an informative population or front of individuals which must be condensed into a label per testing point, and as such the post-processing operation is of more significance than in the case of applying the method to a population which already contains an identified best trained individual as per the comparison algorithms. It is still possible for the post-processing methods to out-perform the BT score on the comparison algorithms if they manage to identify a scheme to utilize the knowledge present in the classifier population more effectively than the best trained method.

Over all of the algorithms, the TC method often localized the scores of the algorithms around the NN score. This is as expected considering the methodology of the TC method, in that it builds upon the NN classification. Focusing upon the PGPC results, under the TC method, PGPC often defeated the base NN score, whereas the comparison algorithms tended to tie it (failed to substantially build upon the initial mapping). In the case where the NN score was the highest (KDD99), PGPC still out performed the other algorithms (moved least away from the initial mapping). Referencing the summary table (Table 5.10), the PGPC algorithm always beat/tied all of the other comparison algorithms under the TC method.

With regards to the G2 method, all of the algorithms under G2 exhibited superior scores when compared to their initial AA/BT score on the Heart data set. However on the Liver set, discounting the simple Cycling and Random algorithms, the converse is true. On the KDD04P set, the G2 method managed to increase the PGPC, Cycling, and Random scores, while retaining the BT scores of the Regular and DSS algorithms. On the Adult data set, the G2 method increased all of the classification scores of all of the algorithms past their initial AA/BT scores. Finally, on the KDD99 set, again discounting the simple Cycling and Random algorithms, the G2 post-processing method exhibited the worst scores. In summary, for the Heart, KDD04P, and Adult sets, the G2 method was beneficial, whereas for the Liver and KDD99 sets, detrimental. Therefore it is deemed to exhibit fairly data dependent characteristics.

In summary, for the PGPC algorithm, the TC method returned the most consistent results (managed to defeat or tie the Regular algorithm on all data sets). However, the G2 and AA methods also provided better results than the Regular algorithm on 4 out of 5 and 3 out of 5 datasets respectively, whilst representing a lower computational overhead than the TC scheme (which requires the costly NN preprocessing step).

In terms of the overall best algorithm-post-processing combination per data set, PGPC-G2 scored best on Heart and Adult, Regular-BT on Liver and KDD04P, DSS-BT tied with Regular-BT on KDD04P, and surprisingly Nearest Neighbour scored best on KDD99.

This re-enforces the notion that the post-processing methods are not as valuable to the comparison algorithms as to the PGPC algorithm in terms of attaining the highest score, however for many of the cases the post-processing methods managed to increase the score of the comparison algorithms past their initial Best Trained scores.

5.3 Assessing the Impact of Finite Archive Sizes - the Significance of Finite Information

Due to the application of pruning upon the archives by the PGPC algorithm, it would be interesting to investigate the effects upon the monotonic progress guarantee of the base IPCA algorithm. In particular, if instances of finite information occur in the PGPC algorithm, and if so, how often and how deep.

The loss of an informative point in the point archive may lead to the inability to differentiate between two learners along that possible underlying objective, leading to a loss of gradient or selection pressure, thus returning to a previous state (hereafter "forgetting"). In the case of a single objective optimization problem, instances of forgetting are easily measured, simply if performance (fitness) decreases. This could be applied to either the best trained individual, or the average fitness of the entire population. However, within a pareto-evolutionary context based on multiple objectives, forgetting is not as easily defined, especially since pareto-front members may move along the front and as such perform trade-offs between the various objectives.

For the purposes of this experiment, a strict pareto-based definition of forgetting will be utilized, namely if during the evolution, the entire pareto-front of learners moves to a position dominated by a front from a previous state. This allows for trade-offs to occur along the front, in the extreme case gaining new ground on one objective while completely losing on the remainder, without being labeled as forgetting.

To measure such occurrences, the derived parameter combinations for each data set were utilized, and at each generation the pareto-front was compared to each of the previous generations fronts. Over 30 independent runs per data set, no such instances of regression were recorded.

It should be noted that the KDD99 data set was not utilized in these experiments, as maintaining all of the previous pareto-fronts in conjunction with the data set itself proved excessive to the memory capabilities of the test machine.

Given these results, it may be inferred that in practice, the pruning of the archives, primarily the point archive (regarding the loss of underlying objectives), results in no, or possibly very minimal detectable instances of regression.

5.4 Unbound Archive Experiments

In order to measure the impact of archive pruning upon the classification performance of the PGPC algorithm, a set of instances were created with no limit placed on the maximum archive size. These instances still retained the original population limits as to allow for a meaningful comparison to the previous results, as the population sizes dictate the rate at which the search space is explored (new individuals per generation).

Each data set was evaluated with 30 independent runs, with the resultant statistics residing in Tables 5.11, 5.12, 5.13, 5.14, and 5.15. It should be noted that the pertinent parameter combination value (point generation method, or G) was set to the optimal derived value per data set (per the AA instance). That is, random for Heart, Liver, and KDD04P, and class balanced for the remainder.

It should be observed that the unbound classification scores tend to either tie or perform worse than the bound scores from the previous experiments, despite having access to the entirety of the pareto fronts during evolution. Thus contradicting the notion that archive pruning would be detrimental to classification performance.

Table 5.11: Statistics on an unbound archive instance of PGPC on Heart, using the random point generator (G0), and population sizes of 25.

Median number of learners	492		
Maximum number of learners	883		
Median number of points	52		
Maximum number of points	66		
Post-processing method	Detection Rate	FP Rate	Score
AA	0.615385	0.108108	0.752945
TC	0.512821	0.162162	0.673943
G2	0.538462	0.108108	0.692308

Table 5.12: Statistics on an unbound archive instance of PGPC on Liver, using the random point generator (G0), and population sizes of 25.

Median number of learners	341		
Maximum number of learners	723		
Median number of points	47		
Maximum number of points	59		
Post-processing method	Detection Rate	FP Rate	Score
AA	0.677419	0.553571	0.558756
TC	0.419355	0.285714	0.566820
G2	0.225806	0.107143	0.536002

Table 5.13: Statistics on an unbound archive instance of PGPC on KDD04P, using the random point generator (G0), and population sizes of 25.

Median number of learners	587		
Maximum number of learners	874		
Median number of points	69		
Maximum number of points	83		
Post-processing method	Detection Rate	FP Rate	Score
AA	0.775091	0.382301	0.675212
TC	0.706167	0.316814	0.694676
G2	0.769649	0.257817	0.753071

Table 5.14: Statistics on an unbound archive instance of PGPC on Adult, using the random point generator (G1), and population sizes of 25.

Median number of learners	639		
Maximum number of learners	1051		
Median number of points	62		
Maximum number of points	73		
Post-processing method	Detection Rate	FP Rate	Score
AA	0.658814	0.192502	0.695904
TC	0.805907	0.483778	0.660344
G2	0.744257	0.287671	0.682935

Table 5.15: Statistics on an unbound archive instance of PGPC on KDD99, using the random point generator (G1), and population sizes of 25.

Median number of learners	63		
Maximum number of learners	151		
Median number of points	39		
Maximum number of points	48		
Post-processing method	Detection Rate	FP Rate	Score
AA	0.911047	0.056377	0.915317
TC	0.910153	0.005743	0.952022
G2	0.811936	0.079053	0.822840

5.5 Maximum Archive Size Parameter Variation

In order to explore the classification performance issues which arose in the unbound archive PGPC instances, the following set of experiments will evaluate the performance of the PGPC algorithm in terms of its sensitivity to the selection of the maximum sizes of the learner and point archives.

The learner archive size will be set to {10, 25, 50}, and the point archive to {5, 10, 25, 50, 100}, and the median score will be measured over 30 independent runs.

The upper limit on the number of learners was arbitrarily chosen, whereas the upper limit on the number of points corresponds to a maximum number of observed members of the point archive during 10 runs on each dataset using a maximum of 50 learners, and the 1-4-1 parameter combination (the P-4 is inconsequential since the point archive was unbound). Table 5.16 illustrates the maximum number of recorded points in the archive for each data set. Past those values any additional space in the archive would be unused and the results would be superfluous as the archive contents would be the same, therefore a common limit of 100 points was selected, in accordance with the other range.

Table 5.16: Maximum number of points stored in an unbound archive, under a learner archive bound to a maximum size of 50 .

Data set	Maximum Points stored
Heart	69
Liver	65
KDD04P	78
Adult	65
KDD99	44

5.5.1 Maximum Archive Size - Parameter Combination Results

Utilizing the previously defined ranges for the maximum archive sizes, the PGPC algorithm was executed 30 times on each data set for each parameter combination (pruning basis + point generation method), with the exception of the Adult and KDD99 data sets, which were evaluated only on their optimal derived parameter combinations due to time constraints.

Statistics on the performance of each parameter combination may be referenced in Appendix F. Furthermore, a parameter combination ranking scheme similar to the one in Appendix C, may be found in Appendix G, again, composed only of the Heart, Liver, and KDD04P data set values.

The optimum derived parameter combination for the balanced data sets over all of the measured maximum archive size combinations may be referenced in Table 5.17.

Table 5.17: Optimum derived parameter combination (L-P-G), using the median score value measured over the set of scores under various maximum archive sizes.

Data sets	AA	TC	G2
Heart, Liver, KDD04P combination	1-3-1	1-4-0	1-4-1
Average rank value:	2.33	1.67	3.33

It should be noted that the aforementioned generalized optimum parameter combination is optimal in the context of the cumulative range of maximum archive sizes evaluated. As such it provides a suitable general set of parameters for PGPC instances utilizing varying maximum archive sizes. This general optimal parameter combination will serve as a comparison to the original optimal parameter combination specialized to the singular combination of archive sizes (maximum of 25 learners and points) in the analysis section only. All remaining experiments will continue to utilize the original optimum parameter combination as indicative of the PGPC algorithm.

5.5.2 Select Maximum Archive Size - Parameter Combination Terrains

The following consist of the surface plots of the PGPC algorithm score under the variance of the learner and point maximum archive values, utilizing the optimum derived parameter combination.

Figures 5.6, 5.8, 5.10, 5.12, and 5.14 illustrate the surface plots for each post-processing method for each data set. With Figures 5.7, 5.9, 5.11, 5.13, and 5.15 illustrating each surface plot for the AA post processing method for each data set independently. The independent TC and G2 surface plots may be referenced in Appendix H.

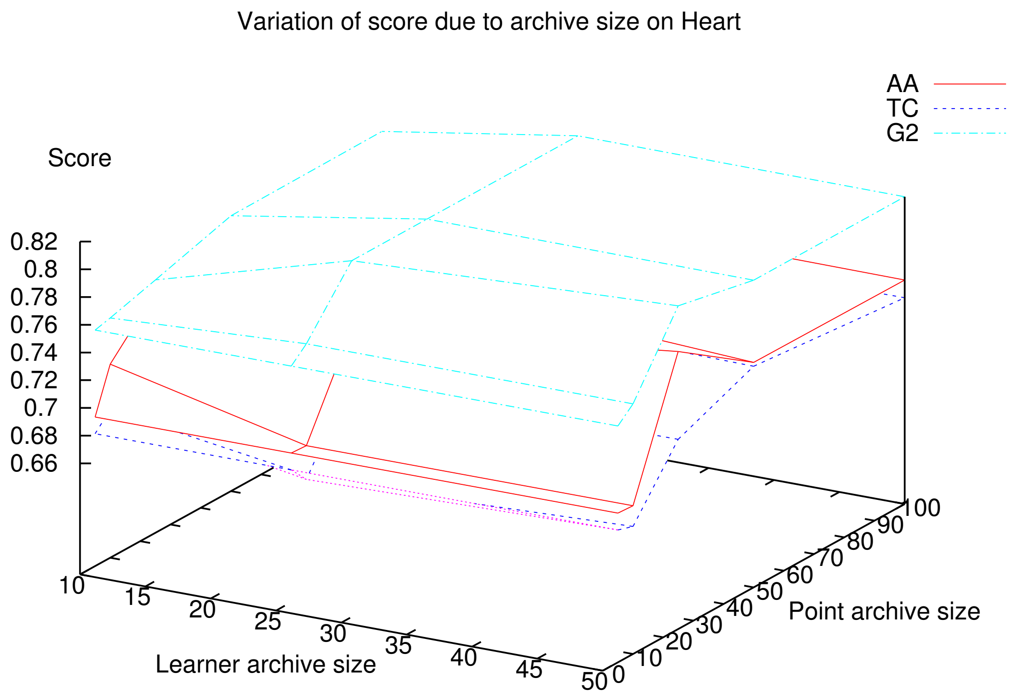


Figure 5.6: Effect of varying the maximum archive sizes upon the scores of the post-processing methods on the Heart data set using the optimal parameter combinations.

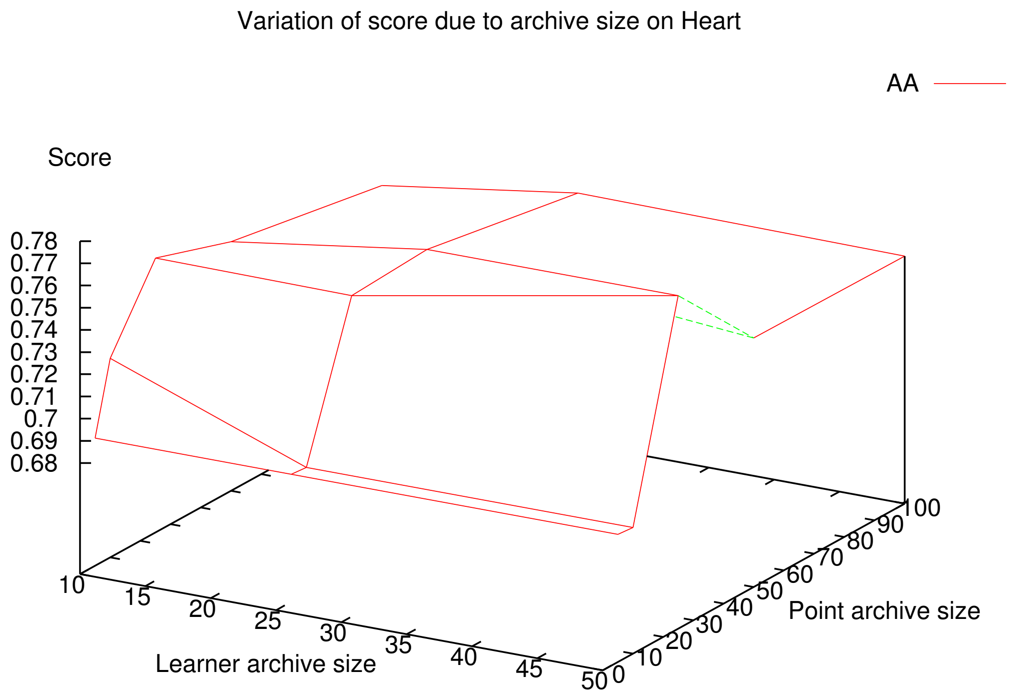


Figure 5.7: Effect of varying the maximum archive sizes upon the scores of the AA post-processing method on the Heart data set using the optimal 1-4-0 parameter combinations.

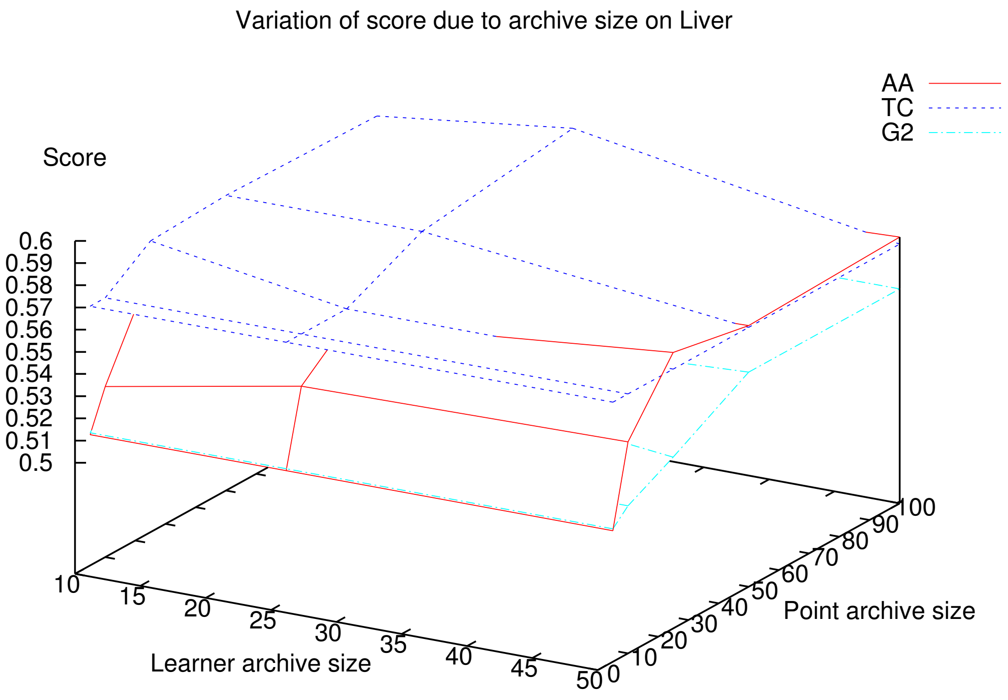


Figure 5.8: Effect of varying the maximum archive sizes upon the scores of the post-processing methods on the Liver data set using the optimal parameter combinations.

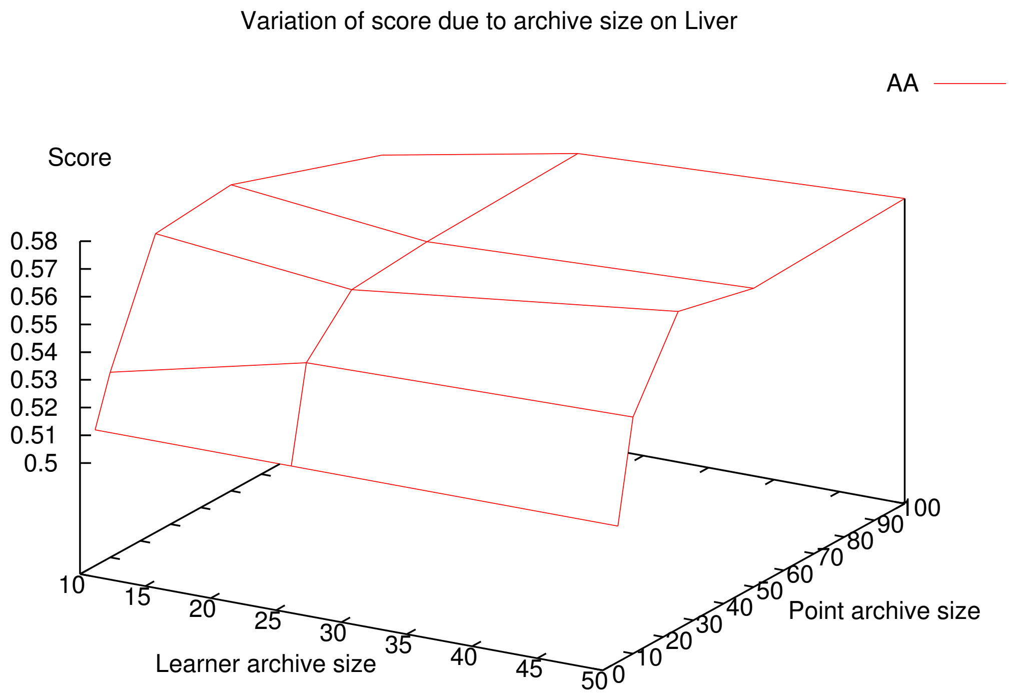


Figure 5.9: Effect of varying the maximum archive sizes upon the scores of the AA post-processing method on the Liver data set using the optimal 1-4-0 parameter combinations.

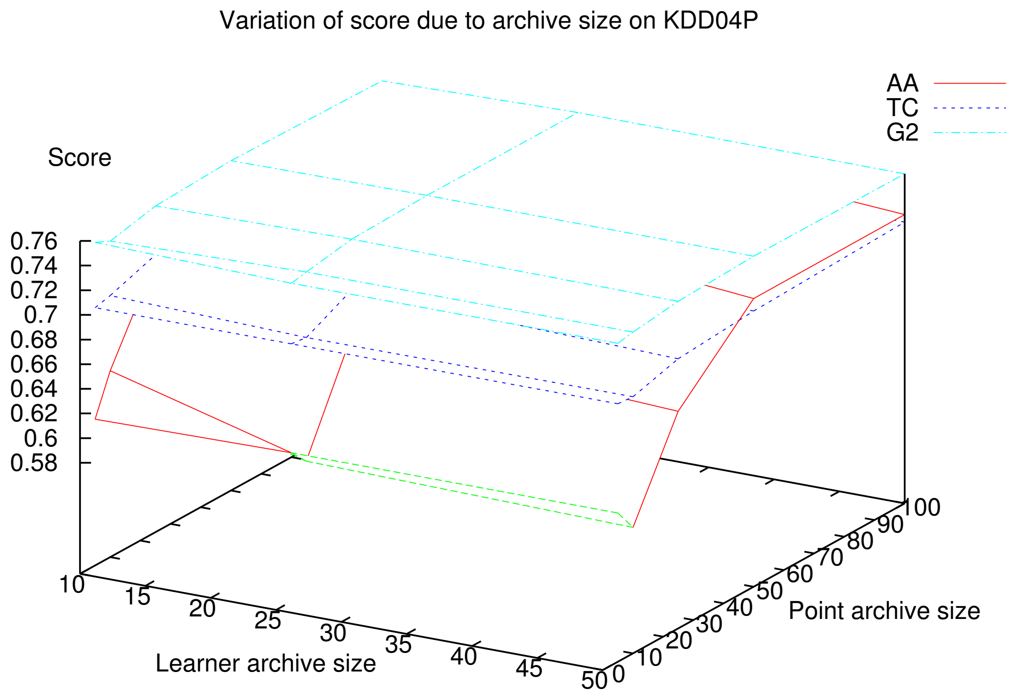


Figure 5.10: Effect of varying the maximum archive sizes upon the scores of the post-processing methods on the KDD04P data set using the optimal parameter combinations.

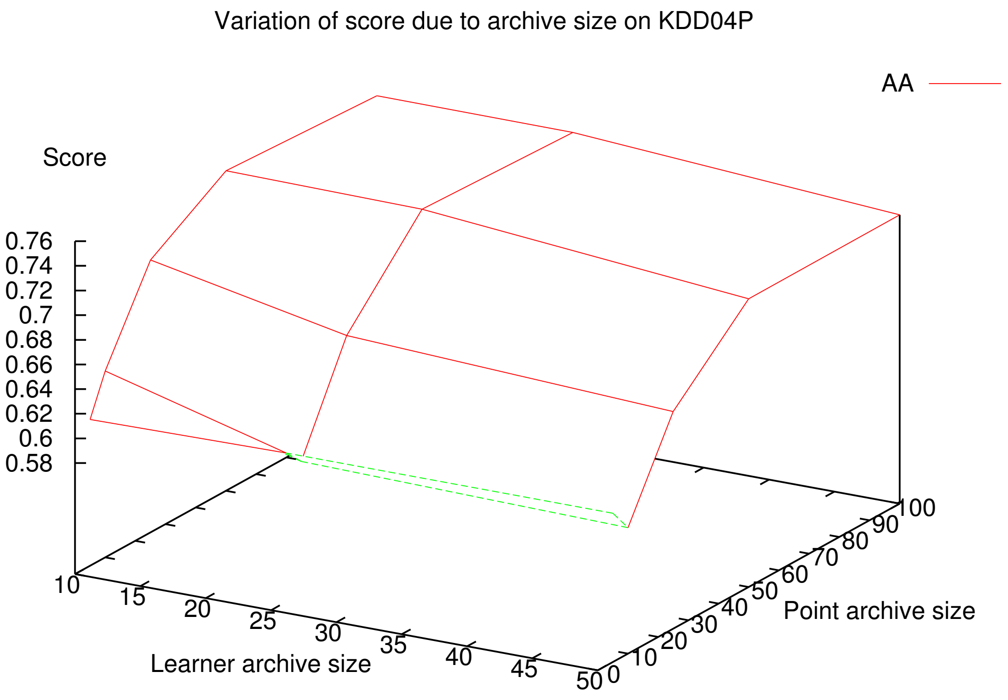


Figure 5.11: Effect of varying the maximum archive sizes upon the scores of the AA post-processing method on the KDD04P data set using the optimal 1-4-0 parameter combinations.

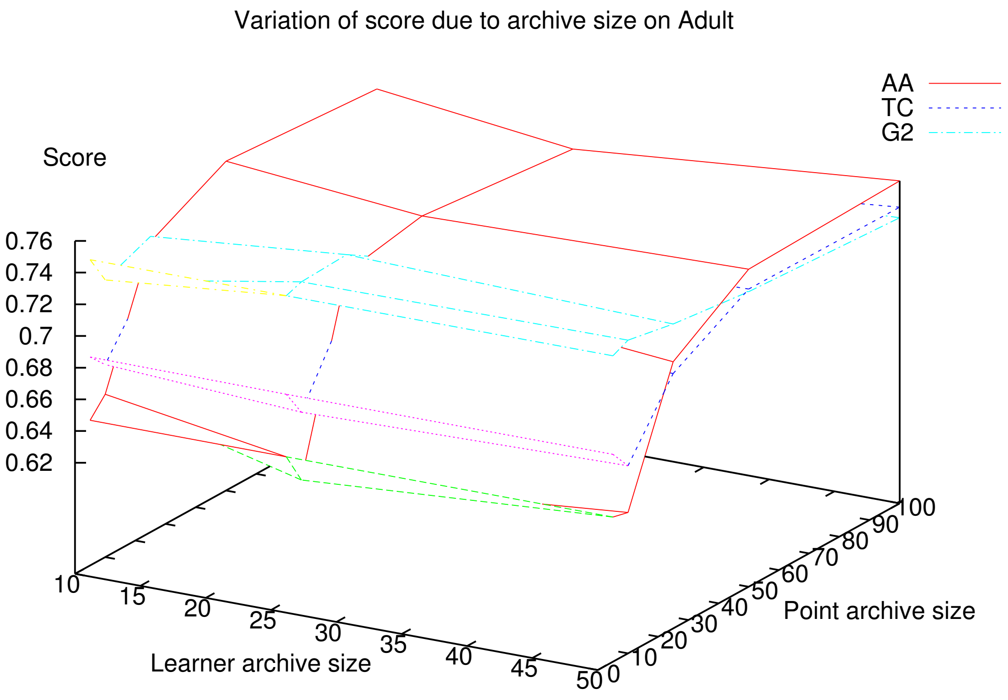


Figure 5.12: Effect of varying the maximum archive sizes upon the scores of the post-processing methods on the Adult data set using the optimal parameter combinations.

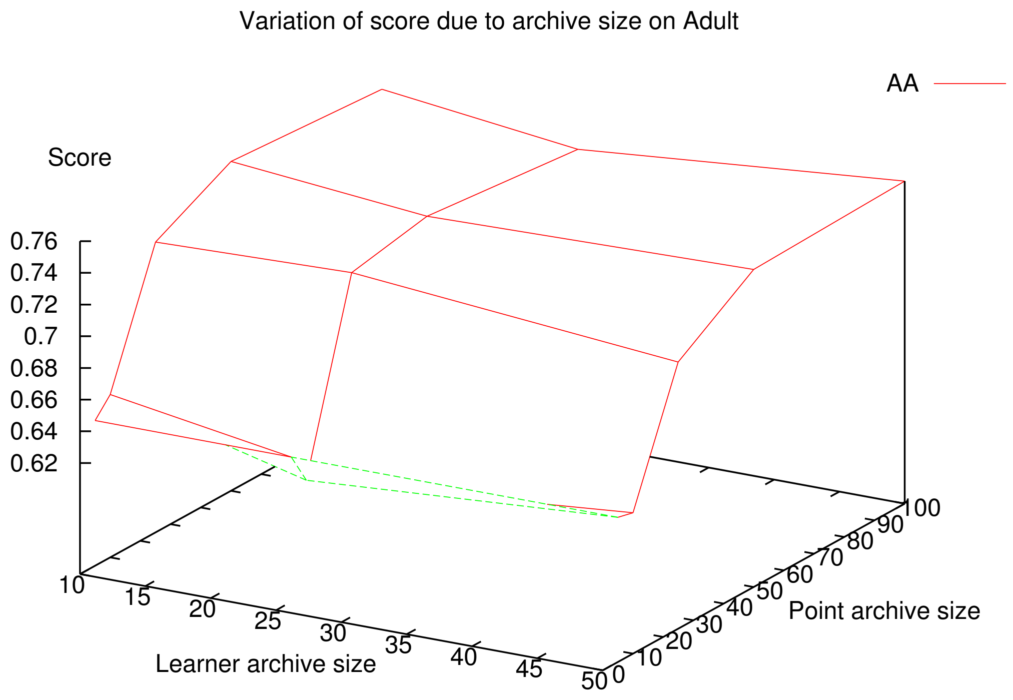


Figure 5.13: Effect of varying the maximum archive sizes upon the scores of the AA post-processing method on the Adult data set using the optimal 1-4-1 parameter combinations.

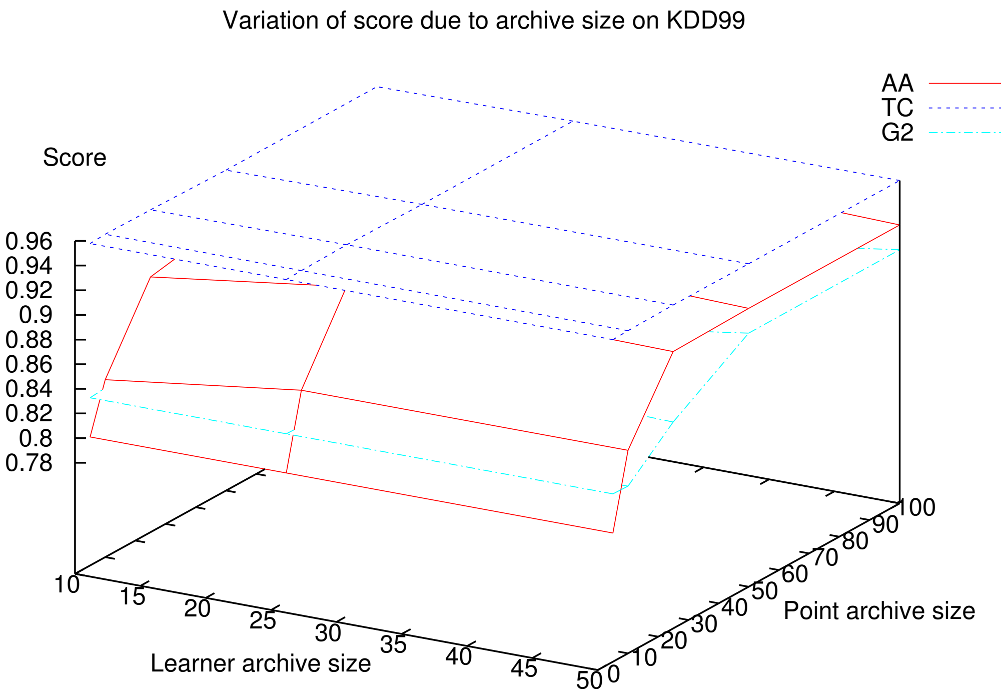


Figure 5.14: Effect of varying the maximum archive sizes upon the scores of the post-processing methods on the KDD99 data set using the optimal parameter combinations.

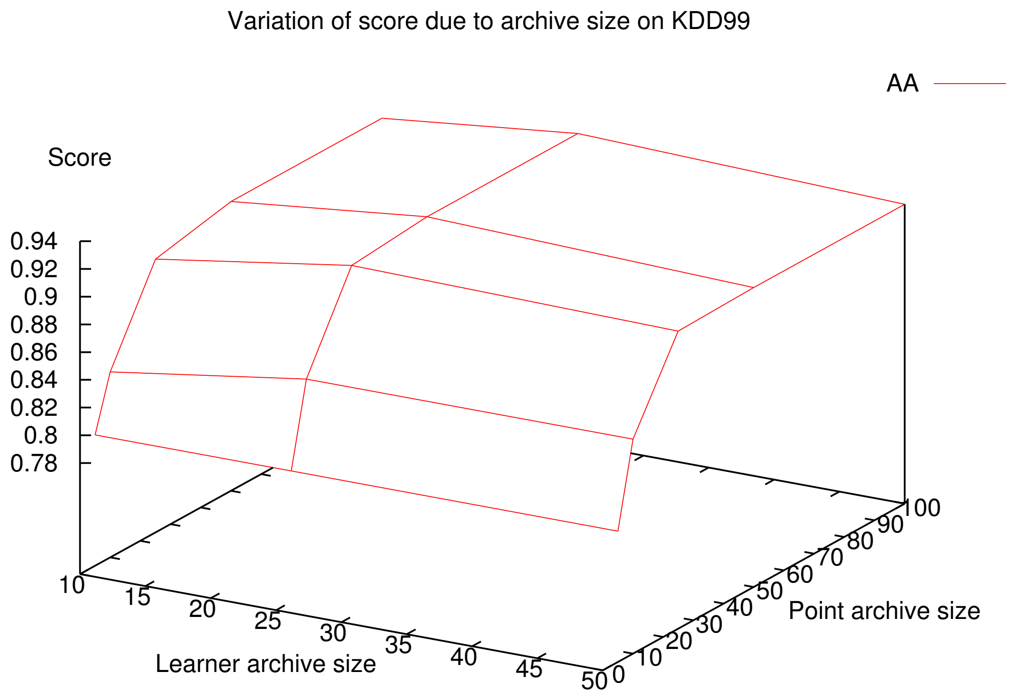


Figure 5.15: Effect of varying the maximum archive sizes upon the scores of the AA post-processing method on the KDD99 data set using the optimal 1-4-1 parameter combinations.

Generally, it may be observed that within the various plots there exist several local optima, and that it is not always the case that more learners and points produce higher scores, nor is the value of the two always equal.

If the AA method results are interpreted as being the raw results of the PGPC algorithm (sans post-processing), then it may be stated that the value of points is higher than the learners, as often an increase in the maximum number of points yields a larger gain in score than the same increase in learners. Furthermore increasing the number of learners has varied results on the different data sets, both increasing and decreasing the recorded score.

It may also be observed that the affects of archive sizes are most prominent upon the AA method, with the TC and G2 surfaces exhibiting a relatively flat profile past an initial phase of low number of learners and points, again with an increase in points being more significant than an increase in learners.

Chapter 6

Discussion

6.1 Analysis

6.1.1 Derived Parameter Combination

The following section discusses possible explanations for the performance of the derived optimal parameter combination upon the data sets utilized.

Learner Pruning Basis

Firstly, it may be observed that the greedy learner pruning basis (L1) was optimal across all post-processing and class balance categories. It may be the case that the pareto-front of learners did not reach a state during the evolution where diversity in the front was required to escape a local optimum. In that case, the greedy method would travel fastest towards an un-obscured optimal pareto-front. It may be possible given more generations and a suitably deceptive problem that the greedy method would stall in terms of progress at a local optimum, whereas the diversity enhancing method would promote a means of escape and continue on, subsequently out-performing the greedy method.

It may also be the case that the post processing methods did not isolate the pertinent learners for each test point, and as such failed to utilize diversity in the front effectively (as promoted by the L0 method).

The AA (Average Archive Value) and TC (Test to Training Point Clustering) methods, being predominantly majority voting schemes, would tend to overshadow the classification recommendation of a learner attuned to the input point (specialized on the subclass), by the noisy votes generated by the remaining uncertain learners. As such, a greedy learner pruning basis would be beneficial to these voting methods, as the average classification score of the front would be promoted.

The G2 (Gaussians of the *gpOut* values for the two classes) method may in theory identify a specialized learner for each input point, based on the learner's ability to produce disjoint *gpOut* values, with respect to each data class around the *gpOut* value of the input point (confidence). This method may effectively utilize diversity within a pareto-front as one learner is chosen to perform the classification, however the selected learner may turn out to be the incorrect choice.

In such a context, assuming the G2 method selects a learner out of the front with near random probability (biased towards the correct selection by the the learner's ability to output disjoint values based on input class), then the greedy learner pruning method would yield a near-homogeneous population of learners each having an average classification score, and the diversity enhancing basis would yield a specialized population having a very high score only on their respective sub-class of points. Depending upon the balance between the score and representation in the population of the optimal individual to be selected by the G2 method, the performance of both of the learner archive pruning basis would be affected. The observed results indicate that for the set of input data sets, the greedy selection method proved superior.

Point Pruning Basis

With regards to the P4 point archive pruning basis being optimal for the majority of the post-processing and class balance combinations, with the exception of the G2 method on balanced data using P3; it can be said that the two basis are in fact very similar. They both prune one of the Euclidean-measured genotypic closest points, with both of the closest point's having the same class as to preserve boundaries between clusters of points. P4 extends onto P3 by requiring that the selected closest points class is over-represented in the point archive, thereby promoting class balance within the archive.

The superior performance of the P3 method on the G2 class balanced data combination may not be easily discounted by the observation that the additional class balance promoted by the P4 method is not required on an already balanced input data set, since the AA and TC methods still preferred the P4 method, even on balanced input data.

It may be the case that class balance in the point archive is important in most cases (thereby requiring P4), however less so under G2. Therefore for unbalanced input data, re-balancing was required, but on the nearly balanced data, a degree of freedom was allowed. Using this degree of freedom, the P3 method was able to select closer pairs of points for pruning, independent of class balance, and thereby removing a more “redundant” point than the P4 method.

It also may be the case that the optimal class balance for the G2 method lies in between perfectly balanced and the input original unbalanced ratio (the unbalanced input data sets ranged from 75-80%). So the P3 method would retain the so called balanced data near their natural 50-56%, and the P4 method would move the unbalanced data class balance closer to that (from 75-80% to 50%).

Justifications for a possible preference of a class balance near equal by the G2 method should be investigated in addition to a more detailed study of the method and its effect upon the traditional GP output.

In regards to the superior performance of the P3 and P4 methods against the other point archive pruning basis, it may be said that they out performed the related P0 method (remove one of the two closest Euclidean measured points, discounting class information) due to the possibility that the P0 method may remove one of two points defining a boundary between two clusters of points; intuitively, very informative points. The additional class considerations of the P3 and P4 methods avoid this scenario, and bias the removal selection to a point within a cluster.

As opposed to the distinction-based phenotypic pruning basis of P1 and P2 (diversity enhancing and greedy, respectively), the superior performance of a genotypic approach which does not make use of any information gleaned from the interactions between the learners and points (distinctions) seems counter-intuitive. One would think that the role distinctions between two learners play in the algorithm, and the positive effects due to avoiding co-evolutionary failure by dis-engagement, would be re-enforced via consideration in the point archive pruning process as to maintain the learning gradient provided by the set of points. Empirically though, the observed results indicate otherwise.

To explore this issue further, statistics were collected on the number of distinctions made by point archive members. The sum of the number of distinctions made by each archive member was recorded, and the resultant set of sums was utilized for the analysis; 30 independent runs were performed and the median values recorded.

Given that a point provides a partition between two sets of learners (pass / fail the point "challenge"), see Figure 6.1, the distribution of the number of distinctions made by a set of points may be indicative of the placement of the points in partitioning the set of learners.

For example, a high number of average distinctions with low variance would mean that a large number of points partitions a large segment of the learners evenly, that is splitting a cluster of learners "through the middle", see Figure 6.2. Conversely, a low average number of distinctions would mean that the points partition the "learner cluster" unequally, or in other words, along the edges of the cluster. The variance of the number of distinctions made can be associated with the similarity of the partitions in terms of placement (distance to the centre of the learner cluster). See Figure 6.3.

It should be noted that using these statistical measures, the positions of the partitions may not be determined, nor the degree in which they span the "learner cluster", as that would require a high-dimensional plot and analysis. Given that points may only enter the point archive if they provide new distinctions between learners, it may be assumed that the distribution of point-partitions around the learners may be relatively uniform with respect to the current set of learners (they may be overlapping, however each point provides a unique distinction), see Figure 6.4.

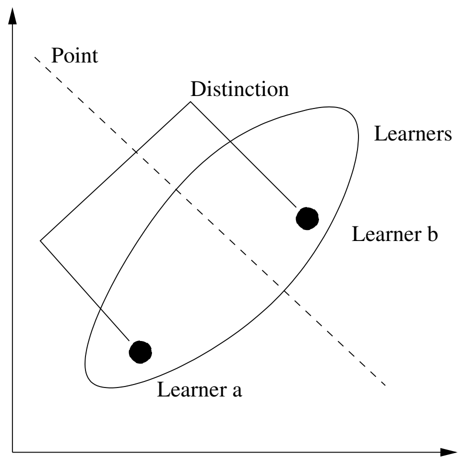


Figure 6.1: A point partitioning the set of learners by providing a distinction between two learners.

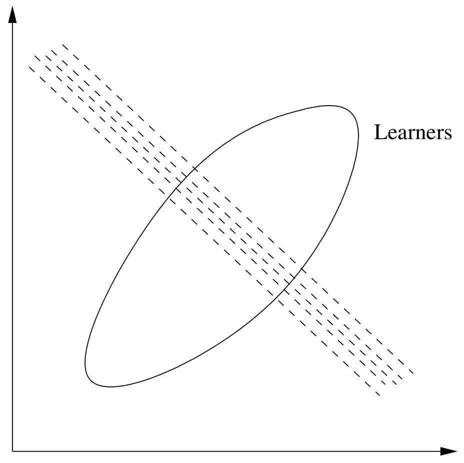


Figure 6.2: A point archive with a high average number of distinctions, with low variance.

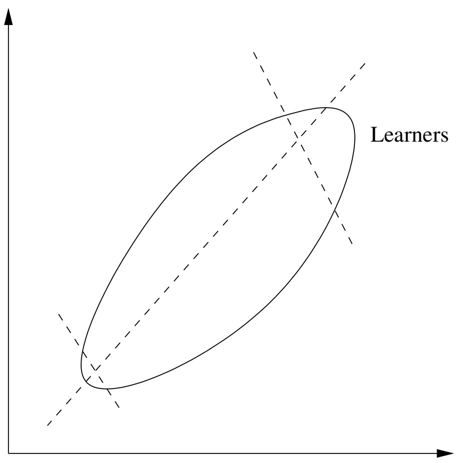


Figure 6.3: A point archive with a low average number of distinctions, with high variance.

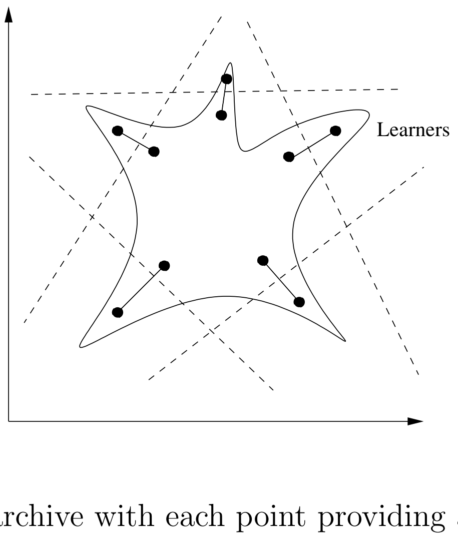


Figure 6.4: A point archive with each point providing a unique distinction.

Tables 6.1, 6.2, 6.3, 6.4, and 6.5 describe the compiled statistics in addition to the AA score for the 1-4-0, 1-2-0, and 1-1-0 parameter combinations for the balanced data sets, and the same for the 1-4-1, 1-2-1, and 1-1-1 combinations for the un-balanced data sets. The number of distinctions is normalized to range from zero to unity.

Table 6.1: Statistics on the number of distinctions made by the point archive under various pruning basis on Heart.

	1-4-0 (genotypic: (diversity enhancing)	1-2-0 (distinction based: greedy)	1-1-0 (distinction based: diversity enhancing)
AA score	0.752945	0.729383	0.740125
Average	0.185822	0.221552	0.207328
Minimum	0.071006	0.155325	0.102071
Maximum	0.250000	0.250000	0.250000
Std Dev	0.057508	0.027521	0.041681

Table 6.2: Statistics on the number of distinctions made by the point archive under various pruning basis on Liver.

	1-4-0 (genotypic: (diversity enhancing)	1-2-0 (distinction based: greedy)	1-1-0 (distinction based: diversity enhancing)
AA score	0.560484	0.516705	0.556740
Average	0.208975	0.226616	0.215863
Minimum	0.102071	0.155325	0.102071
Maximum	0.250000	0.250000	0.250000
Std Dev	0.039827	0.026241	0.037278

Intuitively, one would expect the greedy distinction based method (P2) to have a high average number of distinctions, with a low variance. The diversity-promoting distinction based method (P1) would have a lower average number of distinctions, with a higher variance. The genotypic pruning basis (P4), does not utilize the number of distinctions in its selection, therefore in terms of distinctions, the genotypic method would almost seem to perform a random selection, and thus would be expected to have an average number of distinctions and variance in between the values of the more extreme greedy and diversity based methods.

The results of the distinction analysis indicate that for the most part (ignoring the

Table 6.3: Statistics on the number of distinctions made by the point archive under various pruning basis on KDD04P.

	1-4-0 (genotypic: (diversity enhancing)	1-2-0 (distinction based: greedy)	1-1-0 (distinction based: diversity enhancing)
AA score	0.678974	0.577052	0.640372
Average	0.189406	0.233216	0.213473
Minimum	0.071006	0.177515	0.102071
Maximum	0.250000	0.250000	0.250000
Std Dev	0.055886	0.019461	0.040117

Table 6.4: Statistics on the number of distinctions made by the point archive under various pruning basis on Adult.

	1-4-1 (genotypic: (diversity enhancing)	1-2-1 (distinction based: greedy)	1-1-1 (distinction based: diversity enhancing)
AA score	0.736611	0.616963	0.636335
Average	0.187699	0.227526	0.217285
Minimum	0.071006	0.155325	0.130178
Maximum	0.250000	0.250000	0.250000
Std Dev	0.051113	0.023308	0.035922

KDD99 results), there seems to be a correlation between a high AA score and low average number of distinctions and a high variance.

The greedy distinction based method (P2) yielded an archive providing a high average number of distinctions, with low variance for each of the data sets. The diversity-enhancing distinction based method (P1) yielded an archive providing a medium average number of distinctions with a medium variance when compared to the P2 and P4 methods, for the most part (ignoring KDD99). The genotypic method (P4), yielded the converse of the greedy method; low average number of distinctions, high variance.

This is surprising, since it seems that the genotypic method yields a set of points with the distribution of distinctions being more “diversified” than the distinction based diversity enhancing method, and as such seems to be correlated to a high AA score. The correlation between a diversified set of points and a high AA score seems intuitive in the view that the point archive provides a varied subset of the training

Table 6.5: Statistics on the number of distinctions made by the point archive under various pruning basis on KDD99.

	1-4-1 (genotypic: (diversity enhancing)	1-2-1 (distinction based: greedy)	1-1-1 (distinction based: diversity enhancing)
AA score	0.918419	0.937040	0.907631
Average	0.184513	0.197827	0.178937
Minimum	0.071006	0.102071	0.036982
Maximum	0.248521	0.250000	0.250000
Std Dev	0.054012	0.044837	0.061621

data, exploring multiple aspects and sub-classes of the data, without over-focusing on a limited section.

The ability of the genotypic method to provide a more diversified set of distinctions than the nearest neighbour based (on distinctions) P1 method, without using the distinction data, may possibly be attributed to the genotypic data in fact being domain knowledge, as related to the No Free Lunch theorems (see Section 2.3, and [23])

Since each point must provide a new distinction to gain entry into the archive, the archive contents may be viewed as being pareto-equivalent in terms of distinctions. The use of the genotypic data and associated label, specifically using the class label for archive balance and preserving edges, provides an additional mechanism for ascertaining the "value" of a pareto-equivalent point. Using the heuristical notion that edges within the genotypic clusters of points are important and interesting, a bias to preserve them is introduced in the selection of a point for removal. As shown by the measured results, that bias yields a larger range of distinctions (high variance), than the nearest neighbour on hamming distance on distinctions method, and as such is correlated to a high AA score.

With regards to the KDD99 data set, all of the previously derived relationships seem reversed. A high variance in distinctions is correlated to a low AA score, and a high average number of distinctions to a high AA score. For that data set, the greedy distinction based method yielded the highest AA score, with the genotypic method being second, and diversity promoting being third. The placement of the genotypic method has shifted to being in between the greedy and diversity enhancing methods. That is, a medium average number of distinctions, and a medium variance, correlated to a medium AA score.

This is most likely data set dependent behaviour, in that a greedy distinction based approach is optimal. This may be due to the data set being relatively easy (nearly 80% of the data set represents the denial of service attack), in that variety within the point archive with respect to distinctions is not as pertinent, and as such the faster progressing greedy method would yield similar points providing a strong narrow learning gradient.

Interestingly enough, it seems that the genotypic selection basis, seems to respond to that change, and its resultant distribution of distinctions is more in accordance to the greedy method. This may also be related to the use of domain knowledge of the method, in that genotypically interesting points or boundary points, yield an archive with less diversity and a higher average number of distinctions than the diversity enhancing method.

In summary, the genotypic point pruning basis utilizes additional domain knowledge to yield an archive which is attuned to the input data set in terms of balancing diversity in distinctions and a high average thereof. Within the domain of the input data sets, the genotypic selection basis proved to be superior for the majority of the data sets, however it may be out-performed by heuristics which happen to correspond to data sets with specific biases.

Point Generation Method

In terms of point generation method, for the AA and TC methods it seems intuitive that a class balanced point generator used on unbalanced data would yield superior results as the learners would have equal access to both input types, especially considering that the performance measure is “score” which places equal weight on classification performance on both classes, as opposed to accuracy which may be influenced by degenerate cases yielding values mimicking the natural class distribution.

This is re-enforced by the work of Weiss and Provost [27] (See Section 2.1.2), which conclude that as general guidelines; the best learning class distribution when measuring accuracy tends to be near the natural distribution, and when measuring area under the ROC curve (similar to the "score" measure), the best balance tends to be near the balanced distribution.

Although understandable in the case of applying a class balance enforcing point generator upon unbalanced data, why would a purely uniform random point generator perform better on balanced data? Intuitively, the performance of the two generators on the balanced data should be identical.

This again may be related to the idea of an optimal class balance being somewhere between perfectly balanced and unbalanced. Once again, the experiments of Weiss and Provost measured the optimal training class balance for a variety of problems, with the optimum value varying with the data set, and not necessarily being the natural or balanced distribution.

In such a scenario, the natural distribution of the "balanced" data (50-55%) may be closer to the optimal distribution for learning for those data sets than an artificially balanced one. As such a purely random point generator would yield a class distribution mimicking the input class balance and thereby perform better than the one enforcing a perfectly equal balance.

Assuming the previously stated explanations are correct, then the optimal point generation scheme for balanced and unbalanced data on the G2 method is completely reversed; forcing a 50% balance on balanced data, and preserving the natural balance on the unbalanced data.

Again, the inherent complexity generated by the overlay of the learners behaviour upon the evolution results may be the cause of these results. In that the actual functions and programs of the learners themselves affect the results, not just the evolutionary steps and decisions. In other words, the G2 method applies a transformation to the results of the system, with the parameters being dependent upon the evolution results.

This unpredictable aspect of the G2 method may be observed in the relative rankings of the parameter combinations tested (See Tables 5.3, and 5.4, and Appendix C). When compared to the AA and TC methods, the best average rank value of the G2 method was worse. Even after splitting the data sets based on class balance, the G2 rank value is still relatively worse (tied AA on balanced data, and is worst on unbalanced). Leading to the belief that the performance of the G2 method is more related to the particularities of the input data set itself than a general relationship between the various archive pruning basis and even the input data class balance.

Therefore, until more detailed studies of the G2 method are completed and the mechanisms understood, the optimal G parameter values for the G2 method will be considered to be quirks of the small set of data sets utilized in the experiments.

6.1.2 Computational Complexity Verification

The observed time requirements of the various algorithms do correspond to the predictions made by the complexity analysis section. Using the derived parameter combination, the simplified PGPC algorithm's complexity is $O(gen * subset^4)$. When compared to the Regular algorithm's complexity of $O(gen * L_{pop} * Data_{training})$; for fixed subset and population sizes, the PGPC algorithm will out-perform the Regular algorithm on data sets with training data size greater than $subset^3$ (ignoring any constants in the complexity notation). This may be observed in the execution times of the KDD04P, Adult, and KDD99 sets.

6.1.3 Post-processing Methods

In providing a possible explanation for the varying results of the post-processing methods, it is worthwhile to consider the cases where the utilized methods would succeed or fail, as to understand their heuristical natures.

- In review; the TC method utilizes the Nearest Neighbour mapping as a basis for classification, and applies the evolved GP programs to in effect test for anomalies between the nearest training point, and input test point, as to gauge similarity.

  This method would work well on "nicely" clustered data points, where two nearest points would often have the same class, and be genotypically near each other, as to have GP programs which correctly classify one point, correctly classify the other. Furthermore, programs not specialized on the pertinent sub-class of points (poor classification score on them), would not be utilized, and as such would not weaken the correct vote distribution

  A drawback of the method would be data sets where the Nearest Neighbour points are far apart, as to produce distant GPout values, or have different classes (as in the case of points along cluster boundaries), or are "deceptive", in that a clustering of the training points is not indicative of the testing points. If such data sets coincide with degenerate or overly simplified GP behaviour, then the programs correctly classifying one point degenerately as one class, or weakly as in guessing the correct class with little underlying classification structure present, will apply the same behaviour to the other point, possibly mislabeling it.

  This means that there is a trade-off and associated risk with placing classification responsibility upon learners based on a Nearest Neighbour mapping. Learners not specialized to the pertinent sub-class of points may be beneficially removed, but at the same time pertinent desired learners may also be removed as well.

- The AA method, being a voting scheme over the pareto-front of learners has its success dependent upon the balance of votes cast. Assuming that there are sub-groups within the learners which are specialized to the input point (are desired as they would correctly classify it), destructive to the point (are specialized to an opposing point, and would consistently yield incorrect results on the input point), and ambivalent learners, which correctly classify the point with a uniform probability as they may be specialized on another sub-class of points, but are not necessarily destructive to the input point. Depending upon the distribution of the aforementioned classes, as dictated by the evolution over the input data set, a one vote per each learner scheme may vary in its results.

For example, assuming that the evolutionary process yields a pareto-front of learners specialized on the various sub-classes or aspects of the input data set, then depending upon the input data set, a majority-rules scheme may or may not prove successful. In a scenario where the data set and associated learners are fairly homogeneous, then the number of conflicting destructive classes would remain few, and the AA method would be beneficial. Conversely, if the input data space is diverse and complex, then a number of various sub-classes of data and associated learners would be produced, possibly conflicting with each other, and as such fail to produce a mutually agreed upon majority.

Consider the following examples, both in which it is assumed that the complexity of the evolved learners is limited to performing a linear partition of the search space into two classes. Defining the binary space of the AND operator may be considered relatively simple, and would yield a fairly homogeneous set of learners, with no conflicting or destructive sub-classes of learners. See Figure 6.5. Whereas the XOR space, cannot be defined with linear partitions without incurring a conflict between the two sub-classes of learners (the sub-class isolating state 0,0 from the rest conflicts with the sub-class isolating state 1,1 from the rest). See Figure 6.6. Within the domain of these two problems, the AA method would work well on the AND problem, however the votes cast by the two sub-groups defining the XOR problem would conflict and yield poor results.

- The G2 method builds a local membership function (LMF) on the *gpOut* values for each learner on the training data for each training data class. Given a test

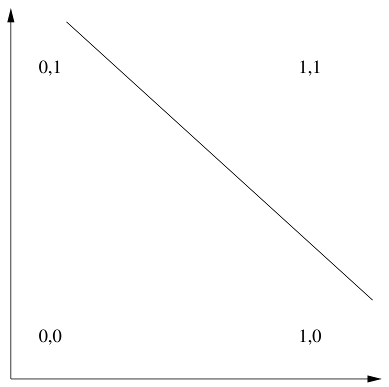


Figure 6.5: Learners restricted to linear partitions defining the AND binary space, with no conflicting or destructive sub-classes of learners.

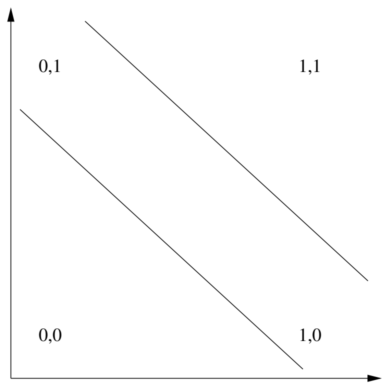


Figure 6.6: Learners restricted to linear partitions defining the XOR binary space, with conflicting sub-classes of learners.

data point, the *gpOut* value is computed on each LMF, and the learner with the highest difference between the memberships of the *gpOut* value on the two class' LMF is chosen as representative of the system, with the higher membership value defining the class.

If the pareto-front of learners is fairly diverse or specialized with regards to the data, possibly with many learned sub-classes of data, then the G2 method would utilize that diversity in building a diverse set of LMFs, and the learner best suited to that data point would be used (assuming the point to learner mapping is correct, and there is little overlap between the two gaussians in the learners, etc). Again, depending on the dataset and evolved learners, the G2 method may easily fail in that it may pick an outlier in regards to the diverse set of learners and base the entire classification on it. Or possibly the LMFs of the learners are "wide" and overlap, yielding a high probability of misclassification.

In summary, the effect upon classification score of the post-processing methods is dependent upon the input data, and the evolved set of learners.

With regards to the difference between the pareto-front approach of the PGPC algorithm, and the best-trained individual approach of the comparison algorithms, the inherent diversity and pareto-equivalence of the PGPC learner archive provides a set of specialized learners for the various sub-classes inherent in the data set. Whereas the best-trained approach, focuses upon a single generally well performing learner. Thereby, the population based approaches employed in the post-processing methods would have a stronger impact upon the PGPC results. Again, it may be possible for the best-trained method to yield an over-generalized learner, with specialized individuals residing in the population (possibly ancestors to the general learner). In such a scenario, a population based method may identify the specialized learner, and make use of them in the classification. Therefore, for the comparison algorithms, the BT method would generally work best, however it is possible for the TC and G2 methods to provide an improvement.

An interesting question warranting further research is to see if it is possible to predict within a measure of certainty, which post processing method would perform best based on the input data set features, in effect identifying and exploiting the data dependent aspects.

6.1.4 Maximum Archive Size Parameter Variation

The following sections analyze the impact of the maximum archive size parameters upon the performance of the PGPC algorithm.

Optimum Derived Parameter Combination

With regards to the optimum parameter combination derived from data generated by the variation of the maximum archive size parameters over the balanced data sets.

It may be observed that the combination is not too dissimilar from the original optimum combination derived from the fixed maximum archive sizes (25 per each archive).

The greedy learner pruning basis (L1) is still superior to the diversity enhancing one (L2). The top point pruning basis are still of the genotypic, Euclidean nearest neighbour, class aware set (P3, P4), and the only difference in terms of optimum point generation method was the switch to a purely random generator for the AA method, possibly having to do with the optimum learning class balance for the AA method being nearer to perfectly balanced over the varying archive sizes. Furthermore, the original parameter combinations still retain fairly high rankings in the generalized results (over the varying archives sizes), with the original 1-4-0 combination being second for AA, the 1-4-0 combination still being first for TC, and the 1-3-1 combination being 5th out of 20 for G2. It should be noted that again the G2 method exhibited a worse average rank value than the other post processing methods, re-enforcing the belief that the complexity of the G2 method inhibits the existence of a general optimal parameter combination suited for it.

Maximum Archive Size Terrains

With regards to the surface plots generated of classification score under the variation of maximum learner and point archive sizes. The low variance of the TC and G2 methods may be attributed to the additional post-processing and identification of specialized or pertinent learners. Even with a smaller archive of learners, if the post-processing methods manage to identify and utilize a specialized learner, additional archive members may be superfluous. By similar logic if the specialized learners to be identified are trained on a small set of points, additional points may be unnecessary, however they may still refine the accuracy of evaluation of the said learners, therefore additional points do have a higher value than additional learners.

Due to the additional complexity incurred by applying the TC and G2 post-processing methods upon the pareto-front of learners, the AA method will be utilized for the subsequent analysis as being indicative of the PGPC algorithm.

In the scenario where memory resources are unlimited, there would be no need for archive pruning, and as such the archives would consist of learners and points that have gained entry into their respective archives by providing classifications and distinctions which were either pareto-dominant or pareto-equivalent to the archives themselves; in a sense filling the archives with a pareto-based balanced approach.

In practice, the archive sizes grow excessively and must be limited via a pruning basis. Depending upon the specified maximum archive size, varying amounts of pruning are performed, that is for small archives, frequent pruning, and for larger archives, less pruning. It should be noted that a maximum possible archive size may be specified as the maximum size that is observed in an unbound archive case, as any additional archive spaces would be unutilized.

In between the extreme archives size cases, there exists a continuum of archive sizes, or in a different view, a continuum of amounts of a pruning basis applied. In the context of learners, under the derived optimal greedy pruning basis, a small archive size may be likened to the BT method as a small number of learners (or one) are greedily retained in the archive. At the opposing end of the spectrum, with little to no pruning, lies the pareto-based approach; diversified amongst multiple underlying objectives, but not necessarily balanced along the front, as if a diversity enhancing "niching" method was applied. In between, there exists a balance between the two, or a degree of applying the greedy pruning basis, and as such is the reason why the surface plots describing the PGPC score performance under various maximum learner archive sizes are not monotonic; there are "sweet spots" where the degree of applying the pruning basis corresponds to an efficient or optimal search strategy for that input data set, as in a balance between exploration and exploitation, or in terms of pruning basis; greedy and pareto-based (i.e., none).

The same logic may be applied to the point archive; intuitively, more visible points would allow for a learner to be evaluated upon a larger section of the input problem, and as such provide a more accurate measure of fitness over the said problem. However, limiting the number of visible points allows for a learner to "focus upon" that subset of the input problem, possibly generating specialized learners attuned to that subset. Once again, there exists an inherent trade-off between the two dictated by the point archive size; a trade-off between learner "generality" and "specialization". It should be noted that the point pruning basis is not directly related to the classification-space search strategy of the algorithm (balance between exploration and exploitation), but is more related to the subset selection and retention strategy, where the point pruning basis dictates the strategy, and the point archive size dictates the amount thereof.

Furthermore, interactions between the two archives complicate matters, including the combination of the various pruning basis and amounts thereof. Since the archives depend upon each other for fitness measurement and inclusion decisions; the contents of one affects the other. It very well may be that a certain amount of learner greedy selection is most optimal in conjunction with a certain amount of the point subset selection strategy for a certain input data set, and as such yielding complex and data set dependent behaviour, as in various peaks and valleys, and general trends within the said surface plots.

The presence of such complexities attributed to the archive size combinations per input data set may explain some of the derived optimal pruning basis results. In that no one pruning parameter combination was optimal for all of the data sets, but grouping the sets by common features allowed for the identification of pruning basis optimized to the data sets. Furthermore, all of the previous results were based on an equal balance in terms of size between the learner and point archives, with the said archive sizes being a fixed value. As shown by the surface plots, if the archive sizes are varied, and not necessarily in a increasing fashion, then the performance of the pruning combination is affected, and therefore possibly providing similar performance to the derived optimal combination using another pruning basis.

All of the previous conjectures tie into the No Free Lunch (NFL) theorems described in Section 2.3, that in addition to stating that all search algorithms exhibit equal performance over all input problem instances, imply that for specific singular instances there exists an optimal search strategy, and identifying the said strategy or group of similar strategies requires the use of problem domain knowledge. In reference to the PGPC algorithm, the NFL theorems support the optimality of the varying amounts of the pruning basis or search strategy on a per input problem basis, in addition to re-enforcing the benefits of utilizing the genotypic training point data and associated class for point archive pruning in conjunction with the distinction based archive entry criterion, as the point data and the heuristically valued cluster boundary points constitute a segment of domain knowledge.

In summary, the pareto-based search strategy commenced by attempting to avoid the selection of an explicit search strategy (explore/exploit) by the algorithm, by in effect exploring all search avenues (sections of the pareto-front). It should be noted that the learner and point generators or breeding operations may be thought of as defining a search strategy, as in either favouring mutation or recombination. However the monotonic guarantees of the pareto-based algorithms only require that all possible individuals are generated with non-zero probability. In effect, the learner and point generation strategies define the search strategy, while the pareto-based section of the algorithm focuses upon identifying and storing pertinent data, in addition to “guiding” the generators towards the current pareto-fronts and observed learning gradients, as such providing a heuristic for the search direction.

In the scenario where memory resources are limited, the pareto-based section must make a decision in terms of a basis for removing a pareto-equivalent individual, in effect moving away from a search strategy semi-neutral role, towards having to explicitly define a strategy via the pruning basis and amounts thereof. Although seemingly violating the pareto-based concepts of retaining equivalently important information, the pruning basis defined search strategy in fact operates in conjunction with the learner and point generator strategies. Although points and learners may be lost and possibly re-visited under pruning, the deviation away from a pareto-balanced search approach is not as significant since the overlayed pruning based search strategy may be considered a part of the generator strategy. For example, applying a greedy pruning basis upon a balanced exploration-exploitation generator may be considered equivalent to performing an unbound archive pareto-based search using a more greedy generator, with the addition that points and learners may be re-visited during the search.

Therefore, the PGPC algorithm allows for a memory efficient search at the expense of possibly re-visiting previously found learners and points. Furthermore, additional control over the search strategy is located in the pruning basis selection, and amount thereof specified by the archive sizes (in addition to the initial generator based strategy), with the capability for domain knowledge such as training point data and label being incorporated.

In conclusion to this section, the PGPC algorithm is not a universal efficient search algorithm (as per NFL), however when practically compared to other contemporary algorithms (using the derived parameter combination / search strategy), it shows comparable and often superior performance when evaluated upon practical input data sets.

6.2 Further Research

Additional experimentation may involve the development and evaluation of alternate archive insertion criteria, pruning basis, and pareto-front evaluation methods. Furthermore, additional experimentation on an increased and more varied set of input data sets would strengthen any optimal derived combination based on data set class balance, and possibly help identify new decision criteria for new optimal parameter combinations. Using a larger number of data sets, it may be possible to search for a set of data set features which may be indicative of the optimum post-processing method; that is in effect search for a classification of data set features, with the output classes being the predicted optimum post-processing method.

Aspects worthy of exploration mentioned in the discussion section include: performing more generations on more data sets as to observe instances where the greedy learner pruning basis becomes trapped in a local optimum, and the observed lower scoring diversity enhancing approach would out-perform it (if it manages to escape the optimum). A more detailed analysis of the effects of genotypic information as it relates to point distinctions in a pruning context, namely why the genotypic approach moved to a distinction distribution closer to the optimal approach per the various data sets; what information is the genotypic approach using? An analysis of the optimal training class balance for the various data sets; performed similarly to the work of Weiss and Provost [27] (See Section 2.1.2), in that, artificially varying the training data distributions, and observing the classification performance outcomes as to identify an optimal balance per data set. A detailed study of the G2 method, including its optimal training class balance, and an understanding of its on-line workings as to understand the effects of archive pruning and the other steps taken during the evolutionary phase upon its results.

Finally, an analysis of the resultant point and learner archives per data sets may be performed to possibly gain insight into the data set itself, and the path taken by the learners during their evolution over it. This may lead to the identification of various sub-classes within the data set and evolved learners, and thus increase the amount of information about the problem domain to the algorithm user.

Chapter 7

Conclusion

An algorithm employing the coevolution of both classifiers and training data subset members within a Genetic Programming environment was presented. The various archive pruning basis and point generation methods were evaluated, with the highest scoring parameter combination used for each pareto-front post processing method. Comparisons were made to a traditional GP classifier employing the training data in its entirety, in addition to classifiers using only a subset of the data selected via either a random, cycling, or dynamic subset selection (DSS) method.

With regards to classification performance, the PGPC algorithm consistently out-performed the comparison Random and Cycling algorithms, and often out-performed the Regular, and DSS algorithms, under the effects of all of the post-processing methods. Thus indicating that the PGPC algorithm is relatively accurate.

With regards to execution time efficiency, training point subset selection provides a dramatic increase in execution speed, yielding an algorithm that is relatively efficient. The algorithm does however take longer to execute than the Random and Cycling algorithms since their subset selection methods are much simpler. Furthermore, the PGPC algorithm only starts to become more time efficient than the comparison algorithms on larger data sets (thousands rather than hundreds of training exemplars).

With respect to the stated objectives of the PGPC algorithm outlined in the introduction (See Section 1), the PGPC algorithm meets and often exceeds the traditional GP classification algorithm's performance while utilizing less time. It should be noted that time efficiency is relatively constant (independent of data set size), so a performance gain against the Regular algorithm is observed on sufficiently large sets. With regards to the second stated objective, the PGPC algorithm exceeds the comparison subset selection algorithm's performance. Furthermore, discounting the overly simple Random and Cycling algorithms, PGPC runs faster than the DSS algorithm, again on sufficiently large sets. Thus allowing for the conclusion that the PGPC algorithm is a practically attractive classification algorithm, providing both superior classification performance and run-time efficiency, and as such is worthy of consideration in applications requiring the accurate classification of large data sets.

Although competitive on the utilized input data sets, the parameter sensitivity analysis of the maximum archive size values indicate that there exists local optima in the selection of the maximum archive sizes as related to score. Indicating that the exhibited performance of the PGPC algorithm may be related to the arbitrary decisions made in the selection of maximum archive sizes. As such, for other data sets, performance may vary as an optimal archive size parameter pair would be unknown, furthermore the observed optimal pruning basis - point generation methods may no longer hold. This possibility is interpreted as stemming from the No Free Lunch theorems, which state that over all input problem instances, search (in the PGPC domain; of the classification space) performance is equivalent to enumeration. As such no one algorithm (search basis) may be consistently superior to others.

Bearing that limitation in mind, the arbitrarily selected maximum archive sizes used in the experimentation proved capable of yielding an algorithm that out-performs the contemporary approaches on real-world data sets, without any claims of being optimal. Furthermore, the observed variance of classification score as related to the maximum archive sizes was not dramatically significant. Finally, due to the run-time efficiency of the PGPC algorithm; if maximum classification performance is desired, many instances may be evaluated in the same time as alloted to one Regular algorithm instance, and an optimal parameter basis - maximum archive size (limited by available memory and time) may be selected. All in all, making use of the multi-objective evolutionary framework applied to both classifiers and data point subset elements.

Bibliography

[1] Robert Curry and Malcom I. Heywood. Towards efficient training on large datasets for genetic programming. *17th Conference of the Canadian Society for Computational Studies of Intelligence*, 3060:161–174, 17-19 May 2004.

[2] Edwin D. de Jong. The incremental pareto-coevolution archive. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 525–536. Springer, 2004.

[3] Edwin D. de Jong. Intransitivity in coevolution. In Xin Yao, Edmund K. Burke, José Antonio Lozano, Jim Smith, Juan J. Merelo Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiño, Ata Kabán, and Hans-Paul Schwefel, editors, *PPSN*, volume 3242 of *Lecture Notes in Computer Science*, pages 843–851. Springer, 2004.

[4] K. Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Marc Schoenauer, K. Deb, G. Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 849–858, Berlin, 2000. Springer.

[5] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. Ensemble techniques for parallel genetic programming based classifiers. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward P. K. Tsang, Riccardo Poli, and Ernesto Costa, editors, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP)*, volume 2610 of *Lecture Notes in Computer Science*, pages 59–69. Springer, 2003.

[6] Jerome H. Freidman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[7] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *PPSN*, volume 866 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 1994.

[8] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub. Co., 1989.

[9] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, 1994. IEEE Service Center.

[10] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

[11] E.D. De Jong. Towards a bounded pareto-coevolution archive. In *Proceedings of the Congress on Evolutionary Computation CEC-04*, volume 2, pages 2341–2348, 2004.

[12] E.D. De Jong and J.B. Pollack. Multi-objective methods for tree size and control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, 2003.

[13] E.D. De Jong and J.B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[14] H. Juille and J.B. Pollack. Massively parallel genetic programming. In P.J. Angeline and K.E. Kinnear, editors, *Advances in Genetic Programming, 2nd ed.*, pages 339–358. MIT Press, Cambridge, MA, USA, 1996.

[15] Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 98–105, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[16] John R. Koza. *Genetic Programming II.* Cambridge, Mass.:MIT Press, 1994.

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

[18] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000.

[19] Sheng Ma and Chuanyi Ji. Performance and efficiency: recent advances in supervised learning. *Proceedings of the IEEE*, 87(9):1519–1535, Sep 1999.

[20] Jason Noble and Richard A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 493–500, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[21] P. Nordin. A compiling genetic programming system that directly manipulates the machine code. In K.E. Kinnear, editor, *Advances in Genetic Programming.*, pages 311–334. MIT Press, Cambridge, MA, USA, 1994.

[22] P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior.*, 5(2):107–140, 1996.

[23] Nicholas J. Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 275–291. Springer, 1995.

[24] 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.

[25] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[26] Richard A. Watson and Jordan B. Pollack. Coevolutionary dynamics in a minimal substrate. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 702–709, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[27] Gary M. Weiss and Foster J. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Artif. Intell. Res. (JAIR)*, 19:315–354, 2003.

[28] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[29] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Appendix A

Computational Complexity Analysis

A.1 Evolution Complexity

The initial storage requirements of the algorithm are:

- The input training data of size $O(Data_{training})$.
- The point population and set of identified useful points, each of size $O(P_{pop})$.
- The point archive of size $O(P_{arch})$.
- The learner population of size $O(L_{pop})$.
- The learner archive of size $O(L_{arch})$.

It shall be assumed that the storage requirements per point, learner, and outcome between the two will be considered constant (or within a constant factor).

To initialize the populations prior to the first generation requires a traversal of each population element. Point (index) random initialization requires constant time, and it will be assumed that learner (tree) initialization will require the same, yielding a time complexity of $O(T_{pop} + L_{pop})$.

The following steps will be performed at each generation, with the number of generations being denoted as *gen*:

- Removing the dominated learners from the learner archive requires the evaluation (play) of every learner in the archive against every point in the point archive. The storage requirements of the outcomes array are simply $O(L_{arch} * P_{arch})$, and assuming that evaluation of a point occurs in constant time, the time complexity for this step would be the same.

To identify dominated learners, each learner pair in the archive is compared over the results against the point archive, yielding a time complexity of $O(L_{arch}^2 * P_{arch})$.

A final pass over the learner archive to remove marked dominated learners may be performed in linear time, yielding a final memory requirement for this step of $O(L_{arch} * P_{arch})$, and an overall complexity of $O(L_{arch}^2 * P_{arch})$.

- The generation of a new population of points requires a linear traversal of the points, $O(P_{pop})$.
- The generation of a new population of learners requires that fitness of each learner in the learner population be evaluated with respect to the point population and archive. Requiring $O(L_{pop} * (P_{pop} + P_{arch}))$ time. It will be assumed that the subsequent breeding operations may be performed in linear time.
- The identification of useful points requires the identification of population learners which are dominated by the learner archive, and their behaviour under the singular inclusion of each population point to the point archive. To aid efficiency, the outcomes of each population learner against the point archive may be cached and re-utilized in the evaluation of each point population member, in addition to the outcomes of the learner archive against the point archive. This requires $O(L_{arch} * P_{arch})$ space and time.

  For each learner in the learner population:

  - The learner is played against the point archive, requiring $O(P_{arch})$ time, and space for the caching of the results.
  - The outcomes of the learner are compared against the outcomes of the learner archive, and domination is tested for, requiring $O(L_{arch} * P_{arch})$ time.
  - For each of the generated points in the point population, the learner and learner archive are evaluated against the point, requiring $O(L_{arch} + 1)$ time per generated point.

A test for non-domination of the learner by the learner archive over the results against the point archive and added point requires $O(L_{arch} * (P_{arch} + 1))$ time per generated point.

Overall, $O(P_{pop} * (L_{arch} * P_{arch}))$ time is required.

The removal of duplicately selected points may be performed in linear time $O(P_{pop})$.

Over all learners in the learner population, $O(L_{pop} * P_{pop} * L_{arch} * P_{arch})$ time is required. With $O(L_{arch} * P_{arch})$ storage for the caching.

- The union of identified useful points (of maximum size P_{pop}), to the point archive may require the selection of an archive member for removal, with the selection being dependent upon the pruning basis parameter.

  The following describes the complexities associated with each pruning basis per point to be inserted.

  - P0: Finding one of the two closest points in the archive using a brute force method selected for simplicity on small archive sizes requires $O(P_{arch}^2)$ time.
  - P1: Computing the distinctions of each point in the archive against the learner archive requires the evaluation of outcomes and conversion into distinctions between each pair of learners. This requires $O(L_{arch} * P_{arch})$ time and space for the outcomes, and $O(L_{arch}^2 * P_{arch})$ time and space for the associated distinctions.

    To find the two closest sets of distinctions (again using a simple brute force method), requires $O(P_{arch}^2 * L_{arch}^2)$ time.
  - P2: This method is similar to P1, except that instead of finding the two closest sets of distinctions, only the worst set must be found, requiring a linear scan. Yielding memory requirements of $O(L_{arch}^2 * P_{arch})$, and a time complexity of $O(P_{arch} * L_{arch}^2)$.

  - P3,P4: Both are similar to P1, except that the class value of the pair of points is considered in the decision of "closest". Requiring $O(P_{arch}^2)$ time.

  Yielding the following point pruning basis dependent costs (per point to be inserted):

  P:0,3,4: Time: $P_{basis} = O(P_{arch})$, Storage: $P_{basis} = O(1)$.
  P:1: Time: $P_{basis} = O(P_{arch} * L_{arch}^2)$, Storage: $P_{basis} = O(L_{arch}^2 * P_{arch})$.
  P:2: Time: $P_{basis} = O(L_{arch}^2)$, Storage: $P_{basis} = O(L_{arch}^2 * P_{arch})$.
  With $O(P_{arch})$ being common to all of the time complexities.

  Yielding final complexities of $O(P_{pop} * P_{arch} * P_{basis})$ time, and $O(P_{basis})$ space.

- Identifying the useful learners in the learner population requires the following steps per each learner:
  - To see if the learner is undefeated by the point archive, requires $O(P_{arch})$ time.
  - To see if any point in the point population defeats an undefeated learner requires $O(P_{pop})$ time.
  - To insert a identified undefeated-learner defeating point into the point archive (and possibly each point in the point archive is identified), requires $O(P_{pop} * P_{arch} * P_{basis})$ time, and $O(P_{basis})$ space.
  - To identify if a learner is useful, once again requires $O(L_{arch} * P_{arch})$ time,
  - The costs of inserting an identified useful learner into the archive depends upon the learner archive pruning method:
    * L0: Computing the distance between each learner pair over the outcomes against the point archive requires $O(L_{arch}^2 * P_{arch})$ time.
    * L1: Finding the worst performing learner archive member requires the evaluation of each learner against the point archive, and a linear scan of the results, incurring $O(L_{arch} * P_{arch})$ time.

  Removing common elements yields the following learner pruning basis costs:

L0: Time: $L_{basis} = O(L_{arch})$
L1: Time: $L_{basis} = O(1)$

With $O(L_{arch} * P_{arch})$ being common to both time complexities, and each requiring $O(L_{arch} * P_{arch})$ space to store the outcomes.

Over all learners, the relevant complexities are $O(L_{pop} * P + arch * ((P_{pop} * P_{basis}) + (L_{arch} * L_{basis})))$ time, and $O(P_{basis} + (L_{arch} * P_{arch})$ space.

- To remove any duplicates from the point and learner archives requires a pairwise comparison of each archive member. Incurring $O(P^2_{arch} + L^2_{arch})$ time.

Combining all the previously described complexities, yields:

Time complexity:	
Learner pruning basis:	
If L = 0	$L_{basis} = O(L_{arch})$
If L = 1	$L_{basis} = O(1)$
Point pruning basis:	
If P = 0,3,4	$P_{basis} = O(P_{arch})$
If P = 1	$P_{basis} = O(L^2_{arch} + P_{arch})$
If P = 2	$P_{basis} = O(L^2_{arch})$
Time complexity =	$O(gen*$ $((L^2_{arch} * P_{arch})+$ $(L_{pop} * P_{pop} * L_{arch} * P_{arch})+$ $(L_{pop} * P_{pop} * P_{arch} * P_{basis})+$ $(L_{pop} * P_{arch} * L_{arch} * L_{basis})+$ $P^2_{arch}))$

Storage complexity:	
Point pruning basis:	
If P = 0,3,4	$P_{basis} = O(1)$
If P = 1,2	$P_{basis} = O(L^2_{arch} + P_{arch})$
Storage complexity =	$O(Data_{training} + P_{pop} + L_{pop}+$ $(L_{arch} * P_{arch}) + P_{basis})$

If a simplification is allowed, and the population and archive sizes of both of the learners and points are set to a common value (denoted *subset*), then the complexities collapse to:

Time complexity:	
Learner pruning basis:	
If L = 0	$L_{basis} = O(subset)$
If L = 1	$L_{basis} = O(1)$
Point pruning basis:	
If P = 0,3,4	$P_{basis} = O(subset)$
If P = 1,2	$P_{basis} = O(subset^2)$
Time complexity =	$O(gen * (subset^4 + (subset^3 * P_{basis}) + (subset^3 * L_{basis})))$

At worst $O(gen * subset^5)$, and at best $O(gen * subset^4)$. With the storage complexity being $O(Data_{training} + subset^2)$; independent of the pruning basis.

A.2 Post-processing Complexity

The following describes the computational and storage costs of performing each of the post processing methods.

- **AA:**

  Evaluating each testing data point on each learner archive member to gain a set of votes requires $O(L_{arch} * Data_{testing})$ time.

- **TC:**

  In the context of utilizing a kd-tree data structure [6] to precompute the nearest neighbour mapping of the set of test points to the static set of training points, the construction of the kd-tree requires $O(Data_{dimension} * Data_{training} * \log Data_{training})$ time, and to query all the testing points requires $O(Data_{testing} * \log Data_{training})$, yielding a sum of $O(((Data_{dimension} * Data_{training}) + Data_{testing}) * \log Data_{training})$. With the resultant mapping requiring $O(Data_{testing})$ storage.

  Testing each learner archive member for success against the nearest training point of each testing point requires $O(L_{arch} * Data_{testing})$ time, with evaluating the testing point having the same complexity. Yielding $O((((Data_{dimension} * Data_{training}) + Data_{testing}) * \log Data_{training}) + (L_{arch} * Data_{testing}))$.

- **G2:**
  Computing the two Local Membership Function gaussians for each archive member over all the training data requires $O(L_{arch} * Data_{training})$ time, and computing the membership values on each learners gaussian for each testing point requires $O(L_{arch} * Data_{testing})$ time. Yielding a total of $O(L_{arch} * Data_{entirety})$.

Appendix B

Parameter Combination Results

The following tables describe the classification performance of the PGPC algorithm per each possible parameter combination (L-P-G) instance for each input data set, ranked by score, and grouped by post-processing method. With each set of instance values being the median of 30 independent runs. All numeric values range from zero to unity, with high accuracy, detection rate, and score being desirable.

B.1 PGPC - AA

Table B.1: The classification results of the various parameter combinations, sorted by AA score over the evolution on Heart.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-3-0	0.776316	0.871795	0.297297	0.772349
1-0-0	0.750000	0.641026	0.108108	0.755024
1-4-0	0.750000	0.589744	0.081081	0.752945
1-4-1	0.736842	0.589744	0.081081	0.740818
1-1-0	0.736842	0.615385	0.135135	0.740125
1-0-1	0.736842	0.641026	0.135135	0.737353
1-2-0	0.723684	0.589744	0.108108	0.729383
1-1-1	0.723684	0.692308	0.162162	0.725225
1-3-1	0.710526	0.538462	0.081081	0.716563
1-2-1	0.684211	0.769231	0.243243	0.691615
0-4-0	0.618421	0.538462	0.270270	0.619889
0-2-0	0.578947	0.512821	0.351351	0.586279
0-0-0	0.578947	0.461538	0.297297	0.577963
0-0-1	0.565789	0.461538	0.270270	0.572765
0-1-1	0.565789	0.435897	0.297297	0.571379
0-4-1	0.565789	0.461538	0.324324	0.568607
0-2-1	0.565789	0.487179	0.351351	0.567914
0-3-1	0.565789	0.487179	0.351351	0.565835
0-3-0	0.552632	0.461538	0.378378	0.553015
0-1-0	0.539474	0.461538	0.324324	0.540194

Table B.2: The classification results of the various parameter combinations, sorted by AA score over the evolution on Liver.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-4-0	0.517241	0.709677	0.589286	0.560484
1-4-1	0.586207	0.580645	0.392857	0.558180
1-3-0	0.655172	0.193548	0.053571	0.557604
1-1-0	0.574713	0.580645	0.357143	0.556740
1-0-1	0.528736	0.612903	0.482143	0.555300
1-1-1	0.494253	0.677419	0.589286	0.553283
1-3-1	0.643678	0.354839	0.178571	0.548099
1-2-1	0.540230	0.709677	0.553571	0.546371
1-0-0	0.643678	0.161290	0.035714	0.539459
0-4-1	0.505747	0.451613	0.464286	0.520449
0-0-0	0.528736	0.548387	0.428571	0.519297
1-2-0	0.482759	0.548387	0.571429	0.516705
0-0-1	0.528736	0.451613	0.410714	0.513537
0-2-1	0.471264	0.548387	0.517857	0.507776
0-1-0	0.540230	0.419355	0.428571	0.504608
0-4-0	0.540230	0.387097	0.375000	0.496544
0-2-0	0.494253	0.483871	0.482143	0.495968
0-3-1	0.517241	0.419355	0.500000	0.495392
0-3-0	0.482759	0.483871	0.464286	0.484735
0-1-1	0.471264	0.451613	0.571429	0.483583

Table B.3: The classification results of the various parameter combinations, sorted by AA score over the evolution on KDD04P.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-3-1	0.716632	0.691657	0.243068	0.715684
1-0-1	0.698417	0.642684	0.223599	0.697970
1-4-0	0.681099	0.520556	0.164012	0.678974
1-0-0	0.680203	0.696493	0.261947	0.678301
1-4-1	0.661093	0.477630	0.157522	0.658786
1-3-0	0.650941	0.669287	0.272566	0.649005
1-2-1	0.644969	0.594317	0.225959	0.643179
1-1-0	0.642580	0.546554	0.256637	0.640372
1-1-1	0.621379	0.543531	0.204130	0.625323
1-2-0	0.573305	0.490931	0.323304	0.577052
0-2-1	0.553598	0.415961	0.325074	0.550127
0-1-1	0.550911	0.382709	0.271976	0.547823
0-2-0	0.540161	0.389964	0.317994	0.536561
0-3-0	0.538668	0.269649	0.219469	0.536095
0-3-1	0.537772	0.331923	0.219469	0.534518
0-4-0	0.535085	0.348851	0.282006	0.532791
0-0-1	0.531203	0.345828	0.251917	0.527294
0-0-0	0.528815	0.318017	0.261357	0.526664
0-4-1	0.517468	0.301088	0.290855	0.513842
0-1-0	0.505225	0.339782	0.330973	0.502289

Table B.4: The classification results of the various parameter combinations, sorted by AA score over the evolution on Adult.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-4-1	0.662038	0.599625	0.105984	0.736611
1-3-1	0.642579	0.558837	0.091925	0.703301
1-0-1	0.687953	0.623887	0.137347	0.700835
1-4-0	0.686007	0.661275	0.274333	0.689353
1-1-1	0.709800	0.694327	0.339942	0.636335
1-3-0	0.761277	0.919128	0.652487	0.627510
1-2-1	0.637095	0.648500	0.294160	0.616963
1-1-0	0.756589	0.931317	0.724225	0.575404
1-0-0	0.763135	0.965190	0.851117	0.548014
1-2-0	0.751636	0.931903	0.843547	0.547062
0-1-1	0.468778	0.401313	0.288032	0.536462
0-2-0	0.477976	0.450188	0.416366	0.533882
0-0-0	0.477976	0.439991	0.411319	0.532180
0-4-0	0.474527	0.442921	0.372747	0.531547
0-3-0	0.477180	0.445617	0.379957	0.528306
0-4-1	0.442862	0.376817	0.310743	0.523133
0-2-1	0.469131	0.390178	0.374910	0.522908
0-0-1	0.458164	0.392171	0.390771	0.512532
0-1-0	0.466301	0.418776	0.409517	0.511916
0-3-1	0.433044	0.366854	0.326604	0.506530

Table B.5: The classification results of the various parameter combinations, sorted by AA score over the evolution on KDD99.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-2-1	0.915377	0.900102	0.012972	0.937040
1-4-1	0.916467	0.914581	0.045039	0.918419
1-3-1	0.890483	0.868881	0.017098	0.908362
1-1-1	0.874889	0.850505	0.007080	0.907631
1-0-1	0.888065	0.868881	0.021438	0.891421
1-4-0	0.908368	0.923869	0.164131	0.871691
1-0-0	0.903385	0.918834	0.158305	0.867321
1-3-0	0.905941	0.918834	0.158305	0.867321
1-1-0	0.906922	0.926708	0.235559	0.842670
1-2-0	0.890110	0.910284	0.145448	0.838649
0-2-1	0.824890	0.826610	0.078443	0.836417
0-0-1	0.851280	0.868437	0.174396	0.821866
0-3-1	0.833173	0.833774	0.199696	0.821866
0-4-1	0.883165	0.910153	0.116699	0.815658
0-3-0	0.768825	0.761300	0.133929	0.792549
0-2-0	0.819466	0.856965	0.156324	0.762122
0-0-0	0.735808	0.738252	0.149954	0.746831
0-1-1	0.711553	0.684658	0.156291	0.746695
0-4-0	0.625657	0.619835	0.213246	0.651262
0-1-0	0.466080	0.373225	0.151390	0.595650

B.2 PGPC - TC

Table B.6: The classification results of the various parameter combinations, sorted by score over the evolution on Heart, using TC.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-3-0	0.736842	0.769231	0.270270	0.735274
1-0-0	0.723684	0.589744	0.135135	0.727997
1-1-0	0.710526	0.615385	0.162162	0.715870
1-4-0	0.710526	0.564103	0.081081	0.715870
1-4-1	0.710526	0.538462	0.081081	0.715870
1-0-1	0.710526	0.589744	0.135135	0.714484
1-2-0	0.710526	0.589744	0.162162	0.713791
1-3-1	0.697368	0.512821	0.108108	0.702356
1-1-1	0.684211	0.589744	0.162162	0.690229
1-2-1	0.684211	0.615385	0.243243	0.685378
0-4-0	0.644737	0.538462	0.216216	0.648302
0-1-1	0.631579	0.487179	0.216216	0.634789
0-2-0	0.618421	0.487179	0.243243	0.624740
0-0-1	0.618421	0.461538	0.216216	0.623354
0-4-1	0.618421	0.487179	0.216216	0.623354
0-2-1	0.618421	0.487179	0.243243	0.619889
0-0-0	0.605263	0.487179	0.216216	0.611920
0-3-1	0.605263	0.461538	0.243243	0.609841
0-1-0	0.592105	0.435897	0.243243	0.597020
0-3-0	0.592105	0.461538	0.270270	0.596327

Table B.7: The classification results of the various parameter combinations, sorted by score over the evolution on LIVER, using TC.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-0-0	0.620690	0.419355	0.285714	0.575749
1-3-1	0.620690	0.419355	0.285714	0.575749
0-0-0	0.609195	0.419355	0.285714	0.566820
0-0-1	0.609195	0.419355	0.285714	0.566820
0-1-0	0.609195	0.419355	0.285714	0.566820
0-1-1	0.609195	0.419355	0.285714	0.566820
0-2-0	0.609195	0.419355	0.285714	0.566820
0-2-1	0.609195	0.419355	0.285714	0.566820
0-3-0	0.609195	0.419355	0.285714	0.566820
0-3-1	0.609195	0.419355	0.285714	0.566820
0-4-0	0.609195	0.419355	0.285714	0.566820
0-4-1	0.609195	0.419355	0.285714	0.566820
1-0-1	0.609195	0.419355	0.285714	0.566820
1-1-0	0.609195	0.419355	0.285714	0.566820
1-1-1	0.609195	0.419355	0.285714	0.566820
1-2-0	0.609195	0.419355	0.285714	0.566820
1-2-1	0.609195	0.419355	0.285714	0.566820
1-3-0	0.609195	0.419355	0.285714	0.566820
1-4-0	0.609195	0.419355	0.285714	0.566820
1-4-1	0.609195	0.419355	0.285714	0.566820

Table B.8: The classification results of the various parameter combinations, sorted by score over the evolution on KDD04P, using TC.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-4-0	0.715736	0.714631	0.284366	0.715788
1-3-1	0.713049	0.717654	0.288496	0.713068
1-4-1	0.711854	0.702539	0.280826	0.711647
1-0-1	0.708570	0.718259	0.300295	0.708577
1-0-0	0.708271	0.727328	0.305015	0.708221
1-3-0	0.706181	0.730351	0.319764	0.706503
1-1-0	0.702299	0.717654	0.313864	0.702434
1-2-1	0.702299	0.703748	0.302065	0.702200
1-1-1	0.701403	0.709794	0.303245	0.701783
1-2-0	0.696327	0.706167	0.312684	0.696593
0-1-1	0.693938	0.698307	0.313864	0.694013
0-2-1	0.693341	0.698307	0.312094	0.693328
0-2-0	0.692744	0.697703	0.316224	0.692716
0-3-1	0.691848	0.695889	0.309145	0.691861
0-1-0	0.690953	0.691657	0.313864	0.690732
0-4-1	0.690654	0.694075	0.311504	0.690695
0-4-0	0.690654	0.695284	0.308555	0.690629
0-0-1	0.690057	0.695889	0.312684	0.690157
0-0-0	0.690057	0.696493	0.313864	0.690113
0-3-0	0.689460	0.691052	0.309145	0.689450

Table B.9: The classification results of the various parameter combinations, sorted by score over the evolution on ADULT, using TC.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-4-1	0.709358	0.704993	0.251983	0.715597
1-3-1	0.713161	0.707220	0.276857	0.701809
1-0-1	0.722448	0.722457	0.344268	0.696505
1-4-0	0.729170	0.769105	0.388609	0.689479
1-3-0	0.778613	0.883966	0.541817	0.670431
1-1-1	0.744560	0.790905	0.412040	0.668971
1-2-1	0.715815	0.766643	0.423937	0.659135
1-0-0	0.773837	0.900961	0.596611	0.648382
1-1-0	0.759950	0.879981	0.570296	0.646813
1-2-0	0.749514	0.858298	0.580389	0.642527
0-4-0	0.660800	0.695030	0.453136	0.622030
0-2-1	0.630816	0.655063	0.439798	0.620972
0-4-1	0.643021	0.662447	0.437635	0.619495
0-3-1	0.628958	0.645218	0.432588	0.618031
0-2-0	0.649567	0.681552	0.460707	0.616524
0-1-1	0.654520	0.678739	0.431867	0.616256
0-3-0	0.642137	0.679208	0.450252	0.615076
0-1-0	0.655227	0.690577	0.468277	0.612317
0-0-0	0.660534	0.688350	0.468998	0.612228
0-0-1	0.643906	0.690342	0.453136	0.610557

Table B.10: The classification results of the various parameter combinations, sorted by score over the evolution on KDD99, using TC.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-4-0	0.927669	0.911898	0.006618	0.952296
1-4-1	0.926904	0.910544	0.006024	0.952254
1-0-0	0.927852	0.912125	0.007344	0.952188
1-3-0	0.927852	0.912125	0.007344	0.952188
0-2-1	0.926836	0.910496	0.006354	0.951999
1-1-1	0.925749	0.909346	0.005050	0.951881
1-1-0	0.928241	0.913123	0.007361	0.951841
1-3-1	0.925794	0.909182	0.005149	0.951816
0-4-1	0.926319	0.910380	0.005925	0.951763
1-0-1	0.925495	0.908675	0.005067	0.951713
0-0-0	0.927061	0.912009	0.008384	0.951661
0-3-1	0.926187	0.910280	0.006800	0.951630
0-0-1	0.926286	0.910540	0.007460	0.951622
0-1-1	0.926061	0.909813	0.006932	0.951611
1-2-0	0.926531	0.910967	0.006981	0.951602
0-3-0	0.927338	0.911894	0.008318	0.951595
1-2-1	0.925052	0.908340	0.005133	0.951589
0-1-0	0.926531	0.910899	0.006981	0.951398
0-4-0	0.926855	0.911211	0.009143	0.951347
0-2-0	0.926531	0.910967	0.007839	0.951194

B.3 PGPC - G2

Table B.11: The classification results of the various parameter combinations, sorted by score over the evolution on Heart, using G2.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-3-0	0.815789	0.871795	0.216216	0.815662
1-0-0	0.815789	0.923077	0.243243	0.812197
1-2-0	0.815789	0.846154	0.189189	0.812197
1-4-0	0.815789	0.871795	0.243243	0.812197
1-3-1	0.802632	0.820513	0.189189	0.802148
1-1-0	0.802632	0.871795	0.270270	0.798683
0-3-0	0.789474	0.820513	0.243243	0.786556
1-1-1	0.789474	0.871795	0.243243	0.785863
1-0-1	0.789474	0.871795	0.270270	0.784477
0-1-1	0.776316	0.769231	0.216216	0.778586
1-4-1	0.776316	0.820513	0.243243	0.777893
0-0-1	0.776316	0.794872	0.216216	0.777200
0-1-0	0.776316	0.820513	0.243243	0.775121
0-4-1	0.776316	0.769231	0.243243	0.775121
0-2-0	0.776316	0.820513	0.243243	0.774428
0-3-1	0.776316	0.794872	0.243243	0.774428
1-2-1	0.776316	0.820513	0.243243	0.774428
0-2-1	0.776316	0.794872	0.270270	0.773735
0-0-0	0.776316	0.794872	0.270270	0.772349
0-4-0	0.763158	0.846154	0.270270	0.762301

Table B.12: The classification results of the various parameter combinations, sorted by score over the evolution on LIVER, using G2.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-0-1	0.643678	0.290323	0.107143	0.561060
1-2-1	0.632184	0.193548	0.089286	0.546947
1-3-1	0.643678	0.225806	0.142857	0.543203
1-0-0	0.620690	0.258065	0.142857	0.541475
1-2-0	0.643678	0.193548	0.125000	0.541475
0-2-1	0.632184	0.161290	0.107143	0.536002
1-4-1	0.643678	0.193548	0.142857	0.536002
1-4-0	0.632184	0.225806	0.142857	0.530530
0-3-1	0.620690	0.193548	0.125000	0.528802
1-3-0	0.620690	0.193548	0.142857	0.528802
0-4-1	0.643678	0.161290	0.107143	0.527074
1-1-1	0.632184	0.161290	0.125000	0.525346
0-0-0	0.620690	0.161290	0.089286	0.521601
0-2-0	0.632184	0.129032	0.089286	0.521601
0-4-0	0.632184	0.193548	0.089286	0.520161
0-0-1	0.632184	0.129032	0.089286	0.519873
0-1-0	0.632184	0.161290	0.107143	0.514401
1-1-0	0.632184	0.161290	0.125000	0.514401
0-1-1	0.632184	0.161290	0.089286	0.509217
0-3-0	0.632184	0.161290	0.071429	0.507200

Table B.13: The classification results of the various parameter combinations, sorted by score over the evolution on KDD04P, using G2.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-3-1	0.756644	0.769045	0.253097	0.756794
1-1-0	0.755748	0.769649	0.252507	0.756084
1-4-1	0.754852	0.772068	0.254277	0.755060
1-0-1	0.753956	0.762394	0.254277	0.754015
1-1-1	0.753359	0.767836	0.256637	0.753585
1-0-0	0.752762	0.772068	0.257227	0.752849
1-4-0	0.751866	0.770859	0.258997	0.752418
1-3-0	0.752165	0.764813	0.260177	0.752274
1-2-1	0.750373	0.771463	0.257817	0.750409
1-2-0	0.745297	0.757557	0.253687	0.745306
0-2-1	0.744999	0.760580	0.272566	0.744660
0-2-0	0.743207	0.766626	0.269027	0.743717
0-0-1	0.741714	0.767836	0.263717	0.741818
0-3-1	0.739922	0.761185	0.264307	0.740282
0-4-1	0.737832	0.733374	0.264897	0.737691
0-1-1	0.736339	0.758162	0.283186	0.736932
0-0-0	0.736041	0.753930	0.267257	0.736418
0-3-0	0.733950	0.739420	0.264897	0.734680
0-1-0	0.729173	0.732164	0.264897	0.728716
0-4-0	0.718423	0.756953	0.303245	0.719460

Table B.14: The classification results of the various parameter combinations, sorted by score over the evolution on ADULT, using G2.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
0-1-1	0.701751	0.646507	0.136986	0.754565
0-0-0	0.701574	0.653774	0.143115	0.750456
0-1-0	0.703078	0.654712	0.144557	0.749135
1-4-0	0.711746	0.668425	0.169430	0.747856
0-4-1	0.704582	0.655063	0.143475	0.746304
1-1-0	0.699540	0.650375	0.149603	0.746192
0-2-1	0.702901	0.650141	0.142394	0.742529
1-2-1	0.704582	0.662447	0.165826	0.742394
0-4-0	0.705378	0.660103	0.144196	0.742061
1-1-1	0.709358	0.677098	0.203677	0.740655
1-0-1	0.709358	0.666198	0.158616	0.739437
0-2-0	0.701574	0.647445	0.162942	0.738853
1-3-1	0.714134	0.690108	0.213050	0.737705
1-3-0	0.706793	0.671824	0.195386	0.737307
1-4-1	0.720325	0.708158	0.256309	0.735775
1-0-0	0.705466	0.663268	0.215213	0.734791
1-2-0	0.705555	0.657993	0.250541	0.734101
0-3-1	0.707058	0.662564	0.197188	0.732688
0-0-1	0.708827	0.692686	0.228190	0.729927
0-3-0	0.705289	0.679442	0.220620	0.729923

Table B.15: The classification results of the various parameter combinations, sorted by score over the evolution on KDD99, using G2.

Combination (L-P-G)	Accuracy	Detection Rate	FP Rate	Score
1-0-0	0.813985	0.828563	0.026786	0.873234
1-3-0	0.813985	0.828563	0.026786	0.873234
1-4-0	0.805374	0.852833	0.034906	0.863319
0-2-0	0.829845	0.814104	0.028766	0.855288
1-2-0	0.843371	0.852066	0.038041	0.855288
1-1-0	0.791045	0.750707	0.031308	0.853925
0-4-1	0.805204	0.809472	0.052284	0.849924
0-4-0	0.810168	0.825736	0.065916	0.849401
0-1-1	0.808130	0.837703	0.126221	0.846791
0-3-1	0.778814	0.754740	0.023369	0.846087
0-3-0	0.764285	0.748547	0.029311	0.842908
0-0-0	0.761523	0.740385	0.029311	0.838164
1-4-1	0.784730	0.773938	0.037447	0.837814
1-1-1	0.810268	0.872806	0.160252	0.835870
1-3-1	0.848075	0.903021	0.111071	0.832807
1-0-1	0.805198	0.813278	0.111071	0.829793
0-2-1	0.798497	0.829489	0.090094	0.829483
1-2-1	0.815997	0.872762	0.141850	0.828574
0-0-1	0.752887	0.711435	0.023369	0.825642
0-1-0	0.782981	0.794307	0.065916	0.817055

Appendix C

Parameter Combination Tournament Rankings

The following tables summarize the relative ranks of the various PGPC parameter combination (L-P-G) scores over the various data sets, grouped by post-processing method. Instances of two parameter combinations having the same numeric score value are resolved by giving them both the same rank value. In addition to the ranks, average rank values are computed for each parameter combination per: the balanced data sets (Heart, Liver, KDD04P), unbalanced data sets (Adult, KDD99), and entire set of data sets. Again, the score values utilized in the computation of the ranks are derived from the median of 30 independent runs.

Table C.1: Tournament rankings of the various parameter combinations (L-P-G) using AA.

(L-P-G)	Rank on Heart	Rank on Liver	Rank on KDD04P	Average rank	Rank on Adult	Rank on KDD99	Average rank	Average entire rank
0-0-0	13	11	18	14.00	13	15	14.00	14.00
0-0-1	14	13	17	14.67	18	11	14.50	14.60
0-1-0	20	15	20	18.33	19	18	18.50	18.40
0-1-1	15	20	12	15.67	11	16	13.50	14.80
0-2-0	12	17	13	14.00	12	14	13.00	13.60
0-2-1	17	14	11	14.00	17	10	13.50	13.80
0-3-0	19	19	14	17.33	15	13	14.00	16.00
0-3-1	18	18	15	17.00	20	11	15.50	16.40
0-4-0	11	16	16	14.33	14	17	15.50	14.80
0-4-1	16	10	19	15.00	16	12	14.00	14.60
1-0-0	2	9	4	5.00	9	7	8.00	6.20
1-0-1	6	5	2	4.33	3	5	4.00	4.20
1-1-0	5	4	8	5.67	8	8	8.00	6.60
1-1-1	8	6	9	7.67	5	4	4.50	6.40
1-2-0	7	12	10	9.67	10	9	9.50	9.60
1-2-1	10	8	7	8.33	7	1	4.00	6.60
1-3-0	1	3	6	3.33	6	7	6.50	4.60
1-3-1	9	7	1	5.67	2	3	2.50	4.40
1-4-0	3	1	3	2.33	4	6	5.00	3.40
1-4-1	4	2	5	3.67	1	2	1.50	2.80

Table C.2: Tournament rankings of the various parameter combinations (L-P-G) using TC.

(L-P-G)	Rank on Heart	Rank on Liver	Rank on KDD04P	Average rank	Rank on Adult	Rank on KDD99	Average rank	Average entire rank
0-0-0	14	2	19	11.67	19	10	14.50	12.80
0-0-1	12	2	18	10.67	20	12	16.00	12.80
0-1-0	16	2	15	11.00	18	17	17.50	13.60
0-1-1	10	2	11	7.67	16	13	14.50	10.40
0-2-0	11	2	13	8.67	15	19	17.00	12.00
0-2-1	13	2	12	9.00	12	4	8.00	8.60
0-3-0	17	2	20	13.00	17	15	16.00	14.20
0-3-1	15	2	14	10.33	14	11	12.50	11.20
0-4-0	9	2	17	9.33	11	18	14.50	11.40
0-4-1	12	2	16	10.00	13	8	10.50	10.20
1-0-0	2	1	5	2.67	8	3	5.50	3.80
1-0-1	4	2	4	3.33	3	9	6.00	4.40
1-1-0	3	2	7	4.00	9	6	7.50	5.40
1-1-1	7	2	9	6.00	6	5	5.50	5.80
1-2-0	5	2	10	5.67	10	14	12.00	8.20
1-2-1	8	2	8	6.00	7	16	11.50	8.20
1-3-0	1	2	6	3.00	5	3	4.00	3.40
1-3-1	6	1	2	3.00	2	7	4.50	3.60
1-4-0	3	2	1	2.00	4	1	2.50	2.20
1-4-1	3	2	3	2.67	1	2	1.50	2.20

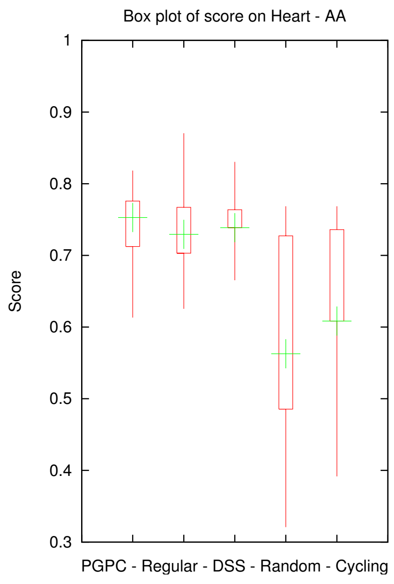
Table C.3: Tournament rankings of the various parameter combinations (L-P-G) using G2.

(L-P-G)	Rank on Heart	Rank on Liver	Rank on KDD04P	Average rank	Rank on Adult	Rank on KDD99	Average rank	Average entire rank
0-0-0	14	10	17	13.67	2	10	6.00	10.60
0-0-1	10	12	13	11.67	19	17	18.00	14.20
0-1-0	11	13	19	14.33	3	18	10.50	12.80
0-1-1	8	14	16	12.67	1	7	4.00	9.20
0-2-0	12	10	12	11.33	12	3	7.50	9.80
0-2-1	13	5	11	9.67	7	15	11.00	10.20
0-3-0	5	15	18	12.67	20	9	14.50	13.40
0-3-1	12	7	14	11.00	18	8	13.00	11.80
0-4-0	16	11	20	15.67	9	6	7.50	12.40
0-4-1	11	8	15	11.33	5	5	5.00	8.80
1-0-0	2	4	6	4.00	16	1	8.50	5.80
1-0-1	7	1	4	4.00	11	14	12.50	7.40
1-1-0	4	13	2	6.33	6	4	5.00	5.80
1-1-1	6	9	5	6.67	10	12	11.00	8.40
1-2-0	2	4	10	5.33	17	3	10.00	7.20
1-2-1	12	2	9	7.67	8	16	12.00	9.40
1-3-0	1	7	8	5.33	14	1	7.50	6.20
1-3-1	3	3	1	2.33	13	13	13.00	6.60
1-4-0	2	6	7	5.00	4	2	3.00	4.20
1-4-1	9	5	3	5.67	15	11	13.00	8.60

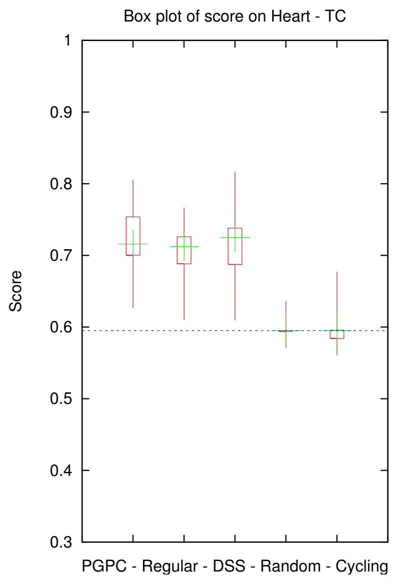
Appendix D

Classification Score Quartile Results

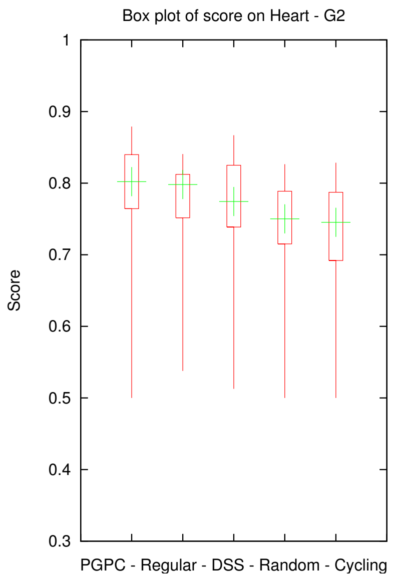
The following box plots illustrate the minimum, first quartile, median, third quartile, and maximum observed scores of the various algorithms across the various post-processing methods, for each of the input data sets.



(a) AA

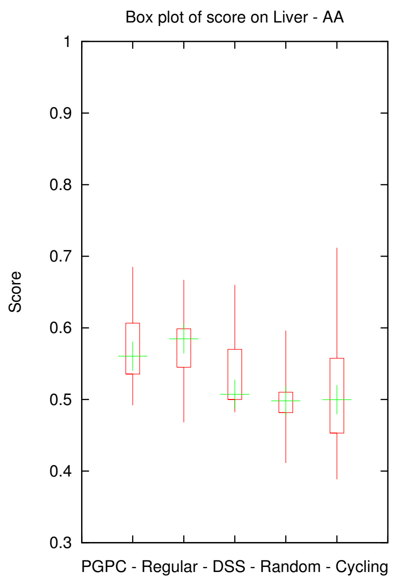


(b) TC

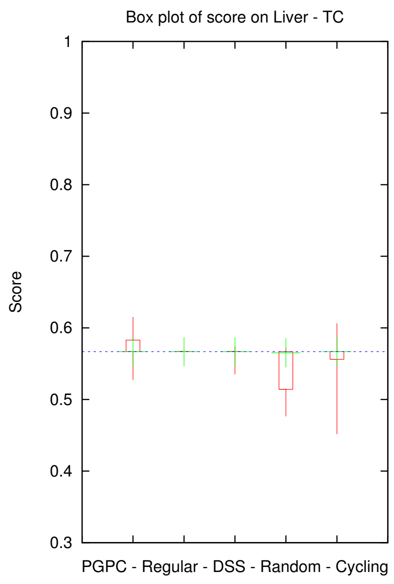


(c) G2

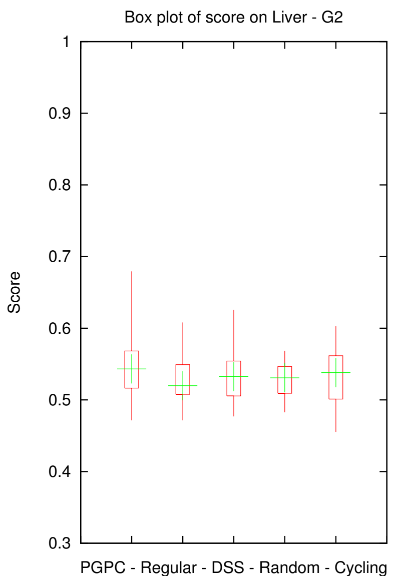
Figure D.1: Classification score on Heart by the various algorithms under various post-processing methods. The NN score is shown as a horizontal line in the TC graph.



(a) AA

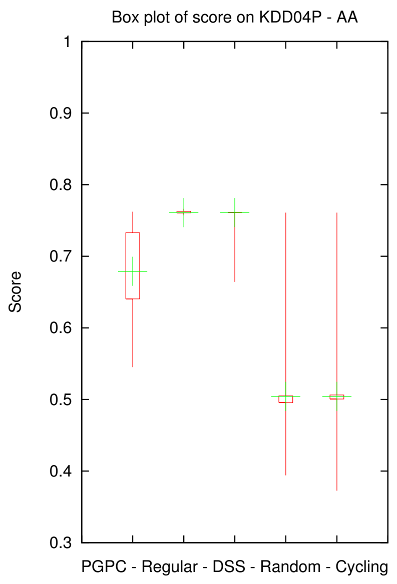


(b) TC

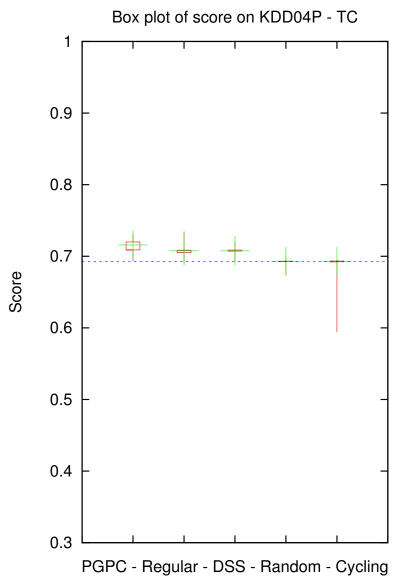


(c) G2

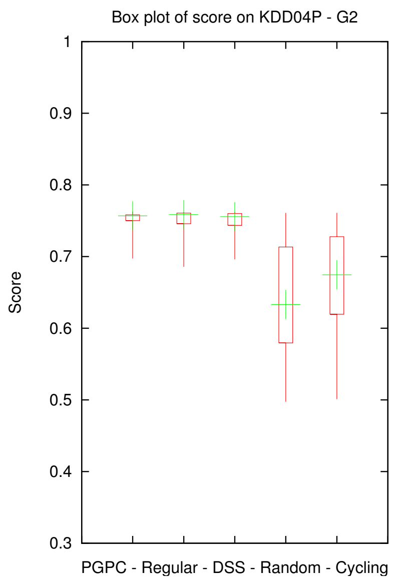
Figure D.2: Classification score on Liver by the various algorithms under various post-processing methods. The NN score is shown as a horizontal line in the TC graph.



(a) AA

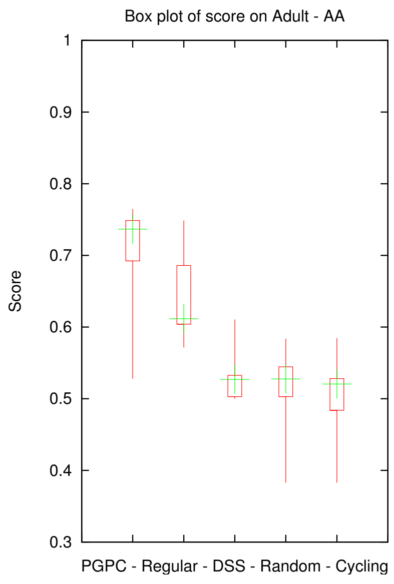


(b) TC

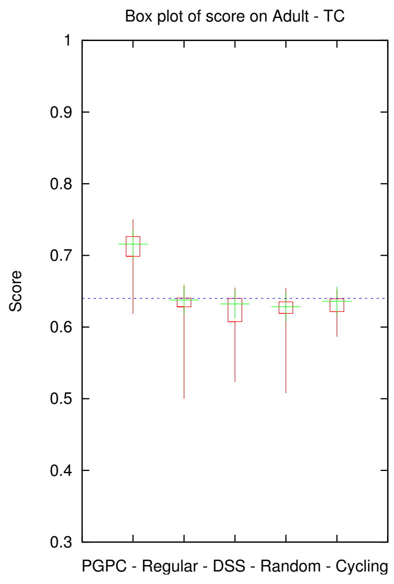


(c) G2

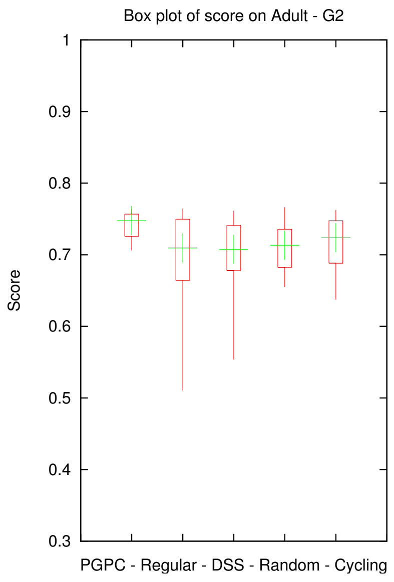
Figure D.3: Classification score on KDD04P by the various algorithms under various post-processing methods. The NN score is shown as a horizontal line in the TC graph.



(a) AA

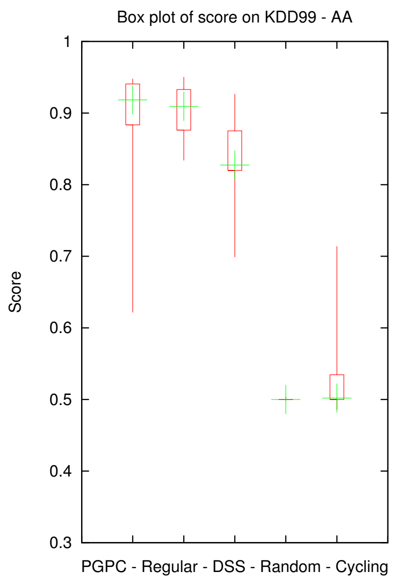


(b) TC

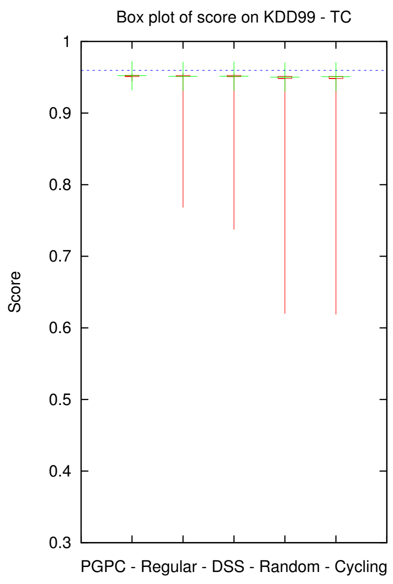


(c) G2

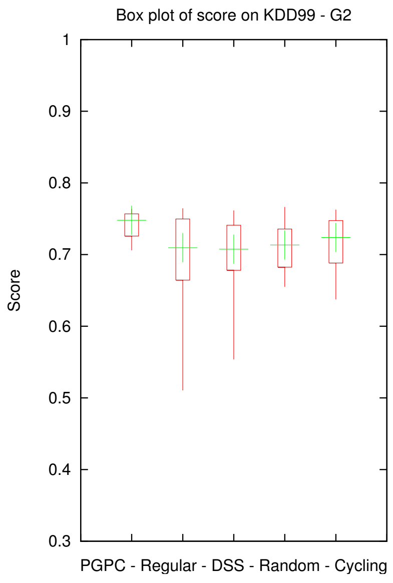
Figure D.4: Classification score on Adult by the various algorithms under various post-processing methods. The NN score is shown as a horizontal line in the TC graph.



(a) AA



(b) TC



(c) G2

Figure D.5: Classification score on KDD99 by the various algorithms under various post-processing methods. The NN score is shown as a horizontal line in the TC graph.

Appendix E

Run-time Quartile Results

The following tables describe the quartile results of the measured run-times of the various algorithms grouped by the post-processing method used for each of the input data sets.

Table E.1: Quartile results of run-time of the post processing instances of the various algorithms on the Heart dataset (in seconds).

Algorithm	Minimum	Q1	Median	Q3	Maximum
PGPC - AA	26.06	32.98	35.42	43.87	79.23
Regular - BT	3.81	7.44	9.46	14.25	47.86
DSS - BT	1.05	1.40	1.77	2.31	4.18
Random - BT	2.22	3.14	3.89	4.16	6.55
Cycling - BT	2.08	2.80	3.56	3.89	5.77
PGPC - TC	26.06	32.97	35.43	43.86	79.22
Regular - TC	3.82	7.45	9.46	14.26	47.92
DSS - TC	1.06	1.41	1.79	2.32	4.19
Random - TC	2.22	3.15	3.90	4.18	6.56
Cycling - TC	2.11	2.81	3.57	3.92	5.81
PGPC - G2	28.75	36.80	39.34	45.07	69.30
Regular - G2	3.82	7.47	9.49	14.28	48.02
DSS - G2	1.06	1.44	1.84	2.34	4.24
Random - G2	2.23	3.19	3.94	4.30	6.64
Cycling - G2	2.20	2.83	3.65	4.03	5.96

Table E.2: Quartile results of run-time of the post processing instances of the various algorithms on the Liver dataset (in seconds).

Algorithm	Minimum	Q1	Median	Q3	Maximum
PGPC - AA	28.48	44.56	54.67	66.65	80.87
Regular - BT	7.68	14.37	15.52	19.55	32.94
DSS - BT	2.18	2.43	2.58	2.81	4.61
Random - BT	1.47	1.65	1.88	1.98	2.33
Cycling - BT	1.23	1.56	1.69	1.89	2.79
PGPC - TC	28.48	44.57	54.68	66.64	80.86
Regular - TC	7.69	14.38	15.52	19.56	32.96
DSS - TC	2.18	2.44	2.59	2.82	4.64
Random - TC	1.47	1.66	1.88	1.99	2.34
Cycling - TC	1.23	1.56	1.70	1.90	2.80
PGPC - G2	27.21	38.67	43.43	52.99	79.86
Regular - G2	7.72	14.42	15.56	19.61	33.05
DSS - G2	2.22	2.49	2.64	2.86	4.74
Random - G2	1.50	1.70	1.91	2.01	2.38
Cycling - G2	1.26	1.58	1.73	1.93	2.84

Table E.3: Quartile results of run-time of the post processing instances of the various algorithms on the KDD04P dataset (in seconds).

Algorithm	Minimum	Q1	Median	Q3	Maximum
PGPC - AA	34.57	38.17	40.79	46.50	58.66
Regular - BT	247.36	289.11	354.00	549.28	1167.65
DSS - BT	3.71	3.83	4.16	4.72	11.66
Random - BT	2.70	3.00	3.16	3.29	3.82
Cycling - BT	2.27	2.43	2.60	2.94	3.41
PGPC - TC	34.63	38.25	40.90	46.59	58.77
Regular - TC	247.63	289.34	354.32	549.95	1169.79
DSS - TC	3.95	4.13	4.50	5.42	12.69
Random - TC	3.10	3.45	3.68	3.85	4.46
Cycling - TC	2.72	2.90	3.13	3.44	4.00
PGPC - G2	32.93	38.05	40.63	47.21	62.22
Regular - G2	248.65	290.29	355.66	552.40	1177.23
DSS - G2	4.88	5.22	5.63	7.39	16.47
Random - G2	4.61	4.94	5.95	6.33	8.72
Cycling - G2	4.14	4.62	5.01	5.73	6.74

Table E.4: Quartile results of run-time of the post processing instances of the various algorithms on the Adult dataset (in seconds).

Algorithm	Minimum	Q1	Median	Q3	Maximum
PGPC - AA	28.84	33.06	41.38	47.68	70.48
Regular - BT	1376.29	1788.85	1973.74	2297.53	2858.88
DSS - BT	10.41	10.82	11.29	11.36	12.74
Random - BT	2.88	3.39	3.63	3.96	5.23
Cycling - BT	2.45	3.01	3.46	3.83	4.86
PGPC - TC	29.06	33.27	41.68	47.92	71.02
Regular - TC	1378.16	1790.21	1975.35	2298.85	2862.85
DSS - TC	11.65	12.67	13.34	14.15	17.51
Random - TC	4.12	4.73	5.56	6.05	8.36
Cycling - TC	3.87	5.09	5.65	6.54	10.62
PGPC - G2	32.11	39.35	41.57	44.98	66.48
Regular - G2	1384.66	1794.99	1981.17	2315.14	2877.01
DSS - G2	14.91	18.27	20.40	22.89	37.96
Random - G2	6.81	8.90	11.57	13.53	25.25
Cycling - G2	7.85	10.00	12.89	15.62	35.07

Table E.5: Quartile results of run-time of the post processing instances of the various algorithms on the KDD99 dataset (in seconds).

Algorithm	Minimum	Q1	Median	Q3	Maximum
PGPC - AA	35.99	47.58	56.97	84.13	290.41
Regular - BT	33446.82	36863.62	40347.74	45640.54	63097.62
DSS - BT	117.76	119.59	120.87	122.82	128.56
Random - BT	2.24	3.36	4.20	5.00	9.49
Cycling - BT	1.34	1.71	2.58	3.62	7.70
PGPC - TC	35.57	46.87	57.21	81.67	282.94
Regular - TC	33468.75	36913.43	40392.51	45733.63	63177.45
DSS - TC	144.07	158.89	166.27	185.02	250.66
Random - TC	8.91	12.76	20.96	27.69	64.68
Cycling - TC	7.76	9.57	17.77	27.35	48.64
PGPC - G2	80.17	104.41	122.69	140.79	210.37
Regular - G2	33559.53	37102.36	40574.29	46094.44	63452.75
DSS - G2	242.26	306.73	343.84	414.72	651.70
Random - G2	63.75	85.35	155.87	242.96	498.69
Cycling - G2	57.28	73.72	140.45	210.94	420.06

Appendix F

Maximum Archive Size Parameter Variation Results

The following tables describe statistics on the score values recorded under variation of the maximum learner and point archive sizes, for each PGPG parameter combination instance (L-P-G), over all the data sets, sorted by the median value, and grouped by post-processing method.

The learner and point archive sizes ranged from {10,25,50}, and {5,10,25,50,100} respectively, with each score value per parameter combination - maximum learner archive size - maximum point archive size being the median of 30 independent runs.

Due to the large amounts of data required for the generation of these statistics, only the smaller data sets were evaluated over all of the parameter combinations. The larger sets were only evaluated using their optimal derived parameter combinations.

F.1 PGPC - AA

Table F.1: Statistics on score of Heart using AA, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-3-0	0.743682	0.742204	0.715177	0.792100	0.023385
1-0-0	0.729915	0.741511	0.686071	0.776507	0.026949
1-4-0	0.726403	0.741511	0.686764	0.779972	0.030426
1-1-0	0.718850	0.731116	0.658351	0.755717	0.032417
1-2-0	0.721945	0.729383	0.690229	0.755024	0.023044
1-4-1	0.700670	0.703049	0.626126	0.765766	0.044167
1-3-1	0.707184	0.703049	0.663895	0.777893	0.035735
1-0-1	0.700393	0.701663	0.626819	0.765766	0.044158
1-2-1	0.691707	0.691615	0.638254	0.765766	0.038087
1-1-1	0.696512	0.690922	0.639986	0.765766	0.038684
0-4-0	0.636198	0.619889	0.561677	0.725918	0.053912
0-1-0	0.621830	0.604990	0.540194	0.713791	0.054689
0-1-1	0.600208	0.601525	0.527374	0.679141	0.050700
0-2-0	0.631924	0.599099	0.540887	0.703742	0.057610
0-0-1	0.605406	0.598753	0.527374	0.701663	0.052886
0-3-1	0.608501	0.598753	0.527374	0.684685	0.056143
0-4-1	0.606838	0.598753	0.527374	0.697505	0.056455
0-2-1	0.602357	0.598753	0.527374	0.683992	0.049279
0-0-0	0.638831	0.597713	0.551629	0.739432	0.065766
0-3-0	0.633634	0.597713	0.553015	0.741511	0.071044

Table F.2: Statistics on score of Liver using AA, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-3-1	0.543702	0.561060	0.494240	0.590438	0.033048
1-4-0	0.548714	0.560484	0.508929	0.574309	0.023318
1-1-1	0.545814	0.553283	0.499712	0.590438	0.028349
1-1-0	0.541782	0.550979	0.499136	0.578341	0.026524
1-0-0	0.546544	0.550979	0.511809	0.597350	0.025071
1-3-0	0.545929	0.550979	0.518145	0.569988	0.019944
1-0-1	0.542492	0.548963	0.500000	0.590438	0.031480
1-2-0	0.537730	0.546659	0.505184	0.569988	0.025523
1-4-1	0.549539	0.544355	0.512097	0.585541	0.024628
1-2-1	0.534312	0.537442	0.487615	0.585541	0.035750
0-4-1	0.512404	0.512097	0.485023	0.542627	0.015674
0-0-0	0.509197	0.511809	0.475806	0.540035	0.020444
0-1-1	0.507546	0.510081	0.483583	0.553283	0.017142
0-3-1	0.507297	0.508353	0.492800	0.528514	0.012217
0-0-1	0.507181	0.508065	0.492800	0.525922	0.010218
0-2-0	0.509140	0.505472	0.475806	0.548675	0.022192
0-2-1	0.500634	0.504608	0.485023	0.519585	0.010874
0-3-0	0.503399	0.504320	0.475806	0.529954	0.016446
0-1-0	0.503975	0.504320	0.468318	0.547235	0.019742
0-4-0	0.503091	0.502592	0.474366	0.545219	0.019563

Table F.3: Statistics on score of KDD04P using AA, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-3-1	0.674469	0.711644	0.577793	0.747442	0.068165
1-4-1	0.682041	0.696137	0.593283	0.751194	0.057425
1-3-0	0.672925	0.685466	0.581442	0.748580	0.064172
1-4-0	0.676394	0.678974	0.592073	0.749483	0.059412
1-0-0	0.677561	0.678301	0.596967	0.753839	0.058955
1-0-1	0.681241	0.673189	0.615207	0.752939	0.053947
1-1-0	0.649448	0.640372	0.530962	0.742839	0.075961
1-2-1	0.658870	0.636360	0.570243	0.747442	0.054044
1-1-1	0.666050	0.631952	0.583004	0.747735	0.060525
1-2-0	0.650264	0.620635	0.577052	0.742839	0.062296
0-2-0	0.564720	0.572206	0.516009	0.609109	0.034438
0-2-1	0.571754	0.555432	0.519136	0.639758	0.042729
0-1-1	0.560402	0.553430	0.519134	0.616099	0.031899
0-0-1	0.571828	0.553430	0.523388	0.629727	0.039858
0-0-0	0.566184	0.552867	0.512954	0.624739	0.041066
0-1-0	0.546459	0.549227	0.500418	0.601576	0.026240
0-4-1	0.562150	0.542562	0.513842	0.632096	0.041545
0-3-0	0.553696	0.538452	0.505068	0.627733	0.036115
0-4-0	0.557605	0.537287	0.509141	0.625631	0.040523
0-3-1	0.554806	0.534518	0.519136	0.608888	0.034355

Table F.4: Statistics on score of Adult using AA, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-1	0.701225	0.733106	0.621430	0.757726	0.051585
1-4-0	0.573114	0.564850	0.505578	0.689353	0.064577

Table F.5: Statistics on score of KDD99 using AA, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-0	0.876722	0.908509	0.794240	0.923470	0.049162
1-4-1	0.804480	0.807886	0.708080	0.871691	0.052990

F.2 PGPC - TC

Table F.6: Statistics on score of Heart using TC, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-3-0	0.717510	0.726611	0.675329	0.752252	0.024323
1-0-0	0.706537	0.713791	0.676022	0.729383	0.021271
1-2-0	0.696997	0.704435	0.651074	0.742897	0.030984
1-4-0	0.703557	0.704435	0.662509	0.742897	0.026262
1-1-0	0.697020	0.702356	0.660430	0.728690	0.026032
1-0-1	0.683622	0.676715	0.648995	0.727997	0.024315
1-4-1	0.677616	0.676715	0.634096	0.717256	0.027274
1-3-1	0.677501	0.675329	0.625433	0.726611	0.029606
1-2-1	0.677293	0.672557	0.656965	0.717256	0.017958
1-1-1	0.678194	0.672557	0.650381	0.717256	0.020496
0-4-0	0.645484	0.646916	0.608455	0.675329	0.023527
0-0-0	0.648048	0.636868	0.607069	0.689536	0.030907
0-1-0	0.642851	0.636868	0.597020	0.698891	0.029236
0-3-0	0.646755	0.636868	0.596327	0.694040	0.033666
0-2-0	0.639871	0.636868	0.605683	0.674636	0.021017
0-0-1	0.633564	0.623354	0.597713	0.674636	0.023648
0-3-1	0.628968	0.623354	0.597713	0.683299	0.024122
0-4-1	0.629822	0.623354	0.597713	0.684685	0.025977
0-2-1	0.633588	0.623354	0.597020	0.675329	0.027323
0-1-1	0.631786	0.623354	0.597713	0.662509	0.023146

Table F.7: Statistics on score of Liver using TC, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
0-0-0	0.566820	0.566820	0.566820	0.566820	0.000000
0-0-1	0.567185	0.566820	0.566820	0.572293	0.001365
0-1-0	0.566820	0.566820	0.566820	0.566820	0.000000
0-1-1	0.567415	0.566820	0.566820	0.574021	0.001817
0-2-0	0.566820	0.566820	0.566820	0.566820	0.000000
0-2-1	0.566705	0.566820	0.565092	0.566820	0.000431
0-3-0	0.566820	0.566820	0.566820	0.566820	0.000000
0-3-1	0.567185	0.566820	0.566820	0.572293	0.001365
0-4-0	0.567415	0.566820	0.566820	0.575749	0.002227
0-4-1	0.566820	0.566820	0.566820	0.566820	0.000000
1-0-0	0.573425	0.566820	0.566820	0.591878	0.008968
1-0-1	0.575077	0.566820	0.566820	0.604551	0.012944
1-1-0	0.573137	0.566820	0.566820	0.591878	0.009632
1-1-1	0.574942	0.566820	0.566820	0.608007	0.013636
1-2-0	0.572197	0.566820	0.566820	0.591878	0.008978
1-2-1	0.575787	0.566820	0.566820	0.604551	0.014330
1-3-0	0.572158	0.566820	0.566820	0.591878	0.008660
1-3-1	0.575557	0.566820	0.566820	0.604551	0.012576
1-4-0	0.572561	0.566820	0.566820	0.591878	0.008440
1-4-1	0.575077	0.566820	0.566820	0.604551	0.012842

Table F.8: Statistics on score of KDD04P using TC, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-0	0.711129	0.713560	0.697382	0.728687	0.011009
1-3-1	0.711905	0.713068	0.697507	0.729053	0.010736
1-4-1	0.711591	0.711647	0.697475	0.729053	0.011770
1-0-1	0.711301	0.708577	0.696632	0.730367	0.011966
1-0-0	0.708877	0.707202	0.694647	0.728687	0.012364
1-3-0	0.709750	0.706503	0.695237	0.728687	0.011699
1-2-1	0.708008	0.703899	0.697670	0.729053	0.010146
1-1-1	0.706354	0.702434	0.688153	0.728687	0.012545
1-1-0	0.708727	0.701783	0.696083	0.729053	0.011642
1-2-0	0.705445	0.696875	0.694552	0.728687	0.011940
0-2-1	0.695054	0.693438	0.687714	0.703899	0.005168
0-2-0	0.693395	0.693379	0.686587	0.699374	0.004202
0-1-1	0.693593	0.692758	0.687585	0.701127	0.003901
0-0-1	0.693842	0.692436	0.684445	0.701544	0.004852
0-4-0	0.692820	0.692039	0.685739	0.699028	0.004632
0-4-1	0.693219	0.691987	0.686714	0.699191	0.004280
0-3-1	0.694116	0.691987	0.687672	0.702699	0.005228
0-1-0	0.692151	0.691875	0.687331	0.698026	0.003015
0-0-0	0.692056	0.691756	0.686897	0.700310	0.003549
0-3-0	0.692354	0.691456	0.686587	0.700798	0.004618

Table F.9: Statistics on score of Adult using TC, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-1	0.704917	0.713896	0.664104	0.737710	0.026441
1-4-0	0.645740	0.648234	0.611287	0.689479	0.025417

Table F.10: Statistics on score of KDD99 using TC, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-0	0.951362	0.951448	0.949875	0.952254	0.000685
1-4-1	0.951566	0.951329	0.951081	0.952457	0.000496

F.3 PGPC - G2

Table F.11: Statistics on score of Heart using G2, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-0	0.788311	0.802841	0.739432	0.814276	0.029126
1-0-0	0.788427	0.801455	0.734581	0.814276	0.029141
1-1-0	0.783368	0.790021	0.727304	0.814276	0.031332
1-2-0	0.787549	0.789328	0.738739	0.814276	0.026689
1-4-1	0.779672	0.788635	0.740125	0.802148	0.022499
1-3-0	0.793463	0.787249	0.765766	0.816355	0.020004
1-1-1	0.773920	0.779279	0.726611	0.802148	0.025857
1-0-1	0.777963	0.779279	0.736660	0.812890	0.023796
1-3-1	0.777754	0.778586	0.750173	0.812890	0.022945
0-0-0	0.771356	0.777200	0.734581	0.800762	0.019259
0-3-0	0.772511	0.777200	0.752252	0.788635	0.011351
1-2-1	0.775306	0.775814	0.739432	0.802148	0.021516
0-1-0	0.764680	0.775121	0.727304	0.790021	0.019764
0-2-0	0.763502	0.773735	0.738739	0.785863	0.016000
0-4-0	0.763456	0.773735	0.739432	0.777200	0.014267
0-0-1	0.760799	0.766459	0.738739	0.789328	0.016831
0-2-1	0.763802	0.766459	0.739432	0.787942	0.014100
0-4-1	0.765835	0.766459	0.740125	0.800069	0.016581
0-1-1	0.760845	0.765073	0.726611	0.778586	0.017139
0-3-1	0.762000	0.760222	0.727997	0.788635	0.015126

Table F.12: Statistics on score of Liver using G2, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-2-1	0.537961	0.544931	0.521601	0.556164	0.011512
1-4-1	0.538748	0.543203	0.516417	0.556164	0.014019
1-0-1	0.533199	0.536290	0.503744	0.561060	0.017953
1-3-1	0.531356	0.533122	0.509793	0.556164	0.016207
1-1-1	0.533641	0.530818	0.514977	0.556164	0.014074
1-0-0	0.524001	0.528802	0.502016	0.546659	0.014275
1-4-0	0.525749	0.528802	0.514401	0.537730	0.008361
1-2-0	0.524443	0.528802	0.500000	0.548675	0.014921
1-3-0	0.526363	0.528802	0.507488	0.537730	0.009115
1-1-0	0.525864	0.527362	0.510945	0.544931	0.010123
0-4-1	0.529013	0.527074	0.506048	0.555876	0.015641
0-2-1	0.527151	0.525634	0.521601	0.541475	0.005046
0-0-1	0.520353	0.523329	0.502304	0.536002	0.009872
0-1-0	0.521390	0.523329	0.504032	0.537730	0.010305
0-0-0	0.518606	0.521601	0.502016	0.539459	0.012353
0-1-1	0.520833	0.521601	0.500576	0.536002	0.009012
0-3-1	0.519393	0.521601	0.503744	0.528802	0.008067
0-3-0	0.519412	0.521601	0.504032	0.537730	0.010318
0-2-0	0.514401	0.516129	0.500000	0.532546	0.011825
0-4-0	0.519048	0.514401	0.504032	0.532546	0.009513

Table F.13: Statistics on score of KDD04P using G2, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-0-1	0.753437	0.755606	0.740358	0.758681	0.005677
1-3-0	0.752753	0.755090	0.745092	0.758571	0.004912
1-4-1	0.747377	0.754685	0.731118	0.758542	0.011108
1-3-1	0.753759	0.754685	0.746365	0.758593	0.004029
1-2-1	0.753330	0.754653	0.740211	0.758203	0.005084
1-1-1	0.753038	0.754575	0.736191	0.758659	0.005535
1-4-0	0.750162	0.753276	0.736030	0.758571	0.008244
1-1-0	0.744474	0.752988	0.699940	0.758571	0.018802
1-0-0	0.751045	0.752849	0.741923	0.758571	0.006461
0-1-1	0.742686	0.745818	0.702000	0.754575	0.012644
1-2-0	0.744676	0.745306	0.695495	0.758571	0.016188
0-2-1	0.746009	0.745248	0.733912	0.755797	0.006337
0-3-1	0.737777	0.742398	0.671980	0.752155	0.019225
0-3-0	0.734855	0.741757	0.695201	0.749568	0.017063
0-0-1	0.738312	0.740789	0.695488	0.755945	0.017154
0-0-0	0.737583	0.740609	0.695488	0.754729	0.012629
0-4-1	0.736446	0.737691	0.696436	0.753474	0.012242
0-2-0	0.732164	0.736381	0.695201	0.751933	0.016242
0-4-0	0.729110	0.736030	0.695201	0.750504	0.017595
0-1-0	0.732601	0.734136	0.699940	0.749168	0.015293

Table F.14: Statistics on score of Adult using G2, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-0	0.731762	0.736626	0.698100	0.747856	0.015914
1-4-1	0.737220	0.735775	0.725090	0.750429	0.007571

Table F.15: Statistics on score of KDD99 using G2, under variation of L and P max.

Combination (L-P-G)	Average	Median	Minimum	Maximum	Standard Deviation
1-4-0	0.852834	0.856009	0.825471	0.895710	0.024989
1-4-1	0.825533	0.825646	0.813322	0.842859	0.007978

Appendix G

Maximum Archive Size - Parameter Combination Tournament Rankings

The following tables summarize the relative ranks of the various PGPC parameter combination (L-P-G) scores under variance of the maximum learner and point archive sizes over the various data sets, grouped by post-processing method. Each parameter combination score is the median of scores recored out of the set of maximum archive size parameter combinations. Instances of two parameter combinations having the same numeric score value are resolved by giving them both the same rank value.

Due to the lack of data, only the "balanced" data sets (Heart, Liver, KDD04P), will be summarized by rank and average rank value across the set. Again, the score values utilized in the computation of the ranks are derived from the median of 30 independent runs.

Table G.1: Tournament rankings of the various parameter combinations using AA under variance of the maximum learner and point archive sizes.

Combination (L-P-G)	Rank on Heart	Rank on Liver	Rank on KDD04P	Average rank
0-0-0	14	10	14	12.67
0-0-1	13	13	13	13.00
0-1-0	10	16	15	13.67
0-1-1	11	11	13	11.67
0-2-0	12	14	11	12.33
0-2-1	13	15	12	13.33
0-3-0	14	16	17	15.67
0-3-1	13	12	19	14.67
0-4-0	9	17	18	14.67
0-4-1	13	9	16	12.67
1-0-0	2	4	5	3.67
1-0-1	6	5	6	5.67
1-1-0	3	4	7	4.67
1-1-1	8	3	9	6.67
1-2-0	4	6	10	6.67
1-2-1	7	8	8	7.67
1-3-0	1	4	3	2.67
1-3-1	5	1	1	2.33
1-4-0	2	2	4	2.67
1-4-1	5	7	2	4.67

Table G.2: Tournament rankings of the various parameter combinations using TC under variance of the maximum learner and point archive sizes.

Combination (L-P-G)	Rank on Heart	Rank on Liver	Rank on KDD04P	Average rank
0-0-0	9	1	18	9.33
0-0-1	10	1	14	8.33
0-1-0	9	1	17	9.00
0-1-1	10	1	13	8.00
0-2-0	9	1	12	7.33
0-2-1	10	1	11	7.33
0-3-0	9	1	19	9.67
0-3-1	10	1	16	9.00
0-4-0	8	1	15	8.00
0-4-1	10	1	16	9.00
1-0-0	2	1	5	2.67
1-0-1	5	1	4	3.33
1-1-0	4	1	9	4.67
1-1-1	7	1	8	5.33
1-2-0	3	1	10	4.67
1-2-1	7	1	7	5.00
1-3-0	1	1	6	2.67
1-3-1	6	1	2	3.00
1-4-0	3	1	1	1.67
1-4-1	5	1	3	3.00

Table G.3: Tournament rankings of the various parameter combinations using G2 under variance of the maximum learner and point archive sizes.

Combination (L-P-G)	Rank on Heart	Rank on Liver	Rank on KDD04P	Average rank
0-0-0	9	11	15	11.67
0-0-1	13	10	14	12.33
0-1-0	11	10	19	13.33
0-1-1	14	11	9	11.33
0-2-0	12	12	17	13.67
0-2-1	13	9	11	11.00
0-3-0	9	11	13	11.00
0-3-1	15	11	12	12.67
0-4-0	12	13	18	14.33
0-4-1	13	8	16	12.33
1-0-0	2	6	8	5.33
1-0-1	7	3	1	3.67
1-1-0	3	7	7	5.67
1-1-1	7	5	5	5.67
1-2-0	4	6	10	6.67
1-2-1	10	1	4	5.00
1-3-0	6	6	2	4.67
1-3-1	8	4	3	5.00
1-4-0	1	6	6	4.33
1-4-1	5	2	3	3.33

Appendix H

Select Maximum Archive Size - Parameter Combination TC and G2 Terrains

The following consist of the surface plots of the PGPC algorithm score under the variance of the learner and point maximum archive values, utilizing the optimum derived parameter combination. Each plot independently illustrates each of the TC and G2 post processing methods on all of the data sets.

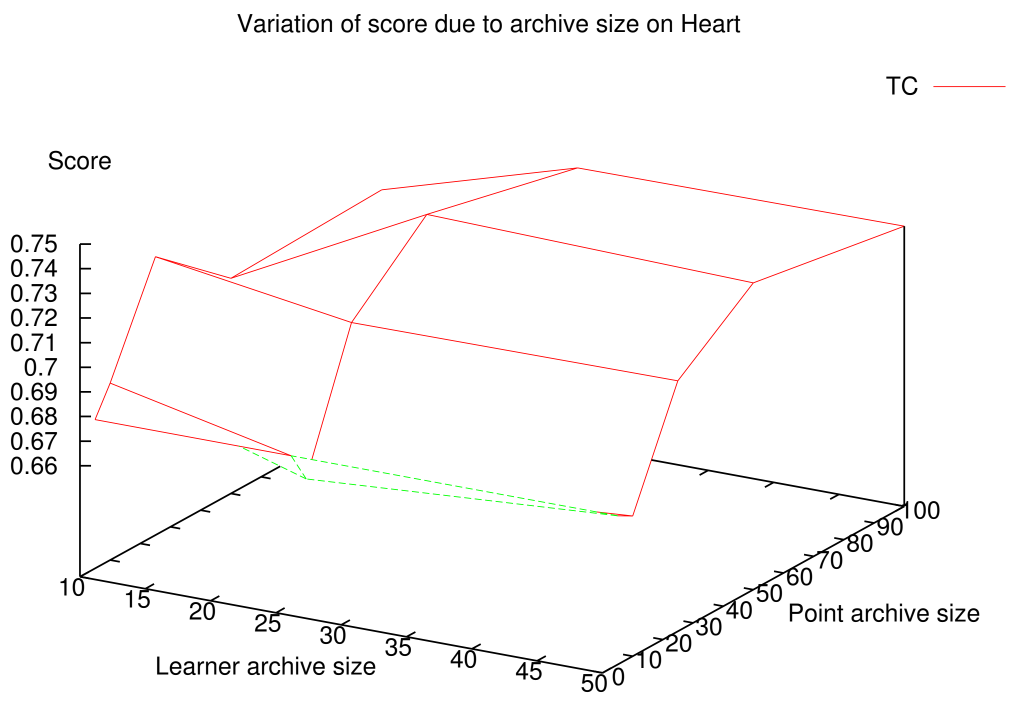


Figure H.1: Effect of varying the maximum archive sizes upon the scores of the TC post-processing method on the Heart data set using the optimal 1-4-0 parameter combination.

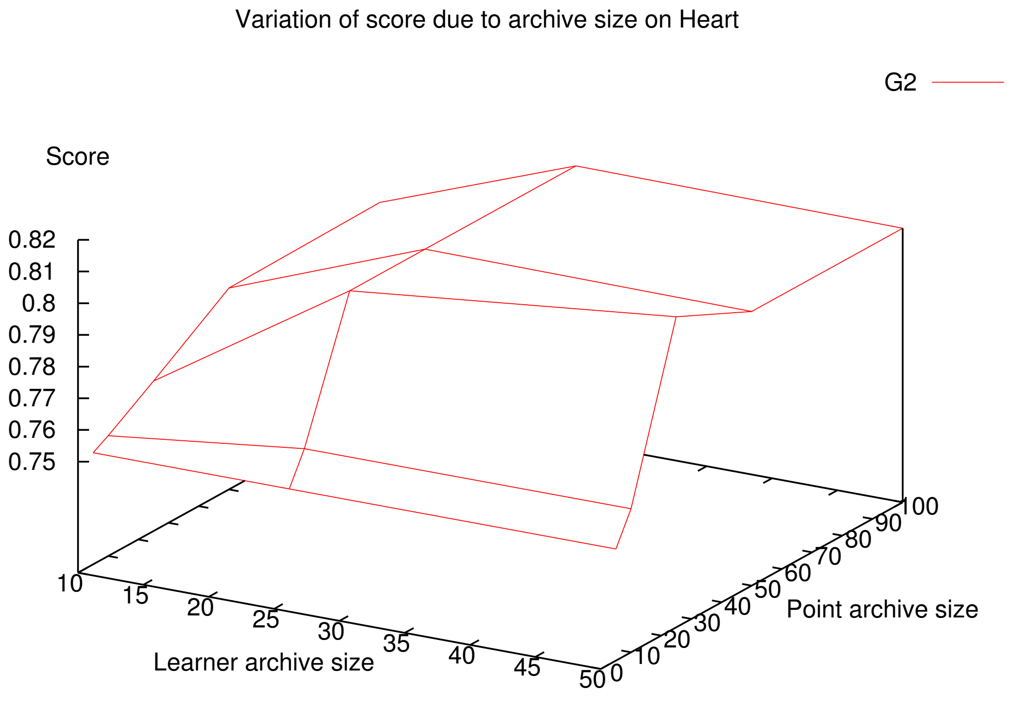


Figure H.2: Effect of varying the maximum archive sizes upon the scores of the G2 post-processing method on the Heart data set using the optimal 1-3-1 parameter combination.

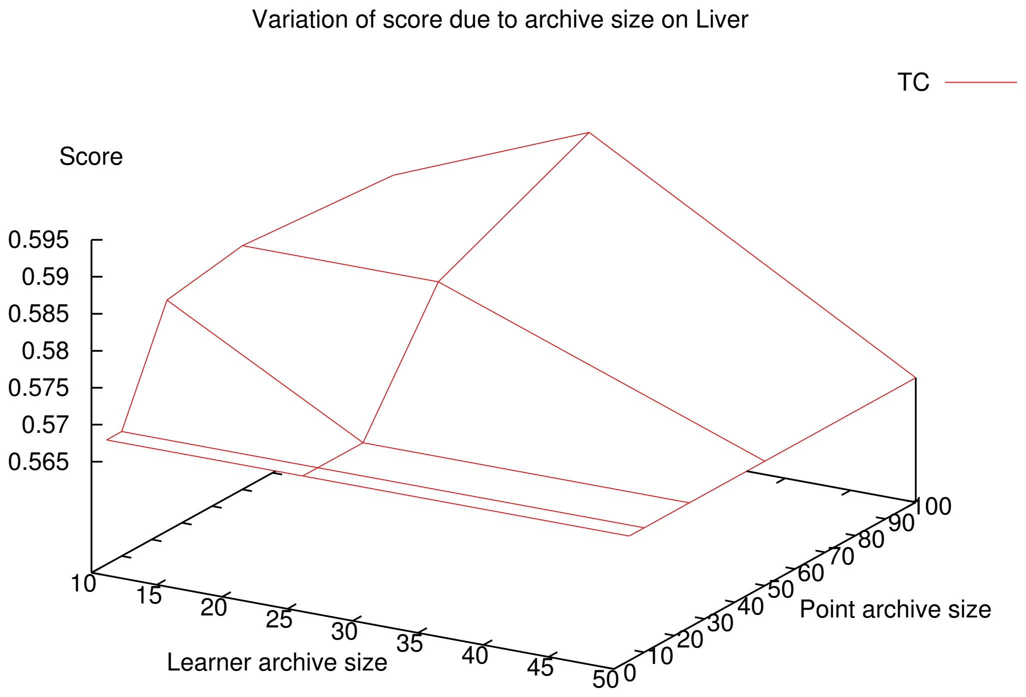


Figure H.3: Effect of varying the maximum archive sizes upon the scores of the TC post-processing method on the Liver data set using the optimal 1-4-0 parameter combination.

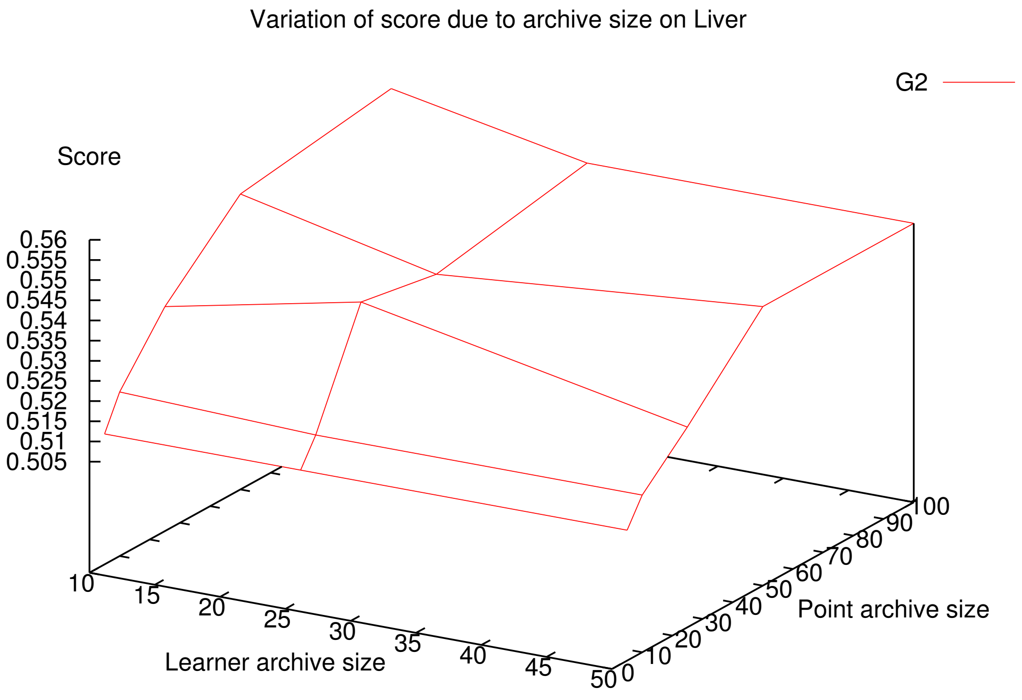


Figure H.4: Effect of varying the maximum archive sizes upon the scores of the G2 post-processing method on the Liver data set using the optimal 1-3-1 parameter combination.

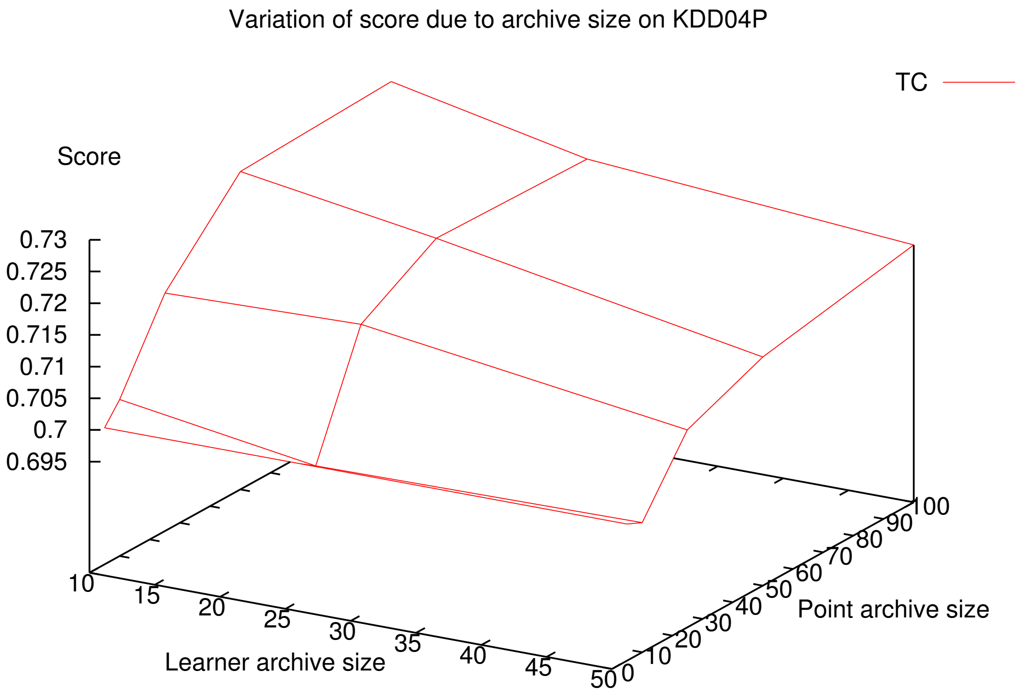


Figure H.5: Effect of varying the maximum archive sizes upon the scores of the TC post-processing method on the KDD04P data set using the optimal 1-4-0 parameter combination.

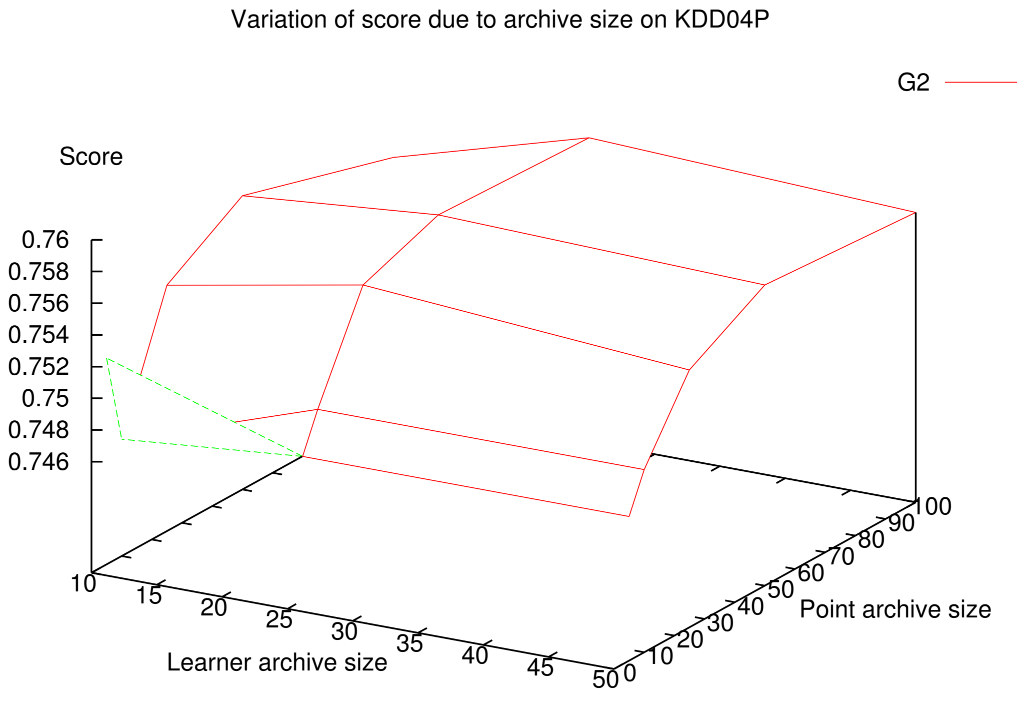


Figure H.6: Effect of varying the maximum archive sizes upon the scores of the G2 post-processing method on the KDD04P data set using the optimal 1-3-1 parameter combination.

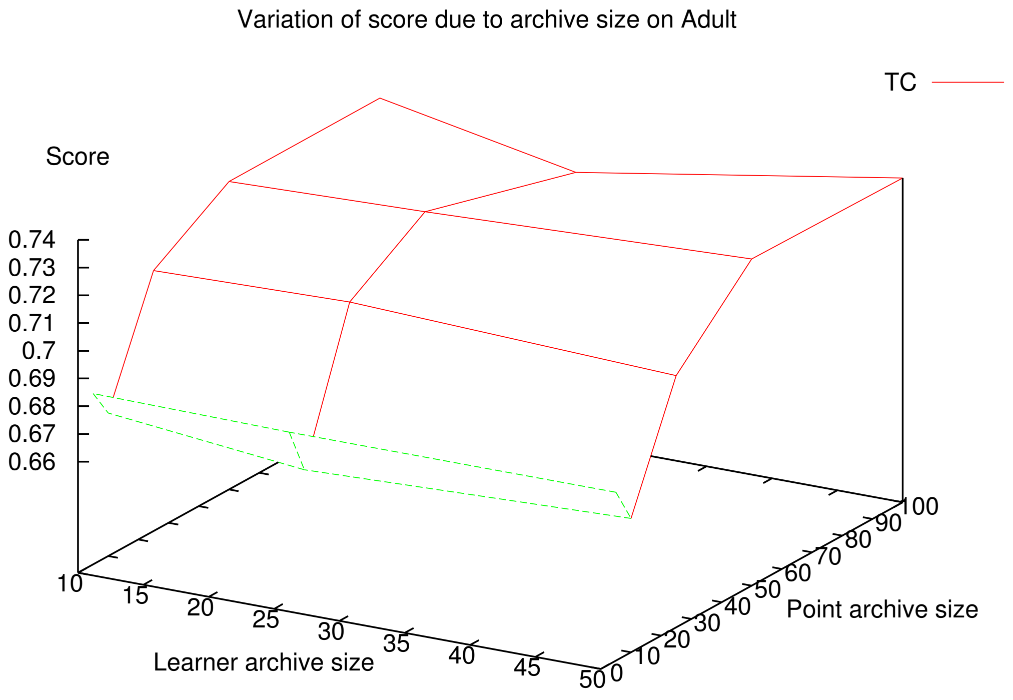


Figure H.7: Effect of varying the maximum archive sizes upon the scores of the TC post-processing method on the Adult data set using the optimal 1-4-1 parameter combination.

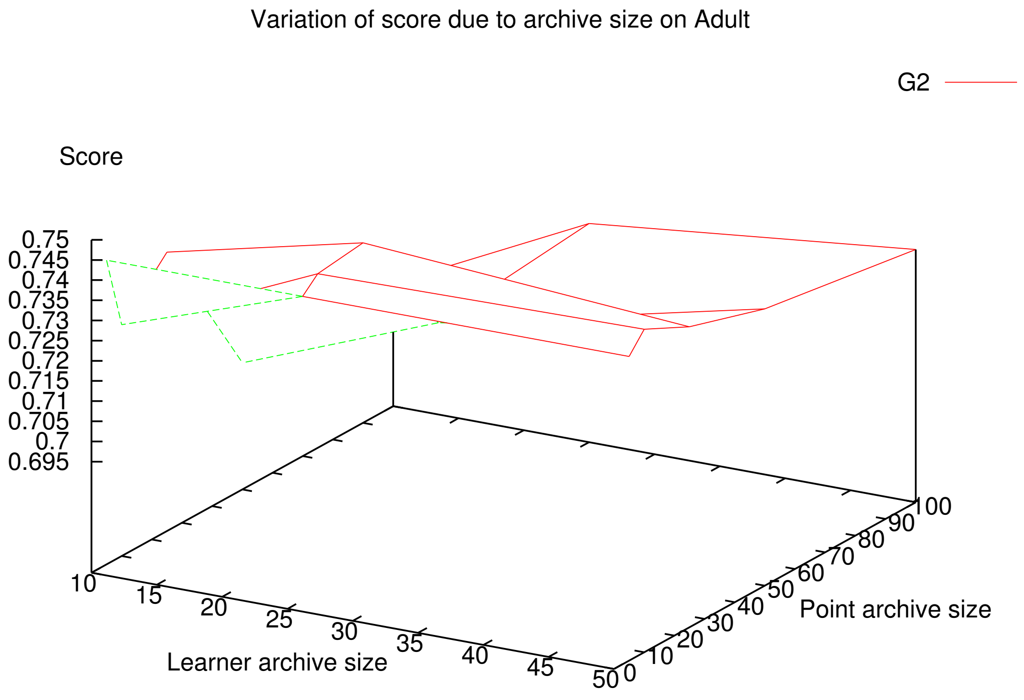


Figure H.8: Effect of varying the maximum archive sizes upon the scores of the G2 post-processing method on the Adult data set using the optimal 1-4-0 parameter combination.

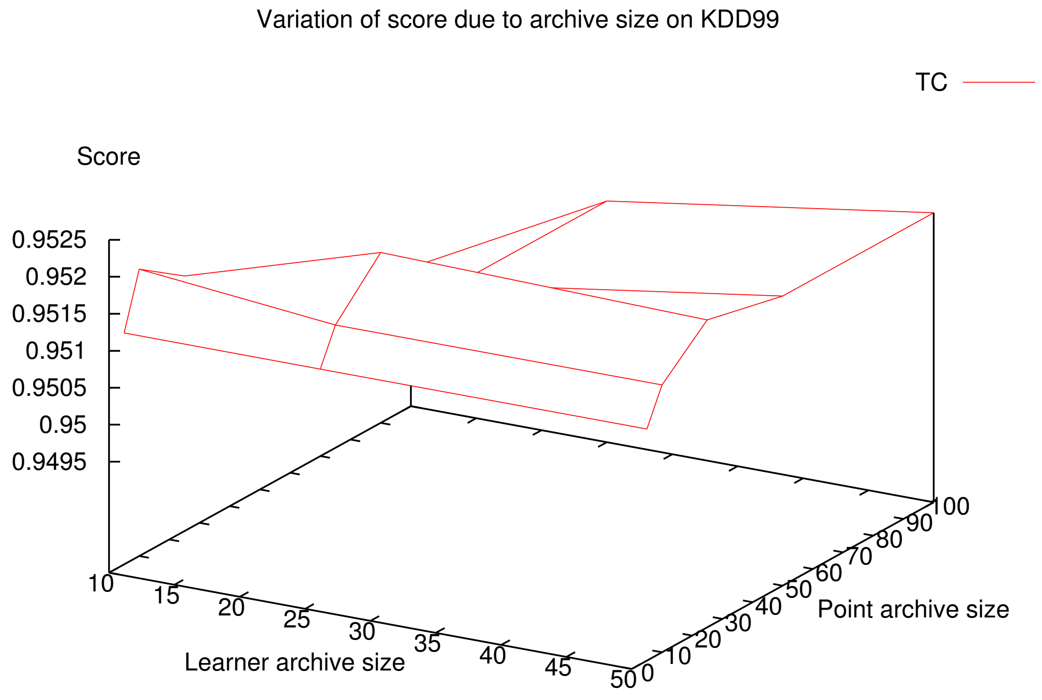


Figure H.9: Effect of varying the maximum archive sizes upon the scores of the TC post-processing method on the KDD99 data set using the optimal 1-4-1 parameter combination.

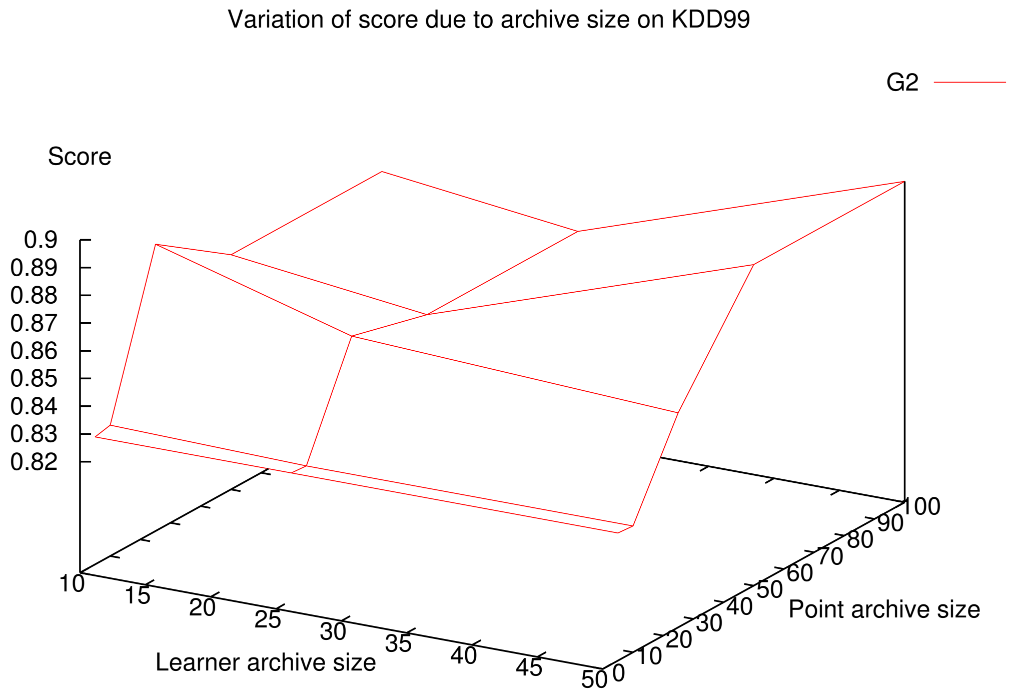


Figure H.10: Effect of varying the maximum archive sizes upon the scores of the G2 post-processing method on the KDD99 data set using the optimal 1-4-0 parameter combination.