TOWARDS EFFICIENT TRAINING ON LARGE DATASETS FOR GENETIC PROGRAMMING

by

Robert M. Curry

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

at

Dalhousie University
Halifax, Nova Scotia
September 2004

DALHOUSIE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "TOWARDS EFFICIENT TRAINING ON LARGE DATASETS FOR GENETIC PROGRAMMING" by Robert M. Curry in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: September 1st, 2004

Supervisor: ______________________
Dr. Malcolm Heywood

Readers: ______________________
Dr. Dirk Arnold

____
Dr. Thomas Trappenberg

DALHOUSIE UNIVERSITY

DATE: September 1st, 2004

AUTHOR: Robert M. Curry

TITLE: TOWARDS EFFICIENT TRAINING ON LARGE DATASETS FOR GENETIC PROGRAMMING

DEPARTMENT OR SCHOOL: Department of Computer Science

DEGREE: MCSC CONVOCATION: October YEAR: 2004

Signature of Author

Table of Contents

4 List of Tables

List of Figures

Abstract

Genetic Programming (GP) has the potential to provide unique solutions to a wide range of supervised learning problems. However, the technique suffers from a widely acknowledged computational overhead. As a consequence, applications of GP are often confined to datasets consisting of hundreds of training exemplars as opposed to tens of thousands of exemplars, thus limiting the applicability of the approach. This work proposes and thoroughly investigates three data sub-sampling algorithms that filter the initial training dataset in parallel with the learning process. The motivation being to focus the GP training on the most difficult or least recently visited exemplars. To do so, we build a hierarchy of subset selections, thus matching the concept of a memory hierarchy. Such an approach provides for the training of GP solutions to data sets with hundreds of thousands of exemplars in tens of minutes whilst matching the classification accuracies of more classical approaches.

Acknowledgements

I would like to thank my supervisor Dr. Malcolm Heywood for his valuable guidance and encouragement. I also acknowledge the support of a Precarn IRIS Emerging Opportunities Graduate Scholarship.

1 Introduction

The interest of this work lies in providing a framework for efficiently training genetic programming (GP) on large datasets. Specifically, the interest is focused on binary classification, where efficient multi-class GP algorithms have been proposed elsewhere [38]. There are at least two aspects to this problem: the cost of fitness evaluation and the overhead in managing datasets that do not reside within RAM alone. The computational overhead associated with the inner loop of GP fitness evaluation has been widely recognized. The traditional approach for addressing this problem has been hardware based. Examples include Beowulf clusters [1], parallel computers [2] and FPGA solutions [3]. In this work we propose to address the problem through the following two observations. Firstly, within the context of supervised learning, the significance of data sampling algorithms have been widely acknowledged albeit with the motivation to improve error performance, e.g. boosting and bagging [4, 5]. Secondly, memory hierarchies are widely used in CPU architectures, where such hierarchies are based on the concept of temporal and spatial locality [6]. The motivation used here, however, is that any learning algorithm need only see a subset of the total dataset, where the sampling process used to identify such a subset of exemplars should also be sympathetic to the memory hierarchy of the computing platform.

To address these issues the method of Dynamic Subset Selection [7] was revisited and extended to a hierarchy of subset selections. Such a scheme was previously applied to the 10% KDD-99 benchmark, a dataset consisting of approximately half a million exemplars [8]. The dataset was first partitioned into blocks that were sufficiently small to reside within RAM alone. Blocks were then chosen from this partition based on Random Subset Selection (RSS). This forms level 1 of the selection hierarchy. At level 2, the method of Dynamic Subset Selection (DSS) was used to stochastically select exemplars from the block identified at the first level. Several rounds of DSS are performed per level 1 block, with exemplars selected with a bias based on exemplar difficulty and age. In this work, this hierarchy is referred to as the RSS-DSS hierarchy.

In this work we concentrate on the parameterization of the algorithm and extending the evaluation to three other datasets (the previous work of Song *et al.* concentrated on application issues associated with the application to an intrusion detection problem [8]).

Furthermore, two alternative hierarchies are introduced: the first, referred to as the DSS-DSS hierarchy, selects level 1 blocks using DSS instead of RSS; and the second, referred to as the Balanced Block hierarchy first partitions the two classes of the dataset and then forms level 1 blocks by selecting a partition from each class using DSS and then combining the two partitions. Such a scheme ensures that each block always consists of the same ratio of exemplars from both the minor and major class. These new hierarchies are shown to improve the error properties of the ensuing solution.

This thesis is organized as follows: Chapter 2 is a literature survey beginning with an introduction to Genetic Programming in 2.1 and a discussion of the difficulties encountered when using Genetic Programming on large datasets in 2.2. Previous work engaged in dealing with these difficulties is discussed in 2.3. The use of subset selection algorithms is introduced in 2.4, which is then extended in this work into a hierarchy of subset selections in 2.5; Chapter 3 deals with methodology, including the generic form of GP utilized here, namely Dynamic Page-Based Linear Genetic Programming in 3.1, and the addition of the hierarchical subset selection algorithms to GP which were developed in this work in 3.2, 3.3 and 3.4; Chapter 4 details the results achieved by these algorithms on four large datasets; and Chapter 5 is a conclusion and a discussion of future work.

2 Literature Survey

This chapter begins with an introduction to Genetic Programming in 2.1 and a discussion of the difficulties encountered when using Genetic Programming on large datasets in 2.2. Previous work engaged in dealing with these difficulties including data sampling algorithms, hardware solutions and software solutions, which will be discussed in 2.3. The use of subset selection algorithms is introduced in 2.4 and in particular Gathercole's Dynamic Subset Selection (DSS) [7]. The use of subset selection algorithms is extended in this work into a hierarchy of subset selections and these algorithms are introduced in 2.5.

2.1 Introduction to Genetic Programming

Genetic programming is an extension to genetic algorithms where the goal, in this case, is to create a computer program that can solve a problem using a simulated evolutionary process [3]. Genetic programming (GP) is one of many computational techniques used to simulate evolution and hence belongs to the computer science field of evolutionary computation (EC). Genetic programming also belongs to the field of machine learning (ML), as the goal is to develop computer programs that improve automatically through experience [21]. Indeed, in this work GP is implemented as a supervised learning algorithm in which GP is presented a set of labeled exemplars, namely the training dataset, from which it is expected to generalize (classify) to previously unseen exemplars. The success of the GP is gauged by running the resulting program on a test dataset in which the labels are known but are not made available to the program [22].

Genetic Programming begins with an initial population of computer programs, each with an associated fitness. The fitness of each program is evaluated by running the program on the entire training dataset. The fitness is a reflection of the program's classification accuracy, which is the number of correctly classified exemplars divided by the total number of exemplars in the training set [24]. GP then transforms the current population into a new population, or generation, through processes based on the Darwinian concepts of natural selection and survival of the fittest [20]. This transformation is achieved through search operators, typically mutation and crossover (sexual recombination). Mutation involves a random change to an individual program in some way and is used to explore the solution space of all programs. Crossover involves interchanging parts of two programs with the hope that an even fitter program will result. It is an attempt to exploit the beneficial properties of fit individuals. Crossover is similar to controlled breeding where the hope is that the mating of two fit individuals can result in an even fitter child. GP will continue to create new generations in this manner until some kind of termination criterion is reached. Termination may occur when a certain classification accuracy is reached or after a fixed number of generations.

A simplified pseudocode Genetic Programming Algorithm can be seen in the following program, Figure 2.1.1.

```
GPAlgorithm (parameters) {
(1) initialize population of programs;
(2) read training dataset into memory;
(3) while (generation < NumGenerations)
            {
(4)         select individuals for training;
(5)         while (individual < NumIndividuals)
                        {
(6)                     while (pattern < NumPatterns)
                                    {
(7)                                 train individual on pattern;
                                    }
(8)                     evaluate individual fitness;
                        }
(9)         apply search operators;
            }
}
```

Figure 2.1.1: Simplified Pseudocode Genetic Programming Algorithm

GP uses pseudo-random numbers to mimic the randomness of natural evolution and uses stochastic processes and probabilistic decision-making throughout [21]. The number of runs needed in order to establish the statistical significance of any GP results depends on the underlying distribution, however the underlying distributions are unknown. The accepted rule of thumb requires a minimum of 30 different initializations in order to establish any statistical significance of results (i.e. verify that the solutions are not due to random chance) [40].

Program Structure: Functional and Terminal Sets – GP assembles variable length programs from basic ingredients termed functions and terminals. Functions perform operations on their inputs, which are either terminals or output from other functions. The actual creation of programs from functions and terminals occurs at the beginning of a run, when the population is initialized. Loosely speaking, terminals provide a value to the system while functions process a value already in the system [21]. The terminal set is typically comprised of the inputs (training dataset features) to the GP program and constants supplied to the GP program (numbers chosen randomly out of a range of integers or reals). The function set is composed of the statements, operators, and functions made available to the GP system. The function set may be application specific and be selected to fit the specific problem domain at hand [21]. The range of available functions to provide to a GP system is very broad. Some examples include Boolean functions (and, or, not, xor), arithmetic functions (plus, minus, multiply, divide), transcendental functions (trigonometric and logarithmic functions) and conditional statements (if, then, else, switch). An important feature of the function set is that each function should be able to handle all values it might receive as an input. This is called closure [20]. For example, the division operator cannot take zero as an input. Division by zero will normally crash the system. Protected division is used instead which functions as normal division except for zero denominator inputs in which the function returns something else (0,1, very big number). All functions (square root, log, sin, cosine) must be able to accept all possible inputs because if there is a way to crash the system GP will certainly find it [21].

In choosing the functions and terminals used in the GP system it is important that they be powerful enough to represent a solution to the problem at hand [21]. For example, a function set consisting only of the addition operator will probably not solve many interesting problems. However, it is better not to use too large a function set for this enlarges the search space and can sometimes make the search for a solution harder. Therefore, a good starting point for a function set might be to use just the arithmetic operators with protected division [21] or conditional and logical operators (applied self computing).

Fitness – The fitness of a GP program is the measure used by GP during evolution as to how well a program has learned to predict the output from the given inputs (features) of the learning domain. The fitness evaluation allows the GP system to determine which individuals should have a higher probability of being allowed to multiply and reproduce and which individuals should have a higher probability of being removed from the population. For a binary classification problem the fitness is typically taken to be the number of correctly classified exemplars in the training dataset [21]. In this case a *wrapper* is used to convert the real valued result of a GP program to a binary outcome where positive results represent class one and all other results represent class zero [20].

Selection Operator – After the application of the fitness function *selection* needs to be performed to select which individuals in the population will create offspring for the next generation [23]. Traditionally *fitness-proportional selection* is used in which each individual is given a slice of a "roulette wheel" in proportion to its fitness divided by the sum of all individuals' fitness. The wheel is then 'spun' however many times necessary to create the new population.

Another popular selection technique is *steady state tournament selection*. Tournament selection involves randomly choosing a number of individuals, called the tournament size (typically 4), from the population. Each individual within the tournament then has its fitness evaluated. The relative fitness of the individuals is used to divide the tournament into two groups. The better half will be parents and remain in the population while the inferior half will be replaced by the offspring created by applying genetic operators to the parents [21]. Steady state tournament selection does not require the comparison of fitness between all individuals in the population, which accelerates evolution time considerably and also provides an easy way to parallelize the algorithm [21]. Moreover, this form of selection operator is elitist (the best individual always survives) and is known to have a faster take over rate (poor solutions die out faster) [37].

Search Operators – The randomly initialized programs created by GP at initialization usually have a very low fitness [21]. Evolution occurs by transforming the initial programs in the population through the combination of selection (see above) and search

operators. The three most popular search operators are crossover, mutation and reproduction. Crossover combines the genetic material of two parents by swapping a part of one parent with a part of the other. Mutation operates on a single individual and is normally applied to each child produced by the crossover operator with a low probability [21]. Mutation makes a random change to the GP program in some way. Reproduction involves copying an individual so that there are now two versions of the individual in the population. The probabilities of applying search operators are parameters of a GP run.
Termination Criteria – The training process will end if the GP has converged (an optimal solution is found), or if the number of generations reaches a pre-set maximum. The latter method is chosen to avoid over-learning and/or on the basis of computational considerations.

2.2 GP on Large Datasets

The interest of this work lies in providing a framework for efficiently training GP on large datasets. The majority of time necessary to train GP is considered to be proportional to the product of the population size, the number of generations and the data size needed for fitness evaluation [28, 29]. In fact, relatively little CPU time is expended on other tasks of the algorithm, such as the creation of the initial random population at the beginning of the run and the execution of genetic operators during the run [3].
Many authors have stressed that for large and complex problems a large GP population is necessary due to the inefficiency of GP, in that it produces a low ratio of fit children at each generation [25, 29]. Therefore for difficult programs, it becomes very unlikely that a particular generation will produce fitter individuals, and thus a larger population size is seen as the main route to increasing this likelihood [25]. However, larger populations require more CPU and memory resources and so the CPU usage is made less efficient and the time taken to process a generation increases.
In this work it is proposed that there are at least two aspects to the problem of efficiently training GP on large datasets. Firstly there is the widely acknowledged computational overhead involved with the inner loop fitness evaluation in GP [1, 3, 26, 29, 31]. The measurement of fitness for just one individual in just one generation (Figure 2.1.1 – Line 8) might involve exposing that individual to hundreds or thousands of different exemplars

(Figure 2.1.1 – Line 7). This problem is magnified when dealing with even larger datasets, such as the ones under investigation here, which can have tens to hundreds of thousands of exemplars. This overhead has traditionally been addressed by hardware solutions, by either increasing the speed of computation such as using fast serial supercomputers or by parallelizing the application by using parallel systems such as Beowulf clusters [1] or Field Programmable Gate Arrays (FPGAs) [3].

Secondly, there is the overhead in managing datasets that cannot reside within RAM alone. For datasets of much more than a thousand or so records, reading the entire dataset into cache is not possible. Therefore, sequentially presenting the entire dataset to each individual at each generation (Figure 2.1.1 – Line 7) will not make efficient use of the localized spatial and temporal access patterns on which cache memory and memory hierarchies are based.

As mentioned in the introduction these problems will be addressed through the following two observations: the significance of data sampling algorithms; and the significance of memory hierarchies widely used in CPU architectures. The motivation used here is that any learning algorithm need only see a subset of the total dataset, where the sampling process used to identify such a subset of exemplars should also be sympathetic to the memory hierarchy of the computing platform.

However, first some previous work attempting to deal with these issues will be discussed in 2.4 and specifically subset selection algorithms discussed in 2.5.

2.3 Previous Work

In this section previous work on improving GP efficiency will be investigated including data sampling algorithms, hardware solutions and software solutions.

2.3.1 Data Sampling Algorithms

Within the context of supervised learning, the significance of data sampling algorithms has been widely acknowledged albeit with the motivation to improve error performance. Bagging and Boosting are both data sampling algorithms that have the capacity to improve weak learners in that they exploit the instability inherent in the learning algorithms. Although GP has not been proven to be a weak learner favorable results have been reported [30].

Bagging is a method for generating multiple versions of a predictor and using these to get an aggregated predictor [22]. The multiple versions are formed through making bootstrap replicates of the training dataset and using these as new training datasets. The replicate datasets are the same size as the original dataset but formed by drawing at random, with replacement, from the original dataset [4]. Bagging improves the error performance of weak learners but GP would have to be run on several datasets of the same size. Even if parallel implementations of GP could be run on the separate datasets there would be no improvement in CPU time.

Boosting is a scheme that improves machine-learning methods without the need for more test cases. It operates by modifying the distribution of the examples in the learning set by emphasizing hard cases. Boosting creates several hypotheses based on different distributions, starting with an initial uniform distribution and then creating new distributions based on the performance of the previous run [22]. The hypotheses are then combined to obtain a final hypothesis. Paris et al. applied boosting to GP with an emphasis on error minimization, and indeed found that boosting greatly improved GP performance [30]. Their focus did not concern large datasets and indeed the use of large datasets would not be feasible under their scheme. CPU time will take much longer since GP has to be run on another dataset of the same size for every round of boosting. Furthermore, improvements cannot be made through parallelization since each new dataset depends on the performance of GP on the previous dataset.

Therefore both bagging and boosting can not directly address the computational overhead of the GP fitness evaluation or the overhead associated with dealing with datasets that do not fit within RAM alone, however the significance of data sampling algorithms is an important observation and have influenced the solution being proposed in this work for applying GP to large datasets.

2.3.2 Hardware Solutions

Hardware solutions to improving GP efficiency include parallel computing methods and the use of fast serial supercomputers. However these types of systems are often not available or affordable to most users.

Parallel Computing – Amenability to parallelization is a recognized feature of genetic programming [29]. The time consuming fitness evaluations of the inner loop of GP are totally decoupled from one another and can be performed independently for each individual in the population. There are three main types of parallelization used: the asynchronous island model, the cellular model and the master-slave model.

Using the asynchronous island model for parallelization the population is divided into semi-isolated subpopulations (demes) with each subpopulation assigned to a separate processor [1]. The local processor is responsible for genetic operations, selection, randomly creating its initial subpopulation and the time consuming task of fitness evaluations. Once a generation has been completed a relatively small percentage of the individuals in each subpopulation are probabilistically selected (based on fitness) for emigration from each processor to other nearby processors. Due to the fitness evaluations being performed independently for each individual at each processor the asynchronous island model for parallelization results in an overall increase in the total amount of work performed that is nearly linear with the number of independent processors [1]. This near 100% efficiency is in marked contrast to the efficiency achieved in parallelizing the vast majority of computer algorithms. In fact the use of semi-isolated subpopulations with occasional migration often delivers a super linear speed-up in terms of the computational effort required to yield a solution [29].

Bennett et al. built a Beowulf-style parallel computer system to increase the computing power of GP and produced 14 results that are competitive with human produced results [1]. Andre and Koza parallelized GP and found a solution to the Boolean even 5 parity problem with half the computational effort of standard GP [29].

Another method of parallelizing GP is to use the cellular model of parallelization. In the cellular model each individual has a spatial location on a low-dimensional grid and the individuals interact locally within a small neighborhood [31]. This model considers the population as a system of active individuals that interact only with their direct neighbors. Different neighborhoods can be defined for the cells and the fitness evaluation is done simultaneously for all individuals. Selection, reproduction and mating take place locally within each neighborhood. Cellular GP uses a cellular automata framework to enable a fine-grained parallel implementation of GP through the diffusion model [31]. The reported advantages of parallelizing GP in this manner include being able to handle large populations in a reasonable time, enabling fast convergence by reducing the number of iterations and execution time and favoring the cooperation in the search for good solutions which in turn improves the accuracy. In [20] a comparison of cellular GP with both standard GP and the island model of parallel GP using benchmark problems of differing complexity was made and the superiority of the cellular approach was shown. In [31], ensemble techniques for cellular GP are introduced. Bagging is a well-known ensemble technique, but when the dataset is too large to fit into main memory, the bags will also be too large and therefore constructing many bags of the same size as the entire dataset is not feasible solution. However, Breiman, who introduced bagging, suggested that when datasets are too large to fit within main memory alone, a possible approach was to partition the dataset into small pieces. A predictor would then be built on each partition, with the resulting predictors combined together through a suitable voting scheme. In [31], the approach was to partition the training data into small subsets, and obtain an ensemble of predictors on the basis of each subset, and then use a voting classification algorithm to predict the class label of new exemplars. A simple majority-voting algorithm was used, similar to that used in bagging. A parallel file system for partitioning the dataset on different processors was used to obtain an efficient data access time [31]. Using this ensemble technique with parallel GP provides comparable accuracy to a single predictor trained on the entire training set, but at a much lower computational cost. Results were reported on several classification datasets and showed a much-reduced CPU time with comparable accuracy to that of cellular GP without ensembles [31]. Another way to parallelize a GP system is to use a Master-Slave architecture and farm out fitness cases to slave processors from the master. For problems of 1000 or more fitness cases a nearly linear speed-up has been reported [21].

Field Programmable Gate Arrays – Field programmable gate arrays (FPGAs) are massively parallel computational devices. Once they are configured the thousands of logical function units operate in parallel at the chip's clock rate. With rapidly reconfigurable FPGAs and the idea of evolvable hardware it is possible to embody each individual of the GP population into hardware [3]. The FPGA can then be used to evaluate all fitness measurements in parallel while a Pentium type computer can be used as a general-purpose host computer to perform all other tasks. In this way the massive parallelism of FPGAs was used to accelerate the time consuming task of fitness measurement in the inner loop of the GP. This setup was used to solve Minimal Sorting Networks managing to find a 16-step 7 sorter, a 19-step 8 sorter and a 25-step 9 sorter which were all proven to be minimal [3]. The limitations of the FPGA is that the clock rate operates much slower than contemporary computers (10-fold) and the operations that can be performed by the logical function units of an FPGA are extremely primitive in comparison to contemporary microprocessor chips. Thus, the instruction set associated with the solution programs might present a significant barrier to understanding a solution once found.

2.3.3 Software Solutions

Software solutions provide an alternative and comparably affordable method of improving GP efficiency. Several software solutions include the use of machine code evolution [21], the limited error fitness algorithm [27] and tricks of the trade such as the removal of structural introns [17] and early run termination [21].

Machine Code Evolution – The fast execution of machine code on computer processors allows a considerable speed-up of GP to be obtained by directly evolving machine code. Figures differ by system, but an acceleration factor of between 60 and 200 compared to traditional GP systems can be safely assumed [21]. In this implementation GP individuals are stored as arrays in memory. These arrays actually contain machine code instructions in the form of binary code, which operate directly on CPU registers. There is no high-level interpretation or compilation step involved since the instructions are executed directly by the processor. In particular, there are no virtual machines, intermediate languages, interpreters or compilers involved. This is a fast approach, about 60 times faster than compiled C code, and is very compact in its use of memory. However, programming this type of system is more difficult than other methods and much greater effort is needed to assure portability, flexibility and maintainability [21].

Limited Error Fitness – Using the limited error fitness (LEF) algorithm of [27] a GP individual's fitness is related to how many of the ordered set of training exemplars it classifies correctly before it makes a certain number of misclassifications. After exceeding this error limit any cases not yet covered by the individual are counted as misclassified. The fitness score is then the total number of misclassified exemplars. Therefore, in general, it is quicker to find the fitness of a poor GP individual than a good GP individual which saves CPU time. Furthermore, the training set order and the error limit are both altered dynamically in response to the performance of the fittest individual in the previous generation. The training set order is altered by "bubbling" – moving the easiest exemplars (misclassified the least) to the end of the ordered set, and thereby moving the harder exemplars one place towards the start of the ordered set [27]. In this way easier exemplars may not be reached because of the error limit and harder exemplars move up the training set and thereby force GP to deal with them. In this way the population is repeatedly forced by LEF to cope with the difficult exemplars. LEF continually emphasizes the relative importance of difficult exemplars and de-emphasizes the importance of easy exemplars [27]. LEF promotes generality, penalizes specialists and maintains diversity in the GP population keeping it in flux even after many thousands of generations preventing premature convergence [27]. This flux allows the use of smaller GP population sizes, which speeds up GP fitness evaluations and allows GP to be run on smaller computers at a reasonable speed. Furthermore, it makes the problem easier for GP to solve in small steps.

GP+LEF was used on the "even N parity problem" for N=6 and N=7 where the training set consists of the 2^N possible combinations of N binary inputs. GP+LEF was able to solve the problem for N=6 and N=7 while standard GP could not. However, GP+LEF has many more parameters than standard GP, which are used to control the change in difficulty of the problem in response to the performance of the population in the previous generation [27]. And it was found the GP+LEF was not very robust to the choice of values for these parameters. Furthermore, even though individual fitness evaluations run more quickly, LEF was found to require many more generations with each run taking several hours to complete.

Structural Intron Removal – Introns are dispensable instructions that do not affect the program execution. In particular, structural introns are single instructions that emerge from manipulating variables that are not used for the calculation of the outputs at that program position [17]. However, introns do play an important role in protecting the effective information holding code from being disrupted by the genetic operators. However, a speed up can be achieved by temporarily removing structural introns from a program before fitness evaluation. This is achieved by copying all of the effective instructions to a buffer with the resulting program then used for the fitness evaluations. This does not cause any changes to the individual during evolution or in behavior but results in an enormous speedup in execution. Linear GP with intone removal was run on several medical classification datasets and resulted in a significant decrease in runtime [17].

Run Termination – In [21] it is noted that it is very helpful to have various criteria for signaling the end of a GP run. Intron explosion could be one such signal since fitness improvement is effectively over once exponential intron growth sets in [21]. Early termination criteria can reduce the run time by a significant amount.

2.4 Subset Selection Algorithms

This work utilizes and extends the active learning concept first proposed by Gathercole denoted Dynamic Subset Selection. The basic approach and two further subset selection algorithms based on dynamic subset selection, namely Topology-Based Subset Selection and Active Data Selection, are summarized.

Dynamic Subset Selection – In [7] and [25], Gathercole and Ross were motivated by the idea that genetic programming need only see a subset of the total dataset during each generation of GP. Furthermore, they found that it was beneficial to focus GP’s attention on difficult exemplars, which are exemplars that are frequently misclassified. It was also beneficial to involve exemplars that had not been looked at for several generations. These observations led to the *dynamic subset selection* algorithm which involves randomly selecting a target number of exemplars from the whole training set at every generation of GP, with a bias, so that an exemplar is more likely to be selected if it is difficult or has not been selected for several generations. GP programs at a given generation are then evaluated against this chosen subset of exemplars instead of the entire training dataset. Besides the dynamic subset selection algorithm, where subsets are chosen randomly but biased by exemplar difficulty and exemplar age, the author's developed two more algorithms: *historical subset selection* (HSS) in which exemplars are chosen randomly with a bias only towards exemplar difficulty and *random subset selection* (RSS) in which exemplars are chosen randomly but with uniform probability.

GP with the DSS, HSS and RSS algorithms were applied to the Thyroid dataset consisting of 3,772 exemplars. The learning task was to classify exemplars as normal or not based upon 21 thyroid gland measurements. The thyroid problem was once considered the limit in terms of difficulty and CPU time of what was practical for GP [25]. All three algorithms required much shorter time then standard GP. Furthermore, DSS produces results as good as those of standard GP, HSS produces results that nearly match standard GP, and RSS performed surprisingly well and could match the performance of standard GP in certain situations. GP+DSS produced the best overall result in comparison to the other subset selection algorithms, standard GP and previous neural network results and the average GP+DSS performance was better than the best of standard GP. It was also better able to generalize from the training data than the neural networks.

GP+DSS was also applied to the Tic-Tac-Toe problem, which consists of 958 cases of all legal 3X3 board configurations. The learning task was for the GP to classify all possible board positions as to whether or not they are a win for 'x'. GP+DSS finds an optimal solution in all runs compared to standard GP's 24% success rate and required less computational effort [25].

The author's hypothesize that the DSS algorithm provides GP with more possibility for small increments in fitness. DSS makes more use of the training set and the changing abilities of the population in supervised learning problems. They reformulate the fitness function in response to the changing abilities of the GP population, in effect presenting a different version of the problem to each generation, emphasizing hard exemplars and de-emphasizing easy exemplars [25]. Furthermore they found that the DSS algorithm seemed to be quite robust performing well with many different parameter settings. However, the selection of useful parameter settings is somewhat of a black art.

These subset selection algorithms are simple, require much less CPU time and could be widely applicable to other supervised training algorithms [7].

Topology-Based Subset Selection – The authors of [26] developed a subset selection method that takes the problem structure into account, while being problem independent at the same time. They gather information about the problem structure being examined during the evolutionary search by creating a topology, or relationship, on the set of fitness cases. The topology is represented by an undirected weighted graph. This topology is induced by the individuals in the evolving population through increasing the strength of the relation between two fitness cases if an individual in the population is able to solve both of them. The strength of the relation is increased by increasing the weight of the edge between the two fitness cases in the graph. Subsets are then chosen during the GP run such that fitness cases in the subset are as distantly related as possible with respect to the induced topology. The authors maintain that this topology-based selection (TBS) helps to improve the performance of GP by allowing dynamically smaller and more suitable subsets to be selected. TBS was applied to two classification problems: the intertwined spiral problem and the thyroid dataset. TBS and DSS were applied with two different settings with TBS having better averages results on both problems under the first setting, and the comparison of the results using the second setting were not statistically significant. However, they claim that on average, runs using the topology-based selection show faster progress than that of dynamic subset selection [26]. However, there is a problem with TBS when the dataset is very large. The most computationally expensive tasks in TBS are the adaptation of the topology and the requirement that the edge values in the graph need to be sorted to select the subset at each generation, which scales approximately quadratically with training set size! The sorting of the edge weights for a dataset of size N in just one generation would be on the order of $O(N^2 \log N^2)$ [26]. Clearly this method is not feasible for very large datasets.

Active Data Selection – In [28] the authors accelerate the evolution of GP through active selection of fitness cases during the GP run. Similar to the method of dynamic subset selection, active data selection operates on only a subset of the given dataset at each generation. However, instead of maintaining the same size the training subset increases incrementally as the generations go on through a process termed Incremental Data Inheritance (IDI). In fact each individual program in the active data selection method will maintain its own training subset starting with a small set of initial cases chosen from the whole training set. After the evolution of individuals through genetic operators each individual's training subset is also evolved. New unseen exemplars are added to each program's training dataset from the original dataset.

The authors applied GP with active data selection to the table transport problem, in which the objective is to find a multi-robot algorithm that, when executed by the robots in parallel, causes efficient table transport behavior in the group. The training and test set performance were slightly improved over standard GP, with almost half the CPU time and with smaller programs evolved [28].

However, with a large dataset this algorithm will still have a problem as the individual training datasets grow. An individual program's subset could grow so that the subset no longer fits within cache or RAM alone, resulting in inefficient use of memory resources. Moreover this problem can occur with each individual since they maintain their own subsets and could result in storage issues since there will now be multiple large datasets.

2.5 Hierarchical Subset Selection

For very large datasets that do not fit within RAM alone, the use of subset selection algorithms of section 2.4 will still require multiple slow hard disk accesses and inefficient use of memory in the selection of subsets. Therefore, in the work of [8] the dynamic subset selection (DSS) algorithm of [7] was extended into a hierarchy of subset selections, to match the concept of a memory hierarchy supported in modern computers. This work has been re-implemented in this thesis. First the entire training dataset is partitioned into blocks that are small enough to fit within RAM, as shown in Figure 2.5.1 below. Blocks are then chosen from the partitioned dataset randomly, with uniform probability, based on random subset selection (RSS). This forms level 1 of the selection hierarchy. Level 2 of the selection hierarchy used the method of DSS to stochastically select exemplars form the block selected at level 1, biased by exemplar difficulty and age. Multiple level 2 subsets are selected for each level 1 block selection.

In this work this hierarchy is referred to as the RSS-DSS hierarchy, and details of this algorithm can be found in 3.2. The work of Song et al. concentrated on issues associated with the application of this hierarchy to an intrusion detection problem (10% KDD'99 dataset) [8]. However, in this work the evaluation of the RSS-DSS algorithm is extended to three further datasets and there is more concentration on the parameterization of the algorithm.

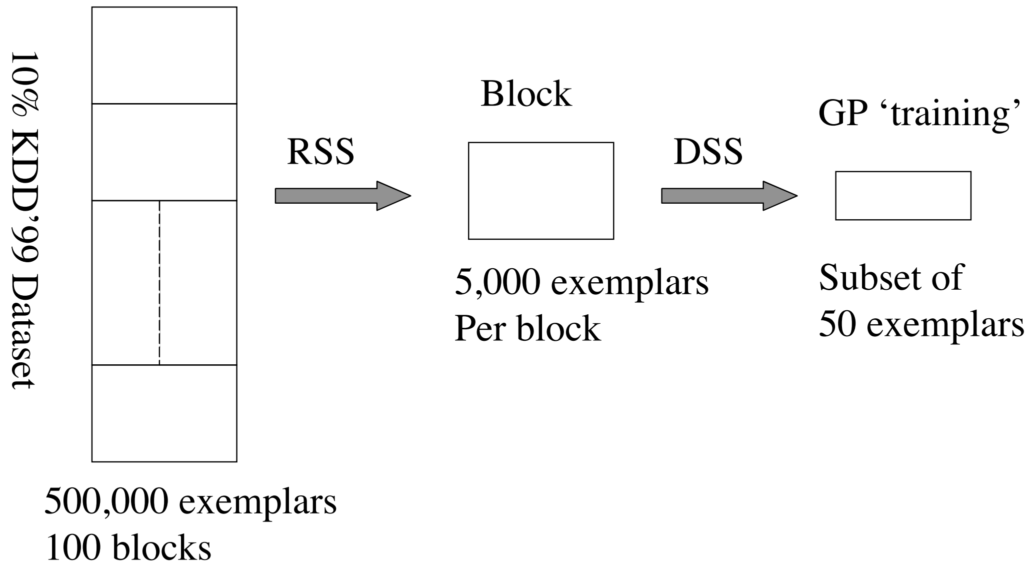

Figure 2.5.1: RSS-DSS Hierarchy

An alternative subset selection hierarchy is considered in this work in which level 1 block selections are also chosen through the use of the DSS algorithm, by introducing a block difficulty and a block age. Blocks can now be chosen randomly, with a bias towards block difficulty and block age. This hierarchy is referred to as the DSS-DSS hierarchy and it relaxes the assumption made by the RSS-DSS hierarchy that all blocks are equally difficult [32]. Details for the DSS-DSS algorithm are presented in Section 3.3.

An extension of the DSS-DSS hierarchy is also investigated in this thesis. Here, the level 1 blocks are 'balanced'. This hierarchy is referred to as the Balanced Block DSS-DSS hierarchy or simply as the Balanced Block algorithm and is outlined in Figure 2.5.2. The original dataset is first sorted into classes and then each class is divided into partitions, though partition sizes between each class are not necessarily the same size. Level 1 blocks are then composed from a partition of each exemplar class, where such partitions are selected in proportion to their difficulty and age by the DSS algorithm. The net effect is that every block consists of a balanced set of exemplars, independent of the initial exemplar data distribution. Level 2 subset selections from this balanced block are conducted by DSS.

Further details of the RSS-DSS, the DSS-DSS and the Balanced Block Hierarchies will be examined in Chapter 3 – Methodology.

2.6 Discussion

In this work, the interest is in developing a single classifier capable of dealing with the entire large dataset and not in creating a committee of classifiers developed on subsets of the dataset that need to be combined after training on their individual subsets.

The two issues involved with applying GP to large datasets are the overhead involved in GP fitness evaluations and the overhead of dealing with datasets that do not fit within RAM alone. It has been shown in 2.4 that the method of dynamic subset selection (DSS) effectively deals with the issue of GP fitness evaluations only requiring the GP to see a subset of the entire dataset at every generation. By focusing on exemplars that are difficult or have not been seen for several generations this technique was shown to speed up fitness evaluations and even outperform standard GP.

However, for a GP classifier to be successful on a large dataset it is not sufficient to only address the issue of GP fitness evaluation but must also explicitly address the size of the dataset. The DSS algorithm does not address the memory overhead involved with large datasets and so it cannot take advantage of the concepts of temporal and spatial locality on which cache memory is based. For this reason, the method of dynamic subset selection is extended upon in this work into a hierarchy of subset selections. Hierarchical subset selection will improve GP fitness evaluations through the use of DSS and will also be sympathetic to the memory hierarchy employed in modern computers by dividing the large datasets into blocks that are small enough to fit within RAM alone.

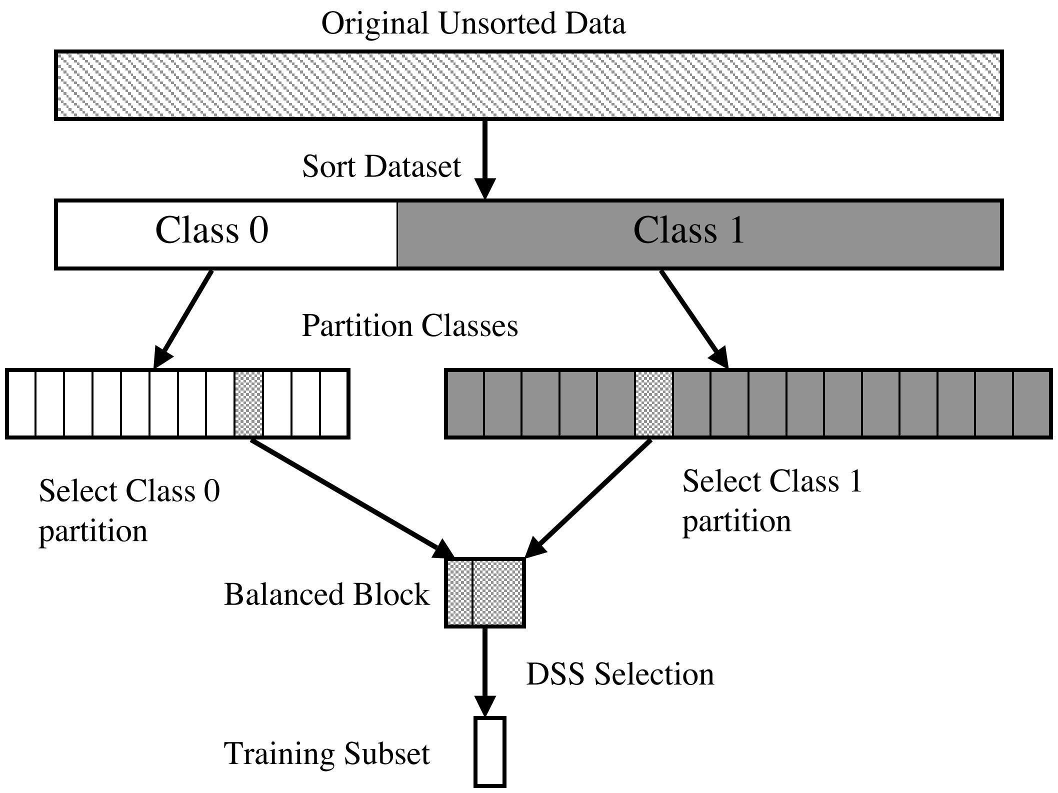

Figure 2.5.2: The Balanced Block Hierarchical Subset Selection Algorithm - The original large dataset is first sorted into the two classes of the binary classification dataset. The two classes are then divided into partitions. A partition from each class is then combined to form a level 1 block. Partitions are selected by DSS over partition ages and difficulties. Once a level 1 block is formed multiple level 2 subset selections are conducted and GP individuals are trained on selected subsets.

3 Methodology

As indicated above, our principle interest lies in the investigation of exemplar sub-sampling algorithms, which filter the dataset in proportion to the ‘age’ and ‘difficulty’ of exemplars as viewed by the learning algorithm, while also incorporating the concept of a memory hierarchy. Such schemes should significantly decrease the time to complete the inner loop of GP, without impacting the error performance. The algorithms are actually independent of the supervised learning algorithm, but in this case they are motivated by the plight of GP in which the inner loop is iterated over a population of candidate solutions [36]. Section 3.1 summarizes the form of GP utilized in this work (any generic form of GP will suffice), whereas the methodologies for the three hierarchical subset selection techniques are detailed in Sections 3.2, 3.3 and 3.4.

3.1 Dynamic Page-Based Linear Genetic Programming

In this work a form of Linearly-structured GP (L-GP) is employed [9-12]. That is to say, rather than expressing individuals using the tree like structure popularized by the work of Koza [13], individuals are expressed as a linear list of instructions which are executed sequentially [9]. Execution of an individual therefore mimics the process of program execution normally associated with a simple register machine. That is, instructions are defined in terms of an opcode and operand (synonymous with function and terminal sets respectively) that modify the contents of internal registers {R[0],…,R[k]}, memory and program counter [9]. Output of a program is taken from the best register upon completion of program execution (or some appropriate halting criterion [11]), where the best register is the register of the best performing individual that generates the greatest number of hits.

Crossover in Linear GP swaps linear segments of code between two parents as shown in the Figure 3.1.1 below (adapted from [21]). The parents are in the left half of the figure while the resulting children are in the right half of the figure. Crossover chooses two individuals as parents based on some selection policy. Crossover then randomly selects a number of instructions from each parent (shown in grey) and swaps the selected sequences between the two parents. Mutation, in linear GP, performs a logical XOR operation between the candidate instruction and a randomly generated bit sequence.

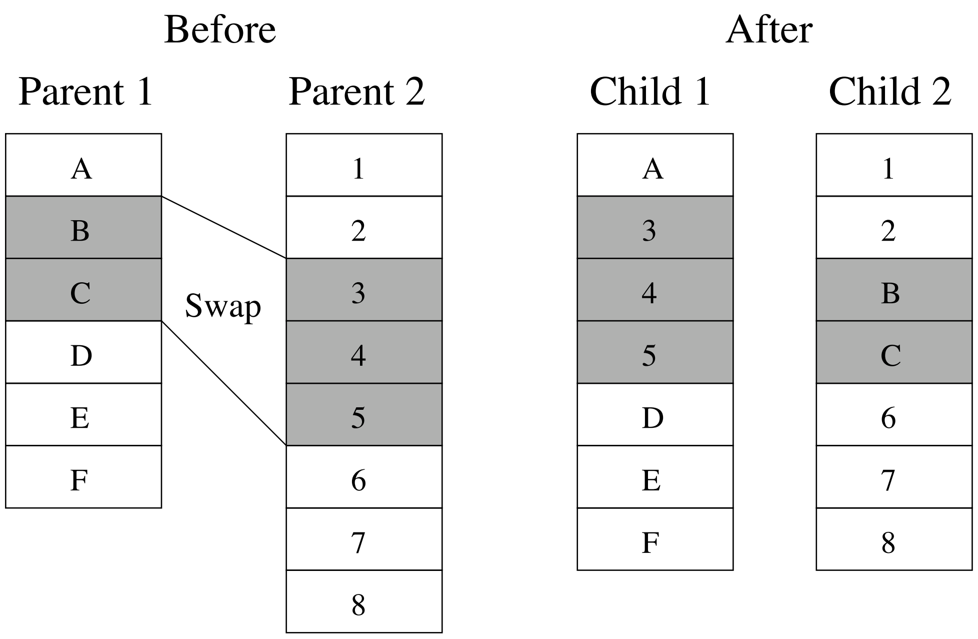

Figure 3.1.1: Crossover in Linear GP

Page-based Linear Genetic Programming with Dynamic Crossover

In an attempt to make the action of the crossover operator less destructive, the location of crossover points remains constant. This scheme is denoted as dynamic page-based LGP, or DPgLGP [12]. In DPgLGP, an individual is described in terms of a number of *pages*, where each page has the *same* number of *instructions*. Crossover is limited to the exchange of *single* pages between two parents, and appears to result in concise solutions across a range of benchmark regression and classification problems. Moreover, a mechanism for dynamically changing page size was introduced, thus avoiding problems associated with the *a priori* selection of a specific number of instructions per page at initialization. Mutation operators take two forms. In the first case the ‘mutation’ operator selects an instruction for modification with uniform probability and performs an XOR with a second instruction, also created with uniform probability. If the ensuing instruction represents a legal instruction the new instruction is accepted, otherwise the process is repeated. The second mutation operator ‘swap’ is designed to provide sequence modification. To do so, two instructions are selected within the same individual with uniform probability and their positions exchanged. The motivation behind the swap mutation operator is that the sequence in which instructions are executed within a program has a significant effect on the solution. Thus a program may have the correct composition of instructions but specified in the wrong order.

The fixed number of instructions per page and the crossover operator being constrained to swapping a single page between programs means that once initialized the length of an individual (the number of pages multiplied by the number of instructions per page) never changes. This differs from classical GP where there is no constraint on crossover, which can result in an increase in the number of instructions and thus the length of individual programs often without a corresponding improvement in performance (code bloat). Despite the fixed length of individuals, pgLGP appears to be capable of providing concise solutions and does not appear to be sensitive to the maximum number of instructions, hence, it does not need extensive fine-tuning of this parameter as might be anticipated [12]. The maximum page size of 256 instructions utilized here was adopted from the previous work of Song et al. [8]. Median results in terms of program size are typically well within this maximum and indicate that this parameterization is satisfactory. Furthermore, the crossover operator in traditional Linear GP is responsible for creating significant overheads in terms of memory management and memory accesses. The main memory management problem coming from the requirement to swap code fragments of differing lengths between pairs of individuals. Enough memory needs to be reserved for each individual up to the maximum program length, and entire blocks of code physically shuffled. DPgLGP only swaps code fragments of equal length and so the memory management problem now simplifies to reading programs and copying the contents of parents to children [12].

3.2 RSS-DSS Hierarchy

The basic formulation for the hierarchical sampling of training exemplars divides the problem into three levels. Level 0 divides the training set into a sequence of equal blocks. Blocks reside in memory and are chosen stochastically, Level 1. Level 2 samples the exemplars of the selected block using the stochastic sampling algorithm Dynamic Subset Selection (DSS), which biases selection towards the more difficult or older exemplars [7]. Program 1 outlines the general relationship between learning algorithm (GP in this case) and hierarchical sampling algorithm for the case of RSS block selection at level 1 and DSS exemplar selection at level 2:

Program 3.1 - Genetic Programming with RSS-DSS Hierarchy

```
{
(1)     divide dataset into blocks (level 0)
(2)     initialize training system and population
(3)     while (RSStermination == FALSE)
            {
(4)         conduct Block Selection (level 1)
(5)         while (DSStermination == FALSE)
                {
(6)             conduct Subset Selection (level 2)
(7)             while (TournamentEnd == FALSE)
                    {
(8)                 conduct tournament selection
(9)                 train tournament individuals on Subset
(10)                update exemplar difficulty
(11)                apply search operators
                    }
                }
(12)            update #Subset selected at next block b instance
            }
(13)    run best individual on entire dataset
(14)    run best individual on test dataset
(15)    record results
(16)    remove introns and translate
}
```

Basic design decisions now need to identify: how a block is identified (level 1), how a subset is selected (level 2) where GP individuals only iterate over the contents of a subset, and how many subsets are selected per block, i.e. the source of the computational speedup. Three basic algorithms are proposed, RSS-DSS (§3.2), DSS-DSS (§3.3) and the Balanced Block algorithm (§3.4).

Level 1 – Block based Random Subset Selection (RSS). At level 0 the datasets are partitioned into 'blocks' (Program 3.1, Line 1), all the blocks exist on the hard disk. A block is then randomly selected with uniform probability (Program 3.1, Line 4) – or Random Subset Selection (RSS) – and read into RAM, level 1 of the hierarchical Subset Selection algorithm. Following selection of block '*b*' a history of training pressure on the block is used to determine the number of iterations performed at level 2 of the RSS-DSS hierarchy – the DSS subset. This is defined in proportion to the error rate over the previous instance of block '*b*' for the 'best' individual over a level 2 subset from block '*b*' (Program 3.1, Line 12), $E_b(i\text{-}1)$. Thus, the number of DSS subset iterations, *I*, on block, *b*, at the current instance, *i*, is

$$I_b(i) = I_{(max)} \times E_b(i-1)\ . \qquad \textbf{(3.1)}$$

Where $I_{(max)}$ is the maximum number of subsets that can be selected on a block; and $E_b(i-1)$ is the error rate (number of block misclassifications) of the best-case subset individual from the previous instance, *i*, of block *b*. Hence, $E_b(i)=1-[hits_b(i)/\ \#exemplars(b)]$, where $hits_b(i)$ is the hit count over block *b* for the best-case individual identified over the entire RSS block at iteration *i* of block *b*; and *#exemplars*(*b*) is the total number of feature vectors in block *b*. To identify the 'best case' individual an array is kept over block *b* to record the hits for each individual in the population. After each tournament the hits of the parents are accumulated while the hits of the newly created individuals are set to zero. The best case individual over block *b* is therefore the individual with the maximum number of hits accumulated. Naturally, denoting the 'best case' individual in this manner prefers individuals that have been chosen for the level 2 subset more often, and so has the potential to miss better performing individuals which might reside in the population. However, an alternate scheme would reintroduce a significant computational cost of the inner loop.

Level 2 – Dynamic Subset Selection (DSS) A simplified version of DSS is employed in the second level of the selection hierarchy [7]. Once a block has been selected using the above RSS process, exemplars within the block are associated with an age and difficulty.

The age is the number of DSS selections since the exemplar last appeared in a DSS subset. The difficulty is the number of GP individuals that misclassified the exemplar the last time it appeared in the DSS subset. Following selection of a block at level 1, all ages are set to one and all difficulties are set to a worst-case difficulty (i.e. no persistence of exemplar difficulties or ages beyond a block selection). Exemplars appear in the DSS subset stochastically, with a fixed chance of being selected by age or difficulty (%difficulty = 100 – %age). Thus two roulette wheels exist per block, one is used to control the selection of exemplars with respect to age and the other difficulty, the roulette wheels being selected in proportion to the corresponding probability for age and difficulty. This process is repeated until the DSS subset is full (Program 3.1, Line 6), the age and difficulty of selected exemplars being reset to the initial values. Exemplars that were not selected from the block have their age incremented by one whereas their difficulties remain the same. Currently DSS uses a subset size of 50 exemplars, with the objective of reducing the number of computations associated with a particular fitness evaluation. Each DSS subset is kept for six steady state tournaments (4 individuals taking part per tournament) before reselection takes place with up to $I_{(b)}(i)$ selections per block (Program 3.1, Line5) – equation (3.1).
The use of the fixed probability of 70% for difficulty ensures that greater emphasis is given to exemplars that resist classification, while the 30% for age ensures that easier exemplars are also visited in an attempt to prevent over-learning.

3.3 DSS-DSS Hierarchy

The RSS-DSS algorithm makes the implicit assumption that all blocks are equally difficult. The following DSS-DSS algorithm relaxes this assumption. To do so, a block difficulty and age is introduced and used to bias the stochastic selection of blocks in proportion to their relative age and difficulty using roulette wheel selection. Thus, the probability of selecting block *i* is,

$$\text{Block}(i)_{\text{weight}} = \frac{\%\text{diff x Block}_{\text{diff}}(i)}{\sum_j (\text{Block}_{\text{diff}}(j))} + \frac{\%\text{age x Block}_{\text{age}}(i)}{\sum_j (\text{Block}_{\text{age}}(j))}$$

$$\text{P(block}(i)) = \frac{\text{Block}(i)_{\text{weight}}}{\square_{\text{j}}\,(\text{Block }(j)_{\text{ weight}})} \qquad (\mathbf{3.2})$$

Where %diff is the fixed difficulty weighting (70%) and %age the age weighting (100 - %diff); $\text{Block}_{\text{diff}}(i)$ and $\text{Block}_{\text{age}}(i)$ are the respective block difficulty and age for block *i*; and *j* indexes all blocks.

At initialization each block has an age of one and worst-case difficulty. Therefore, block selection will initially be uniform. The age of a block is the number of RSS block selections since last selected. The difficulty of a block takes the form of a weighted running average, thus persistent across block selections, or,

$$\text{Block}(i, 0) = \text{Block }(i - 1);$$
$$\text{Block}(i, \text{j}) = \square\ \textit{exemplar}_{\text{diff}}(j) + (1 - \square)\ \text{Block }(i, \text{j} - 1);\ \square\text{j}\ \square\ \{1, \ldots, \text{P}_{\text{subset}}\}$$
$$\text{Block}(i) = \text{Block}(i, \text{P}_{\text{subset}})$$

Where $\textit{exemplar}_{\text{diff}}(j)$ is the difficulty of exemplar *j*; j indexes all exemplars in the subset of P_{subset} exemplars; and, □ is a constant (0 < □ < 1), set to 0.1 in this case. The difficulty of a block will be updated before each new level 2-subset selection (Program 3.1 – Line 11 and before returning to Program 3.1 – Line 5). Since □ is small, each of the exemplars in subset have the ability to influence the overall block difficulty by a small amount. If the exemplar difficulty is high, the overall block difficulty will increase, whereas if the exemplar difficulty is small, the overall block difficulty will decrease. Level 2 of this hierarchy uses the same DSS process as the RSS-DSS hierarchy.

3.4 Balanced Block Hierarchy

For many datasets the distribution of class exemplars is not uniform, thus some classes are represented excessively while others are represented very infrequently. This is typically the case with the anomaly detection problem on computer networks such as the classification problem of the 10% KDD'99 dataset. Some attacks, such as Denial of Service, occur very frequently (the basic objective is to overload the target network node) making up almost 80% of the training data, while others, such as a probe or User to Root (U2R) attack, are very infrequent, and account for only a small percentage of the overall audit data [33]. The following tables (Tables 3.4.1 and 3.4.2) show the unbalanced distribution of each of the training datasets used in Chapter 4. Note that the Census Income dataset, the Shuttle 2 dataset and the Shuttle 4 dataset are particularly unbalanced.

Dataset	10%KDD'99	Adult	Census-Income
Class 0	19.69%	24.78%	5.76%
Class 1	80.31%	75.22%	94.24%

Table 3.4.1: Dataset Distributions

Shuttle	1	2	3	4
Class 0	20.84%	99.60%	85.14%	94.42%
Class 1	79.16%	0.40%	14.86%	5.58%

Table 3.4.2: Shuttle Datasets Distributions

One approach to the problem of unevenly distributed data might be to stratify the data such that each level one block had the same distribution as the original data [33]. Unfortunately, when some exemplar classes are very rare this scheme will also fail since rare exemplars are still encountered very infrequently.

The approach taken here is based on the previous DSS-DSS algorithm, but has the added goal of always having class 0 and class 1 exemplars represented in the Level 1 block at a fixed percentage. In this way both classes can always be represented regardless of the original size and distribution of the dataset. An optimal percentage of each class in the level 1 block now needs to be determined. This approach is termed the Balanced Block algorithm.

The first stage of the balanced block algorithm is to sort the dataset and to select the desired percentage of class 0 and class 1 exemplars to appear in the Level 1 balanced block. Once a percentage has been selected, say 25/75 (25% of the block class 0 and 75% of the block class 1), the class 0 and class 1 exemplars in the sorted dataset can be divided up into partitions matching the size of its block percentage. For example, a block size of 1000 with a 25/75 block partition ratio would require class 0 partitions of 250 exemplars and class 1 partitions of 750 exemplars. A block can now be created by selecting a partition from each class and combining the two partitions. Partitions are selected to be in the Level 1 block by the DSS algorithm, which biases selection by partition difficulty and partition age.

The net effect is that every block consists of a balanced set of exemplars, independent of the initial data distribution.

Program 3.2: Balanced Block Algorithm Level 2 Subset Selection

```
{
(1) for ( i < level 2 subset size )
    {
(2)             Select partition to choose exemplar from by roulette wheel selection
                over partition difficulty
(3)             Select whether to choose exemplars by age or difficulty by roulette
                wheel selection over the age/difficulty ratios
(4)             Select exemplar for subset by roulette wheel selection over total
                partition age or difficulty as chosen in line 3
    }
}
```

The level 2 subset selection algorithm has also been altered for this algorithm. Once the partitions have been selected for a level 1 block, exemplars are chosen for the level 2 subset by the algorithm in Program 3.2.

Program 3.3 - Genetic Programming with Balanced Block Hierarchy

```
{
(1) sort dataset into classes
(2) divide each class into partitions (level 0)
(3) initialize training system and population
(4) while ( BLOCKtermination == FALSE)
    {
(5)     conduct class 0 partition selection (level 1)
(6)     conduct class 1 partition selection (level 1)
(7)     combine partitions to form Block (level 1)
(8)     while (DSStermination == FALSE)
        {
(9)             conduct Balanced Block Subset Selection as in Program 3.2  (level 2)
(10)            while (TournamentEnd == FALSE)
                {
(11)                    conduct tournament selection
(12)                    train tournament individuals on subset
(13)                    update exemplar difficulty
(14)                    apply search operators
                }
        }
(15)    update #Subset selected at next block b instance
    }
(16) run all individuals on entire dataset
(17) run best hits and best performance individuals on test dataset
(18) record results for both
(19) remove introns and translate both
```

Program 3.3 outlines the relationship between the GP learning algorithm and the Balanced Block hierarchical sampling algorithm. An additional design decision for this algorithm is the choosing of a block partition ratio (i.e. 25%/75%, 50%/50%), which will also determine the number of partitions and the size of the partitions for each class, which are necessary in order to partition the classes at Level 0 of the hierarchy (Program 3.3 – Line 2).

Partition ages and difficulties are much like the block ages and difficulties introduced in the DSS-DSS algorithm (Section 3.3), and are used to bias the stochastic selection of partitions in proportion to their relative age and difficulty. The difficulty of partitions, like block difficulty, takes the form of a weighted running average, and is thus persistent across partition selections with a condition that only exemplars chosen from a partition are allowed to update that partition. In this way each exemplar chosen for the Level 2 subset has the ability to update the overall partition difficulty. The difficulty of a partition is updated before each new Level 2 subset selection (Program 3.3 – Line 9). Moreover, following the creation of a block at Level 1 of the hierarchy (Program 3.3 – Lines 5, 6 and 7), a history of training pressure on the two partitions of the block are used to determine the number of iterations to perform at Level 2 (Program 3.3 – Line 9) of the Balanced Block algorithm. The number of iterations performed on a block is calculated using the average of the two partition error rates as opposed to the block error rate of Equation 3.1.

4 Results

4.1 Experimental Setup

As indicated in the introduction, the principle interest of this work is in establishing a framework for applying Genetic Programming to large datasets. To this end experiments are reported using four large binary classification datasets: the KDD-99 Intrusion Detection dataset, taken from the 5th ACM SIGKDD Knowledge Discovery and Data Mining Competition (1999) [14]; and the Adult dataset, the Census Income dataset and the Shuttle dataset all taken from the UCI Machine Learning Repository [15].

The stochastic nature of the GP algorithm requires that runs be conducted over at least 30 different initializations in order to establish the statistical significance of the results, thus verifying that the solutions found are not due to random chance [40]. To this end, all of the experiments on these datasets are based on 40 GP runs using one of the Hierarchical Subset Selection algorithms and Dynamic page-based Linear-GP [12]. Runs differ only in their choice of random seeds used for initializing the population, with all other parameters remaining unchanged. 40 GP runs are used, instead of only the necessary 30 in order to establish statistical significance, due to the occurrence of degenerate solutions. Degenerate solutions are GP programs that label all exemplars as one class (i.e. label every exemplar class 1). They are of no interest and are removed if found in the results. Degenerate solutions seem more likely to occur on imbalanced datasets, several of which will be studied here. The number of degenerate solutions found is recorded so we can be sure that we have achieved 30 results in order to verify statistical significance.

Training was performed on a dual G4 1.33 GHz Mac Server with 2 GB RAM. For the experiments on the datasets mentioned above, the best individuals of each of the 40 GP runs are recorded in terms of two metrics: the Hits metric; and the TNR+TPR metric. The Hits metric is just a counter for the number of correct classifications of either class by a GP program (i.e., classification accuracy). To achieve the greatest number of hits the GP programs will usually concentrate on labeling the class that has a larger representation in the dataset. The trade off to this method is often a very poor accuracy on the smaller class.

The TNR+TPR metric stands for the true negative rate (TNR) plus the true positive rate (TPR). The true negative rate is the number of class 0 exemplars predicted correctly to be class 0 by the GP program divided by the total number of class 0 exemplars (TNR is equal to 1-false positive rate (1 - FPR), see Table 4.1.1 below). The true positive rate is the number of class 1 exemplars predicted correctly to be class 1 by the GP program divided by the total number of class 1 exemplars (TPR is the same as the detection rate (DR), see Table 4.1.1 below). This metric puts equal importance on achieving good classification accuracy on both classes of a binary classification dataset regardless of the size of each class. The TNR+TPR metric credits the accuracy of GP programs on the smaller class but the trade off is typically not detecting as much of the larger class resulting in a lower overall classification accuracy than the Hits metric.

Note that all training is performed with the equivalent of a ‘hits’ based count. Selection of the ‘best’ individual from a population is calculated post-training in terms of the Hits Metric and the TNR+TPR metric as evaluated across the training data.

The best individual program of a GP run is determined by running all of the programs in the population on the training data and selecting the programs the have the highest number of hits and the highest value for TNR+TPR.

The performance of each of the 40 best individuals is recorded in terms of run time in minutes on the training dataset, and accuracy (hits rate), detection rate (DR), and false positive rate (FPR) on both the training and test datasets and program size both before and after simplification. Furthermore, the best individuals programs are output and then translated. Detection rates and false positive rates are estimated as shown in Table 4.1.1.

Detection Rate = 1 – (# of False Negatives / Total # of Positives)
False Positive Rate = (# of False Positives / Total # of Negatives)

Table 4.1.1: Detection Rate and False Positive Rate

These results from the 40 best individuals in terms of the Hits metric and the TNR+TPR metric are then summarized in terms of the first, second (median) and third quartiles. The use of the mean (average) and variance would suffice if the data conformed to a normal distribution (which it rarely does). The normal distribution would need to be verified, which requires a sufficient number of points. However, the shape of the distribution can not be established here due to the computational overhead involved in conducting the hundreds of trials necessary to achieve the sufficient number of points. The use of quartiles, on the other hand, makes a minimum number of assumptions regarding the distribution of the data and is therefore considered a robust statistic. The spread of quartile plots are an indication of the degree of variation in the results, with tighter quartiles representing greater consistency of the solution.

Removal of Structural Introns – Once evolution is complete, programs can be simplified by the removal of structural introns [17]. As mentioned in Chapter 2, introns are dispensable instructions that do not have any effect on the output of the program. Structural introns, in particular, are single instructions that emerge from manipulating variables that are not used in the sequence of instructions contributing to the calculation of the output [17]. Introns do play an important role in protecting effective code from being disrupted by genetic operators, such as crossover, during GP training, but once training is complete these instructions can be safely removed.

GP Parameters – Table 4.1.2 lists the common parameter settings for all GP runs.

	Parameter	Setting
1	Population size	125
2	Maximum # of pages	32
3	Page size	8 instructions
4	Maximum working page size	8 instructions
5	Crossover probability	0.9
6	Mutation probability	0.5
7	Swap probability	0.9
8	Tournament size	4
9	Number of Registers	8
10	Instruction type 1 probability	0.5/5.5
11	Instruction type 2 probability	4/5.5
12	Instruction type 3 probability	1/5.5
13	Function set	{+,-,*,/}
14	Terminal set	{0, …, 255} □ {exemplar features}
15	Level 2 subset size	50
16	Block selection iterations	1000
17	Max subset selection iterations (6 tournaments/iteration)	100
18	Wrapper function	0 if output □ 0, otherwise 1
19	Cost function	Increment by 1/(# in class) for each misclassification

Table 4.1.2: Parameter Settings for Dynamic Page-based Linear GP

The first 14 parameters are standard GP parameters [12] while parameters 15, 16 and 17 define the number of epochs before training stops. The 18th parameter merely reflects the problem type – classification – and is used to map the continuous GP output into a binary classifier. The last parameter, 19, is the 'hits' count used to measure fitness during training.

Instruction Set – The GP instructions employ a 2-address format in which provision is made for: up to 16 internal registers, up to 64 inputs (Terminal Set), 5 opcodes (Functional Set) – the fifth is retained for a reserved word denoting end of program – and an 8-bit integer field representing constants (0-255) [12]. Two mode bits toggle between one of three instruction types: opcode with internal register reference; opcode with reference to input; target register with integer constant. Extension to include further inputs or internal register merely increases the size of the associated instruction field. The output is taken from the best performing register.

In addition to the hierarchical subset selection algorithms the C5.0 tree induction algorithm was run on each of the datasets tested here (see [39]). C5.0 provides an alternative machine-learning algorithm for comparative purposes. C5.0 was chosen since it is readily available, it is able to handle large datasets and it is easy to use. In addition to a vanilla C5.0 run various pruning parameterizations were attempted in order to reduce the number of tree nodes and/or rules returned by the algorithm.

The rest of Chapter 4 investigates the performance of the RSS-DSS, the DSS-DSS and the Balanced Block algorithm on the Adult dataset (4.2), the 10% KDD'99 dataset (4.3), the Census Income dataset (4.4) and the Shuttle dataset (4.5). Each section contains a parameterization of the Balanced Block algorithm in terms of the optimal ratio of the Level 1 block into class partitions. This optimal block partition ratio is then used in the comparison to the other algorithms. Section 4.2 uses the Adult dataset to further investigate the parameterization of the RSS-DSS, DSS-DSS and the Balanced Block algorithms.

4.2 The Adult Dataset

The Adult Dataset represents a prediction problem taken from the 1994 Census Bureau database and can be found in the UCI Machine Learning Repository [15]. It is the smallest dataset considered here, with a training set of over 30,000 exemplars. The learning task is to predict whether a person's income exceeds $50,000 per year based on 14 census data features. Each exemplar represents a person and is made up of 14 personal and demographic characteristics plus a label. All exemplars with missing features were removed from both the training and test data. The data distribution for the Adult dataset is approximately 25% class 0 and 75% class 1 for both training and test sets, Table 4.2.1.

The Adult Dataset	Training		Test	
Class 0 (>$50K)	7474	24.78%	3700	24.57%
Class 1 (<=$50K)	22688	75.22%	11360	75.43%
Total	30162	100%	15060	100%

Table 4.2.1: The Adult Dataset and Data Distribution

In the following section, the Adult dataset is used to conduct a range of parameter sensitivity experiments. RSS-DSS and DSS-DSS parameterization is presented in 4.2.1 while Balanced Block parameterization is presented in 4.2.2.

4.2.1 RSS-DSS and DSS-DSS Algorithm Parameterization

4.2.1.1 Block Sizes

Recall from Chapter 3 that the RSS-DSS and DSS-DSS algorithms initially divide the large dataset into a series of blocks. The block size is chosen so that a single block can fit entirely within RAM alone. However, within the limitation of fitting in RAM it is not known what the optimal block size should be. To this end, experiments with different block sizes were run for the RSS-DSS and DSS-DSS algorithms. Adult dataset block sizes and the corresponding number of blocks can be found in Table 4.2.2. Note that for each dataset the final block will not contain the full block size but the remaining number of exemplars.

Block Size	250	500	750	1000	2500	5000
# of blocks	121	61	41	31	13	7

Table 4.2.2: Adult Dataset Block Sizes and Number of Blocks

Tables 4.2.3 and 4.2.4 list median results in terms of run time in minutes, test accuracy, test detection rate and program size after intron removal for both the Hits metric and the TNR+TPR metric. Full results, including first, second (median) and third quartiles as well as training results can be found in Appendix A, Tables A.1 and A.2.

RSS-DSS						
Block Size	**250**	**500**	**750**	**1000**	**2500**	**5000**
Time	7.26	8.06	7.22	8.69	9.19	12.35
Hits Metric						
Accuracy	80.59%	81.00%	79.42%	80.77%	79.52%	79.21%
DR	0.9229	0.9197	0.9578	0.9148	0.9297	0.9601
FPR	0.5181	0.4918	0.7108	0.4953	0.6159	0.7541
TNR+TPR	1.4048	1.4279	1.2470	1.4195	1.3138	1.2060
Size	89	79	58	91	51.5	63
TNR+TPR Metric						
Accuracy	75.13%	75.20%	73.48%	76.16%	72.93%	73.87%
DR	0.7158	0.7140	0.6849	0.7276	0.6809	0.6934
FPR	0.1368	0.1338	0.1138	0.1377	0.1211	0.1212
TNR+TPR	1.5790	1.5802	1.5711	1.5899	1.5598	1.5722
Size	75	81	65	76.5	53	59

Table 4.2.3: Median RSS-DSS Results for Differing Block Sizes

DSS-DSS						
Block Size	**250**	**500**	**750**	**1000**	**2500**	**5000**
Time	7.45	7.98	7.44	7.88	9.64	11.20
Hits Metric						
Accuracy	80.83%	80.60%	80.81%	79.64%	79.79%	79.58%
DR	0.9244	0.9325	0.9327	0.9293	0.9404	0.9420
FPR	0.4828	0.5604	0.5554	0.6024	0.6143	0.6035
TNR+TPR	1.4416	1.3721	1.3773	1.3269	1.3261	1.3385
Size	80	73	70	70.5	65.5	48.5
TNR+TPR Metric						
Accuracy	75.64%	75.08%	74.46%	74.53%	74.77%	74.56%
DR	0.7188	0.7136	0.7082	0.7066	0.7069	0.7004
FPR	0.1300	0.1342	0.1268	0.1305	0.1232	0.1276
TNR+TPR	1.5888	1.5794	1.5814	1.5761	1.5837	1.5728
Size	75.5	87	72	72	66.5	61

Table 4.2.4: Median DSS-DSS Results for Differing Block Sizes

Tables' 4.2.3 and 4.2.4 show that as the block size increases, then the run times of the algorithm seem to increase. The smaller block size resulting in a significant speedup in

CPU time mirrors cache block size design, in which larger blocks encounter a higher transfer penalty if it is not leveraged into an extended utilization.

It is also apparent that the Hits metric results in greater classification accuracy for both algorithms while the TNR+TPR metric not surprisingly has the greater TNR+TPR value. The Hits metric performs better in terms of detection rate while the TNR+TPR metric has a much better false positive rate. This coincides with our expectations for these metrics. In terms of test accuracy, test detection rate and test false positive rate; both algorithms appear to be quite robust across blocks sizes within each metric. Therefore, we will dismiss the larger block sizes of 2500 and 5000 as the optimal block size due to slower run times. All of the other block sizes compare well in terms of consistency (spread of quartiles – see Appendix A, Tables A.1 and A.2). Any of the other four block sizes can be chosen as optimal and a block size of 1000 is used throughout this section corresponding to our original parameterization of the algorithm. Upon reflection, it may have been better to choose a block size of 750 due to its superior median run times and program sizes after intron removal with all other results being comparable. However, no significance testing was conducted to verify a significant difference in results.

Block Size	**250**	**500**	**750**	**1000**	**2500**	**5000**
RSS-DSS	16.95	16.95	17.56	16.99	16.72	17.56
DSS-DSS	16.46	16.63	16.95	16.95	17.06	17.03

Table 4.2.5: Best Case Adult Dataset Error Rates

Block Size	**250**	**500**	**750**	**1000**	**2500**	**5000**
RSS-DSS	1.6085	1.6069	1.6026	1.6058	1.6107	1.6061
DSS-DSS	1.6088	1.6070	1.6086	1.6011	1.6042	1.6071

Table 4.2.6: Best Case Adult Dataset TNR+TPR Values

Tables 4.2.5 and 4.2.6 list the error rates of the best case RSS-DSS and DSS-DSS algorithms with varying block sizes in terms of both error rate and TNR+TPR values. For the DSS-DSS algorithm as block size decreases the error rate also tends to decrease while in most cases the DSS-DSS error rate was better than the RSS-DSS error rate. For TNR+TPR values there does not seem to be a trend with a decrease of block size but the DSS-DSS algorithm does tend to find a marginally better solution in most cases.

4.2.1.2 Block and Exemplar Age/Difficulty Ratios

Table 4.2.7 list the median results for varying age/difficulty ratios for the RSS-DSS algorithm for both the Hits metric and the TNR+TPR metric.

Recall that exemplars appear in the DSS subset stochastically, with a fixed chance of being selected by age or difficulty. For example, a 30/70 age/difficulty ratio means that for each exemplar to be selected for the DSS subset there is a 30% chance of being selected by age and a 70% chance of being selected by difficulty. Age/difficulty ratios of 30/70, 10/90 and 0/100 were run, although the other extreme of 100/0 was not tested. The 10/90 ratio decreases the probability of being selected by age and increases the probability of being selected by difficulty. The 0/100 ratio means that all exemplars are chosen for the DSS subset by difficulty not taking age in to account at all. The 0/100 ratio is similar to boosting algorithms in wide spread use which are based on difficulty alone. Full results for this experiment can be found in Appendix A, Table A.3. These results show that the three ratios seem to be comparable in terms of consistency (spread of quartiles in Appendix A, Table A.3).

RSS-DSS			
Exemplar Age/Diff	**30/70**	**10/90**	**0/100**
Time	8.58	8.25	8.21
Hits Metric			
Test Accuracy	80.77	79.40	79.64
Test DR	0.9148	0.9518	0.9491
Test FPR	0.4953	0.7061	0.6408
TNR+TPR	1.4195	1.2457	1.3083
Program Size	91	76	77
TNR+TPR Metric			
Test Accuracy	76.16%	74.36%	74.32%
Test DR	0.7276	0.7030	0.6988
Test FPR	0.1377	0.1265	0.1242
TNR+TPR	1.5899	1.5765	1.5746
Program Size	76.5	79.5	77.5

Table 4.2.7: Median RSS-DSS Exemplar Age/Difficulty Ratios

Table 4.2.7 shows that the 0/100 ratio had the lowest median run time. This could be a result of removing the first roulette wheel based selection between age and difficulty in choosing an exemplar for the DSS subset.

The 30/70 age/difficulty ratio achieves the best median test accuracy and the best TNR+TPR value for both the Hits metric and the TNR+TPR metric. The 10/90 ratio achieves a higher median test accuracy than the 0/100 ratio but a worse TNR+TPR value for the Hits metric, while outperforming the 0/100 ratio for both values for the TNR+TPR metric. These results suggest that it is indeed beneficial to consider the age of exemplars when deciding which exemplars to choose for the DSS subset.

Table 4.2.8 shows the median results of varying the ratio of choosing blocks by age and difficulty as well as varying the ratio of choosing exemplars by age and difficulty (as in Table 4.2.7).

DSS-DSS									
Block Age/Diff	**30/70**			**10/90**			**0/100**		
Exemp. Age/Diff	**30/70**	**10/90**	**0/100**	**30/70**	**10/90**	**0/100**	**30/70**	**10/90**	**0/100**
Time	7.04	7.45	7.71	7.13	7.22	7.06	6.98	7.33	7.06
Hits Metric									
Acc.	79.64	79.29	80.27	80.53	80.13	80.02	79.63	79.49	79.89
DR	0.929	0.959	0.929	0.936	0.930	0.942	0.959	0.959	0.941
FPR	0.602	0.733	0.528	0.557	0.538	0.610	0.740	0.738	0.578
Size	70.5	69	85	83.5	72	80	64.5	75.5	72.5
TNR+TPR Metric									
Acc.	74.44	73.49	75.40	75.27	75.14	74.47	74.35	73.05	74.32
DR	0.706	0.690	0.720	0.714	0.714	0.703	0.703	0.679	0.704
FPR	0.129	0.126	0.137	0.138	0.131	0.128	0.126	0.120	0.125
Size	70.5	74	72	72.5	69.5	67.5	65.5	75.5	75

Table 4.2.8: Median DSS-DSS Block and Exemplar Age/Difficulty Ratios

Unlike the exemplar age/difficulty ratio the block ratio is used to determine a weight for each block in proportion to its age and difficulty. Roulette wheel selection of blocks is then applied over all block weights. Varying the block age/difficulty ratio will therefore alter the effect of block age and block difficulty on each blocks weighting. Block ratios of 30/70, 10/90 and 0/100 were tested each with corresponding exemplar ratios of 30/70, 10/90 and 0/100. Full results of this experiment can be found in Appendix A, Table A.4. The median results shown in Table 4.2.8 are surprisingly similar across the table with time, test accuracy, test detection rate and test false positive rates differing by a very small margin. Consulting the quartiles in Appendix A, Table A.4 show that all of the results seem to be comparably consistent as well. This suggests that the DSS-DSS algorithm is actually quite robust in terms of both block age/difficulty ratio and exemplar age/difficulty ratio.

4.2.2 Dataset Manipulation

4.2.2.1 Sorting the Adult Dataset

Leading up to the development of the Balanced Block algorithm, experiments were run to test the effect of sorting the dataset before application of the DSS-DSS algorithm. Two block sizes were used: 1000 (31 blocks) and 5000 (7 blocks). Sorting the dataset creates only one block in between the two classes in which both classes are represented.

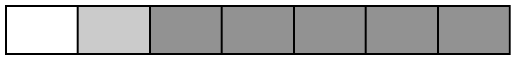
Figure 4.2.1: Sorted Adult Dataset with 7 Blocks

Figure 4.2.1 shows the sorted Adult dataset divided into seven blocks of size 5000. Block one is made up entirely of class 0 exemplars (white block) while blocks three through seven are made up entirely of class 1 exemplars (dark blocks). Only block two has exemplars from both classes (grey block – with 2474 class 0 and 2526 class 1).

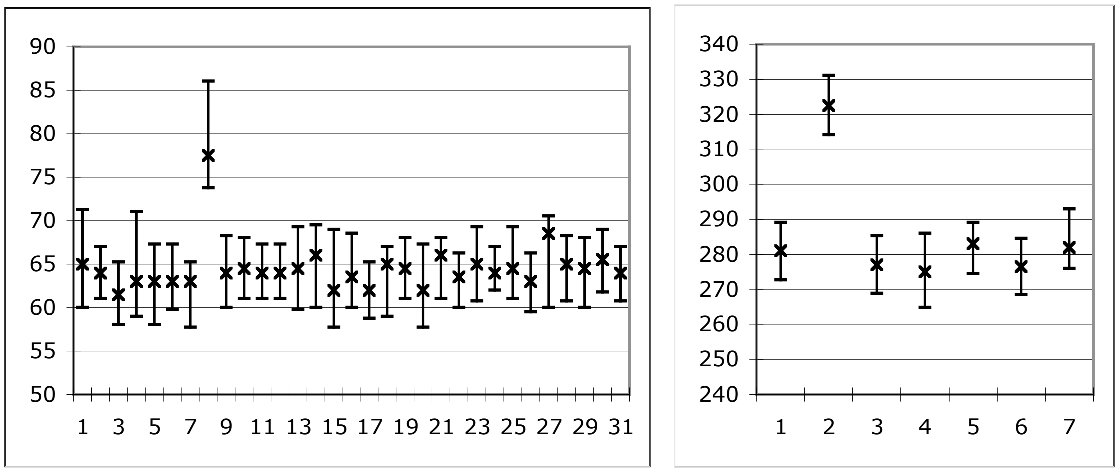

Figure 4.2.2: Quartile Plots of Block Selection Frequencies on Sorted Adult Dataset for Block Sizes of 1000 (31 Blocks) and 5000 (7 Blocks).

The frequencies shown in Figure 4.2.2 show that the single mixed block (block 8 for block sizes of 1000, and block 2 for block size of 5000) is chosen more often for both block sizes. It seems logical that it would be harder for the algorithm to do well on a block that has both class 0 and class 1 unless it can learn to solve both classes. This suggests that the mixed block would consistently be the most difficult and it is indeed chosen more often by the DSS-DSS algorithm. This verifies that the DSS-DSS algorithm is functioning as anticipated.

Table 4.2.9 lists the median results of running the DSS-DSS algorithm on the sorted Adult dataset. Full results of this experiment can be found in Appendix A, Table A.5.

Block Size	1000	5000
Run Time (mins)	2.34	7.38
Hits Metric		
Accuracy	77.48	77.92
Detection Rate	0.9587	0.9618
False Positive Rate	0.7892	0.7892
TNR+TPR	1.1695	1.1726
Program Size	34	51
TNR+TPR Metric		
Accuracy	71.05	70.48
Detection Rate	0.6618	0.6415
False Positive Rate	0.1399	0.1039
TNR+TPR	1.5219	1.5376
Program Size	24.5	41.5

Table 4.2.9: Median DSS-DSS Results on Sorted Adult Dataset

The sorting of the Adult dataset has resulted in faster median run times and usually a smaller median program size than for DSS-DSS for block sizes of 1000 and 5000 on the unsorted Adult dataset (see Table 4.2.4). However, in almost all other instances, including accuracy and TNR+TPR value, the sorted median results were worse than those of the unsorted results (except for DR for the Hits metric).

Dataset	Sorted Adult		Unsorted Adult	
Block Size	1000	5000	1000	5000
Best Accuracy	80.33	79.40	83.05	82.97
Best TNR+TPR	1.5512	1.5711	1.6011	1.6071

Table 4.2.10: Best Case DSS-DSS Results on Sorted and Unsorted Adult Dataset

Table 4.2.10 shows the best accuracy and the best TNR+TPR value for DSS-DSS algorithm with block sizes of 1000 and 5000 on both the sorted and unsorted Adult datasets. It is readily apparent that the unsorted Adult dataset results are superior in both accuracy and TNR+TPR. The basic hypothesis here is that blocks of the same exemplar class encourage degenerate GP solutions, thus impacting the detector quality as a whole.

4.2.2.2 Manually Mixing the Adult Dataset

Further testing was conducted on the DSS-DSS algorithm where the dataset was initially separated into its respective classes. Creating blocks with a fixed proportion of class 0 exemplars and class 1 exemplars then formed a new dataset. When the smaller class ran out, the remainder of the larger class was put together at the end of the new dataset. Different block proportions were attempted for these manually mixed datasets as shown in Table 4.2.11.

Dataset	Block Size	Num Blocks	Breakdown
Mixed 1	1000	31	15 blocks 50/50, 15 blocks class 1, 1 block rest
Mixed 2	1000	31	30 blocks 25/75, 1 block rest
Mixed 3	5000	7	3 blocks 50/50, 3 blocks class 1, 1 block rest
Mixed 4	5000	7	6 blocks 25/75, 1 block rest

Table 4.2.11: Manually Mixed Adult Datasets

Figure 4.2.3: Mixed 3 – Manually Mixed Adult Dataset

Figure 4.2.3 shows an example of a manually mixed Adult dataset. This is the Mixed 3 dataset which is the third dataset shown in Table 4.2.11 with 3 blocks mixed (blocks 1 to 3 shown in grey), followed by three blocks of class 1 exemplars (blocks 4 to 6 shown in dark), followed by a single block of the remainder of the dataset (block 7 shown in grey). Table 4.2.12 shows the median results of the DSS-DSS algorithm on the four mixed Adult datasets of Table 4.2.11. Full results can be found in Appendix A, Table A.6. The Mixed 2 and the Mixed 4 dataset both have all of their blocks representing both class 0 and class 1 exemplars. The DSS-DSS algorithm seems to take longer on these two datasets as compared to Mixed 1 and Mixed 3 respectively. However, the Mixed 2 and Mixed 4 results show an improvement over the results of Mixed 1 and Mixed 3, which only have half of their block with representatives of both classes. The greater quantity of mixed blocks results in higher median accuracies and median TNR+TPR values for both

Block Size	1000		5000	
Dataset	Mixed 1	Mixed 2	Mixed 3	Mixed 4
Run Time (mins)	4.89	7.73	8.68	11.81
Hits Metric				
Accuracy	77.71	79.15	77.77	79.70
Detection Rate	0.9586	0.9569	0.9586	0.9413
False Positive Rate	0.7892	0.7215	0.7892	0.6073
TNR+TPR	1.1694	1.2354	1.1694	1.3340
Program Size	63.5	62.5	46	71
TNR+TPR Metric				
Accuracy	71.11	72.62	70.64	74.73
Detection Rate	0.6506	0.6746	0.6439	0.7056
False Positive Rate	0.1014	0.1208	0.1011	0.1315
TNR+TPR	1.5492	1.5538	1.5428	1.5741
Program Size	59	56	33	73.5

Table 4.2.12: Median DSS-DSS Results on Mixed Adult Datasets

block sizes of 1000 and 5000, and for both the Hits metric and the TNR+TPR metric. Furthermore, the results from the manually mixed Adult dataset show a definite improvement over the results of the previous section's sorted Adult dataset.

Block Size	1000			5000		
Dataset	Mixed 1	Mixed 2	UnMixed	Mixed 3	Mixed 4	UnMixed
Best Hits	81.02	83.14	83.05	79.87	83.02	82.97
Best Performance	1.5729	1.6057	1.6011	1.5802	1.6017	1.6071

Table 4.2.13: Best Case DSS-DSS Results on Mixed Adult Datasets
and UnMixed Adult Dataset

Table 4.2.13 shows the best results found by the DSS-DSS algorithm on the four mixed datasets as well as the best results on the unmixed Adult dataset for comparison. The Mixed 2 (Mixed 4) dataset with all blocks mixed again shows better results in terms of best accuracy and best TNR+TPR value than Mixed 1 (Mixed 3). Furthermore, the best Mixed 2 results outperform the best results found on the unmixed Adult dataset with block size of 1000, while the best Mixed 4 results have a higher accuracy than the unmixed Adult with block size 5000 and a comparable TNR+TPR value.
The improvement in performance of the datasets in which each block has a mixture of class 0 and class 1 exemplars helped to motivate the Balanced Block algorithm.

4.2.3 The Balanced Block Algorithm Parameterization

This section takes a closer look at the Balanced Block algorithm. In 4.2.3.1 the result of varying the □ parameter is investigated. Recall that the □ parameter controls the amount of influence that the difficulty of each individual exemplar chosen to be in the level 2 DSS subset has over its' partition difficulty (partition difficulty is calculated as a running average similarly to block difficulty in the DSS-DSS algorithm see 3.3). A larger □ increases the influence each exemplar has on the partition difficulty while a smaller □ decreases this influence.

In 4.2.3.2 the difficulties of the two partitions making up the level 1 balanced block and the number of exemplars from each class in the level 2 DSS subset are listed for the first 20 DSS iterations. This table is investigated to confirm whether the balanced block algorithm is working as expected.

In 4.2.3.3 several block partition ratios are investigated to discover the optimal setting for the balanced block algorithm on the Adult dataset. The block partition ratio controls the proportion of exemplars from each class that make up the level 1 block (e.g., a 25/75 block partition ratio denotes 25% of the level 1 block to class 0 exemplars and 75% of the block to class 1 exemplars).

4.2.3.1 □ Parameter

Tables' 4.2.14 and 4.2.15 show the best results in terms of the Hits metric and the TNR+TPR metric for the Balanced Block algorithm with various □ parameters and various block partition ratios. Recall that the □ parameter is used to update the running average of partition difficulties from exemplars chosen to be in the DSS subset from that particular partition.

HITS Metric		□		
		0.05	0.1	0.2
Block Partition Ratio	10/90	83.294	83.234	82.636
	15/85	82.417	82.444	82.889
	25/75	82.058	82.311	82.875
	30/70	81.833	81.700	82.072
	37.5/62.5	83.572	82.689	82.191
	50/50	83.061	82.855	82.616
	62.5/37.5	82.503	81.985	81.580

Table 4.2.14: Best Case Accuracies for Balanced Block Algorithm □ Parameters using Hits Metric

TNR+TPR Metric		□		
		0.05	0.1	0.2
Block Partition Ratio	10/90	1.5986	1.5941	1.5930
	15/85	1.5994	1.6007	1.6004
	25/75	1.6004	1.5921	1.5947
	30/70	1.5997	1.5998	1.5969
	37.5/62.5	1.6053	1.5899	1.5761
	50/50	1.6004	1.6031	1.5981
	62.5/37.5	1.6044	1.5921	1.5841

Table 4.2.15: Best Case TNR+TPR for Balanced Block Algorithm
□ Parameters using TNR+TPR Metric

For both the Hits and the TNR+TPR metrics the best combination was an □ of 0.05 and a block partition ratio of 37.5/62.5. However, all of the best hits and performance measures were very close together, differing by only a small amount, suggesting that the algorithm is in fact quite robust in terms of these parameters.

4.2.3.2 Balanced Block DSS Selection

Once partitions have been selected to create a 1st level block of the Balanced Block algorithm the DSS subset needs to be filled. The subset is filled through choosing exemplars from one of the two partitions. A partition is chosen through a roulette wheel selection over partition difficulties. Table 4.2.16 shows the first 20 iterations of DSS for the first block created by the Balanced Block algorithm on the Adult dataset. The second and third columns of Table 4.2.16 show the partition difficulties over the 20 rounds with both partition difficulties originally starting at 25. The fourth and fifth columns show each partition's percentage of the total partition difficulty. The sixth and seventh columns show how many exemplars from each class were chosen for the DSS subset (a total of 50 exemplars per DSS subset) and columns eight and nine show the percentage of each exemplar in the DSS subset. The selection of a partition from which to choose an exemplar for the DSS subset is a stochastic process and so the proportion of total partition difficulty and the proportion of each class in the DSS subset should be similar though not necessarily the same. An examination of Table 4.2.16 shows that this is indeed the case, verifying that the Balanced Block algorithm is working as expected.

#	Partition Difficulty		% of Total Diff		# Class		% of Total	
	Class 0	Class 1	Class 0	Class 1	0's	1's	0's	1's
1	25.00	25.00	50.00	50.00	20	30	40	60
2	4.76	22.87	17.22	82.78	12	38	24	76
3	11.43	6.55	63.58	36.42	28	22	56	44
4	2.25	21.59	9.43	90.57	3	47	6	94
5	8.41	1.15	88.02	11.98	40	10	80	20
6	1.49	16.06	8.51	91.49	4	46	8	92
7	8.00	1.43	84.84	15.16	45	5	90	10
8	2.63	10.44	20.14	79.86	13	37	26	74
9	10.43	7.02	59.78	40.22	33	17	66	34
10	1.73	20.04	7.95	92.05	4	46	8	92
11	8.39	1.45	85.25	14.75	39	11	78	22
12	1.84	16.64	9.97	90.03	5	45	10	90
13	10.22	1.16	89.79	10.21	45	5	90	10
14	1.22	10.69	10.28	89.72	9	41	18	82
15	9.88	4.70	67.78	32.22	35	15	70	30
16	1.37	19.16	6.67	93.33	1	49	2	98
17	3.73	1.10	77.17	22.83	42	8	84	16
18	1.04	14.44	6.69	93.31	2	48	4	96
19	5.59	1.16	82.87	17.13	46	4	92	8
20	1.34	8.81	13.17	86.83	9	41	18	82

Table 4.2.16: Balanced Block DSS Selection

The partition difficulties in columns two and three of Table 4.2.16 do seem to fluctuate a good deal between each DSS round. This seems to suggest that exemplars may have too much influence over the partition difficulties at each DSS round. The amount of influence exemplars have over the partition difficulty is controlled by the □ parameter, which was set to 0.1 in this experiment. However, in the previous subsection experiments with different □ parameters did not seem to have a very large affect on the outcome of the GP experiments.

4.2.3.3 Balanced Block – Block Partition Ratios

Table 4.2.17 show the corresponding sizes and number of partitions for each of the block partition ratios that will be used to parameterize the Balanced Block algorithm below. A block partition ratio of 10/90 indicating that Level 1 blocks are made up of 10% class 0 and 90% class 1. For the block size of 1000 used in this experiment this means that the class 0 partitions are of size 100 and class 1 partitions are of size 900.

Block Ratios		10/90	15/85	25/75	30/70	37.5/62.5	50/50	62.5/37.5
Zero Partitions	Size	100	150	250	300	375	500	625
	#	75	50	30	25	20	15	12
One Partitions	Size	900	850	750	700	625	500	375
	#	26	27	31	33	37	46	61

Table 4.2.17: Balanced Block Algorithm Adult Dataset – Number of Partitions and Size of Partitions

Block partition ratios have been chosen to try to ensure a good number of partitions for each class, and to try to fit as many exemplars into partitions as possible for each class (extra exemplars are put into a final partition which is usually much smaller than the other partitions) and just to test a variety of different block partition ratios.

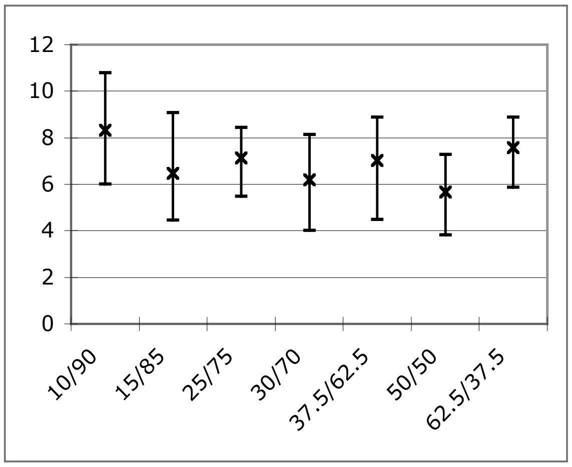

Figure 4.2.4: Adult Dataset Balanced Block Partition Ratios – Run Time (minutes)

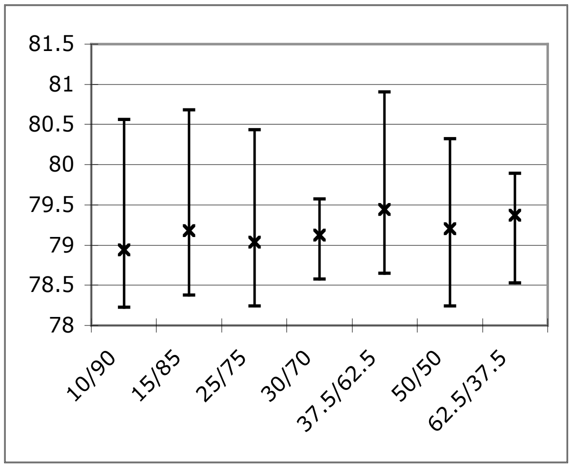

Figure 4.2.5: Adult Dataset Balanced Block Partition Ratios – Test Accuracy Hits Metric

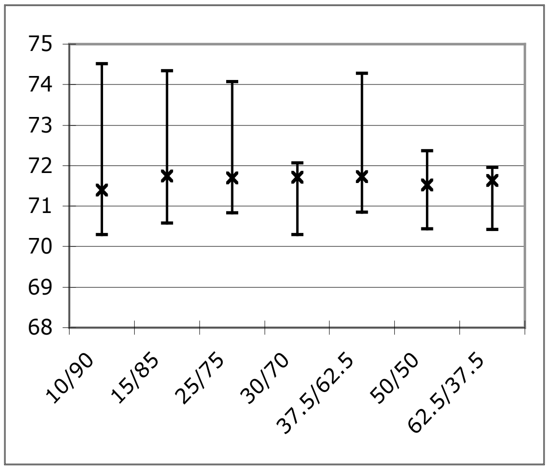

Figure 4.2.6: Adult Dataset Balanced Block Partition Ratios – Test Accuracy TNR+TPR Metric

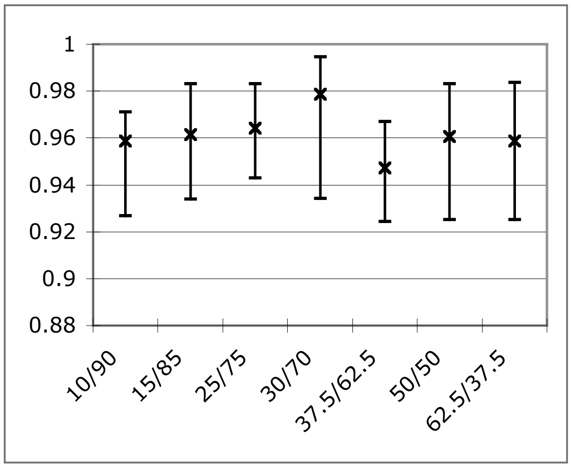

Figure 4.2.7: Adult Dataset Balanced Block Partition Ratios – Test Detection Rate Hits Metric

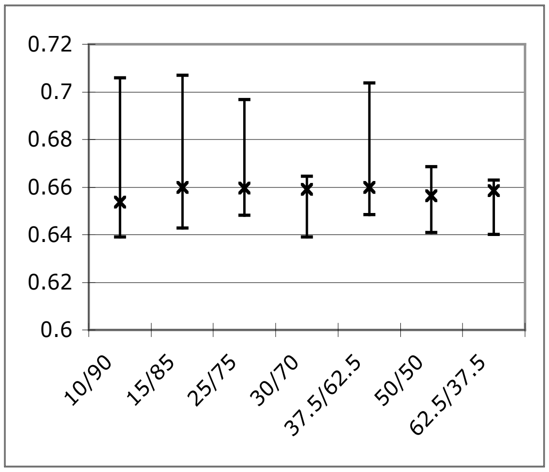

Figure 4.2.8: Adult Dataset Balanced Block Partition Ratios – Test Detection Rate TNR+TPR Metric

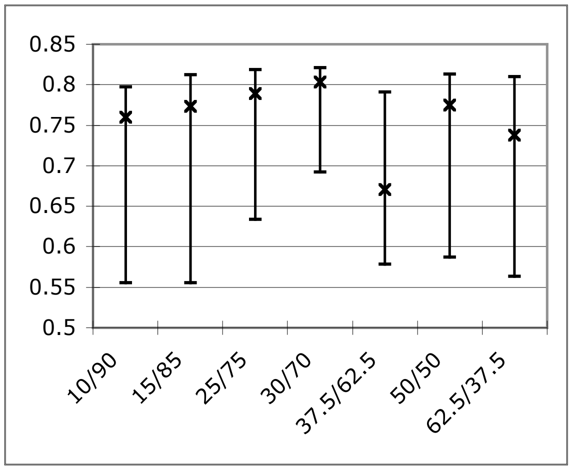

Figure 4.2.9: Adult Dataset Balanced Block Partition Ratios – Test False Positive Rate Hits Metric

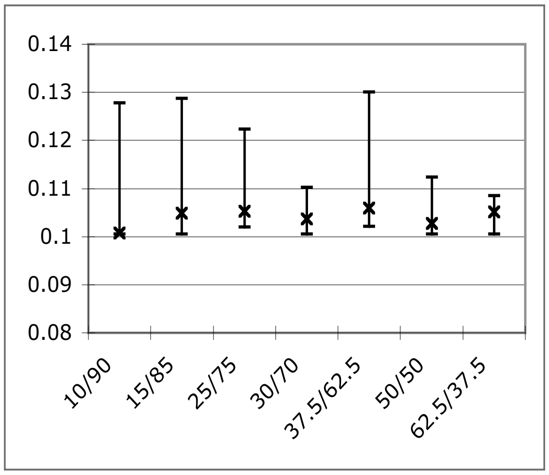

Figure 4.2.10: Adult Dataset Balanced Block Partition Ratios – Test False Positive Rate TNR+TPR Metric

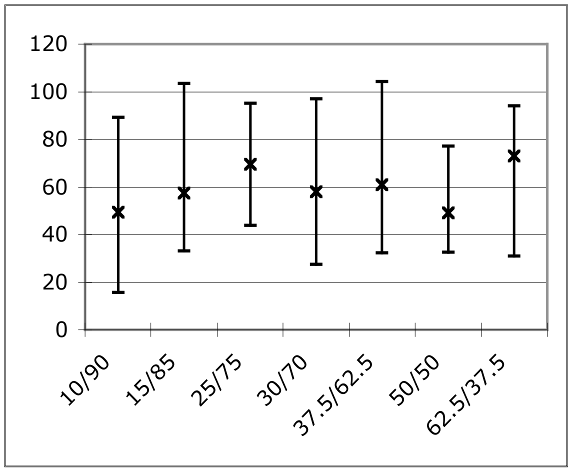

Figure 4.2.11: Adult Dataset Balanced Block Partition Ratios – Program Size after Intron Removal Hits Metric

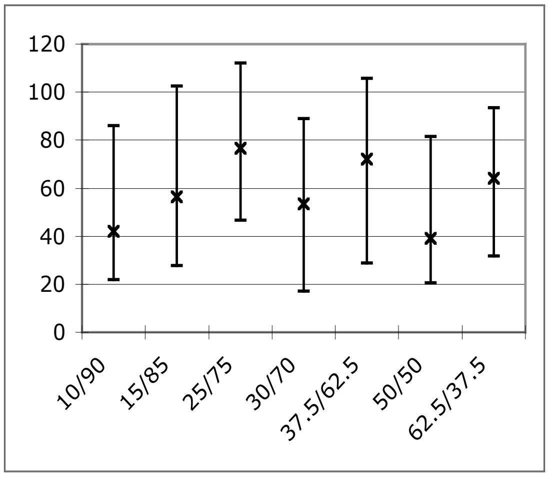

Figure 4.2.12: Adult Dataset Balanced Block Partition Ratios – Program Size after Intron Removal TNR+TPR Metric

Figures 4.2.4 to 4.2.12 show quartile plots of the results of the Balanced Block algorithm with varying block partition ratios on the Adult dataset. Results are reported in terms of run time, test accuracy, test detection rate, test false positive rate and program size after intron removal for both the Hits metric and the TNR+TPR metric. Full results from this experiment can be found in Appendix A, Table A.7.

The results shown in Figures 4.2.4 to 4.2.12 show that median values across partition ratios are, for the most part, quite similar. However, the spread of the quartile plots are quite different. For the Hits metric the 30/70 ratio seems to be the best partitioning it has the highest median detection rate but in a classic trade-off has the worst false positive rate. However, this partitioning is always competitive and is the most consistent of all the partitions (smallest spread in quartile plots). For the TNR+TPR metric the 30/70 ratio again seems to be the best partitioning being among the lowest median run times, the highest median test accuracies, the highest median detection rates, the lowest median false positive rates (lower is better) and in the middle in terms of program size after intron removal. However, again the 30/70 partitioning appears to be the most consistent of all the partitions (smallest spread in quartile plots).

Therefore, the 30/70 partitioning will be chosen as the best parameterization of the Balanced Block algorithm on the Adult dataset and will be used for comparison against the RSS-DSS and DSS-DSS algorithms in the next section.

4.2.4 GP Algorithm Comparison

Figures 4.2.13 to 4.2.21 below show quartile plots of the results for the RSS-DSS, the DSS-DSS and the Balanced Block algorithm with block partition ratio of 30/70 on the Adult Dataset. Results are summarized in terms of run time, test accuracy, test detection rate, test false positive rate and program size after intron removal for both the Hits metric (left figures) and the TNR+TPR metric (right figures). Full results may be found in Appendix A, Table A.8.

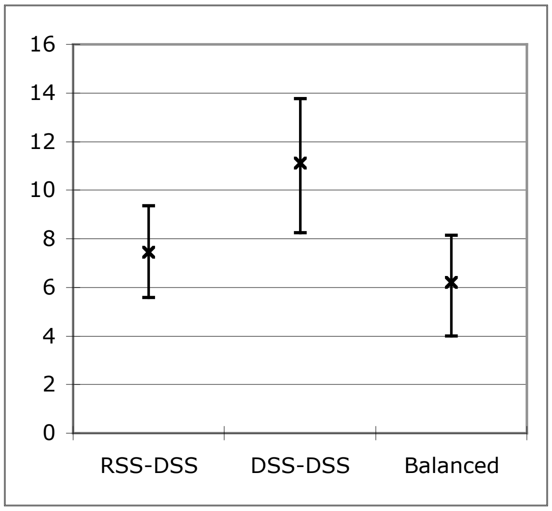

Figure 4.2.13: Adult Dataset – Run Time (minutes)

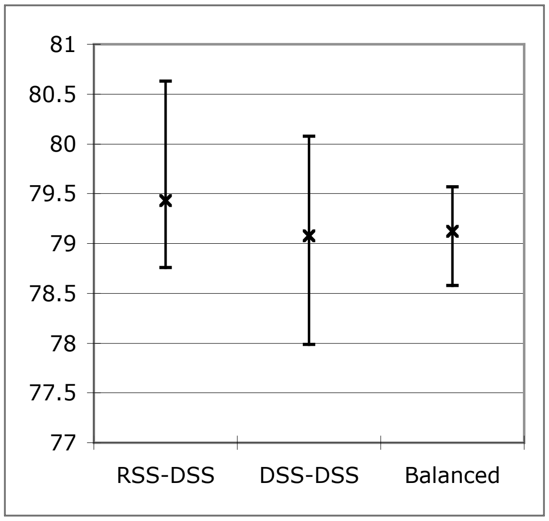

Figure 4.2.14: Adult Dataset – Test Accuracy Hits Metric

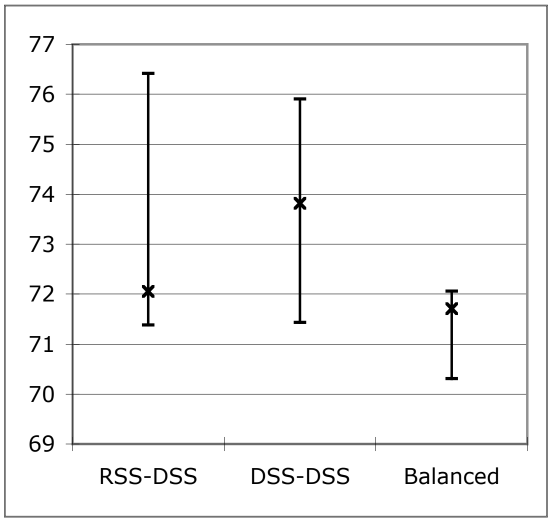

Figure 4.2.15: Adult Dataset – Test Accuracy TNR+TPR Metric

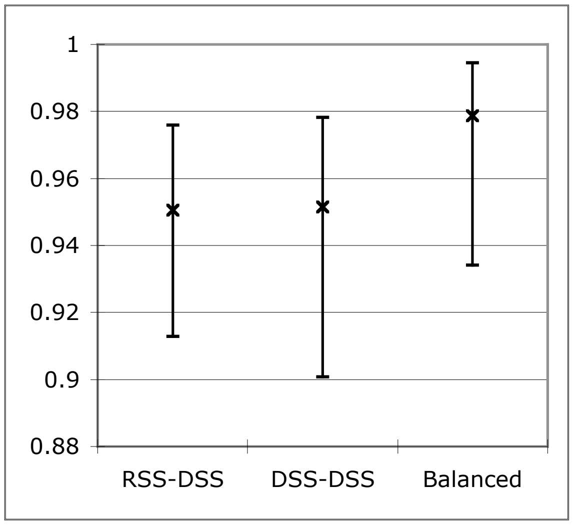

Figure 4.2.16: Adult Dataset – Test Detection Rate Hits Metric

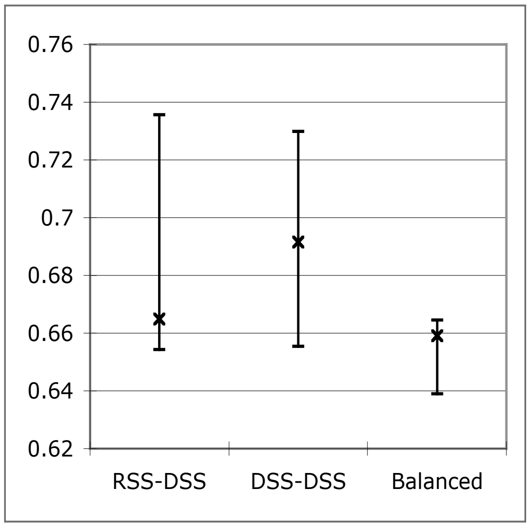

Figure 4.2.17: Adult Dataset – Test Detection Rate TNR+TPR Metric

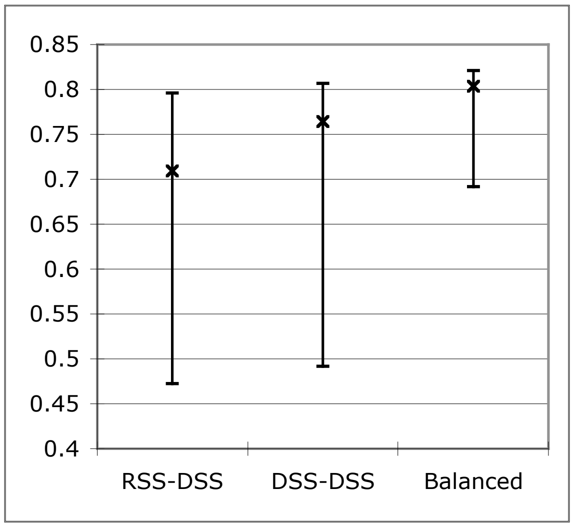

Figure 4.2.18: Adult Dataset – Test False Positive Rate Hits Metric

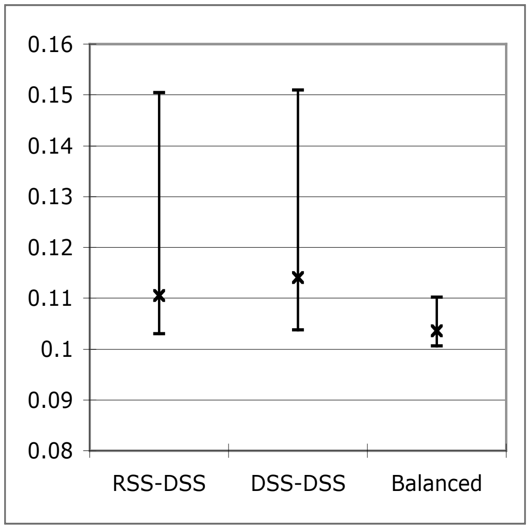

Figure 4.2.19: Adult Dataset – Test False Positive Rate TNR+TPR Metric

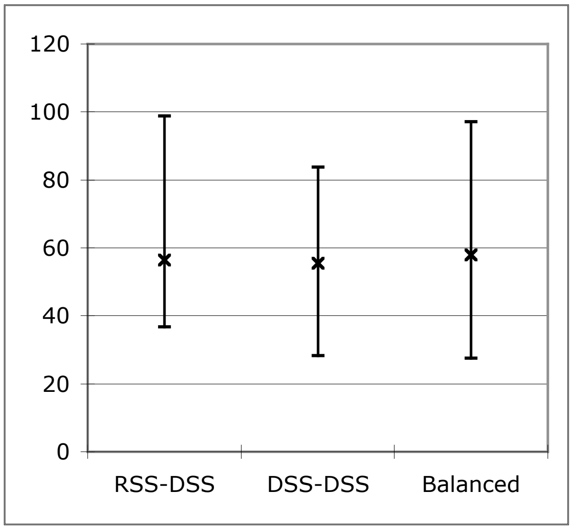

Figure 4.2.20: Adult Dataset – Program Size after Intron Removal Hits Metric

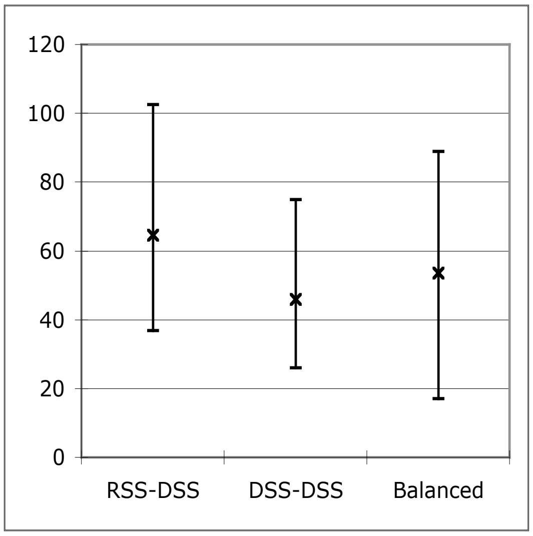

Figure 4.2.21: Adult Dataset – Program Size after Intron Removal TNR+TPR Metric

The DSS-DSS algorithm had the slowest median run time (Figure 4.2.13) and somewhat surprisingly the balanced block algorithm had the fastest median run time. The step up in run time between the RSS-DSS and the DSS-DSS algorithm at first appeared to be a natural reflection of the overhead in conducting an additional roulette wheel based calculation for block selection in DSS-DSS, as opposed to the uniform selection in the RSS-DSS algorithm. However, under this reasoning the Balanced Block algorithm should also suffer from extra calculations due to selecting partitions to make up the Level 1 block, and extra roulette selections to determine which partitions of the Level 1 block will contribute exemplars to the DSS subset. Upon further consideration it is believed that run time will be more affected by the size (complexity) of GP programs throughout training than extra roulette wheel selections due to the far greater number of program executions for fitness evaluation.

The Hits metric has much higher test accuracy than the TNR+TPR metric (Figures 4.2.14 and 4.2.15), a much higher test detection rate (Figures 4.2.16 and 4.2.17) than the TNR+TPR metric but a much higher (worse) false positive rate (Figures 4.2.18 and 4.2.19) than the TNR+TPR metric. This follows our reasoning that the Hits metric will concentrate on the larger class 1 and try to maximize detection rate, and thereby achieving a better overall accuracy while the TNR+TPR metric attempts to concentrate on both classes with equal importance thereby improving the false positive rate but suffering a lower test accuracy.

The RSS-DSS algorithm had the highest median test accuracy (Figure 4.2.14) and the lowest median test false positive rate (Figure 4.2.18) for the Hits metric. The DSS-DSS algorithm had the highest median test accuracy (Figure 4.2.15), the highest median detection rate (Figure 4.2.17) and the lowest program size after intron removal (Figure 4.2.21) for the TNR+TPR metric.

The Balanced Block algorithm had the lowest median run time (Figure 4.2.13), the highest median detection rate for the hits metric (Figure 4.2.16) and the lowest median false positive rate for the TNR+TPR metric (Figure 4.2.19).

The Balanced Block algorithms results always seemed competitive with the RSS-DSS and DSS-DSS results and also seemed to show the most consistency (tightness of quartile plots) out of all three algorithms.

4.2.5 Comparison with Other Algorithms

In order to provide some qualification of the classifier performance Table 4.2.18 details (best case) error rates of alternative machine learning algorithms summarized as part of the UCI repository [15]. Unfortunately no information is provided regarding the learning algorithms, the reliability with which these results were attained or any preprocessing or algorithm parameterizations necessary to achieve these results.

Algorithm	Error	Algorithm	Error
FSS Naïve Bayes	14.05	Voted ID3 (0.6)	15.64
NBTree	14.10	CN2	16.00
C4.5-auto	14.46	Naïve-Bayes	16.12
IDTM (decision table)	14.46	Voted ID3 (0.8)	16.47
HOODG	14.82	T2	16.84
C4.5 rules	14.94	1R	19.54
OC1	15.04	Nearest-neighbor (3)	20.35
C4.5	15.54	Nearest-neighbor (1)	21.42

Table 4.2.18: Error Rates on Adult Dataset [15]

Table 4.2.19 lists results reported by Folino et al. in [31] where ensemble techniques for parallel GP's are investigated. In particular the cellular model of parallel genetic programming (CGPC – Cellular Genetic Programming Classifier) was employed which is a fine-grained parallel implementation of GP through the diffusion model using cellular automata as a framework.

Algorithm	Accuracy	Error Rate
CGPC	82.74	17.26
BagCGPC	83.42	16.58
BagCGPC without com.	81.74	18.26

Table 4.2.19: Parallel GP Results [31]

BagCGPC is an extension to CGPC where an ensemble of classifiers is created. Each classifier is trained on a different subset of the overall dataset. The classifiers are then combined to classify test data by applying a majority-voting algorithm as in bagging. Furthermore, in order to take advantage of the cellular model of GP the subpopulations of each ensemble classifier are not independently evolved, but exchange the outermost individuals in an asynchronous way. The third entry BagCGPC without communication removes the communication used in BagCGPC.

Algorithm	Best Accuracy	Best Error Rate	Best TNR+TPR
RSS-DSS	83.66	16.34	1.6042
DSS-DSS	82.60	17.40	1.6074
Balanced Block (30/70)	83.57	16.43	1.6053

Table 4.2.20: Best Case Adult Dataset Results

Table 4.2.20 shows the best results found by each GP algorithm in terms of accuracy and TNR+TPR value in order to compare with Table 4.2.18. The RSS-DSS found the best overall test accuracy achieving 83.66% while the DSS-DSS algorithm found the best overall TNR+TPR value at 1.6074. It is readily apparent that the GP solutions are ranked towards the end of Table 4.2.18, however, they also appear before the step change in errors from an error of 17 to an error of 19.5. Again, there is no additional information on how the algorithms in 4.2.18 fare on the smaller class, and so it is unknown how they would compare in terms of TNR+TPR. The results in Table 4.2.19 also appear before the step change in errors in Table 4.2.18. However, despite the advantages of parallelizing GP and using an ensemble of classifiers our algorithms in Table 4.2.20 show comparable results. In particular, the best case RSS-DSS and the best case Balanced Block algorithm outperform all of the CGPC algorithms.

Algorithm	Time	Size	Accuracy	Error	DR	FPR	TNR+TPR
Tree	0.18	291	86.1	13.9	0.9324	0.3695	1.5629
Tree (0.1)	0.20	36	86.1	13.9	0.9583	0.4555	1.5028
Tree (0.001)	0.16	22	85.7	14.3	0.9612	0.4789	1.4823
Tree (0.00001)	0.15	18	85.7	14.3	0.9631	0.4878	1.4753
Rules	0.76	97	86.5	13.5	0.9441	0.3924	1.5517
Rules (0.1)	0.24	22	86.1	13.9	0.9582	0.4548	1.5034
Rules (0.001)	0.19	10	85.7	14.3	0.9600	0.4774	1.4826
Rules (0.00001)	0.18	10	85.7	14.3	0.9628	0.4867	1.4761

Table 4.2.21: C5.0 Results on the Adult Dataset (pruning settings in brackets)

Additional experiments with the C5.0 tree induction algorithm, Table 4.2.21 row 1, indicate that high classification accuracies are indeed reliably achieved (test set error rate of 13.9%), albeit at the expense of solution complexity (over 290 nodes, verses 60-170 instructions in GP). Thus, GP solutions appear to be trading complexity for precision under the parameterization considered here.

However, a manipulation of the pruning options provided with C5.0 were able to reduce the size of the solution to 36 tree nodes while maintaining the error rate of 13.9% (Table 4.2.21, row 2 – Tree (0.1)). Further tweaking of the pruning option led to even smaller trees but resulted in some degradation of the error rate. Results for C5.0 Rules with various pruning values are also shown with standard C5.0 rules achieving the lowest error rate of all.

In terms of run time, all C5.0 algorithms finished in less than one minute as compared to GP solutions that were on the order of 10 to 15 minutes for one run, with 40 runs requiring up to 10 hours.

However, the last column of Table 4.2.21 shows the TNR+TPR value for the C5.0 algorithms. GP results in Table 4.2.20 show larger TNR+TPR values than those of any C5.0 algorithm (i.e., better false positive rates under solutions found by GP). However, C5.0 is not trained to optimize this metric.

CPU time (hrs)	% train	% test	Best error
16.3	82.95	82.91	16.65

Table 4.2.22: Tree Structured GP on Adult Dataset, no RSS/DSS algorithm

One final comparison is made. In this case results are reported for a vanilla implementation of Koza's original Tree structured GP [13] using lilgp [19]. This provides GP baseline performance figures for classification accuracy, run time and solution complexity. Lilgp represents an efficient C code implementation of Tree-structured GP and is parameterized using the default values of: Pop. Size 4000; Crossover prob. 90%; Reproduction prob. 10%; Mutation prob. 0% (i.e., parameterization as per Koza [92][94]). As per the page-based linear GP results a node limit of 256 nodes was utilized (as opposed to a depth limit). Results are reported in terms of median CPU time and best case error for a total of 10 trials in Table 4.2.22. Table 4.2.22 shows that the RSS-DSS, the DSS-DSS and the Balanced Block algorithms have not negatively impacted on the classification performance of GP whilst retaining a computational speedup of 4 orders of magnitude.

4.3 The 10% KDD'99 Dataset

The second large dataset under investigation is the KDD-99 Intrusion Detection dataset, taken from the 5th ACM SIGKDD Knowledge Discovery and Data Mining Competition (1999) [14]. Two partitions of the dataset are used, the 10% KDD'99 for training and the Corrected Test for test, as per the original competition [14]. The 10% KDD'99 dataset represents a real data-mining problem, containing approximately 500,000 exemplars. Given the considerably large dataset a block size of 5000 was used resulting in 99 blocks compared to the Adult's block size of 1000 and 31 blocks. The data distribution of the 10% KDD'99 training data and the Corrected Test can be seen in Table 4.3.1.

Class	Training		Test	
0	97,249	19.69%	60,577	19.48%
1	396,744	80.31%	250,424	80.52%
Total	493,993	100%	311,001	100%

Table 4.3.1: 10% KDD'99 and Corrected Test Data Distribution

Each exemplar is described in terms of 41 features, comprising of 9 basic features and 32 derived features describing temporal and content information. Here we only use the first 8 basic features, but express these in terms of a shift register with 8 taps taken at intervals of 8 sequential exemplars. Such a scheme requires that the learning algorithm also identify the useful temporal properties, rather than relying on the *a priori* selected features (see [16]). Each entry of the KDD dataset represents a connection, labeled in terms of one of five categories: Normal, Denial of Service (DoS), Probe, User to Root (U2R) and Remote to Local (R2L). In this work we are only interested in distinguishing between Normal and any of the four attack categories. Moreover, 79% of the training data represent instances of DoS, 20% Normal and the remainder Probe, U2R and R2l, Table 4.3.2. Thus, as well as representing a large training set it is also unbalanced, introducing the possibility for degenerate solutions (i.e., a detector that labels every exemplar as attack). The test data on the other hand increases the contribution of the second smallest attack category, R2L, to 5% of the dataset and introduces 14 attack types unseen during training, Table 4.3.2. Moreover, there are many unseen attacks in the test data, relative to those encountered during training.

Data Type	Training	Test
Normal	19.69%	19.48%
Probe	0.83%	1.34%
DoS	79.24%	73.90%
U2R	0.01%	0.07%
R2L	0.23%	5.20%

Table 4.3.2: Distribution of Attacks

4.3.1 Balanced Block – Block Partition Ratios

Several different block partition ratios were tested to help parameterize the Balanced Block algorithm on the 10% KDD'99 dataset and to choose a best partitioning for comparison against the RSS-DSS and DSS-DSS algorithms. Block partition ratios of 10%/90% (10% class 0, 90% class 1), 20%/80%, 25%/75% and 50%/50% were chosen for variety and convenience. The results of these experiments are summarized in Figures 4.3.1-4.3.9 for both the Hits Metric and the TNR+TPR Metric in terms of run time, test set accuracy, test detection rate, test false positive rate and program size after structural introns have been removed. Full results, including training data results, can be found in Appendix B, Table B.1.

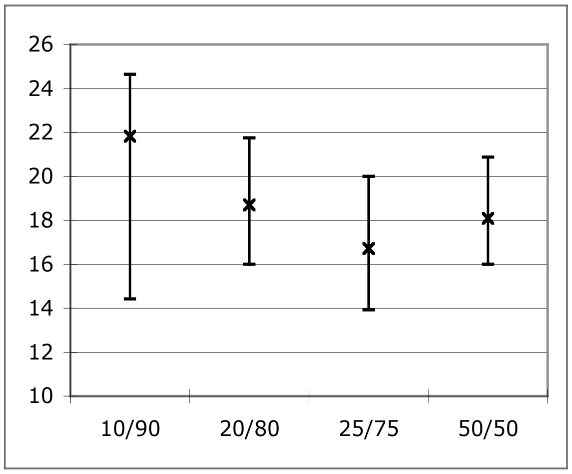

Figure 4.3.1: KDD Dataset: Balanced Block Partitions – Run Time (mins)

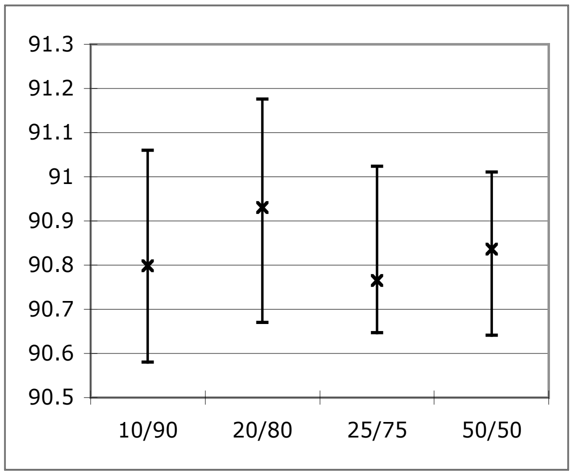

Figure 4.3.2: KDD Dataset: Balanced Block Partitions – Test Accuracy Hits Metric

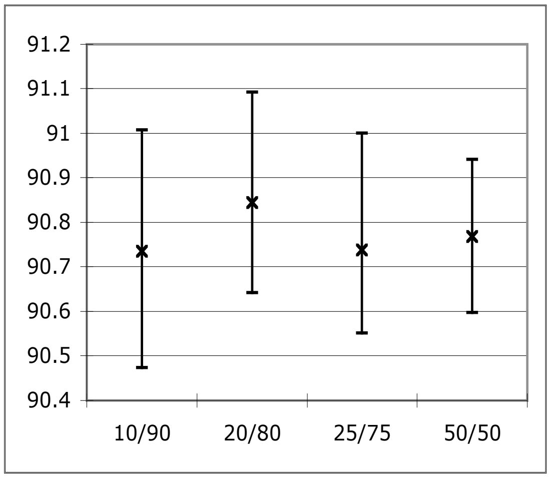

Figure 4.3.3: KDD Dataset: Balanced Block Partitions – Test Accuracy TNR+TPR Metric

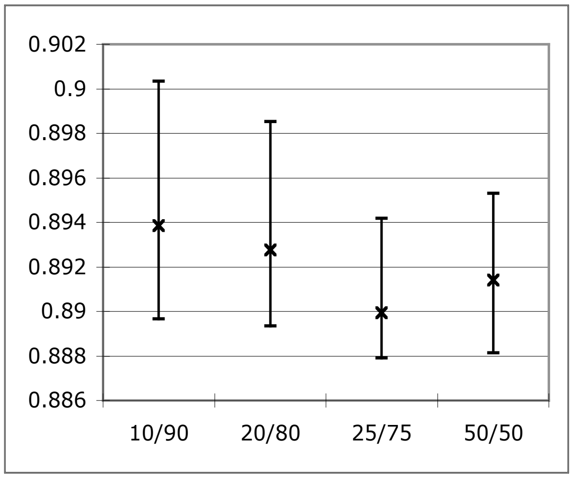

Figure 4.3.4: KDD Dataset: Balanced Block Partitions – Test Detection Rate Hits Metric

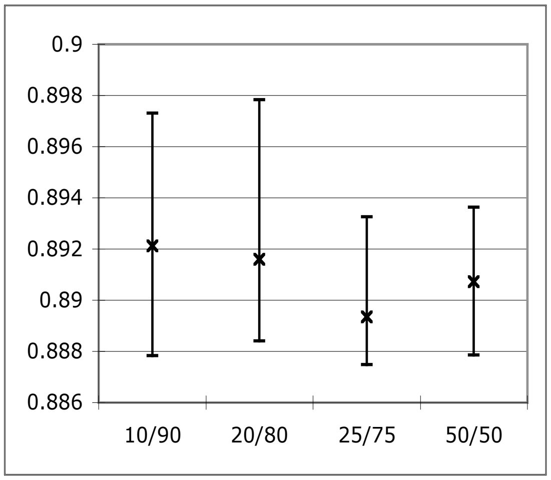

Figure 4.3.5: KDD Dataset: Balanced Block Partitions – Test Detection Rate TNR+TPR Metric

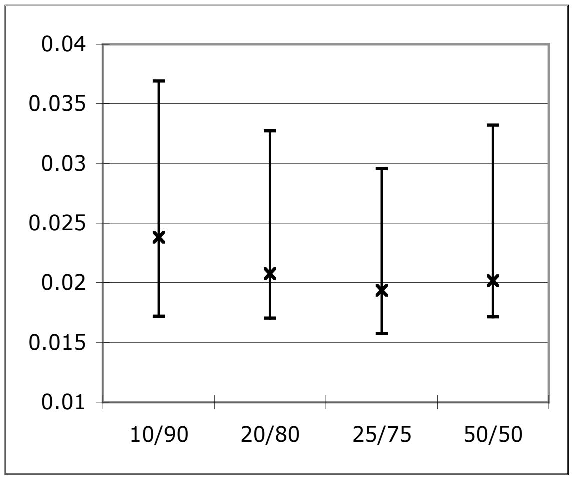

Figure 4.3.6: KDD Dataset: Balanced Block Partitions – Test False Positive Rate Hits Metric

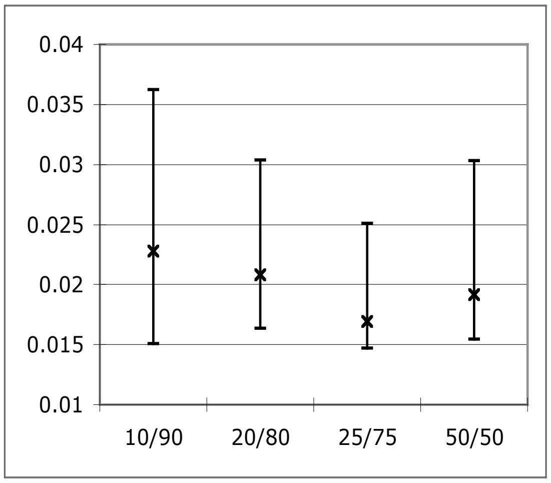

Figure 4.3.7: KDD Dataset: Balanced Block Partitions – Test False Positive Rate TNR+TPR Metric

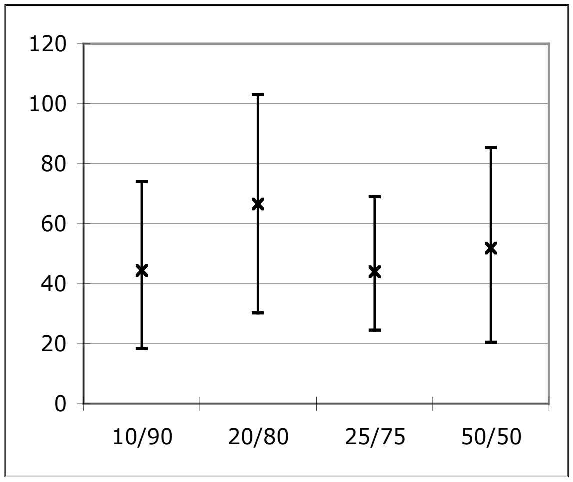

Figure 4.3.8: KDD Dataset: Balanced Block Partitions – Program Size after Intron Removal Hits Metric

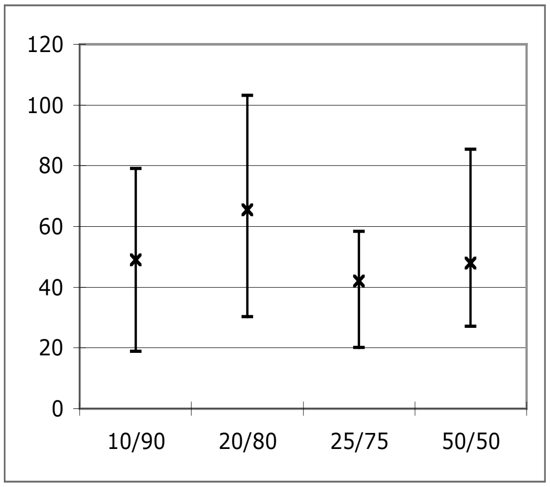

Figure 4.3.9: KDD Dataset: Balanced Block Partitions – Program Size after Intron Removal TNR+TPR Metric

The 25/75 block partition ratio is the best parameterization of the Balanced Block algorithm for the KDD'99 dataset. The 25/75 ratio has the best median run time (Figure 4.3.1), test false positive rates (Figures 4.3.6 and 4.3.7) and program size after intron removal (Figures 4.3.8 and 4.3.9). It is competitive in terms of test classification accuracy (Figures 4.3.2 and 4.3.3) and test detection rate (Figures 4.3.4 and 4.3.5) and it seems to be the most consistent across all of the figures for both the Hits metric and the TNR+TPR metric.

4.3.2 GP Algorithm Comparison

Figures 4.3.10 through 4.3.18 are quartile plots comparing the RSS-DSS algorithm, the DSS-DSS algorithm and the Balanced Block algorithm with a 25/75 block partition ratio on the 10% KDD'99 dataset. The Hits Metric results are shown in the left figures while the TNR+TPR Metric results are shown on the right for easy comparison. Full results of this experiment can be found in Appendix B, Table B.2.

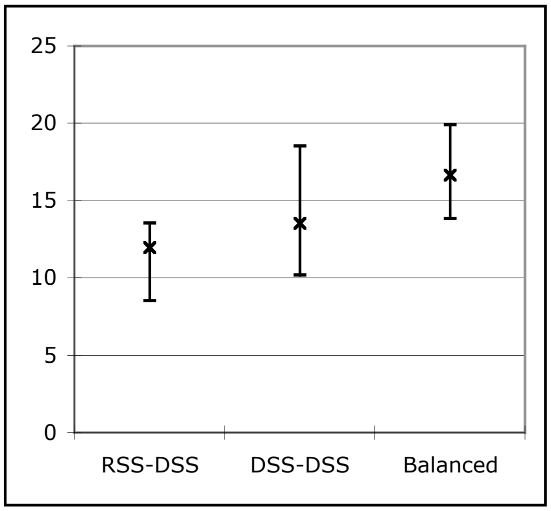

Figure 4.3.10 : KDD Dataset – Run Time (minutes)

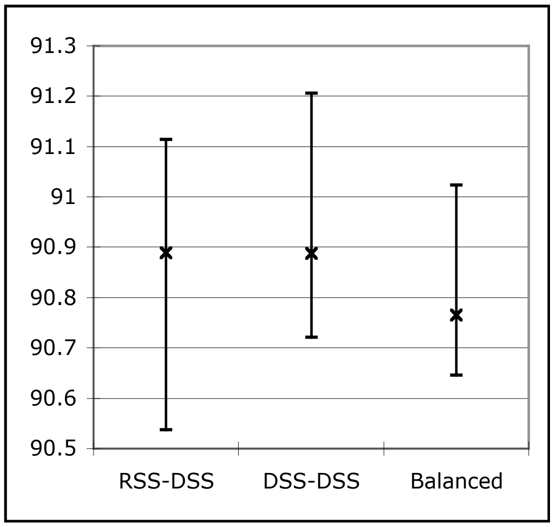

Figure 4.3.11: KDD Dataset – Test Accuracy Hits Metric

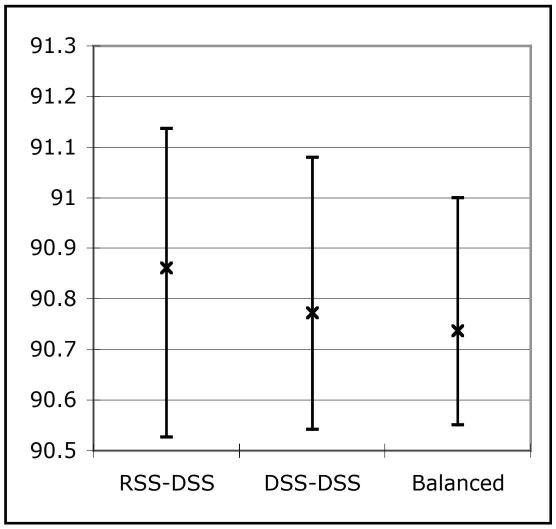

Figure 4.3.12: KDD Dataset – Test Accuracy TNR+TPR Metric

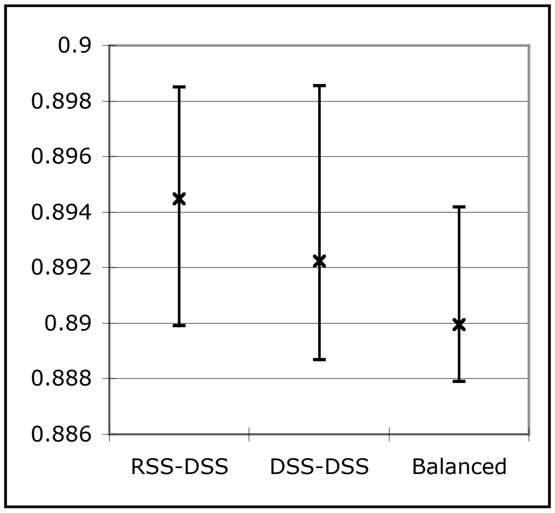

Figure 4.3.13: KDD Dataset – Test Detection Rate Hits Metric

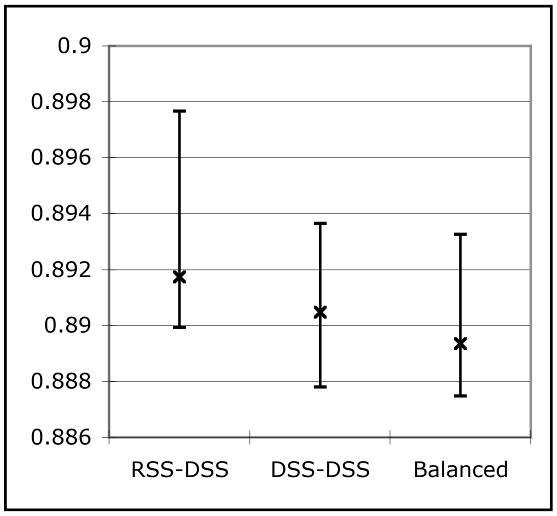

Figure 4.3.14: KDD Dataset – Test Detection Rate TNR+TPR Metric

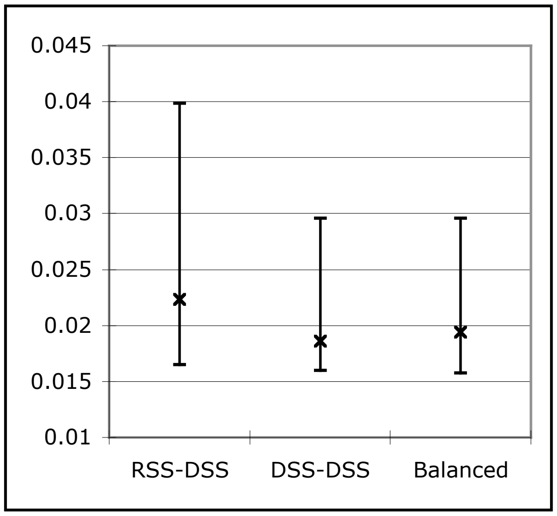

Figure 4.3.15: KDD Dataset – Test False Positive Rate Hits Metric

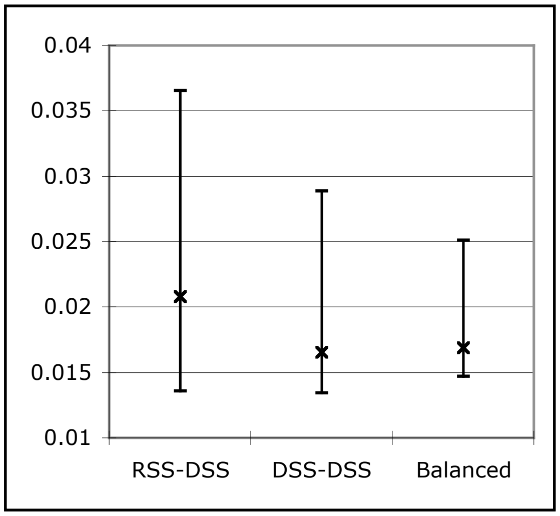

Figure 4.3.16: KDD Dataset – Test False Positive Rate TNR+TPR Metric

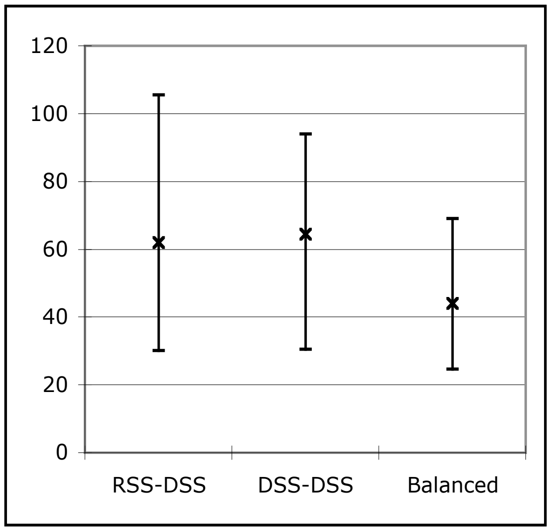

Figure 4.3.17: KDD Dataset – Program Size after Intron Removal Hits Metric

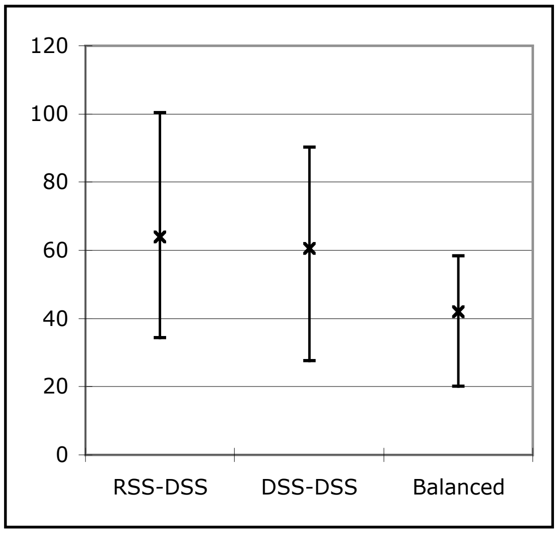

Figure 4.3.18: KDD Dataset – Program Size after Intron Removal TNR+TPR Metric

Figure 4.3.10 shows the quartile plots for algorithm run times in minutes. It is clear that the RSS-DSS algorithm has the fastest median run time and is the most consistent in terms of run time. The RSS-DSS hierarchy typically taking less time to train than the DSS-DSS hierarchy and the Balanced Block algorithm appears to be a natural reflection of the overhead in conducting additional roulette wheel based calculations for block selections for DSS-DSS and Balanced Block, as opposed to the uniform selection of the RSS-DSS algorithm. However, all three algorithms are two orders of magnitude better than previously reported research in which GP was applied directly to the entire 10% KDD’99 dataset [18]. Such a direct application required 48 hours to complete a single trial (Pentium III, 800Mhz), whereas here each trial takes less than 20 minutes; 40 trials therefore completing in less than 15 hours. Relative to the Adult dataset, five less minutes appear to be necessary in run time indicating that although the Adult dataset is smaller, there is more diversity in the dataset, where this is also reflected in the additional complexity in the solutions.

In terms of test accuracy and detection rate for both metrics the four figures (4.3.11 to 4.3.14) show a similar pattern. The RSS-DSS algorithm shows the highest median accuracies and detection rates. The accuracy and detection rates get worse with DSS-DSS and worse again with the Balanced Block algorithm but these algorithms both show an improvement in test false positive rates than the RSS-DSS algorithm, Figures 4.3.15 and 4.3.16. Furthermore, the DSS-DSS algorithm shows an improved consistency (tighter quartile plots) than the RSS-DSS algorithm in terms of accuracy, detection rate and false positive rate. The Balanced Block algorithm shows a further improvement in consistency than the DSS-DSS algorithm.

Finally, in terms of program size after intron removal, Figures 4.3.17 and 4.3.18, the Balanced Block algorithm produced the smallest median program sizes and was also the most consistent of the three algorithms.

For 10% KDD'99 dataset results it is interesting to note that the differences in results between the Hits metric and the TNR+TPR metric is not very large with many of the quartile plots appearing to be almost the same between the two metrics. Indeed it is the case that the program with the best hits on the 10% KDD'99 dataset was often the same program that achieved the best TNR+TPR value.

4.3.3 GP Algorithm Comparison on KDD Attack Types

Figures 4.3.19 to 4.3.26 depict the classification accuracy of each algorithm for both metrics on the four separate attack categories (Probe, Denial of Service, User to Root and Remote to Local). Full results, including training accuracies, can be found in Appendix B, Table B.3.

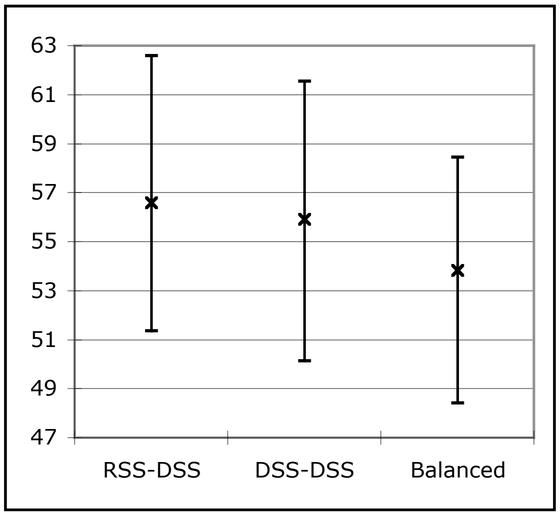

Figure 4.3.19: KDD Dataset – Probe Accuracy Hits Metric

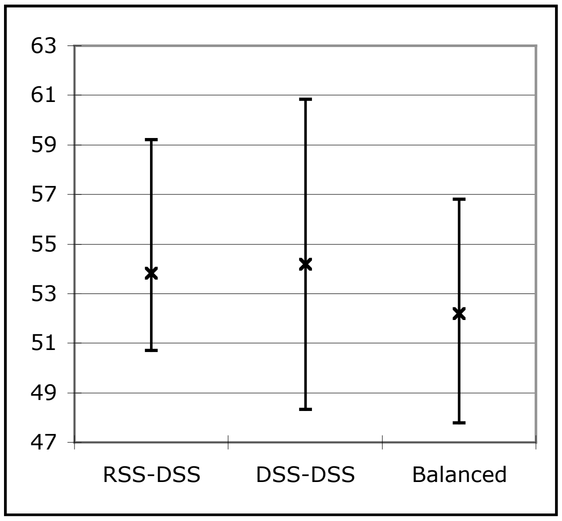

Figure 4.3.20: KDD Dataset – Probe Accuracy TNR+TPR Metric

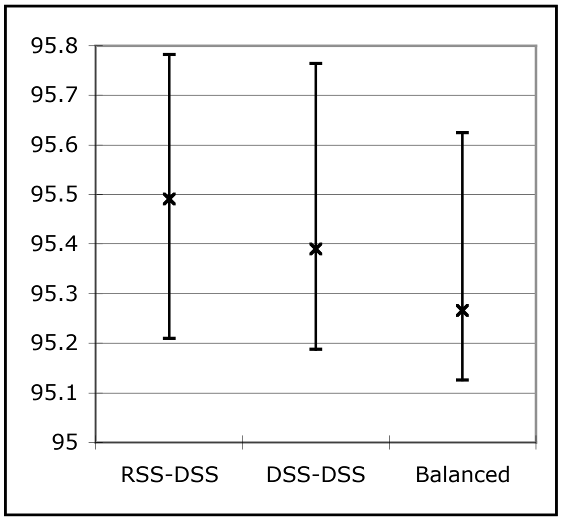

Figure 4.3.21: KDD Dataset – DoS Accuracy Hits Metric

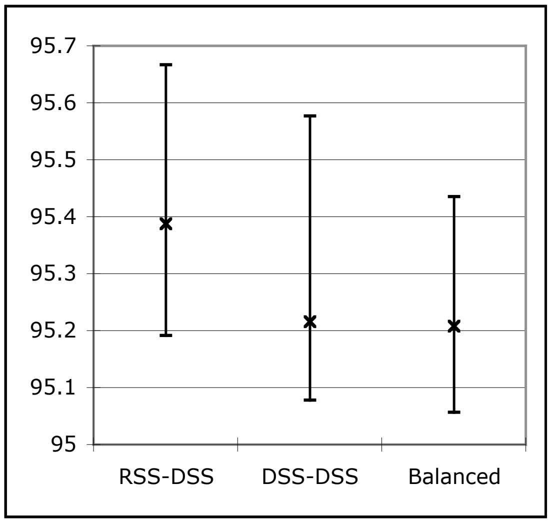

Figure 4.3.22: KDD Dataset – DoS Accuracy TNR+TPR Metric

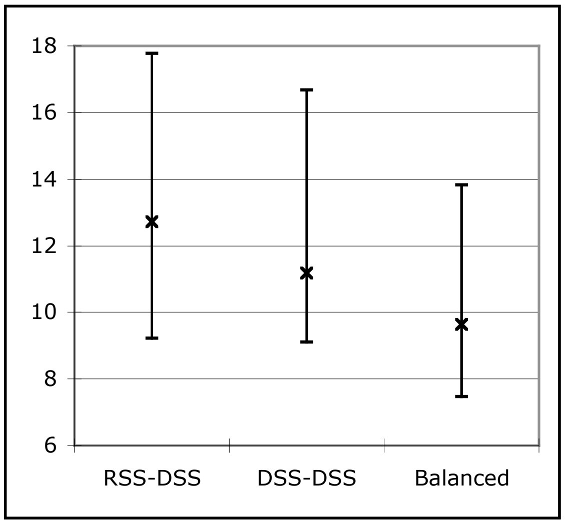

Figure 4.3.23: KDD Dataset – U2R Accuracy Hits Metric

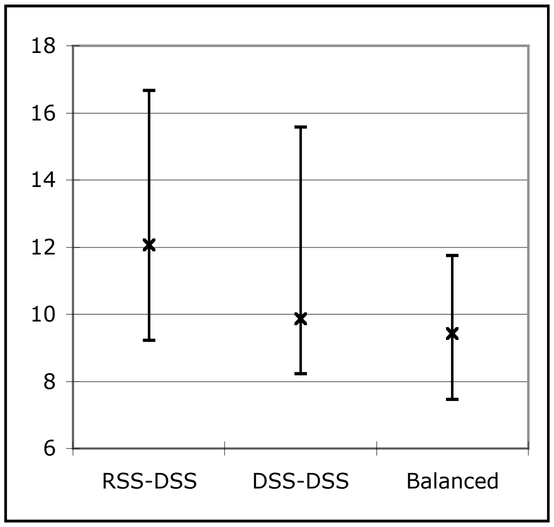

Figure 4.3.24: KDD Dataset – U2R Accuracy TNR+TPR Metric

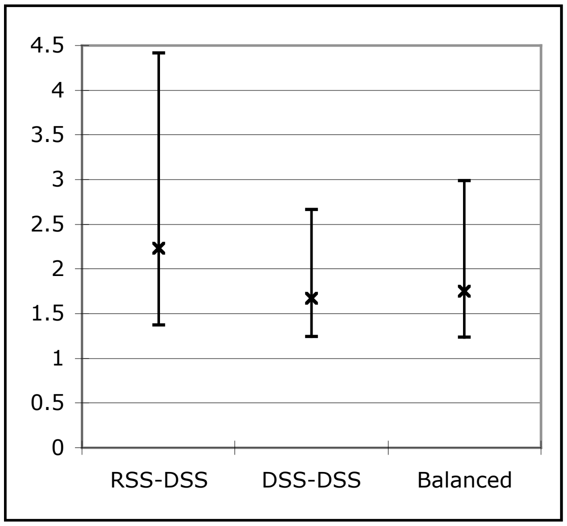

Figure 4.3.25: KDD Dataset – R2L Accuracy Hits Metric

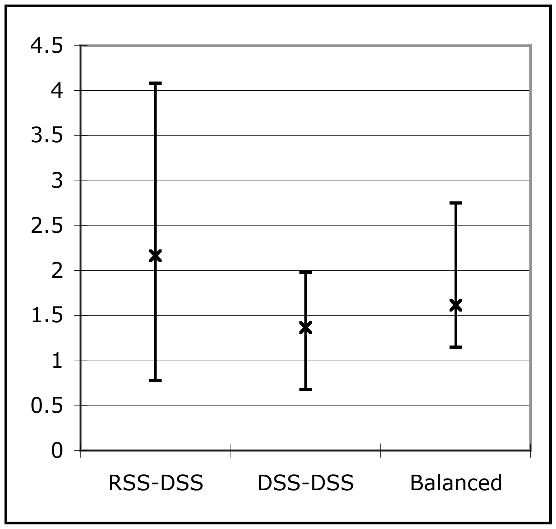

Figure 4.3.26: KDD Dataset – R2L Accuracy TNR+TPR Metric

The RSS-DSS algorithm has the highest classification accuracies for each of the attack categories, which corresponds to its higher detection rate (Figures 4.3.13 and 4.3.14). However, in general the RSS-DSS algorithm had the worst consistency across all of the attack categories with the Balanced Block algorithm with 25/75 block partition ratio being the most consistent.

For all three algorithms classification accuracy was the highest for the DoS category with results over 95% accuracy while results on the other attack categories are significantly worse with Probe achieving median results of approximately 53-57%, U2R achieving 9-13%, and U2R achieving only 1.5-2.5% accuracy. However, the overall detection rate remains very high for all three algorithms due to DoS making up 73.9% of the entire test dataset and 91.79% of the attacks.

It is also interesting to note that on the training dataset the three algorithms were achieving median accuracies of 14-32% on the R2L category (see Appendix B, Table B.3), which dropped off to 1.5-2.5% accuracy on the test dataset, Figures 4.3.25 and 4.3.26. The increase in the contribution of the R2L attack category to the test dataset and the introduction of 14 attack types unseen during training may contribute to this fall off. The DoS and Probe categories also experience a reduction in classification accuracy between training and test (Appendix B, Table B.3).

4.3.4 KDD Smallest GP Program

Table 4.3.3 shows the smallest program found by all three algorithms on the KDD dataset after structural introns have been removed.

	Opcode	Destination	Source
1	MUL	R[4]	Input[1][1]
2	DIV	R[4]	Input[1][5]
3	DIV	R[4]	Input[1][5]
4	MUL	R[4]	Input[1][1]
5	DIV	R[4]	Input[1][5]
6	SUB	R[4]	Input[7][5]

Table 4.3.3: KDD Smallest GP Program (Balanced Block partition 50/50)

The program contains six instructions and was found by the Balanced Block algorithm with a 50/50 partitioning. Inputs are read as Input[temporal][feature]. All registers are initialized to zero and therefore register four, R[4], will initially start at zero. Due to R[4] starting at zero the output of this program will be:

$$R[4] = 0 - Input[7][5] = - Input[7][5] \qquad \textbf{(4.3.1)}$$

with the exception of one scenario. The exception occurs when Input[1][5] is equal to zero. In this case dividing by Input[1][5] will be protected division which returns a one. Therefore, Line 5 in Table 4.3.3 will result in R[4] = R[4] / 0 = 1 and the output of line 6 will be:

$$R[4] = 1 - Input[7][5] \qquad \textbf{(4.3.2)}$$

The only effective code for this program is in lines 5 and 6 of Table 4.3.3, and these lines can be re-expressed as:

```
IF ( Input[1][5] == 0)
        R[4] = 1 – Input[7][5]
ELSE
        R[4] = - Input[7][5]
```

or as the expression:

$$Output = R[4] = (0 / Input[1][5]) - Input[7][5] \qquad \textbf{(4.3.3)}$$

The results of this program actually fare very well on the KDD test data as shown in Table 4.3.4.

	Accuracy	DR	FPR	Normal	Probe	DoS	U2R	R2L
Results	90.475	0.8946	0.0535	94.65%	50.53%	95.65%	15.35%	3.62%

Table 4.3.4: KDD Smallest Program Results

The results of this program being so small and faring so well seem rather surprising at first. However, looking at the program in Table 4.3.3 we see that it makes use of only feature five of the KDD dataset, which is SRC (the number of data bytes from destination to source). For Normal traffic the SRC value is usually a positive integer. This fact, along with the fact that normal traffic usually occurs consecutively, means that Input[1][5] and Input[7][5] are typically both positive integers. This results in line 5 of Table 4.3.3 being zero and line 6 returning a non-zero negative Input[7][5]. The wrapper function of our GP maps all values less than or equal to zero as a normal connection and so normal traffic is classified correctly as evidenced in Table 4.3.4.

On the other hand, DoS attacks typically have an SRC value of zero. This fact, along with the fact that DoS traffic usually occurs consecutively, means that Input[1][5] and Input[7][5] are typically both zero. This results in Line 5 of the program in Table 4.3.3 returning a one due to protected division. Furthermore, Input[7][5] also being typically zero means the output becomes a one. The wrapper function sets all output values greater than zero as an attack and so DoS attacks are labeled correctly again evidenced in Table 4.3.4.

4.3.5 Comparison to Other Algorithms

Table 4.3.5 shows the winning and second place entries from the KDD'99 competition versus the corresponding GP cases.

Algorithm	Detection Rate	False Positive Rate	TNR+TPR
Winning Entry	0.908819	0.004472	1.9043
Second Place	0.915252	0.005760	1.9095
Best GP (RSS-DSS)	0.903050	0.011160	1.8927
Best GP (DSS-DSS)	0.906116	0.014115	1.8920
Best GP (Balanced (20/80))	0.903712	0.004031	1.8997

Table 4.3.5: Comparison Against KDD Winning Entries

Both the winning entries were based on decision trees and were trained using a boosting algorithm on different partitions of the original 10% KDD'99 dataset. Furthermore, they developed independent detectors for each class over all 41 features, all of which make a direct comparison difficult. The winning entries took roughly 24 hours to complete training, while GP training, on the other hand, took approximately 10-15 hours to complete (15-20 minutes for each GP run (Figure 4.3.10) for 40 runs). The winning KDD'99 solutions were complex with 500 C5.0 decision trees in the case of the winning entry [34] while GP programs contained approximately 50-70 instructions on median, Figures 4.3.17 and 4.3.18. However, it is not clear how instructions and C5.0 tree nodes or C5.0 rules can be compared. It is apparent however, that the GP solutions are competitive to the KDD'99 competition winners, Table 4.3.5, with the Balanced Block algorithm using a 20/80 partitioning finding the best GP solution.

C5.0	Time (mins)	Size	Test Accuracy	Test DR	Test FPR	TNR+TPR
Tree	4.87	188	92.8%	0.9176	0.0052	1.9124
Tree (0.1)	5.15	135	93.1%	0.9128	0.0045	1.9083
Tree (0.001)	6.83	100	92.9%	0.9155	0.0056	1.9099
Tree (0.00001)	5.26	131	93.0%	0.9166	0.0054	1.9112
Rules	10.58	46	93.3%	0.9121	0.0055	1.9066
Rules (0.1)	9.16	48	92.9%	0.9160	0.0049	1.9111
Rules (0.001)	9.44	26	93.1%	0.9136	0.0052	1.9084
Rules (0.00001)	9.05	34	93.2%	0.9139	0.0050	1.9089

Table 4.3.6: C5.0 KDD results (binary classification with pruning levels in brackets)

Table 4.3.6 provides test results of running C5.0 on the 10% KDD'99 dataset provided to our GP algorithms (binary classification problem distinguishing only between attacks and normal connections with only the first eight basic connection features plus a temporal history) as compared to KDD'99 competition winners who developed C5.0 classifiers for each of the 41 features. These C5.0 results have very high test accuracies, outperforming all three of our algorithms best results in most cases (Table 4.3.7), with C5.0 rules achieving the highest accuracy at 93.3%. In terms of TNR+TPR C5.0 outperforms all of our algorithms in all cases (Table 4.3.7 below) and even outperforms the KDD competition winners in some cases. However, the Balanced Block algorithm with 20/80 partitioning found the lowest false positive rate (Table 4.3.5) at 0.004031 in comparison to all of our results, the KDD winning entries, and C5.0, whilst remaining competitive in terms of Detection Rate and TNR+TPR.

Max Hits		Max TNR+TPR	
RSS-DSS	91.98%	RSS-DSS	1.8927
DSS-DSS	92.17%	DSS-DSS	1.8920
Balanced (50/50)	92.89%	Balanced (20/80)	1.8997

Table 4.3.7: Best Case KDD Results

C5.0 actually ran quite quickly on the 10% KDD'99 dataset normally taking between 4 to 10 minutes to run (Table 4.3.6). GP results typically take between 8 to 20 minutes for 1 run and so total training time takes between 10 to 15 hours (See Figure 4.3.10).
In terms of solution size, C5.0 ranged from 100 to 188 tree nodes, and C5.0 Rules ranged from 26 to 48 rules. GP results typically ran from 20 to 110 instructions after intron removal with median results between 40 to 70 instructions. Again it is difficult to know how C5.0 tree nodes and C5.0 rules can be compared to GP instructions.

4.4 The Census Income Dataset

The Census Income dataset is taken from the UCI Machine Learning Repository [15]. Similar to the Adult dataset, the learning task is to predict whether a person's income exceeds $50,000 per year. Each exemplar represents a person and is made up of 40 personal and demographic characteristics (or features) plus a label. All exemplars with missing features were removed from both the training and test datasets. The Census Income dataset is approximately three times the size of the Adult dataset with each exemplar having 40 features compared to the Adult Dataset's 14 features per exemplar. The data distribution for the Census Income dataset is shown in Table 4.4.1.

	Training		Test	
Class 0 (>$50K)	5479	5.76%	2683	5.66%
Class 1 (<=$50K)	89651	94.24%	44708	94.34%
Total	95130	100%	47391	100%

Table 4.4.1: The Census Income Dataset Data Distribution

From Table 4.4.1, it is readily apparent that Census Income is an imbalanced dataset having approximately 6% class 0 and 94% class 1 for both training and test data. This means that a degenerate classifier (one that labels all exemplars as one class) can achieve 94% accuracy. Degenerate classifiers on the Census Income dataset will typically achieve a 100% detection rate with a false positive rate close to 0%. These classifiers are of no interest. Instead, we are looking for classifiers who perform well on both classes. Avoiding degenerate solutions on imbalanced data is a difficult task for GP while using the Hits metric. This metric rewards programs that classify the most data correctly regardless of class and therefore degenerate solutions will fare quite well. On the other hand, programs that are learning to classify the smaller class will have a lower hits count and will therefore have a greater chance of being overwritten. So degenerate programs will be encouraged while other programs cannot survive long enough to learn to be competitive on both classes. For all three algorithms, RSS-DSS, DSS-DSS and Balanced Block, degenerate solutions were returned for every run using the Hits metric.

The second metric, TNR+TPR, encourages the GP programs to classify both classes with equal importance. No degenerate solutions were found for any of the algorithms when this metric was employed. Therefore, all results reported on the Census Income dataset use the TNR+TPR metric only.

4.4.1 Balanced Block – Block Partition Ratios

Several different block partition ratios were tested to help parameterize the Balanced Block algorithm on the Census Income dataset and to choose a best partitioning for comparison against the RSS-DSS and DSS-DSS algorithms. Block partition ratios of 5%/95% (5% class 0, 95% class 1), 10%/90%, 20%/80%, 25%/75% and 50%/50% were chosen for variety and convenience. The results of these experiments are summarized in Figures 4.4.1-4.4.5 in terms of run time, test set accuracy, test detection rate, test false positive rate and program size after structural introns have been removed. Full results, including training data results, can be found in Appendix C, Table C.1.

All results for different block partition ratios seem to have medians that are quite close to one another, suggesting that the algorithm is fairly robust in terms of partitions. The best partition ratio to use for comparison seems to be the 10/90 ratio. The 10/90 and 50/50 ratios had the highest test set accuracies (Figure 4.4.2) and detection rates (Figure 4.4.3) but the 50/50 ratio had the worst median run time (Figure 4.4.1) and median program size (Figure 4.4.5). The 10/90 ratio also has the best median run time (Figure 4.4.1) and was typically more consistent than the other block partition ratios. Therefore, the Balanced

Block algorithm with a 10/90 block partition ratio is used to compare to the RSS-DSS and DSS-DSS algorithms.

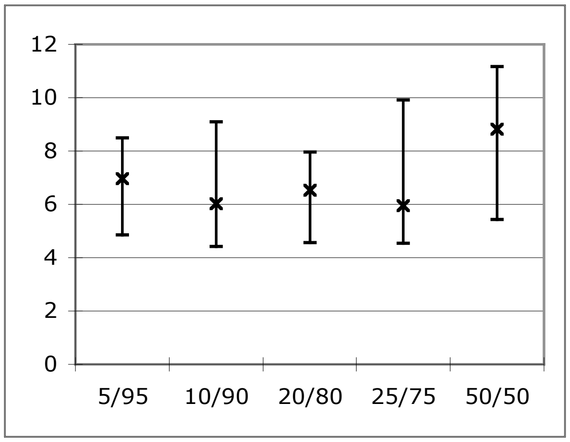

Figure 4.4.1: Census Income: Balanced Block Partitions – Run Time (minutes)

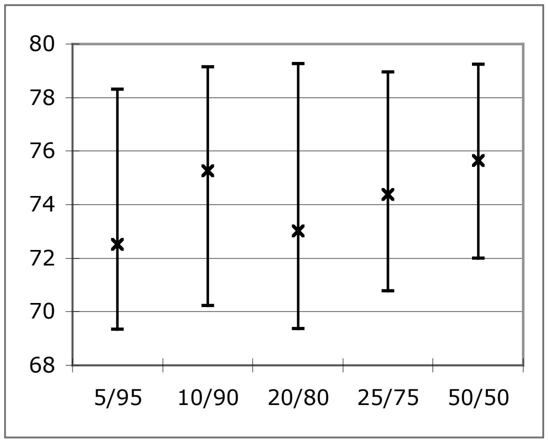

Figure 4.4.2: Census Income: Balanced Block Partitions – Test Accuracy

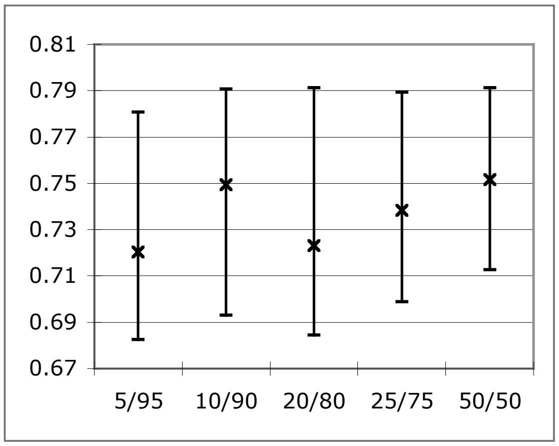

Figure 4.4.3: Census Income: Balanced Block Partitions – Test Detection Rate

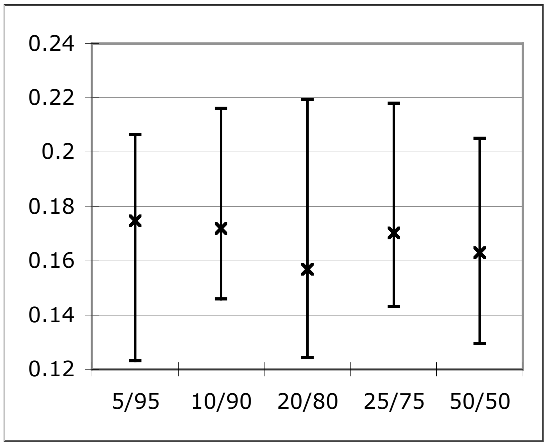

Figure 4.4.4: Census Income: Balanced Block Partitions – Test False Positive Rate

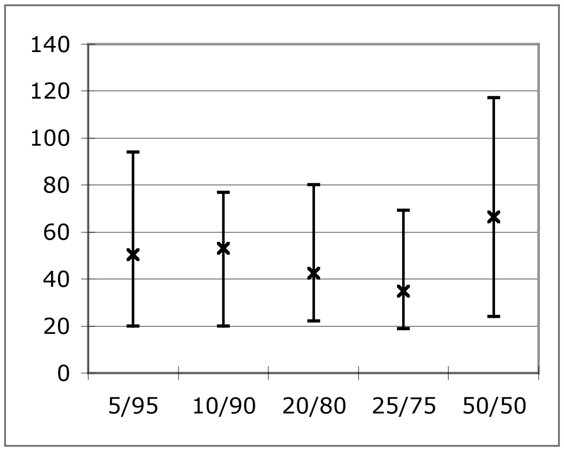

Figure 4.4.5: Census Income: Balanced Block Partitions – Program Size after Intron Removal

4.4.2 GP Algorithm Comparison

Figures 4.4.6 through 4.4.10 are quartile plots comparing the RSS-DSS algorithm, the DSS-DSS algorithm and the Balanced Block algorithm with a 10/90 block partition ratio on the Census Income dataset. Full results from this experiment can be found in Appendix C, Table C.2.

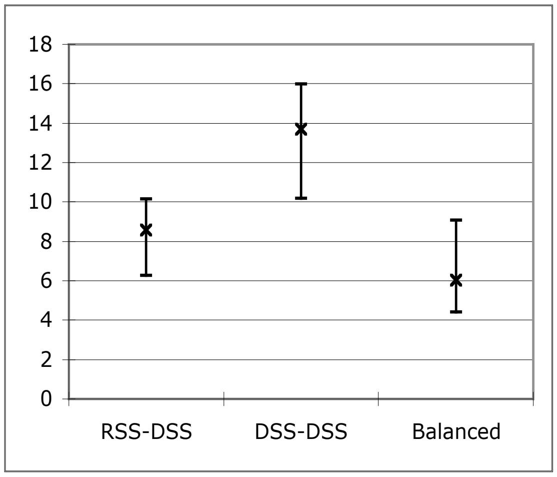

Figure 4.4.6: Census Income: Run Time (minutes)

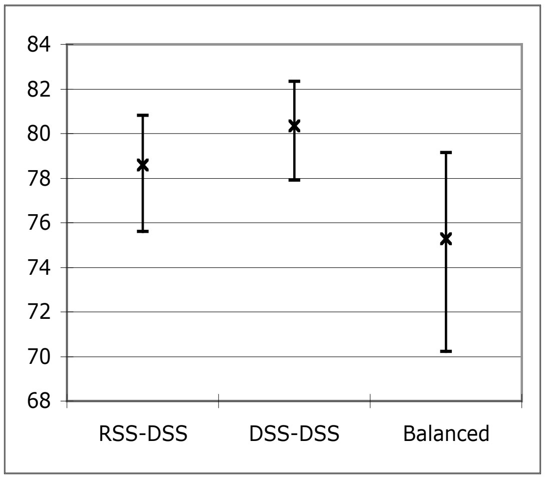

Figure 4.4.7: Census Income: Test Classification Accuracy

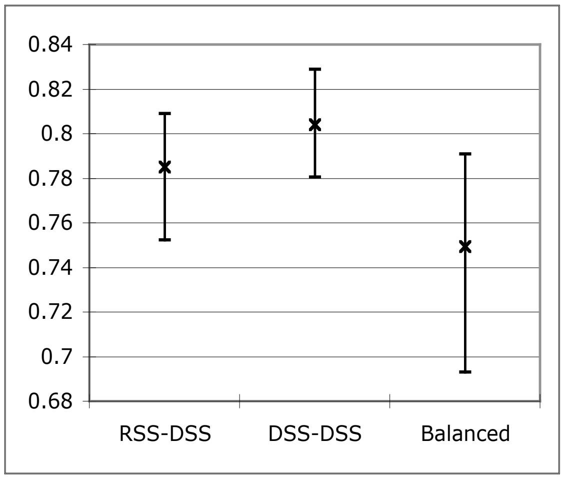

Figure 4.4.8: Census Income: Test Detection Rate

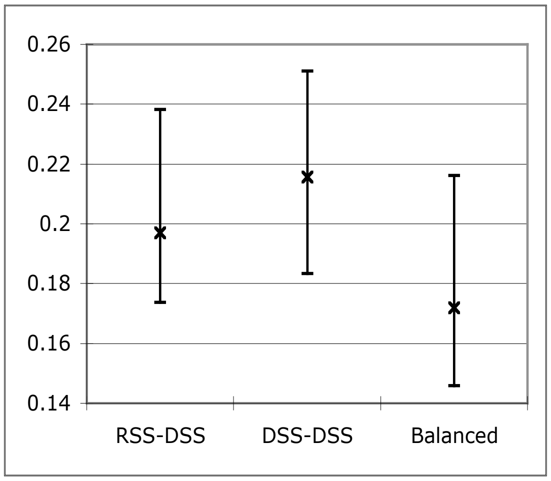

Figure 4.4.9: Census Income: Test False Positive Rate

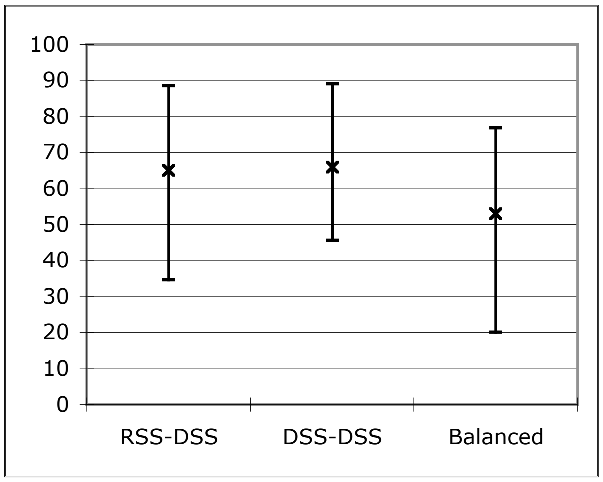

Figure 4.4.10: Census Income: Program Size after Intron Removal

The DSS-DSS algorithm has the worst median run time (Figure 4.4.6) and test false positive rate (Figure 4.4.9), but has the best test classification accuracy (Figure 4.4.7) and Detection Rate (Figure 4.4.8). It also appears to be the most consistent of the three algorithms. The Balanced Block algorithm had the best median run time (Figure 4.4.6), test false positive rate (Figure 4.4.9) and solution size after intron removal (Figure 4.4.10). The Balanced Block algorithm seems to have made the classical trade-off of achieving a better false positive rate at the expense of a worse Detection Rate. The lowering of the detection rate in turn lowers the overall classification accuracy due to class 1 making up the majority of the dataset. The Balanced Block algorithm does not appear to be as consistent as the DSS-DSS algorithm. The RSS-DSS algorithm always seemed to find the middle ground between the DSS-DSS algorithm and Balanced Block algorithm.

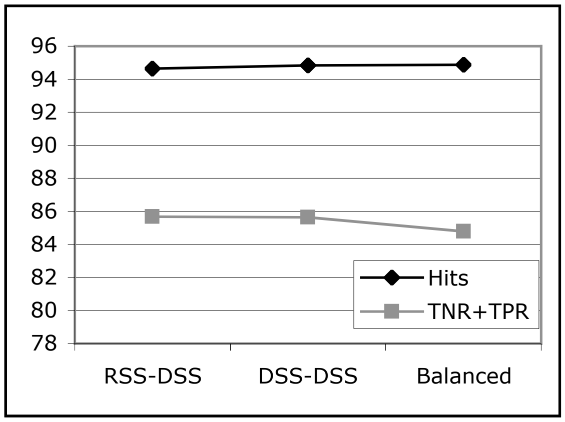

Figure 4.4.11: Census Income: Best Test Classification Accuracy

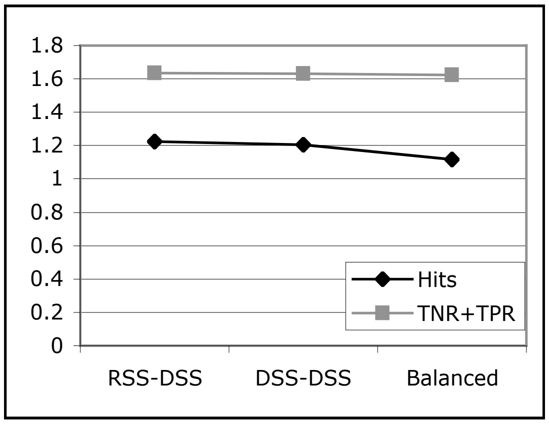

Figure 4.4.12: Census Income: Best Test TNR+TPR

Figure 4.4.11 and 4.4.12 are shown to demonstrate the difference between the Hits metric and the TNR+TPR metric on the Census Income dataset. Figure 4.4.11 shows the best test classification accuracy achieved by each program in terms of the Hits metric (diamonds) and the TNR+TPR metric (squares). Figure 4.4.12 show the best combined true negative rate and true positive rate (TNR+TPR value) achieved by each program in terms of the Hits metric and the TNR+TPR metric. It is evident in Figure 4.4.11 that the Hits metric returns individuals with much higher classification accuracy than those using the TNR+TPR metric. However, notice that the Hits metric classification accuracy is around 94.5 - 95% which is just above the degenerate test set accuracy of 94.34% and in fact the algorithm does return mostly degenerate solutions and even the best results had very poor false positive rates (accuracy on small class). Figure 4.4.12 shows that indeed the Hits metric results in a much lower best TNR+TPR value which is due to the Hits metric very poor true negative rate (true positive rate or detection rate is usually near 1 while the true negative rate is near 0). On the other hand the best TNR+TPR metric programs achieve only about 83 – 85% classification accuracy but have a much higher best TNR+TPR than the Hits metric and return no degenerate solutions.

4.4.3 Comparison with Other Algorithms

Table 4.4.2 shows some results reported on the Census Income dataset [15]. C4.5 and all C5.0 algorithms performed better than the default accuracy of 94.34% but no other information was reported such as run time, solution size and how well the algorithms performed on each class (TNR+TPR).

Algorithm	Test Classification Accuracy
C4.5	95.2%
C5.0	95.3%
C5.0 Rules	95.3%
C5.0 Boosting	95.4%
Naïve-Bayes	76.8%

Table 4.4.2: Other Results Reported on Census Income Dataset [15]

To obtain these figures and to verify the results reported above, C5.0 was run on the Census Income dataset. Results are reported in Table 4.4.3.

Algorithm	Time (mins)	Size	Test Accuracy	Test DR	Test FPR	TNR+TPR
C5.0	0.67	263	95.61%	0.9919	0.6396	1.3523
C5.0 (0.1)	0.70	48	95.56%	0.9922	0.6545	1.3377
C5.0 (0.001)	0.53	12	95.20%	0.9965	0.7898	1.2067
C5.0 (0.00001)	0.52	12	95.20%	0.9965	0.7898	1.2067
C5.0 Rules	2.60	76	95.60%	0.9946	0.6865	1.3081
C5.0 Rules (0.1)	1.15	17	95.37%	0.9944	0.7238	1.2706
C5.0 Rules (0.001)	0.70	7	95.15%	0.9962	0.7920	1.2042
C5.0 Rules (0.00001)	0.68	7	95.15%	0.9962	0.7920	1.2042

Table 4.4.3: C5.0 Results on Census Income Dataset (Pruning level in brackets)

C5.0 and C5.0 Rules achieved an even higher accuracy than that reported in Table 4.4.2. Furthermore, all runs of C5.0 and C5.0 Rules achieved an almost perfect true positive (or detection rate) while achieving a rather poor false positive rate. The C5.0 algorithm was very fast and managed to find very short trees and rules when the level of pruning was increased.

Algorithm	Best Test Accuracy	Best Test TNR+TPR
RSS-DSS (Hits)	94.63%	1.2230
DSS-DSS (Hits)	94.85%	1.2042
Balanced (10/90) (Hits)	94.49%	1.1203
RSS-DSS (TNR+TPR)	85.69%	1.6335
DSS-DSS (TNR+TPR)	85.66%	1.6308
Balanced (10/90) (TNR+TPR)	84.80%	1.6243

Table 4.4.4: Best Results on Census Income Dataset

All of the algorithms reported in Table 4.4.4 have a better test classification accuracy than Naïve Bayes but worse than the other algorithms reported in Table 4.4.2 and worse than C5.0 results in Table 4.4.3. The three algorithms using the Hits metric in Table 4.4.4 achieved results that were very similar to that of C5.0 with a very high test classification accuracy, but a poor TNR+TPR value. The three algorithms using the TNR+TPR metric had quite a bit lower classification accuracy but a much higher TNR+TPR value. This means that the TNR+TPR metric learns to classify the smaller class much better than the Hits metric and C5.0.

Algorithm	Median Run Time (mins)	Median Size after Intron Removal
RSS-DSS (TNR+TPR)	8.57	65
DSS-DSS (TNR+TPR)	13.70	66
Balanced (10/90) (TNR+TPR)	6.02	53

Table 4.4.5: Median Run Times and Program Sizes on Census Income Dataset

Table 4.4.5 shows that the run times of the three algorithms are much slower than that of C5.0. Furthermore, although it is difficult to make a comparison between rules or tree nodes and program instructions it seems that C5.0 results in smaller solutions than our algorithms.

4.5 Shuttle Dataset

The Shuttle dataset is taken from the Statlog Project Databases of the UCI Machine Learning Repository [15]. The shuttle dataset has a training dataset of 43,500 exemplars and a test dataset of 14,500 exemplars. Approximately 80% of the training and test data belongs to class 1, Tables 4.5.1 and 4.5.2, and therefore the default accuracy is about 80%. The aim suggested with the dataset is to obtain an accuracy of 99 - 99.9%. This dataset is originally a seven class problem however due to classes 2, 3, 6 and 7 being so small, Table 4.5.1, it has been reformulated into a 4-class problem by combining these classes. Classes 2, 3, 6 and 7 have been combined to form class 2, Table 4.5.2.

Class		Training	Test
1	Rad Flow	34,108	11,478
2	Fpv Close	37	13
3	Fpv Open	132	39
4	High	6,748	2,155
5	Bypass	2,458	809
6	Bpv Close	6	4
7	Bpv Open	11	2

Table 4.5.1: Original Shuttle Distribution

Class	Training		Test	
1	34,108	78.41%	11,478	79.16%
2	186	0.43%	58	0.40%
3	6,748	15.51%	2,155	14.86%
4	2,458	5.65%	809	5.58%
Total	43,500	100%	14,500	100%

Table 4.5.2: New Shuttle Distribution

Moreover, we are only developing binary classifiers so the problem has been broken down into 4 binary classification problems where the object is to classify in class and out of class for each of the four classes. Therefore, we are actually building classifiers for four datasets Shuttle 1, 2, 3 and 4. The distributions of the four shuttle training and test datasets are shown in Tables 4.5.3 and 4.5.4.

	Training		Test	
Shuttle 1	9,392	34,108	3,022	11,478
Shuttle 2	43,314	186	14,442	58
Shuttle 3	36,752	6,748	12,345	2,155
Shuttle 4	41,042	2,458	13,691	809

Table 4.5.3: Shuttle Datasets 1-4 Distributions

	Training		Test	
Shuttle 1	21.59%	78.41%	20.84%	79.16%
Shuttle 2	99.57%	0.43%	99.60%	0.40%
Shuttle 3	84.49%	15.51%	85.14%	14.86%
Shuttle 4	94.35%	5.65%	94.42%	5.58%

Table 4.5.4: Shuttle Datasets 1-4 Percentage Distributions

4.5.1 Balanced Block – Block Partition Ratios

Several different block partition ratios were tested to help parameterize the Balanced Block algorithm on the Shuttle datasets and to choose a best partitioning for comparison against the RSS-DSS and DSS-DSS algorithms. Block partition ratios of 25%/75% (25% class 0, 75% class 1), 37.5%%/62.5%% and 50%/50% were chosen for variety and convenience for Shuttle 1. Only one block partition ratio of 95%/5% was chosen for Shuttle 2 due to the small size of class 1 for this dataset so no comparison is made in this section. Three block partition ratios of 70%/30%, 62.5%/37.5% and 50%/50% were chosen for Shuttle 3, and two block partition ratios were chosen for Shuttle 4 of 62.5%/37.5% and 50%/50%. The results of these experiments for each Shuttle dataset are summarized in subsections 4.5.2.1 to 4.5.2.3 in terms of run time, test set accuracy, test detection rate, test false positive rate and program size after structural introns have been removed for both the Hits metric and the TNR+TPR metric. Full results, including training data results, can be found in Appendix D, Tables D.1, D.2 and D.3.

4.5.1.1 Shuttle 1

Figures 4.5.1 to 4.5.9 below show the results of the three block partition ratios attempted for the balanced block algorithm on the Shuttle 1 dataset.

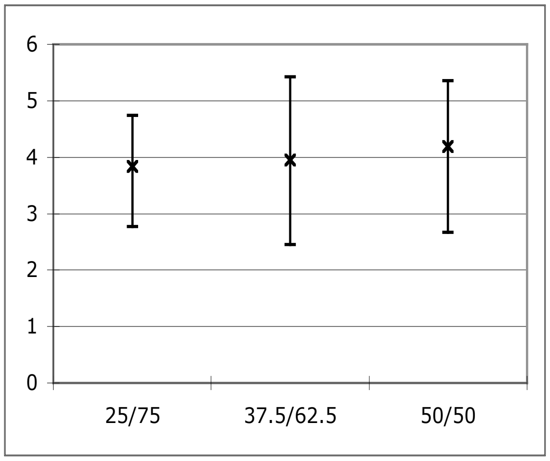

Figure 4.5.1: Shuttle 1 Partitions - Run Time (minutes)

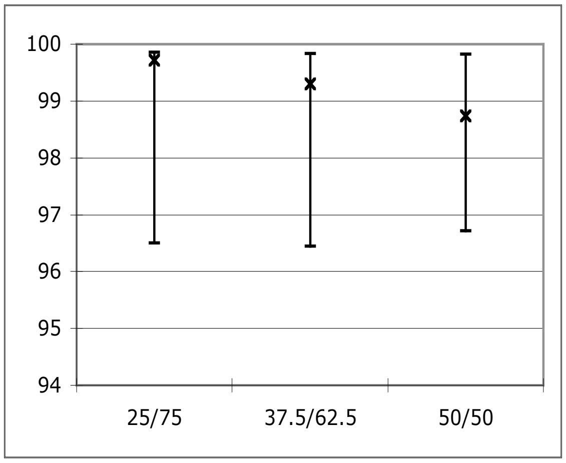

Figure 4.5.2: Shuttle 1 Partitions - Test Accuracy Hits Metric

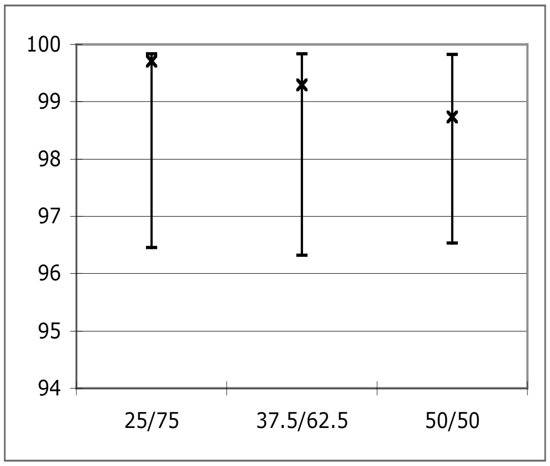

Figure 4.5.3: Shuttle 1 Partitions - Test Accuracy TN + TP Metric

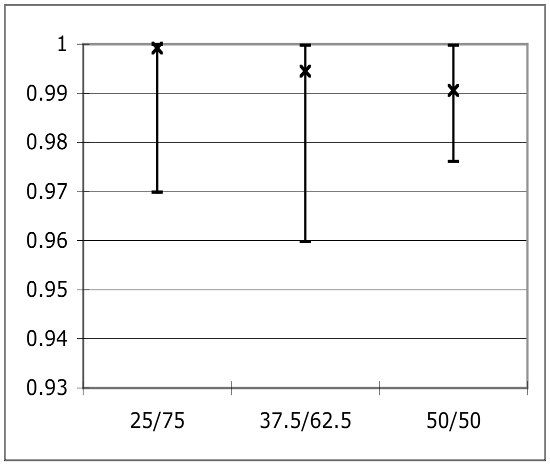

Figure 4.5.4: Shuttle 1 Partitions - Test Detection Rate Hits Metric

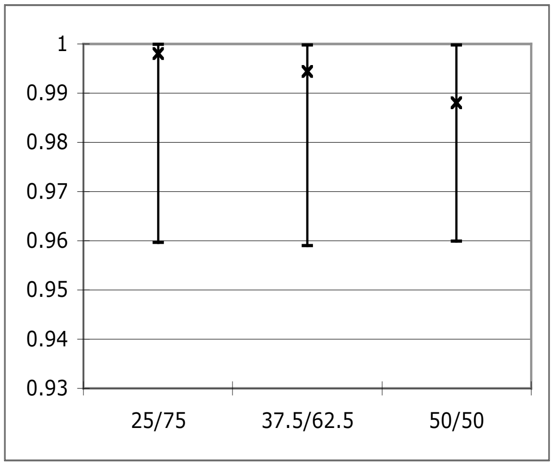

Figure 4.5.5: Shuttle 1 Partitions - Test Detection Rate TN + TP Metric

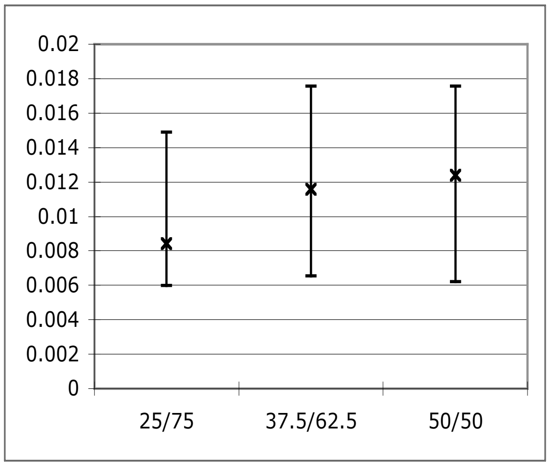

Figure 4.5.6: Shuttle 1 Partitions - Test False Positive Rate Hits Metric

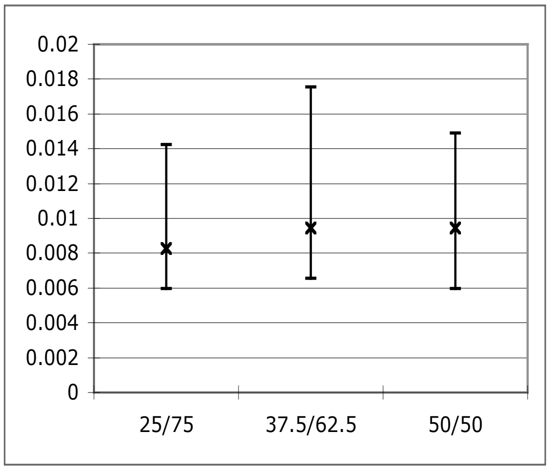

Figure 4.5.7: Shuttle 1 Partitions - Test False Positive Rate TN + TP Metric

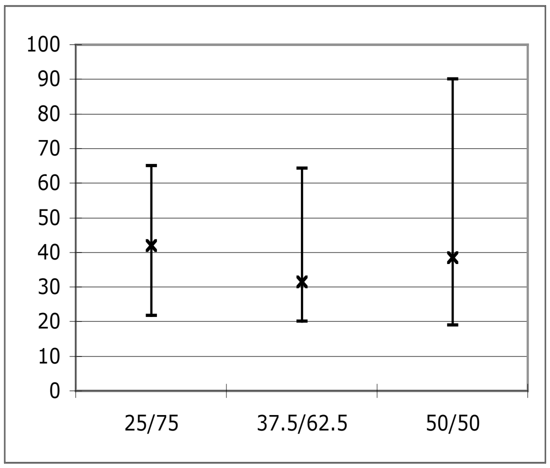

Figure 4.5.8: Shuttle 1 Partitions – Program Size after Intron Removal Hits Metric

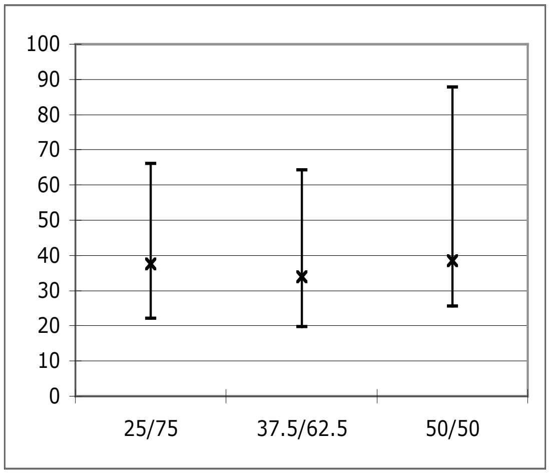

Figure 4.5.9: Shuttle 1 Partitions – Program Size after Intron Removal TN + TP Metric

The results in Figures 4.5.1 to 4.5.9 show that the 25/75 partition of the Balanced Block algorithm has the best median run time, test accuracy, test detection rate and test false positive rate for both metrics. Moreover, the 25/75 partition is the most consistent in terms of run time and false positive rate and has similar consistency to the other partitions for accuracy, detection rate and program size after intron removal. Therefore, the 25/75 block partition ratio will be chosen as the best parameterization of the Balanced Block algorithm on the Shuttle 1 dataset.

4.5.1.2 Shuttle 3

Figures 4.5.10 to 4.5.13 below show the results of the three block partition ratios attempted for the balanced block algorithm on the Shuttle 3 dataset. For this dataset individuals returned by the Hits metric and the TNR+TPR metric were the same and so only one graph of accuracy, detection rate, false positive rate and program size after intron removal is shown.

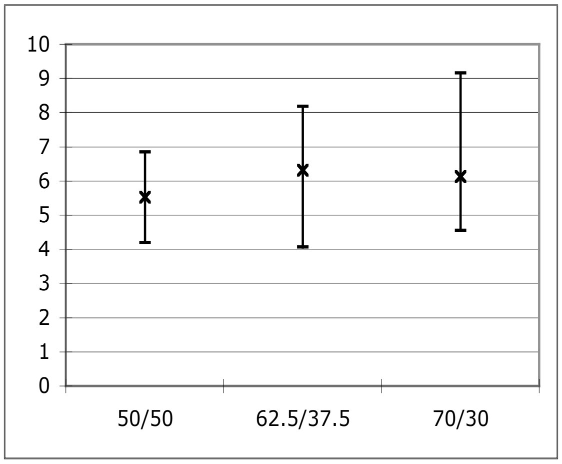

Figure 4.5.10: Shuttle 3 Partitions – Run Time (minutes)

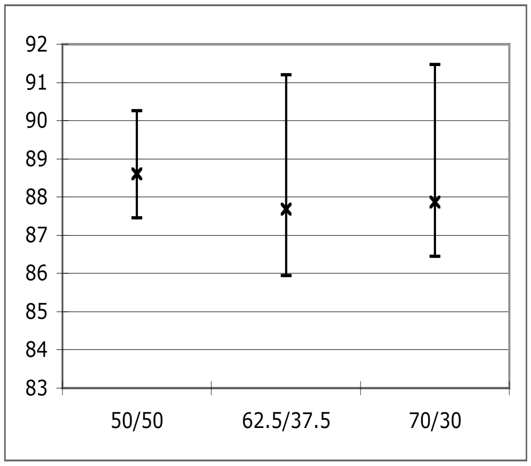

Figure 4.5.11: Shuttle 3 Partitions - Test Accuracy

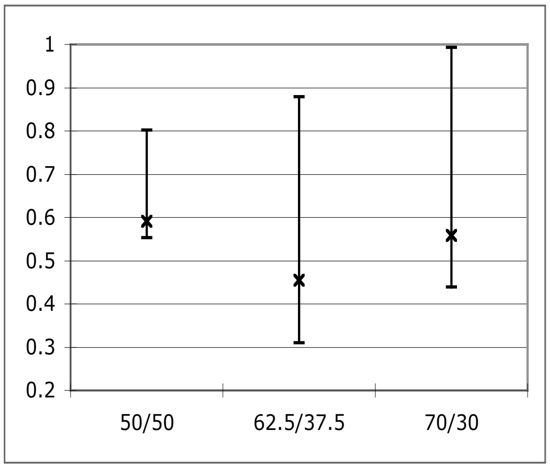

Figure 4.5.12: Shuttle 3 Partitions – Test Detection Rate

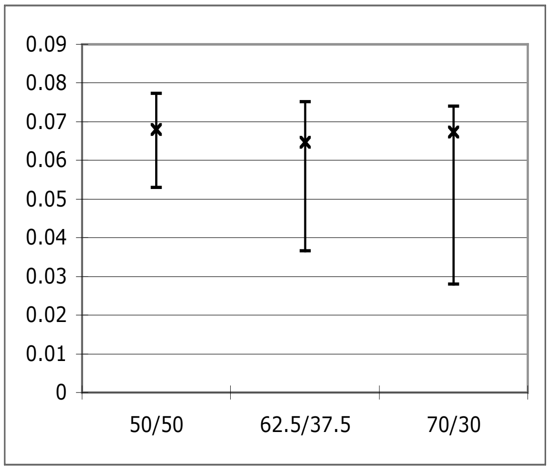

Figure 4.5.13: Shuttle 3 Partitions - Test False Positive Rate

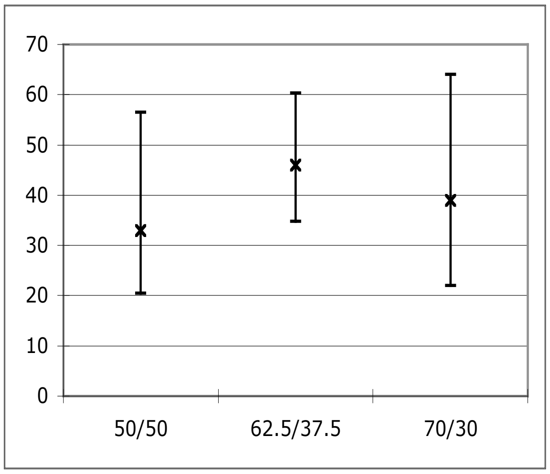

Figure 4.5.13: Shuttle 3 Partitions - Program Size after Intron Removal

The 50/50 ratio has the best median run time, test accuracy, test detection rate and program size after intron removal. It also has a similar median false positive rate to the other partitions. Furthermore, it is the most consistent partitioning in all but program size. Therefore, the 50/50 ratio is chosen as the best parameterization of the balanced block algorithm for the Shuttle 3 dataset.

4.5.1.3 Shuttle 4

Two block partition ratios were attempted for the balanced block algorithm on the Shuttle 4 dataset. For this dataset individuals returned by the Hits metric and the TNR+TPR metric were again the same and so only one graph of accuracy, detection rate, false positive rate and program size after intron removal is shown.

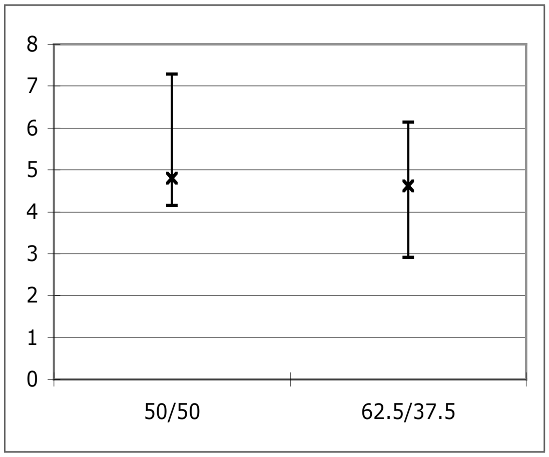

Figure 4.5.14: Shuttle 4 Partitions – Run Time (minutes)

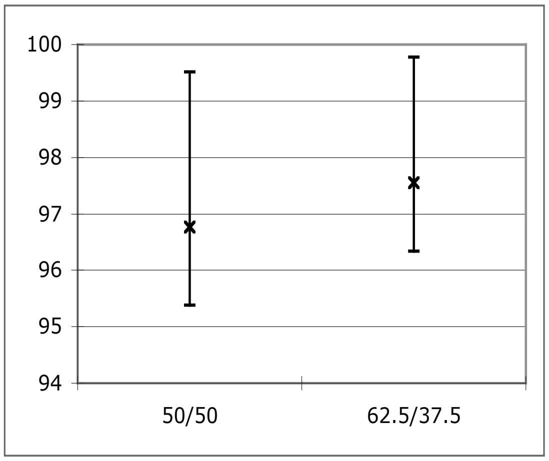

Figure 4.5.15: Shuttle 4 Partitions – Test Accuracy

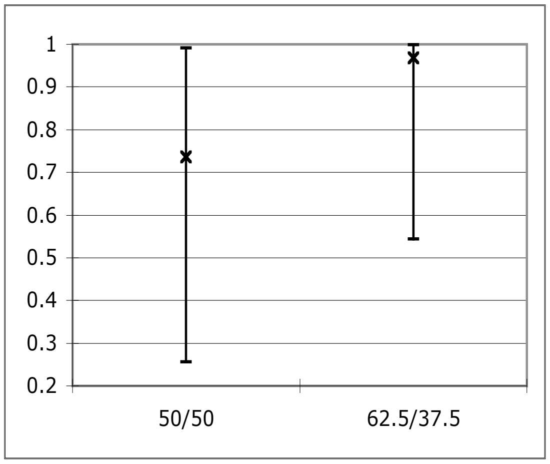

Figure 4.5.16: Shuttle 4 Partitions – Test Detection Rate

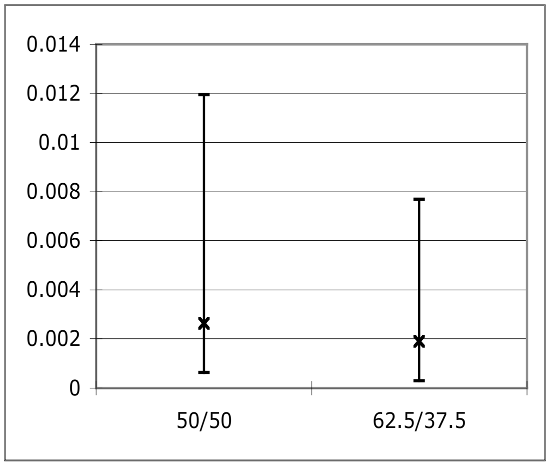

Figure 4.5.17: Shuttle 4 Partitions – Test False Positive Rate

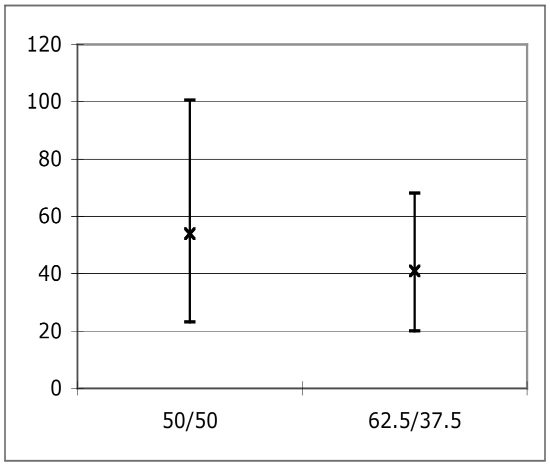

Figure 4.5.18: Shuttle 4 Partitions – Program Size after Intron Removal

The 62.5/37.5 ratio had the best median run time, test accuracy, test detection rate, test false positive rate and program size after intron removal. Furthermore it was more consistent than the 50/50 ratio in each figure. Therefore, the 62.5/37.5 ratio is chosen as the best parameterization of the balanced block algorithm for the Shuttle 4 dataset.

4.5.2 Degenerates

Table 4.5.5 lists the number of degenerate solutions out of 40 runs for each partitioning of each Shuttle dataset.

Dataset	Partition	Hits Metric	TN+TP Metric
Shuttle 1	25/75	0	0
	37.5/62.5	0	0
	50/50	0	0
Shuttle 2	95/5	27	27
Shuttle 3	50/50	17	17
	62.5/37.5	16	16
	70/30	15	15
Shuttle 4	50/50	13	13
	62.5/37.5	7	7

Table 4.5.5: Balanced Block Partitions Degenerates

Recall degenerate solutions are those programs that label all of the dataset as one class. The Shuttle 1 dataset has no degenerate solutions for any partition, however the remaining three Shuttle datasets have quite a few, with Shuttle 2 having an excessive amount. Recall that the stochastic nature of the GP algorithm requires that runs be conducted over at least 30 different initializations in order to establish the statistical significance of any results (verifying that the solutions are not due to random chance). Therefore, the Balanced Block results for the Shuttle 2 and Shuttle 3 dataset fail to meet this requirement since degenerate solutions are not considered.

Table 4.5.6 lists the number of degenerate solutions of each algorithm on each dataset with the Balanced Block algorithm using the best partitioning found in the previous section. All three algorithms successfully avoid degenerates on the Shuttle 1 dataset. Furthermore, both the RSS-DSS and DSS-DSS algorithms perform far better in terms of degenerates on the remaining three datasets only resulting in degenerate solutions on the Shuttle 2 dataset at a far less rate than the Balanced Block algorithm.

	Algorithm	Hits Metric	TNR+TPR Metric
Shuttle 1	RSS-DSS	0	0
	DSS-DSS	0	0
	Balanced (25/75)	0	0
Shuttle 2	RSS-DSS	5	5
	DSS-DSS	5	5
	Balanced (95/5)	27	27
Shuttle 3	RSS-DSS	0	0
	DSS-DSS	0	0
	Balanced (50/50)	17	17
Shuttle 4	RSS-DSS	0	0
	DSS-DSS	0	0
	Balanced (62.5/37.5)	7	7

Table 4.5.6: Algorithm Comparison of Degenerate Solutions

The high frequency of degenerate solutions found by the Balanced Block algorithm occurs on the more imbalanced Shuttle datasets (see Table 4.5.4) and especially on the extremely imbalanced Shuttle 2. These results seem to favor the choice of the RSS-DSS and DSS-DSS algorithms.

4.5.3 GP Algorithm Comparison

Despite the frequency of degenerates and the questionable significance of the Balanced Block algorithm results, the three algorithms will be compared on the four Shuttle datasets in the following subsections. Full results of these experiments can be found in Appendix D, Tables D.4, D.5, D.6 and D.7.

4.5.3.1 Shuttle 1

Figures 4.5.19 to 4.5.27 below show the comparison between the RSS-DSS, the DSS-DSS and the Balanced Block algorithm with 25/75 block partition ratio on the Shuttle 1 dataset.

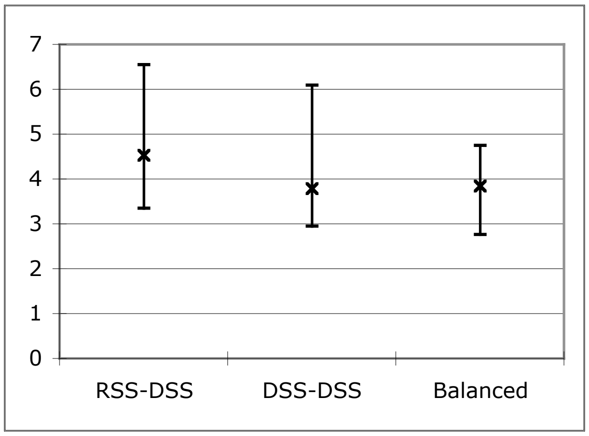

Figure 4.5.19: Shuttle 1 Algorithm Comparison - Run Time (minutes)

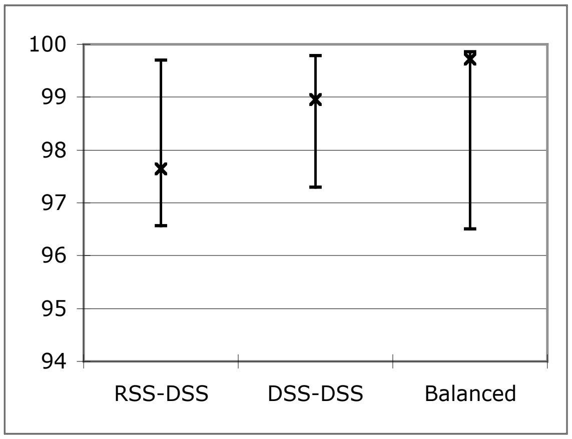

Figure 4.5.20: Shuttle 1 Algorithm Comparison – Test Accuracy Hits Metric

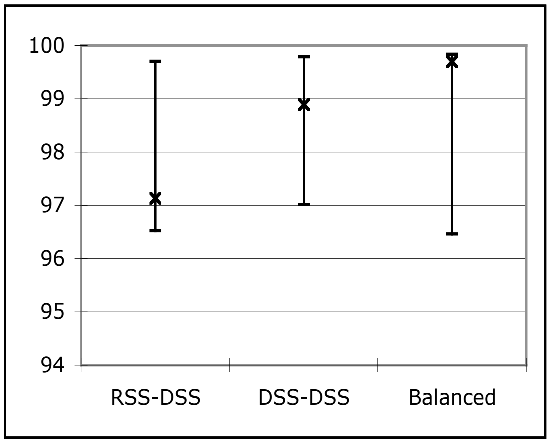

Figure 4.5.21: Shuttle 1 Algorithm Comparison – Test Accuracy TNR+TPR Metric

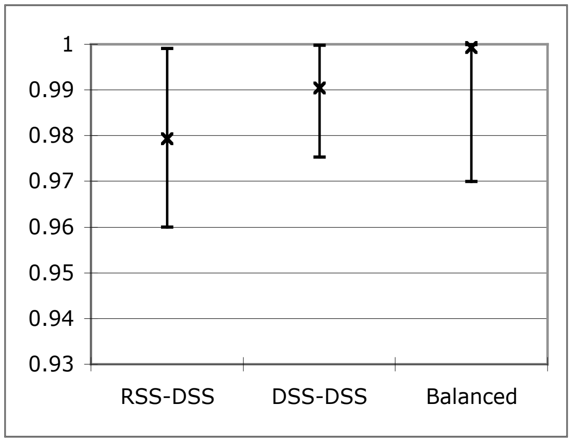

Figure 4.5.22: Shuttle 1 Algorithm Comparison – Test Detection Rate Hits Metric

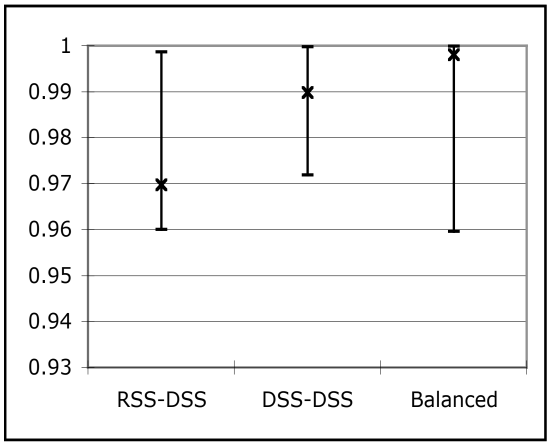

Figure 4.5.23: Shuttle 1 Algorithm Comparison – Test Detection Rate TNR+TPR Metric

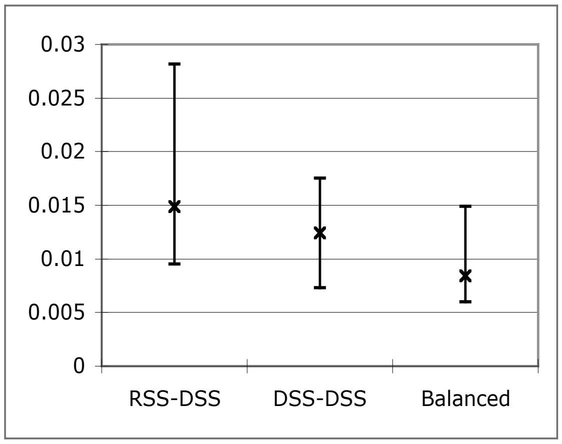

Figure 4.5.24: Shuttle 1 Algorithm Comparison – Test False Positive Rate Hits Metric

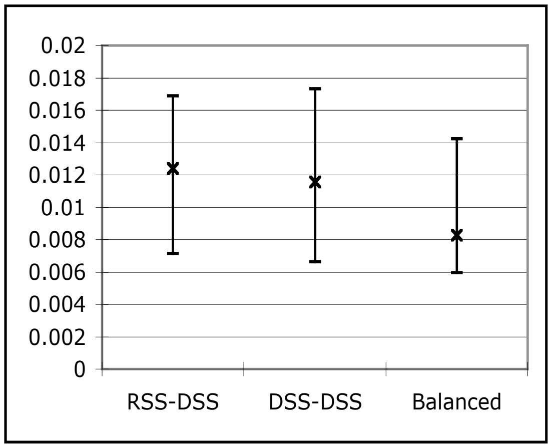

Figure 4.5.25: Shuttle 1 Algorithm Comparison – Test False Positive Rate TNR+TPR Metric

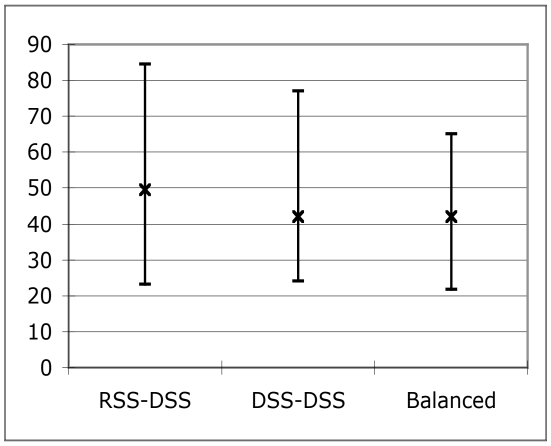

Figure 4.5.26: Shuttle 1 Algorithm Comparison – Program Size after Intron Removal Hits Metric

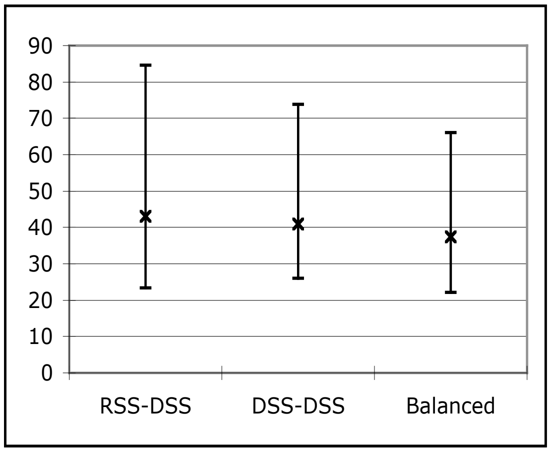

Figure 4.5.27: Shuttle 1 Algorithm Comparison – Program Size after Intron Removal TNR+TPR Metric

The Balanced Block algorithm with 25/75 block partition ratio achieves the highest median test accuracy (Figure 4.5.20 and 4.5.21), the highest median test detection rate (Figure 4.5.22 and 4.5.23), the lowest median test false positive rate (Figure 4.5.24 and 4.5.25) and the lowest median program size after intron removal (Figure 4.5.26 and 4.5.27) for both the Hits metric and the TNR+TPR metric. Furthermore, this algorithm had the most consistency in terms of run time, test false positive rate and program size after intron removal.

The DSS-DSS algorithm achieves the lowest median run time (Figure 4.5.19) and has all median values better than that of the RSS-DSS algorithm. The DSS-DSS algorithm also had the most consistency in terms of test accuracy and test detection rate. The DSS-DSS algorithm was always more consistent than the RSS-DSS algorithm.

4.5.3.2 Shuttle 2

Figures 4.5.28 to 4.5.32 below show the comparison between the RSS-DSS, the DSS-DSS and the Balanced Block algorithm with 95/5 block partition ratio on the Shuttle 2 dataset. For this dataset the Hits metric and the TNR+TPR metric returned the same individual as best for all algorithms and so only one figure is shown for each comparison.

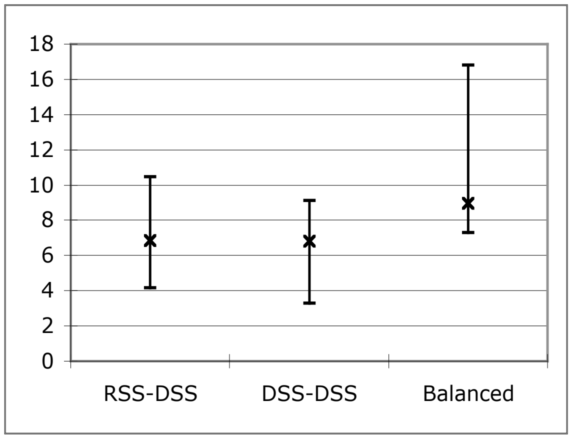

Figure 4.5.28: Shuttle 2 Algorithm Comparison – Run Time (minutes)

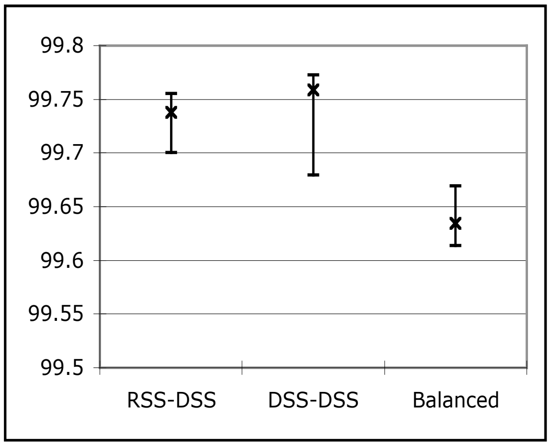

Figure 4.5.29: Shuttle 2 Algorithm Comparison – Test Accuracy

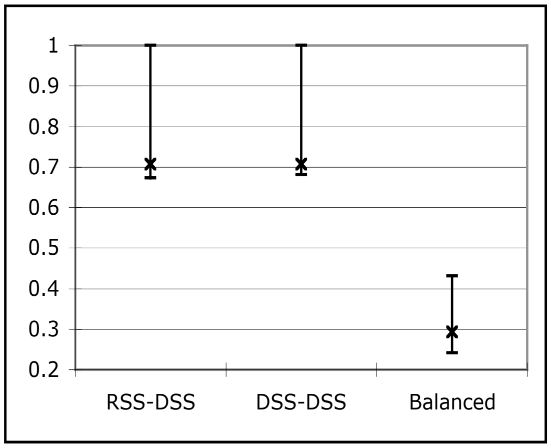

Figure 4.5.30: Shuttle 2 Algorithm Comparison – Test Detection Rate

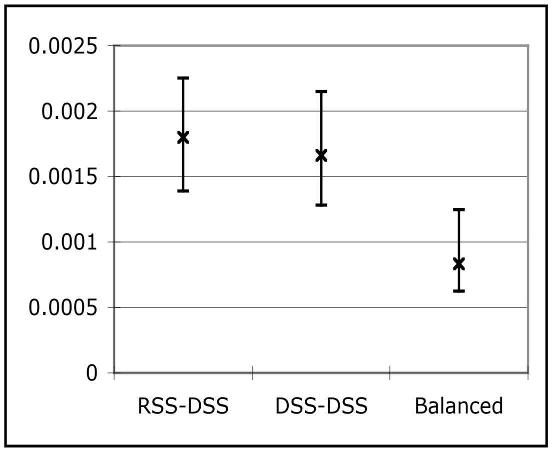

Figure 4.5.31: Shuttle 2 Algorithm Comparison – Test False Positive Rate

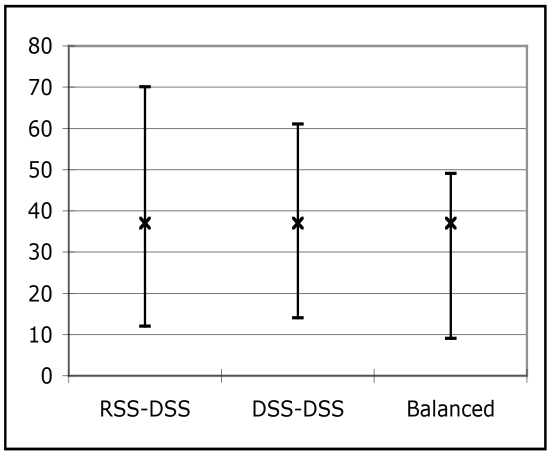

Figure 4.5.32: Shuttle 2 Algorithm Comparison – Program Size after Intron Removal

The DSS-DSS algorithm has the highest median test accuracy (Figure 4.5.29), and a better test false positive rate (Figure 4.5.31) than the RSS-DSS algorithm. The RSS-DSS algorithm and the DSS-DSS algorithm seem to have the same median run time (Figure 4.5.28), median test detection rates (Figure 4.5.30) and median program size after intron removal (Figure 4.5.32). The Balanced Block algorithm with 95/5 block partition ratio had the worst median run time and only outperformed the other algorithms in median test false positive rate. However, the Balanced Block algorithm does have the greatest consistency in all figures except for run time.

4.5.3.3 Shuttle Three

Figures 4.5.33 to 4.5.37 show the comparison of results of the three algorithms on the Shuttle 3 dataset. The Balanced Block algorithm used a 50/50 block partition ratio. The Hits metric and the TNR+TPR metric returned the same best programs for each of the algorithms and so only one set of figures needs to be shown.

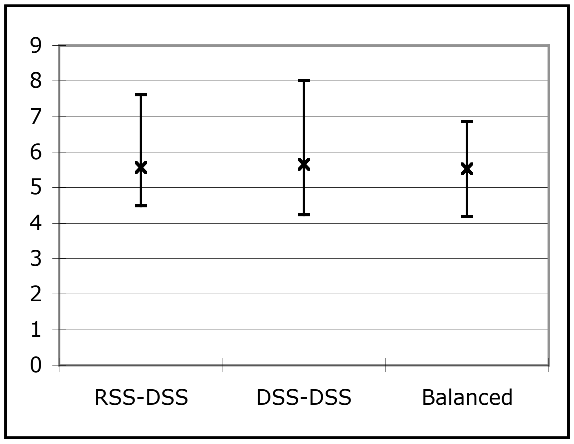

Figure 4.5.33: Shuttle 3 Algorithm Comparison – Run Time (minutes)

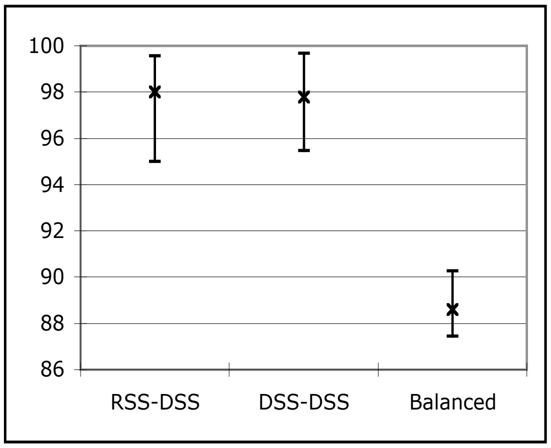

Figure 4.5.34: Shuttle 3 Algorithm Comparison – Test Accuracy

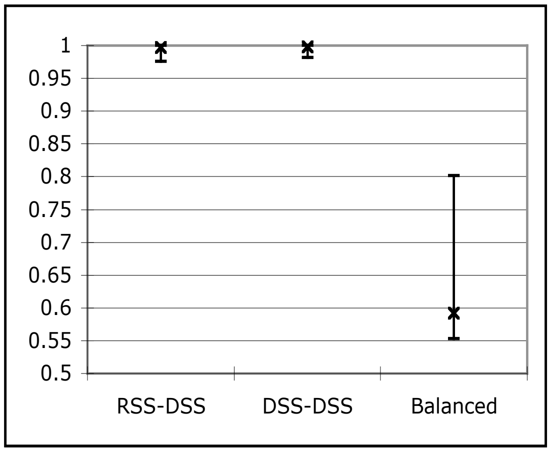

Figure 4.5.35: Shuttle 3 Algorithm Comparison – Test Detection Rate

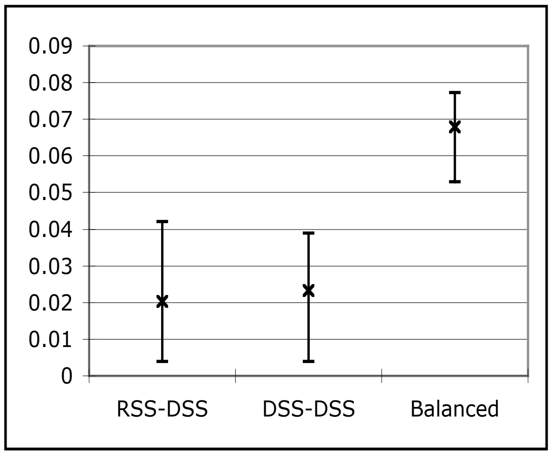

Figure 4.5.36: Shuttle 3 Algorithm Comparison – Test False Positive Rate

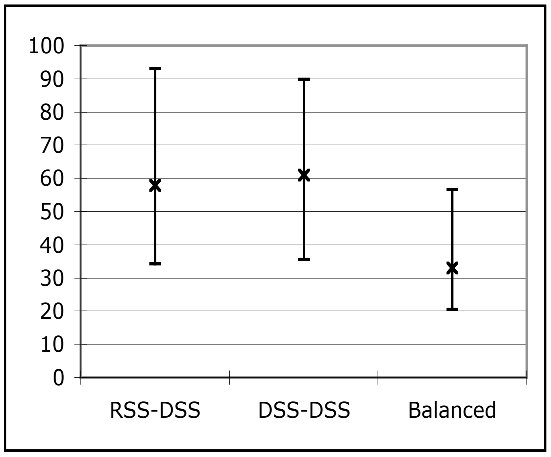

Figure 4.5.37: Shuttle 3 Algorithm Comparison – Program Size after Intron Removal

On this dataset the RSS-DSS algorithm and the DSS-DSS algorithm seemed to perform similarly. The RSS-DSS algorithm had the highest median test accuracy (Figure 4.5.34) and the lowest median false positive rate (Figure 4.5.36). All three algorithms seemed to fare the same in terms of run time (Figure 4.5.33). The Balanced Block algorithm with block partition ratio 50/50 fared significantly worse than the other two algorithms in terms of test accuracy, test detection rate and test false positive rate and only had a lower median solution size after intron removal than the other two algorithms. However, the Balanced Block algorithm again appeared to be the most consistent algorithm only faring worse in terms of detection rate of which the RSS-DSS and DSS-DSS algorithm get almost perfect results.

4.5.3.4 Shuttle 4

Figures 4.5.38 to 4.5.42 show the comparison of results between the three GP algorithms on the Shuttle 4 dataset. The Balanced Block algorithm used a 62.5/37.5 block partition ratio here. Again, for this dataset the Hits metric and the TNR+TPR metric again returned the same programs as the best results and so only one set of figures needs to be shown.

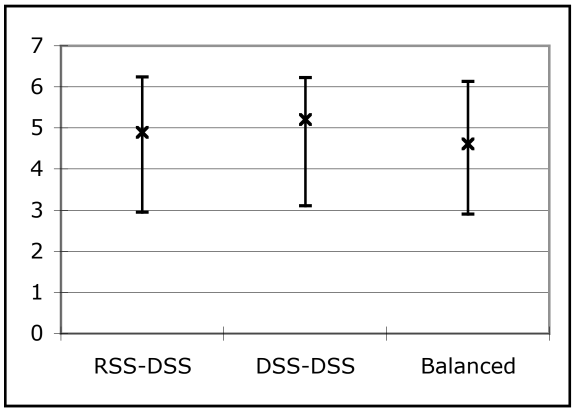

Figure 4.5.38: Shuttle 4 Algorithm Comparison – Run Time (minutes)

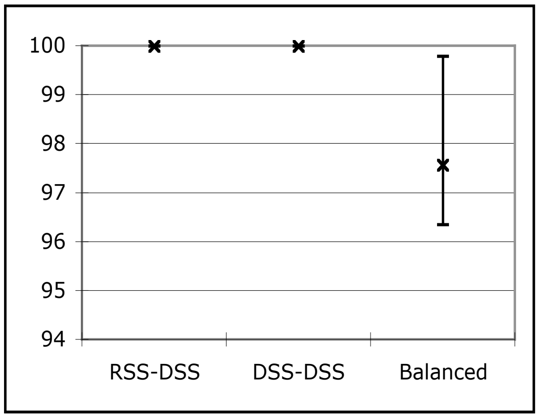

Figure 4.5.39: Shuttle 4 Algorithm Comparison – Test Accuracy

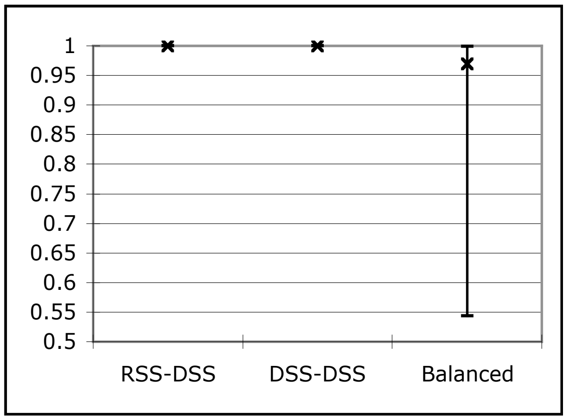

Figure 4.5.40: Shuttle 4 Algorithm Comparison – Test Detection Rate

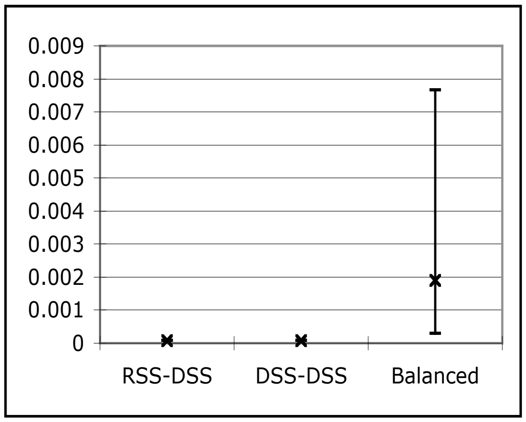

Figure 4.5.41: Shuttle 4 Algorithm Comparison – Test False Positive Rate

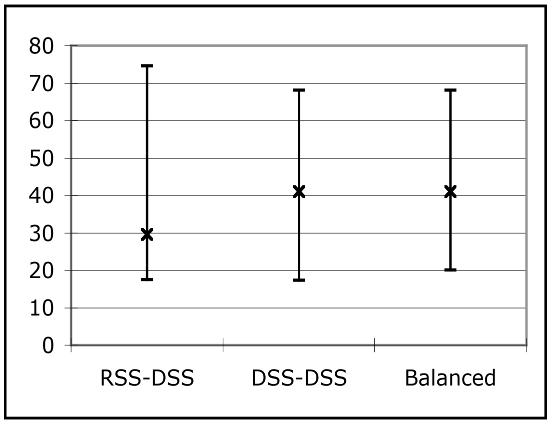

Figure 4.5.42: Shuttle 4 Algorithm Comparison – Program Size after Intron Removal

For this dataset the RSS-DSS and DSS-DSS algorithms achieve nearly perfect results in terms of test accuracy (Figure 4.5.39), test detection rate (Figure 4.5.40) and test false positive rate (Figure 4.5.41). The RSS-DSS algorithm also had the best median solution size after intron removal (Figure 4.5.42). The Balanced Block algorithm with block partition ratio 62.5/37.5 had the best median run time but in general did not fare as well on Shuttle 4 as the other two GP algorithms.

4.5.4 Shuttle Programs

In this section some small programs from the Shuttle dataset will be examined. The Balanced Block algorithm found all the following programs in Tables 4.5.7 to 4.5.10. The programs are small but perform rather well and demonstrate the power of GP to find simple transparent solutions. Furthermore, we can glean information about the relationship between a class and the features that determine whether an exemplar belongs to that class, through examining the features used by a program.

	Opcode	Destination	Source
1	DIV	R[7]	Input[8]
2	LOD	R[7]	53
3	SUB	R[7]	Input[0]
4	DIV	R[7]	Input[8]
5	DIV	R[7]	Input[8]

Table 4.5.7: Shuttle One – Balanced Block 25/75 Smallest Program

This program was found by the Balanced Block algorithm using the best partitioning of 25/75. This small five instruction program can be further simplified. Loading register seven with 53 in instruction two makes instruction one's result inconsequential. The resulting four instruction program is very small but achieves an accuracy of 95.37% with a test detection rate of 0.9447, a test false positive rate of 0.0122 for a TNR+TPR value of 1.9325. Furthermore, this five instruction program makes use of Input[0] and Input[8], which corresponds to the first feature and the last feature of an exemplar. This result gives us insight into class one exemplars since their classification relies heavily on only two of an exemplars nine features.

	Opcode	Destination	Source
1	ADD	R[1]	Input[6]
2	LOD	R[1]	20
3	SUB	R[1]	Input[8]
4	LOD	R[1]	20
5	ADD	R[1]	Input[1]
6	DIV	R[4]	R[3]
7	SUB	R[4]	R[1]
8	SUB	R[4]	Input[1]

Table 4.5.8: Shuttle 2 – Balanced Block 95/5 Smallest Program

This small eight instruction program was found by the Balanced Block algorithm using the 95/5 block partitioning. This program can also be further simplified. Loading register one with the value 20 at instruction four makes the previous three instructions redundant. Therefore, this program simplifies to a five instruction program which achieves a 99.73% accuracy with a test detection rate of 0.7069, a test false positive rate of 0.0015 for a

TNR+TPR value of 1.7054. Furthermore, this five instruction program only makes use of Input[1], which corresponds to the second feature of the exemplars. This result gives us insight into class two exemplars since there classification relies heavily on the second feature of an exemplar.

	Opcode	Destination	Source
1	SUB	R[4]	Input[0]
2	ADD	R[4]	Input[6]
3	LOD	R[0]	76
4	MUL	R[4]	Input[8]
5	MUL	R[4]	Input[8]
6	SUB	R[5]	R[0]
7	SUB	R[5]	R[4]
8	SUB	R[5]	R[0]

Table 4.5.9: Shuttle 3 – Balanced Block 50/50 Smallest Program

This program was found by the Balanced Block algorithm on the Shuttle 3 dataset using the best partitioning 50/50. This small eight instruction program can not be further simplified. This program is still small and achieves an accuracy of 94.12% with a test detection rate of 1, a test false positive rate of 0.0690 for a TNR+TPR value of 1.9310. Class three exemplars seem to rely on inputs 0, 6 and 8 corresponding to features 1, 7 and 9 of an exemplar.

	Opcode	Destination	Source
1	SUB	R[1]	Input[6]
2	ADD	R[5]	Input[6]
3	LOD	R[1]	80
4	LOD	R[1]	13
5	ADD	R[1]	R[5]
6	LOD	R[1]	13
7	SUB	R[1]	Input[6]
8	SUB	R[1]	Input[6]

Table 4.5.10: Shuttle 4 – Balanced Block 62.5/37.5 Smallest Program

This program was found by the Balanced Block algorithm using the best partitioning of 62.5/37.5. This small eight instruction program can be further simplified. Loading register one with 13 in instruction six makes the previous five instructions inconsequential. The resulting three instruction program is very small and achieves an accuracy of 99.99% with a test detection rate of 1, a test false positive rate of 0.000073 for a TNR+TPR value of 1.999927. This three instruction program also gives us insight into class four exemplars, in that they are almost perfectly classified through only considering the seventh feature (features 1 to 9 correspond to Input[0] to Input[8]).

4.5.5 Combining Programs to Build an Overall Classifier

In order to obtain an overall result on the full Shuttle dataset for comparison with other algorithms it is necessary to combine the individual classifiers together as in [38]. This was achieved by taking the best TNR+TPR classifiers from each of the individual Shuttle datasets (Shuttle 1 to 4) for an algorithm. The program with the lowest number of false positives out of these best programs is then applied to the dataset. Every exemplar that is classified as negative by this program is then passed on to the program with the next lowest number of false positives. This process is repeated until each of the four programs has been run. The resulting overall classification accuracy is calculated by adding up the correctly classified results from each program and dividing by the total number of test exemplars.

For example in the RSS-DSS algorithm the best results from each shuttle dataset are shown in Table 4.5.11.

Dataset	Accuracy	TPR+TNR	False Negatives	False Positives
Shuttle 1	99.86%	1.9938	3	18
Shuttle 2	99.83%	1.9983	0	25
Shuttle 3	99.88%	1.9985	0	18
Shuttle 4	99.99%	1.9999	0	1

Table 4.5.11: Best Case TNR+TPR Results by RSS-DSS Algorithm on Shuttle Datasets

So, we then apply the best programs in order of 4, 3, 1 and then 2 according to the lowest number of false positives.

Program	# of exemplars	Accuracy	Correct	Incorrect
4	14500	99.99%	809	1
3	13690	99.91%	2155	13
1	11522	99.82%	11473	18
2	31	100%	28	0
	3 leftover		14465	32

Table 4.5.12: Results after Each Program on Test Data

The first line of Table 4.5.12 shows the result of applying the best program of the RSS-DSS algorithm on the overall Shuttle dataset. The program labels 810 exemplars, 809 of which were correct and 1 of which was not. The remaining 13,960 exemplars were deemed negative and so they are passed on to the program with the next least number of false positives, which comes from the Shuttle 3 results. This process is continued until all of the programs have run.

The programs are applied in order of the least number of false positives so that the programs that classify the most data incorrectly will effectively see less of the data. This gives the poorer programs less chance to label the data incorrectly. This effect can be seen in Table 4.5.12 as the number of incorrectly classified exemplars for program three has dropped to 13 compared to the number of false positives normally achieved by program three of 18. Furthermore, program four has improved classifying no exemplars incorrectly.

The result of applying the programs in this manner has resulted in 14,465 exemplars classified correctly and 35 exemplars (32 incorrect plus 3 unclassified) were incorrectly classified. Dividing 14,465 by the total number of exemplars 14,500 gives an overall classification accuracy of 99.76%.

4.5.6 Comparison with Other Algorithms

Table 4.5.13 shows the best-case results of the hierarchical GP algorithms versus results reported using parallel genetic programming with an ensemble of classifiers.

Shuttle	Test Accuracy
CGPC	94.82%
BagCGPC	94.63%
BagCGPC without comm.	91.46%
RSS-DSS	99.76%
DSS-DSS	99.78%
Balanced Block	99.87%

Table 4.5.13: Algorithms versus Ensemble Parallel GP's

The last three entries of Table 4.5.13 show the best test accuracies achieved by the three GP algorithms on the entire Shuttle dataset. The overall accuracies are calculated following the method outlined in section 4.5.5. The first three entries in Table 4.5.13 are

results reported by Folino et al. in [31] where ensemble techniques for parallel GP's are investigated (as discussed in 4.2.4). In particular the cellular model of parallel genetic programming (CGPC – Cellular Genetic Programming Classifier) was employed. BagCGPC is an extension to CGPC where an ensemble of classifiers is created. Each classifier is trained on a different subset of the overall dataset and exchange outermost individuals in an asynchronous way (communication). The third entry BagCGPC without communication removes the communication used in BagCGPC.
All of our classifiers perform very well outperforming the ensemble parallel GP's and approaching the upper limit goal of 99-99.9% accuracy. The Balanced Block algorithm achieved the best test accuracy with 99.87%. Despite the advantages of parallelizing GP and using an ensemble of classifiers our algorithms outperform these results by almost 5%. However, our classifiers were built on a four class problem evolving classifiers for each of the four classes while it is assumed that CGPC results were obtained on the original 7 class dataset.

	Run Time	Size	Accuracy
C5.0 Tree	0.065	16	99.9%
C5.0 Tree (0.1)	0.067	16	99.9%
C5.0 Tree (0.001)	0.063	15	99.9%
C5.0 Tree (0.0001)	0.063	13	99.8%
C5.0 Rule	0.108	13	99.9%
C5.0 Rule (0.1)	0.108	13	99.9%
C5.0 Rule (0.001)	0.102	13	99.9%
C5.0 Rule (0.00001)	0.095	11	99.8%

Table 4.5.14: C5.0 Results on Shuttle Dataset (pruning level in brackets)

Table 4.5.14 shows the results of running C5.0 on the full Shuttle dataset. C5.0 obtains almost perfect classification accuracy with a very small run time and small trees or rules for all cases. C5.0 test accuracies are as good or better than our GP test accuracies (C5.0 rounds up to the first decimal place).

5 Conclusions and Future Work

The computational overhead of the inner loop fitness evaluation of GP is addressed by introducing a hierarchy of training subset selections. This enables the scaling up of the size of problems for which approaches based on GP may be applied. To do so, the original problem is divided into a series of blocks. Blocks are either selected uniformly (RSS), relative to their age and difficulty (DSS) or can be 'balanced' consisting of partitions chosen relative to their respective partition age and difficulty (Balanced Block). Exemplars are sampled from the chosen subset relative to their exemplar age and difficulty (DSS). Such schemes match the observations used to formulate the memory hierarchy typically employed in computer architectures.

The hierarchical subset selection algorithms presented here were compared to a standard tree GP implementation (lilgp) on the Adult dataset demonstrating that these hierarchies have not negatively impacted classification performance whilst resulting in a computational speed up of one order of magnitude. Furthermore, these algorithms provided faster results than previously reported on the 10% KDD'99 dataset by three orders of magnitude, although previous results are not the binary version of the dataset utilized here.

The RSS-DSS algorithm and the DSS-DSS algorithm proved to be robust towards a variety of parameters settings, including block size, exemplar age/difficulty ratios and block age/difficulty ratios for the DSS-DSS algorithm. The DSS-DSS algorithm outperformed the RSS-DSS algorithm for the most part, and was typically the more consistent algorithm (tighter spread of quartiles). However, both algorithms appeared sensitive to unbalanced datasets returning degenerate solutions.

The Balanced Block algorithm was motivated by the encouraging results discovered when datasets were manually balanced in to blocks and by the plight of the GP algorithm when faced with unbalanced datasets. The Balanced Block algorithm proved to be robust in terms of several parameters, including the alpha parameter and the block partition ratio. Furthermore, the algorithm was typically competitive with the RSS-DSS and DSS-DSS algorithms and was often more consistent (tighter spread of quartiles) than these algorithms. However, contrary to the motivation behind the algorithm the Balanced Block algorithm suffered on the datasets that were the most unbalanced and frequently

returned degenerate solutions. In effect, enforcing a balanced block content may be resulting in seeing too much of the minor class and therefore overlearning on this class. Alternatively, the number of training epochs may need to be reconsidered, where this was a constant across all datasets.
Therefore, the optimal hierarchical subset selection algorithm appears to be the DSS-DSS algorithm, which is more consistent and outperforms the RSS-DSS algorithm, and often outperforms the Balanced Block algorithm while remaining less prone to degenerate solutions.

Two GP evaluation metrics were also examined in this work: the Hits metric and the TNR+TPR metric. These metrics were used to determine the best GP programs post training. The Hits metric usually returns a higher accuracy than the TNR+TPR metric and concentrates on learning to classify the larger class in the dataset. The TNR+TPR metric, on the other hand, tries to maximize the sum of the true negative rate plus the true positive rate. This metric treats both classes as equally important regardless of the original distribution. This usually results in a much better accuracy on the smaller class than the Hits metric but the trade off is a poorer accuracy on the larger class that translates to a lower overall classification accuracy. However, for heavily unbalanced datasets the TNR+TPR metric was shown to help the GP algorithm to avoid useless degenerate solutions. This metric could also be useful for problems where learning to classify the smaller class is of greater importance than a high accuracy on the larger class.

These methods differ from alternative sampling algorithms such as ‘boosting’ by introducing the concept of age and difficulty (boosting is explicitly based on exponentially weighted difficulties) and utilizing a hierarchy of samples. The latter point is fundamental in efficiently scaling the algorithm to larger datasets than is normally possible without specialist hardware. That is to say, competitive solutions are located in minutes rather than hours. Furthermore the framework of these algorithms is not specific to GP and thus potentially applicable to other learning algorithms such as neural networks or unsupervised learning algorithms [36].

The C5.0 tree induction algorithm was also run on the datasets found here for comparison. C5.0 obtains very good results in terms of accuracy on all datasets performing as well as, and typically better than, the GP algorithms. C5.0 provided solutions to the classification problems much faster than the GP algorithms and usually required a lower number of tree nodes or rules than GP program instructions. However, it is unknown how instructions and C5.0 tree nodes or rules compare. On the other hand, GP results using the TNR+TPR metric were able to provide better results than C5.0 in terms of this metric and in accuracy on the smaller class. In all, these results suggest that GP needs improvement to be considered an equivalent alternative to C5.0 for binary classification problems.

Relative to other work in which GP is explicitly applied to large datasets the proposed methodology appears not to degrade the quality of solutions found (Adult and 10% KDD'99) whilst providing speedups in the range of several orders of magnitude. Moreover, no specialist hardware resources are required.

Future work

Currently for all of the hierarchical subset selection algorithms used here there is no persistence of exemplar ages and difficulties beyond the current block. Future work will involve investigating the maintaining of exemplar values globally beyond the current block through possibly maintaining a history of the most difficult or significant exemplars. Then these exemplars could be revisited and concentrated on in later generations.

The plight of our GP algorithms on unbalanced datasets and Balanced Block algorithm's failure to be of benefit on unbalanced data warrants further investigation. In particular, the use of a fixed stopping criterion that is independent of block fitness used in this work may be negatively impacting the Balanced Block algorithm. The inclusion of an algorithm for early stopping could potentially prevent any over learning and provide an additional speedup in run time. Furthermore, the Balanced Block algorithm's ability to achieve superior consistency, could be related to this problem, and is also worth investigating.

Other work could involve the building of a cascade of hierarchical subset selection classifiers where each additional classifier could take as inputs the original dataset as well as the output from the original classifier. This should make it possible to incrementally improve on the performance of the previous classifier. Such an architecture was considered utilizing the RSS-DSS hierarchy, resulting in the Cascade GP algorithm [36].

Bibliography

[1] Bennett III F.H., Koza J.R., Shipman J., Stiffelman O.: Building a Parallel Computer System for $18,000 that Performs a Half Petra-Flop per Day. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan Kaufmann (1999) 1484-1490

[2] Juillé H., Pollack J.B.: Massively Parallel Genetic Programming. In: Angeline P.J., Kinnear K.E. (eds): Advances in Genetic Programming 2, Chapter 17. MIT Press, Cambridge, MA (1996) 339-358

[3] Koza J.R., Bennett III F.H., Hutchings J.L., Bade S.L., Keane M.A., Andre D.: Evolving Computer Programs using Reconfigurable Gate Arrays and Genetic Programming. Proceedings of the ACM 6th International Symposium on Field Programmable Gate Arrays. ACM Press. (1998) 209-219

[4] Breiman L.: Bagging predictors. Machine Learning. 24(2) (1996) 123-140

[5] Freund Y., Schapire R.E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and Systems Sciences. 55 Academic Press. (1997) 119-139

[6] Hennessy J.L., Patterson D.A.: Computer Architecture: A Quantitative Approach. 3rd Edition. Morgan Kaufmann, San Francisco, CA (2002)

[7] Gathercole C., Ross P.: Dynamic Training Subset Selection for Supervised Learning in Genetic Programming. Parallel Problem Solving from Nature III. Lecture Notes in Computer Science. Vol. 866. Springer Verlag. (1994) 312-321

[8] Song D., Heywood M.I., Zincir-Heywood A.N.: A Linear Genetic Programming Approach to Intrusion Detection. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). Lecture Notes in Computer Science. Vol. 2724 Springer-Verlag. (2003) 2325-2336

[9] Cramer N.L.: A Representation for the Adaptive Generation of Simple Sequential Programs. Proceedings of the International Conference on Genetic Algorithms and Their Application (1985) 183-187

[10] Nordin P.: A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In: Kinnear K.E. (ed.): Advances in Genetic Programming, Chapter 14. MIT Press, Cambridge, MA (1994) 311-334

[11] Huelsbergen L.: Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations. Proceedings of the 3rd Conference on Genetic Programming. Morgan Kaufmann, San Francisco, CA (1998) 158-166

[12] Heywood M.I., Zincir-Heywood A.N.: Dynamic Page-Based Linear Genetic Programming. IEEE Transactions on Systems, Man and Cybernetics – PartB: Cybernetics. 32(3) (2002), 380-388

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

[14] Elkan C.: Results of the KDD'99 Classifier Learning Contest. SIGKDD Explorations. ACM SIGKDD. 1(2), (2000) 63-64

[15] UCI Machine Learning Repository. (2003) http://www.ics.uci.edu/~mlearn/MLRepository.html

[16] Lichodzijewski P., Zincir-Heywood A.N., Heywood M.I.: Host-Based Intrusion Detection Using Self-Organizing Maps. IEEE-INNS International Joint Conference on Neural Networks. (2002) 1714-1719

[17] Brameier M., Banzhaf W.: A Comparison of Linear genetic Programming and Neural Networks in Medical data Mining. IEEE Transaction on Evolutionary Computation. 5(1) (2001) 17-26

[18] Chittur A.: Model Generation for Intrusion Detection System using Genetic Algorithms. http://www1.cs.columbia.edu/ids/publications/ (2001) 17 pages

[19] Punch B., Goodman E.: lilgp Genetic Programming System, v 1.1. http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html (1998).

[20] Koza J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA (1994)

[21] Banzhaf W. et al.: Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers, San Francisco, CA (1998)

[22] Witten I., Frank E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, (2000)

[23] Mitchell M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA (1996)

[24] Freitas A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag, (1998)

[25] Gathercole C., Ross P.: Small Populations over Many Generations can beat Large Populations over Few Generations in Genetic Programming. Genetic Programming 1997:

Proceedings of the Second Annual Conference, 13-16, Morgan Kaufmann, Stanford University, CA, 111-118, (1997)

[26] Lasarczyk C., Dittrich P., Banzhaf W.: Dynamic Subset Selection Based on a Fitness Case Topology. Evolutionary Computation, 12 (2), 223-242, (Summer 2004)

[27] Gathercole C., Ross P.: Tackling the Boolean Even N Parity Problem with Genetic Programming and Limited-Error Fitness. Genetic Programming 1997: Proceedings of the Second Annual Conference, 13-16, Morgan Kaufmann, Stanford University, CA, USA, 119-127, (1997)

[28] Zhang B.T., Cho D.Y.: Genetic Programming with Active Data Selection. Lecture Notes in Computer Science, 1585, 146-153, 1999

[29] Andre D., Koza J.R.: A Parallel Implementation of Genetic Programming that Achieves Super-Linear Performance. In Arabnia, Hamid R.(editor), Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. Athens, GA: CSREA. Volume III. Pages 1163-1174.

[30] Paris G., Robilliard D., Fonlupt C.: Applying Boosting Techniques to Genetic Programming. EA 2001, Lecture Notes in Computer Science, 2310, 267-278, Springer Verlag, 2002

[31] Folino G., Pizzut C., Spezzano G.: Ensemble Techniques for Parallel Genetic Programming Based Classifiers. EuroGP 2003 Proceedings, Lecture Notes in Computer Science, Vol. □610, 2003

[32] Curry R., Heywood M.: Towards Efficient Training on Large Datasets for Genetic Programming. 17th Canadian Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, 3060, 161-174, Springer-Verlag, 2004

[33] Dong S., Curry R., Heywood M.I., Zincir-Heywood A. N., On the Efficient Mining of Network Audit Data using Genetic Programming. Proceedings of the Workshop on Military and Security Applications of Evolutionary Programming (Accepted for publication), GECCO'2004, Seattle, USA, 2004.

[34] Pfahringer B.: Winning the KDD99 Classification Cup: Bagged Boosting. SIGKDD Explorations, 1(2): 65-66, 2000

[35] Lichodzijewski P., Heywood M.I., Zincir-Heywood A.N.: Cascaded GP Models for Data Mining, IEEE Congress on Evolutionary Computation, CEC 04, Portland, Vol.2: 2258-2264, 2004, ISBN 0-7803-8515-2.

[36] Wetmore L.: Hierarchical Dynamic Subset Selection Applied to Self-Organizing Maps. Master of Computer Science Thesis. Dalhousie University. (2004) 101 pages.

[37] Syswerda G.: A Study of Reproduction in Generational and Steady State Genetic Algorithms. Foundations of GAs, Morgan Kaufmann, 1991, Rawlins (ed)

[38] McIntyre A.R., Heywood M.I.: On Multi-Class Classification by way of Niching. Proceedings of Genetic and Evolutionary Computation Conference (GECCO), 581-592, Lecture Notes in Computer Science 3103, 2004

[39] Quinlan R.: Data Mining Tools See5 and C5.0. (2004) http://www.rulequest.com/see5-info.html

[40] Wineberg M., Christensen S.: Using Appropriate Statistics – Statistics for Artificial Intelligence. Tutorial Program of Genetic and Evolutionary Computation Conference (GECCO), 544-564, 2004

Appendix A Adult Dataset Results

Table A.1 RSS-DSS Block Sizes

Hits Metric

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				250				
5.79	79.36	0.8996	0.4195	79.50	0.8974	0.4279	117	53.25
7.26	80.58	0.9227	0.5174	80.59	0.9229	0.5181	159	89
9.24	81.71	0.9433	0.6097	81.65	0.9433	0.6068	201	119.5
				500				
5.98	79.30	0.8965	0.4306	79.36	0.8940	0.4336	127	45
8.06	81.06	0.9213	0.4928	81.00	0.9197	0.4918	167	79
9.69	81.71	0.9583	0.6423	81.60	0.9576	0.6426	217	101.75
				750				
5.11	78.61	0.8998	0.4361	78.68	0.8976	0.4355	102.25	38.75
7.22	79.31	0.9586	0.7089	79.42	0.9578	0.7108	143	58
9.43	81.24	0.9833	0.7998	81.23	0.9837	0.8030	191	84.5
				1000				
7.17	79.36	0.8821	0.4089	79.41	0.8814	0.4124	135	66.25
8.69	80.86	0.9168	0.4985	80.77	0.9148	0.4953	167	91
10.64	81.58	0.9593	0.6869	81.65	0.9590	0.6857	223	126
				2500				
6.23	78.43	0.9166	0.4735	78.62	0.9158	0.4770	77	30.5
9.19	79.29	0.9298	0.6211	79.52	0.9297	0.6159	142	51.5
12.07	80.53	0.9674	0.7876	80.77	0.9698	0.7892	191	102.5
				5000				
10.28	78.28	0.9304	0.5348	78.57	0.9298	0.5325	101	42.75
12.35	79.14	0.9597	0.7574	79.21	0.9601	0.7541	135	63
15.04	80.65	0.9809	0.7993	80.96	0.9830	0.8027	175	97

TNR+TPR Metric

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				250				
5.87	72.78	0.6791	0.1217	72.74	0.6800	0.1195	119	48
7.28	75.38	0.7221	0.1391	75.13	0.7158	0.1368	159	75
9.24	77.06	0.7447	0.1568	76.95	0.7434	0.1553	203	100.5
				500				
6.34	72.97	0.6837	0.1237	72.96	0.6809	0.1195	135	43.5
8.22	75.10	0.7123	0.1371	75.20	0.7140	0.1338	167	81
9.71	76.48	0.7425	0.1580	76.71	0.7456	0.1562	219	108.5
				750				
5.29	71.60	0.6578	0.1078	71.73	0.6589	0.1036	103	34.5
7.28	73.27	0.6838	0.1188	73.48	0.6849	0.1138	143	65
9.43	75.13	0.7134	0.1432	75.42	0.7175	0.1388	191	93.5
				1000				
7.17	74.19	0.6972	0.1233	74.19	0.7004	0.1199	135	53
8.69	75.85	0.7242	0.1416	76.16	0.7276	0.1377	167	76.5
10.64	77.68	0.7556	0.1627	77.45	0.7534	0.1631	223	113.75
				2500				
6.38	71.29	0.6545	0.1086	71.39	0.6554	0.1036	79	25.25
9.37	72.86	0.6794	0.1232	72.93	0.6809	0.1211	149	53
12.12	74.95	0.7116	0.1359	75.12	0.7154	0.1330	191	103.5
				5000				
10.25	71.45	0.6559	0.1090	71.51	0.6562	0.1047	103	36.25
12.66	73.86	0.6955	0.1247	73.87	0.6934	0.1212	135	59
15.16	74.68	0.7107	0.1349	74.95	0.7094	0.1339	175	93

Table A.2 DSS-DSS Block Sizes

Hits Metric

Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size After
				250				
5.57	79.32	0.9027	0.4329	79.57	0.9026	0.4371	111	36.75
7.45	80.53	0.9252	0.4785	80.83	0.9244	0.4828	155	80
8.99	82.18	0.9342	0.6246	82.20	0.9354	0.6228	193	107
				500				
5.50	78.95	0.9189	0.4536	79.06	0.9202	0.4528	113	40.75
7.98	80.53	0.9344	0.5583	80.60	0.9325	0.5604	183	73
9.52	81.95	0.9736	0.7927	81.76	0.9736	0.7959	219	113.5
				750				
5.08	78.93	0.8967	0.4442	79.13	0.8947	0.4480	103	42.5
7.44	80.71	0.9320	0.5551	80.81	0.9327	0.5554	151	70
9.03	81.38	0.9638	0.7699	81.56	0.9637	0.7673	193	91.25
				1000				
5.93	78.85	0.9051	0.4501	79.04	0.9016	0.4532	109	48
7.88	79.52	0.9309	0.6047	79.64	0.9293	0.6024	151	70.5
9.41	81.07	0.9808	0.7978	81.06	0.9812	0.8013	207	111.75
				2500				
6.65	78.64	0.8972	0.4302	78.74	0.8955	0.4357	101	38.25
9.64	79.57	0.9412	0.6180	79.79	0.9404	0.6143	159	65.5
11.76	81.24	0.9744	0.7900	81.08	0.9752	0.7922	191	107.75
				5000				
8.76	78.54	0.9143	0.4637	78.81	0.9158	0.4643	93	31.75
11.20	79.30	0.9418	0.6086	79.58	0.9420	0.6035	135	48.5
14.15	81.68	0.9783	0.8001	81.62	0.9798	0.8035	183	86

TNR+TPR Metric

Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size After
				250				
5.70	72.84	0.6798	0.1226	72.80	0.6785	0.1174	113	53
7.63	75.25	0.7145	0.1311	75.64	0.7188	0.1300	163	75.5
9.00	77.47	0.7553	0.1590	77.42	0.7538	0.1560	197	104.75
				500				
5.50	73.65	0.6933	0.1244	73.42	0.6915	0.1209	113	51.75
7.98	74.72	0.7111	0.1376	75.08	0.7136	0.1342	183	87
9.52	77.50	0.7499	0.1623	77.51	0.7507	0.1598	217	105.5
				750				
5.33	71.52	0.6579	0.1100	71.60	0.6590	0.1065	103	30
7.83	74.31	0.7045	0.1275	74.46	0.7082	0.1268	159	72
9.07	77.43	0.7536	0.1632	77.40	0.7533	0.1638	199	105
				1000				
5.97	71.60	0.6570	0.1071	71.72	0.6581	0.1026	111	50
8.03	74.70	0.7094	0.1323	74.53	0.7066	0.1305	159	72
9.48	76.16	0.7368	0.1573	76.15	0.7395	0.1561	203	113
				2500				
6.68	72.32	0.6726	0.1144	72.42	0.6728	0.1121	97	36
9.64	74.49	0.7024	0.1273	74.77	0.7069	0.1232	159	66.5
11.85	75.85	0.7286	0.1475	76.11	0.7325	0.1441	191	103
				5000				
8.80	71.60	0.6581	0.1088	71.71	0.6593	0.1057	87	36
11.37	74.23	0.6979	0.1289	74.56	0.7004	0.1276	135	61
14.07	75.61	0.7344	0.1569	75.84	0.7314	0.1546	183	83

Table A.3 RSS-DSS Exemplar Age/Difficulty Ratios

Hits Metric

Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size After
				30/70				
7.10	79.36	0.8821	0.4089	79.41	0.8814	0.4124	135	66.25
8.58	80.86	0.9168	0.4985	80.77	0.9148	0.4953	167	91
10.71	81.58	0.9593	0.6869	81.65	0.9590	0.6857	223	126
				10/90				
6.28	78.23	0.9149	0.4678	78.50	0.9137	0.4685	118.5	28.5
8.25	79.27	0.9509	0.7062	79.40	0.9518	0.7061	167	76
10.68	81.49	0.9781	0.8033	81.40	0.9779	0.8114	231	100.25
				0/100				
6.33	78.60	0.9152	0.4681	78.74	0.9143	0.4636	125	42.5
8.21	79.51	0.9476	0.6436	79.64	0.9491	0.6408	159	77
10.25	80.67	0.9728	0.7879	80.92	0.9736	0.7897	207.5	104.5

TNR+TPR Metric

Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size After
				30/70				
7.10	74.19	0.6972	0.1233	74.19	0.7004	0.1199	135	53
8.58	75.85	0.7242	0.1416	76.16	0.7276	0.1377	167	76.5
10.71	77.68	0.7556	0.1627	77.45	0.7534	0.1631	223	113.75
				10/90				
6.28	71.62	0.6582	0.1097	71.67	0.6591	0.1057	118.5	46.5
8.25	74.46	0.7054	0.1317	74.36	0.7030	0.1265	163	79.5
10.68	76.25	0.7329	0.1518	76.28	0.7324	0.1498	219	110.75
				0/100				
6.33	71.83	0.6612	0.1099	71.98	0.6629	0.1069	125	55
8.21	74.20	0.6966	0.1270	74.32	0.6988	0.1242	159	77.5
10.25	75.21	0.7202	0.1539	75.37	0.7162	0.1492	207.5	120.25

Table A.4 DSS-DSS Block Age/Difficulty Ratios and Exemplar Age/Difficulty Ratios

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				30/70 – 30/70				
5.28	78.85	0.9051	0.4501	79.04	0.9016	0.4532	109	48
7.04	79.52	0.9309	0.6047	79.64	0.9293	0.6024	151	70.5
8.52	81.07	0.9808	0.7978	81.06	0.9812	0.8013	207	111.75
				30/70 – 10/90				
5.92	78.50	0.9261	0.5553	78.63	0.9246	0.5605	119	47.5
7.45	79.12	0.9593	0.7281	79.29	0.9586	0.7328	163	69
9.83	80.26	0.9770	0.7939	80.39	0.9787	0.7964	225	102.5
				30/70 - 0/100				
6.06	78.90	0.8870	0.4057	79.16	0.8822	0.4087	127	64.75
7.71	80.46	0.9298	0.5294	80.27	0.9294	0.5282	167	85
9.69	81.55	0.9642	0.7876	81.38	0.9642	0.7892	231	122.75
				10/90 – 30/70				
5.34	78.92	0.9035	0.4140	79.06	0.9034	0.4168	111	53.5
7.13	80.80	0.9336	0.5581	80.53	0.9355	0.5569	163	83.5
8.33	81.85	0.9685	0.7886	81.84	0.9708	0.7902	201	112.25
				10/90 – 10/90				
5.41	78.54	0.9110	0.4528	78.70	0.9082	0.4577	119	40
7.22	79.86	0.9282	0.5348	80.13	0.9300	0.5384	159	72
9.38	82.07	0.9645	0.7881	81.84	0.9655	0.7892	215	98
				10/90 – 0/100				
5.70	78.84	0.9030	0.4238	79.04	0.9010	0.4287	125	45
7.06	79.86	0.9418	0.6127	80.02	0.9420	0.6101	167	80
9.38	81.45	0.9756	0.7989	81.47	0.9777	0.8039	209	104.25
				0/100 – 30/70				
5.01	78.49	0.9129	0.4609	78.55	0.9113	0.4618	109	42.5
6.98	79.44	0.9585	0.7397	79.63	0.9585	0.7396	151	64.5
8.84	81.37	0.9811	0.8109	81.26	0.9816	0.8142	207	104.25
				0/100 – 10/90				
5.68	78.81	0.9213	0.5221	79.09	0.9219	0.5182	119	38
7.33	79.42	0.9584	0.7305	79.49	0.9585	0.7380	159	75.5
9.01	81.15	0.9810	0.7975	81.09	0.9817	0.8018	195	114.25
				0/100 – 0/100				
5.40	78.48	0.9002	0.4312	78.44	0.8985	0.4375	119	53.5
7.06	79.76	0.9393	0.5773	79.89	0.9413	0.5776	167	72.5
9.28	81.11	0.9706	0.7919	81.25	0.9715	0.7928	223	99.5

TNR+TPR Metric

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				30/70 – 30/70				
5.28	71.49	0.6560	0.1064	71.58	0.6568	0.1016	109	46.75
7.04	74.56	0.7073	0.1315	74.44	0.7057	0.1288	151	70.5
8.52	76.14	0.7351	0.1561	76.02	0.7372	0.1555	201	111.5
				30/70 – 10/90				
5.92	71.64	0.6593	0.1107	71.72	0.6604	0.1080	119	48.25
7.45	73.32	0.6892	0.1288	73.49	0.6897	0.1259	159	74
9.83	75.04	0.7211	0.1493	75.34	0.7172	0.1436	225	108.5
				30/70 – 0/100				
6.06	72.63	0.6777	0.1239	72.77	0.6786	0.1199	127	52.75
7.71	75.26	0.7175	0.1378	75.40	0.7197	0.1366	167	72
9.69	76.96	0.7444	0.1681	77.20	0.7454	0.1666	231	112.25
				10/90 – 30/70				
5.34	72.83	0.6774	0.1184	73.03	0.6797	0.1151	117	47.75
7.13	75.05	0.7155	0.1390	75.27	0.7143	0.1376	163	72.5
8.33	77.11	0.7471	0.1580	77.13	0.7482	0.1575	187	107
				10/90 – 10/90				
5.35	71.77	0.6607	0.1111	71.93	0.6623	0.1074	115	36.75
7.11	74.89	0.7124	0.1343	75.14	0.7140	0.1311	159	69.5
9.34	76.53	0.7374	0.1564	76.57	0.7384	0.1572	215	98.75
				10/90 – 0/100				
5.70	71.43	0.6558	0.1096	71.59	0.6574	0.1070	117	52.25
7.06	74.38	0.7028	0.1301	74.47	0.7030	0.1276	163	67.5
9.38	76.00	0.7298	0.1578	76.35	0.7341	0.1505	207	96
				0/100 – 30/70				
5.01	71.50	0.6573	0.1087	71.67	0.6583	0.1049	109	38.25
6.98	74.32	0.7016	0.1283	74.35	0.7032	0.1259	151	65.5
8.84	76.94	0.7432	0.1573	76.94	0.7434	0.1561	189	110
				0/100 – 10/90				
5.68	71.62	0.6584	0.1081	71.75	0.6592	0.1036	119	51.75
7.33	73.01	0.6803	0.1262	73.05	0.6789	0.1199	159	75.5
9.01	75.12	0.7164	0.1464	75.38	0.7195	0.1482	195	92.5
				0/100 – 0/100				
5.40	71.37	0.6546	0.1087	71.47	0.6559	0.1037	117	47.5
7.06	74.53	0.7043	0.1275	74.32	0.7043	0.1250	163	75
9.28	76.13	0.7351	0.1553	75.94	0.7334	0.1503	211	94

Table A.5 DSS-DSS Sorted Adult Dataset
Hits Metric

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric								
1000								
1.55	76.70	0.9584	0.7708	77.24	0.9585	0.7891	63	11.25
2.03	76.81	0.9588	0.7921	77.48	0.9587	0.7892	99	34
2.86	77.42	0.9700	0.8051	77.62	0.9695	0.8066	167	65.25
5000								
5.37	76.93	0.9584	0.7708	77.49	0.9585	0.7892	87	21.25
6.78	77.41	0.9620	0.7921	77.92	0.9618	0.7892	131	51
9.53	78.09	0.9810	0.8237	78.58	0.9817	0.8224	205	93.75
TNR+TPR Metric								
1000								
1.45	70.29	0.6395	0.1052	70.30	0.6391	0.1005	61.25	9.25
2.34	70.91	0.6597	0.1424	71.05	0.6618	0.1399	111	24.5
3.15	70.91	0.6616	0.1477	71.05	0.6633	0.1446	167	53.25
5000								
5.38	70.28	0.6391	0.1052	70.29	0.6389	0.1005	85	18
7.38	70.42	0.6410	0.1052	70.48	0.6415	0.1005	143	41.5
9.65	70.98	0.6538	0.1085	71.10	0.6542	0.1039	201	76.5

Table A.6 Manually Mixed Adult Datasets

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric								
Mixed 1								
6.69	77.29	0.9513	0.7324	77.49	0.9513	0.7326	95	27
8.68	77.44	0.9586	0.7876	77.77	0.9586	0.7892	143	46
10.38	78.54	0.9737	0.7969	78.73	0.9723	0.8000	183	80.5
Mixed 2								
9.36	78.50	0.8873	0.4393	78.66	0.8865	0.4347	109	50.25
11.81	79.69	0.9424	0.6120	79.70	0.9413	0.6073	151	71
13.83	80.67	0.9700	0.7892	80.85	0.9724	0.7909	191	86
Mixed 3								
3.88	77.29	0.9584	0.7876	77.49	0.9585	0.7892	103	28
4.89	77.55	0.9587	0.7876	77.71	0.9586	0.7892	135	63.5
7.21	78.03	0.9720	0.8006	78.30	0.9723	0.8028	215	95.5
Mixed 4								
5.75	78.03	0.9293	0.5630	78.25	0.9297	0.5652	101	42.25
7.73	79.03	0.9578	0.7159	79.15	0.9569	0.7215	151	62.5
9.91	80.95	0.9719	0.7965	81.03	0.9732	0.7999	195	102.5
TNR+TPR Metric								
Mixed 1								
6.43	70.29	0.6394	0.1052	70.32	0.6393	0.1005	83	26
8.58	70.55	0.6431	0.1059	70.64	0.6439	0.1011	135	33
10.19	71.32	0.6536	0.1074	71.41	0.6541	0.1030	163	69.5
Mixed 2								
9.57	72.03	0.6656	0.1139	72.22	0.6674	0.1112	111	48.5
12.02	74.49	0.7048	0.1341	74.73	0.7056	0.1315	151	73.5
13.98	76.36	0.7337	0.1535	76.49	0.7363	0.1506	191	101
Mixed 3								
3.93	70.38	0.6396	0.1052	70.41	0.6391	0.1005	103	27
5.02	70.98	0.6493	0.1064	71.11	0.6506	0.1014	151	59
7.25	71.61	0.6587	0.1115	71.69	0.6599	0.1076	223	78
Mixed 4								
5.95	71.21	0.6525	0.1079	71.28	0.6527	0.1036	103	26
7.88	72.61	0.6750	0.1233	72.62	0.6746	0.1208	151	56
9.96	74.69	0.7080	0.1349	74.59	0.7077	0.1335	207	98.5

Table A.7 Balanced Block - Block Partition Ratios

Hits Metric

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				10/90				
5.99	77.88	0.9275	0.5593	78.23	0.9269	0.5552	87	15.5
8.33	78.79	0.9587	0.7638	78.94	0.9586	0.7600	127	49.5
10.80	80.62	0.9716	0.7959	80.56	0.9710	0.7974	185	89
				15/85				
4.45	78.06	0.9340	0.5537	78.37	0.9338	0.5557	85	33
6.47	78.95	0.9617	0.7724	79.18	0.9615	0.7735	151	57.5
9.07	80.65	0.9808	0.8079	80.68	0.9829	0.8122	207	103.25
				25/75				
5.48	77.81	0.9441	0.6346	78.24	0.9429	0.6338	135	43.75
7.14	78.83	0.9641	0.7876	79.03	0.9642	0.7892	167	69.5
8.43	80.53	0.9824	0.8145	80.43	0.9832	0.8188	207	95
				30/70				
4.00	78.20	0.9346	0.6894	78.57	0.9341	0.6916	88	27.5
6.20	78.99	0.9772	0.8015	79.12	0.9787	0.8035	151	58
8.14	79.33	0.9943	0.8163	79.57	0.9945	0.8205	200	97
				37.5/62.5				
4.48	78.41	0.9244	0.5757	78.65	0.9243	0.5780	101.25	32.25
7.03	79.20	0.9458	0.6682	79.45	0.9473	0.6703	171	61
8.88	80.52	0.9667	0.7886	80.90	0.9669	0.7907	215	104.25
				50/50				
3.80	77.81	0.9258	0.5891	78.24	0.9250	0.5868	79	32.5
5.67	78.87	0.9601	0.7757	79.21	0.9607	0.7746	131	49
7.27	80.50	0.9808	0.8099	80.32	0.9829	0.8130	171	77
				62.5/37.5				
5.87	78.24	0.9246	0.5648	78.52	0.9251	0.5631	100.75	31
7.58	79.21	0.9587	0.7328	79.37	0.9586	0.7381	159	73
8.87	79.72	0.9840	0.8056	79.89	0.9836	0.8094	191	94

TNR+TPR Metric

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				10/90				
5.99	70.32	0.6393	0.1052	70.29	0.6388	0.1005	87	21.75
8.33	71.28	0.6532	0.1057	71.39	0.6536	0.1008	127	42
10.80	74.25	0.7048	0.1297	74.51	0.7059	0.1277	185	86
				15/85				
4.45	70.58	0.6429	0.1052	70.58	0.6427	0.1005	93	27.75
6.47	71.66	0.6586	0.1093	71.74	0.6598	0.1049	151	56.5
9.07	74.06	0.7021	0.1317	74.34	0.7069	0.1286	207	102.5
				25/75				
5.48	70.78	0.6477	0.1064	70.83	0.6482	0.1019	135	46.5
7.14	71.62	0.6587	0.1095	71.70	0.6597	0.1053	167	76.5
8.43	74.01	0.6957	0.1250	74.06	0.6966	0.1224	207	112
				30/70				
4.14	70.34	0.6395	0.1052	70.30	0.6390	0.1005	91	17
6.21	71.68	0.6582	0.1078	71.71	0.6591	0.1036	147	53.5
8.01	71.99	0.6641	0.1154	72.06	0.6645	0.1102	196.5	88.75
				37.5/62.5				
4.48	70.88	0.6483	0.1069	70.84	0.6483	0.1021	101.25	28.75
7.03	71.65	0.6588	0.1099	71.73	0.6598	0.1059	171	72
8.88	74.39	0.7019	0.1315	74.27	0.7037	0.1300	215	105.75
				50/50				
3.78	70.48	0.6417	0.1052	70.43	0.6409	0.1005	79	20.5
5.82	71.45	0.6558	0.1076	71.53	0.6563	0.1027	135	39
7.36	72.43	0.6701	0.1165	72.37	0.6685	0.1123	175	81.5
				62.5/37.5				
5.93	70.44	0.6403	0.1052	70.42	0.6400	0.1005	107	31.75
7.99	71.61	0.6578	0.1092	71.64	0.6585	0.1051	163	64
9.15	71.86	0.6614	0.1110	71.96	0.6629	0.1085	191	93.5

Table A.8 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric								
RSS-DSS								
5.58	78.55	0.9136	0.4634	78.76	0.9128	0.4721	109	36.75
7.46	79.30	0.9502	0.7041	79.43	0.9505	0.7091	139	56.5
9.34	80.74	0.9749	0.7910	80.62	0.9758	0.7957	199	98.75
DSS-DSS								
8.25	77.96	0.9023	0.4934	77.98	0.9006	0.4916	93	28.25
11.13	78.98	0.9519	0.7593	79.08	0.9515	0.7645	135	55.5
13.77	80.07	0.9773	0.8028	80.08	0.9781	0.8061	169	83.75
Balanced (30/70)								
4.00	78.20	0.9346	0.6894	78.57	0.9341	0.6916	88	27.5
6.20	78.99	0.9772	0.8015	79.12	0.9787	0.8035	151	58
8.14	79.33	0.9943	0.8163	79.57	0.9945	0.8205	200	97
TNR+TPR								
RSS-DSS								
5.58	71.34	0.6542	0.1076	71.37	0.6543	0.1029	109	36.75
7.68	71.95	0.6635	0.1133	72.05	0.6648	0.1105	147	64.5
9.88	76.55	0.7394	0.1511	76.42	0.7356	0.1505	201	102.5
DSS-DSS								
8.20	71.36	0.6546	0.1080	71.43	0.6553	0.1038	89	26
11.13	73.76	0.6906	0.1207	73.82	0.6915	0.1141	135	46
13.80	75.98	0.7323	0.1573	75.90	0.7297	0.1509	173	74.75
Balanced (30/70)								
4.14	70.34	0.6395	0.1052	70.30	0.6390	0.1005	91	17
6.21	71.68	0.6582	0.1078	71.71	0.6591	0.1036	147	53.5
8.01	71.99	0.6641	0.1154	72.06	0.6645	0.1102	196.5	88.75

Appendix B 10% KDD'99 Dataset Results

Table B.1 Balanced Block – Block Partition Ratios

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				Hits Metric				
				50/50				
16.00	98.61	0.9882	0.0116	90.64	0.8881	0.0171	141	20.5
18.09	98.76	0.9888	0.0154	90.84	0.8914	0.0202	179	52
20.87	98.92	0.9895	0.0226	91.01	0.8953	0.0332	215	85.25
				(25/75)				
13.90	98.34	0.9880	0.0089	90.65	0.8879	0.0157	109	24.5
16.71	98.78	0.9886	0.0144	90.76	0.8900	0.0194	159	44
19.97	98.91	0.9894	0.0318	91.02	0.8942	0.0296	209	69
				(20/80)				
15.98	98.04	0.9886	0.0124	90.67	0.8893	0.0170	133	30.25
18.71	98.33	0.9892	0.0406	90.93	0.8928	0.0208	167	66.5
21.75	98.89	0.9902	0.0569	91.18	0.8985	0.0327	225	103
				(10/90)				
14.42	97.97	0.9885	0.0147	90.58	0.8897	0.0172	101	18.25
21.83	98.53	0.9893	0.0342	90.80	0.8939	0.0238	195	44.5
24.64	98.85	0.9901	0.0626	91.06	0.9003	0.0369	215	74
				TNR+TPR Metric				
				50/50				
16.00	98.61	0.9879	0.0092	90.60	0.8878	0.0154	141	27
18.09	98.75	0.9886	0.0138	90.77	0.8907	0.0192	179	48
20.87	98.90	0.9892	0.0226	90.94	0.8936	0.0303	215	85.25
				25/75				
13.90	98.22	0.9877	0.0075	90.55	0.8875	0.0147	109	20
16.71	98.78	0.9884	0.0139	90.74	0.8893	0.0169	159	42
19.97	98.90	0.9890	0.0318	91.00	0.8933	0.0251	209	58.25
				20/80				
15.98	97.98	0.9878	0.0101	90.64	0.8884	0.0163	133	30.25
18.71	98.33	0.9889	0.0324	90.84	0.8916	0.0208	167	65.5
21.75	98.83	0.9901	0.0569	91.09	0.8978	0.0304	225	103
				10/90				
14.42	97.84	0.9873	0.0109	90.47	0.8878	0.0151	101	18.75
21.83	98.52	0.9889	0.0231	90.73	0.8921	0.0228	195	49
24.64	98.81	0.9896	0.0419	91.01	0.8973	0.0362	215	79

Table B.2 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				Hits Metric				
				RSS-DSS				
8.53	98.44	0.9873	0.0117	90.54	0.8899	0.0165	93	30
11.95	98.68	0.9882	0.0174	90.89	0.8945	0.0223	179	62
13.54	98.80	0.9891	0.0341	91.11	0.8985	0.0398	215	105.5
				DSS-DSS				
10.19	98.63	0.9880	0.0077	90.72	0.8887	0.0160	103	30.5
13.56	98.85	0.9886	0.0127	90.89	0.8922	0.0186	171	64.5
18.51	98.95	0.9898	0.0204	91.21	0.8986	0.0296	215	94
				Balanced Block (25/75)				
13.82	98.34	0.9880	0.0089	90.65	0.8879	0.0157	109	24.5
16.65	98.78	0.9886	0.0144	90.76	0.8900	0.0194	159	44
19.91	98.91	0.9894	0.0318	91.02	0.8942	0.0296	209	69
				TNR+TPR Metric				
				RSS-DSS				
8.53	98.41	0.9863	0.0088	90.53	0.8899	0.0136	93	34.25
11.95	98.66	0.9881	0.0157	90.86	0.8917	0.0208	179	64
13.54	98.79	0.9888	0.0341	91.14	0.8977	0.0366	215	100.25
				DSS-DSS				
10.19	98.60	0.9870	0.0053	90.54	0.8878	0.0134	103	27.5
13.56	98.79	0.9878	0.0090	90.77	0.8905	0.0166	171	60.5
18.51	98.91	0.9882	0.0155	91.08	0.8937	0.0289	215	90.25
				Balanced Block (25/75)				
13.82	98.22	0.9877	0.0075	90.55	0.8875	0.0147	109	20
16.65	98.78	0.9884	0.0139	90.74	0.8893	0.0169	159	42
19.91	98.90	0.9890	0.0318	91.00	0.8933	0.0251	209	58.25

Table B.3 Algorithm Comparison on KDD Attack Categories

Training Probe	Training DoS	Training U2R	Training R2L	Test Probe	Test DoS	Test U2R	Test R2L
Hits Metric							
RSS-DSS							
76.96	99.23	0.00	6.94	51.36	95.21	9.21	1.37
79.17	99.26	4.81	24.64	56.58	95.49	12.72	2.23
83.53	99.29	31.25	38.95	62.57	95.78	17.76	4.42
DSS-DSS							
77.46	99.26	0.00	7.17	50.12	95.19	9.10	1.25
80.17	99.28	2.88	20.80	55.92	95.39	11.18	1.67
84.25	99.32	21.15	37.03	61.53	95.76	16.67	2.66
Balanced Block (25/75)							
77.39	99.24	0.00	24.16	48.42	95.13	7.46	1.23
79.57	99.26	1.92	31.28	53.83	95.27	9.65	1.75
83.92	99.30	4.81	48.46	58.44	95.62	13.82	2.99
TNR+TPR Metric							
RSS-DSS							
76.56	99.13	0.00	2.67	50.70	95.19	9.21	0.78
78.67	99.23	3.85	23.10	53.82	95.39	12.06	2.17
82.45	99.28	27.88	34.15	59.19	95.67	16.67	4.08
DSS-DSS							
76.66	99.18	0.00	3.89	48.33	95.08	8.22	0.68
78.63	99.24	1.92	13.16	54.18	95.22	9.87	1.37
81.20	99.28	18.27	27.19	60.83	95.58	15.57	1.98
Balanced Block (25/75)							
77.03	99.22	0.00	23.30	47.77	95.06	7.46	1.14
78.71	99.24	1.92	30.92	52.20	95.21	9.43	1.62
82.71	99.28	3.85	43.13	56.80	95.43	11.73	2.75

Appendix C Census Income Dataset Results

Table C.1 Balanced Block – Block Partition Ratios

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
TNR+TPR Metric								
5/95								
4.84	69.14	0.6809	0.1278	69.35	0.6823	0.1230	85	19.75
6.97	72.46	0.7190	0.1661	72.51	0.7203	0.1746	151	50.5
8.49	78.54	0.7830	0.2066	78.29	0.7807	0.2064	199	94
10/90								
4.42	70.00	0.6902	0.1423	70.21	0.6929	0.1458	69	20
6.02	75.38	0.7503	0.1664	75.27	0.7493	0.1718	119	53
9.07	79.30	0.7920	0.2153	79.13	0.7907	0.2160	201	76.75
20/80								
4.56	69.09	0.6818	0.1303	69.36	0.6843	0.1241	71	22
6.52	72.79	0.7213	0.1585	73.03	0.7231	0.1569	119	42.5
7.95	79.26	0.7925	0.2226	79.27	0.7913	0.2193	161	80
25/75								
4.54	70.55	0.6957	0.1399	70.77	0.6987	0.1431	63	18.75
5.95	74.32	0.7375	0.1612	74.38	0.7384	0.1703	103	35
9.91	79.08	0.7917	0.2177	78.95	0.7893	0.2180	199	69.25
50/50								
5.42	71.90	0.7112	0.1309	72.00	0.7125	0.1294	77	24
8.83	75.73	0.7528	0.1585	75.64	0.7518	0.1629	171	66.5
11.17	79.29	0.7920	0.2030	79.24	0.7910	0.2050	231	117.25

Table C.2 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
TNR+TPR Metric								
RSS-DSS								
6.26	75.80	0.7546	0.1743	75.59	0.7523	0.1737	103	34.5
8.57	78.72	0.7858	0.1924	78.60	0.7850	0.1970	167	65
10.15	80.97	0.8102	0.2328	80.80	0.8088	0.2381	201	88.5
DSS-DSS								
10.18	77.81	0.7791	0.1788	77.89	0.7804	0.1834	109	45.5
13.70	80.51	0.8053	0.2168	80.35	0.8042	0.2156	175	66
15.96	82.44	0.8298	0.2516	82.32	0.8287	0.2509	215	89
Balanced (10/90)								
4.42	70.00	0.6902	0.1423	70.21	0.6929	0.1458	69	20
6.02	75.38	0.7503	0.1664	75.27	0.7493	0.1718	119	53
9.07	79.30	0.7920	0.2153	79.13	0.7907	0.2160	201	76.75

Appendix D Shuttle Dataset Results

Table D.1 Shuttle 1 Balanced Block – Block Partition Ratios

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric								
50/50								
2.66	96.84	0.9778	0.0076	96.72	0.9761	0.0062	71	19
4.19	98.75	0.9909	0.0132	98.73	0.9906	0.0124	127	38.5
5.35	99.80	0.9996	0.0183	99.82	0.9997	0.0175	169	90
37.5/62.5								
2.45	96.49	0.9605	0.0074	96.44	0.9597	0.0065	53.25	20
3.95	99.29	0.9949	0.0129	99.30	0.9945	0.0116	111	31.5
5.43	99.80	0.9996	0.0183	99.84	0.9997	0.0175	193	64.25
25/75								
2.76	96.54	0.9713	0.0070	96.50	0.9699	0.0060	79	21.75
3.84	99.70	0.9992	0.0094	99.72	0.9993	0.0084	103	42
4.74	99.83	0.9998	0.0156	99.85	0.9998	0.0149	157.5	65
TNR+TPR Metric								
50/50								
2.66	96.61	0.9607	0.0073	96.53	0.9599	0.0060	71	25.5
4.19	98.75	0.9881	0.0101	98.73	0.9881	0.0094	127	38.5
5.35	99.80	0.9996	0.0150	99.82	0.9997	0.0149	169	87.75
37.5/62.5								
2.45	96.38	0.9600	0.0073	96.31	0.9590	0.0065	53.25	19.75
3.95	99.29	0.9949	0.0105	99.30	0.9945	0.0094	111	34
5.43	99.80	0.9996	0.0183	99.84	0.9997	0.0175	193	64.25
25/75								
2.76	96.50	0.9604	0.0070	96.46	0.9596	0.0060	79	22
3.84	99.70	0.9986	0.0089	99.70	0.9980	0.0083	103	37.5
4.74	99.82	0.9998	0.0146	99.83	0.9998	0.0142	157.5	66

Table D.2 Shuttle 3 Balanced Block – Block Partition Ratios

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric (TNR+TPR Metric the same)								
70/30								
4.55	86.40	0.4470	0.0272	86.45	0.4381	0.0280	79	22
6.13	87.80	0.5553	0.0662	87.87	0.5592	0.0674	111	39
9.15	91.51	0.9932	0.0735	91.46	0.9926	0.0740	191	64
50/50								
4.18	87.49	0.5514	0.0513	87.44	0.5531	0.0529	63	20.5
5.53	88.72	0.5864	0.0668	88.60	0.5916	0.0680	87	33
6.85	90.26	0.7940	0.0756	90.25	0.8019	0.0772	123	56.5
62.5/37.5								
4.06	85.89	0.3296	0.0353	85.94	0.3096	0.0366	85	34.75
6.31	87.49	0.4529	0.0632	87.69	0.4548	0.0648	111	46
8.18	91.29	0.8774	0.0727	91.19	0.8784	0.0751	145	60.25

Table D.3 Shuttle 4 Balanced Block – Block Partition Ratios

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric (TNR+TPR Metric the same)								
62.5/37.5								
2.90	96.45	0.5315	0.0006	96.34	0.5439	0.0003	63	20
4.62	97.52	0.9714	0.0020	97.55	0.9691	0.0019	119	41
6.13	99.81	0.9983	0.0082	99.77	0.9988	0.0077	207	68
50/50								
4.14	95.46	0.2745	0.0008	95.37	0.2565	0.0006	83	23
4.80	96.73	0.7251	0.0030	96.77	0.7367	0.0026	135	54
7.28	99.54	0.9932	0.0116	99.51	0.9913	0.0119	175	100.5

Table D.4 Shuttle 1 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
				Hits Metric				
				RSS-DSS				
3.34	96.63	0.9607	0.0111	96.56	0.9599	0.0095	79	23.25
4.53	97.61	0.9800	0.0152	97.65	0.9794	0.0149	107	49.5
6.55	99.67	0.9990	0.0282	99.70	0.9990	0.0281	185	84.5
				DSS-DSS				
3.34	96.63	0.9607	0.0111	96.56	0.9599	0.0095	79	23.25
4.53	97.61	0.9800	0.0152	97.65	0.9794	0.0149	107	49.5
6.55	99.67	0.9990	0.0282	99.70	0.9990	0.0281	185	84.5
				Balanced Block (25/75)				
2.76	96.54	0.9713	0.0070	96.50	0.9699	0.0060	79	21.75
3.84	99.70	0.9992	0.0094	99.72	0.9993	0.0084	103	42
4.74	99.83	0.9998	0.0156	99.85	0.9998	0.0149	157.5	65
				TNR+TPR Metric				
				RSS-DSS				
3.34	96.59	0.9607	0.0081	96.51	0.9599	0.0071	79	23.25
4.53	97.29	0.9704	0.0130	97.13	0.9697	0.0124	107	43
6.55	99.67	0.9986	0.0172	99.70	0.9986	0.0169	185	84.5
				DSS-DSS				
2.94	97.10	0.9735	0.0073	97.01	0.9719	0.0066	61	26
3.78	98.90	0.9901	0.0125	98.89	0.9898	0.0116	95	41
6.08	99.74	0.9996	0.0179	99.79	0.9997	0.0173	175	73.75
				Balanced Block (25/75)				
2.76	96.50	0.9604	0.0070	96.46	0.9596	0.0060	79	22
3.84	99.70	0.9986	0.0089	99.70	0.9980	0.0083	103	37.5
4.74	99.82	0.9998	0.0146	99.83	0.9998	0.0142	157.5	66

Table D.5 Shuttle 2 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric (TNR+TPR Metric the same)								
RSS-DSS								
4.17	99.73	0.7339	0.0013	99.70	0.6724	0.0014	74	12
6.83	99.77	0.7688	0.0016	99.74	0.7069	0.0018	135	37
10.44	99.77	0.9839	0.0020	99.76	1.0000	0.0023	199	70
DSS-DSS								
3.28	99.72	0.7392	0.0013	99.68	0.6810	0.0013	42.5	14
6.80	99.77	0.7688	0.0016	99.76	0.7069	0.0017	119	37
9.13	99.79	1.0000	0.0020	99.77	1.0000	0.0021	191	61
Balanced Block (95/5)								
7.28	99.59	0.2120	0.0005	99.61	0.2414	0.0006	39	9
8.97	99.60	0.2337	0.0007	99.63	0.2931	0.0008	71	37
16.80	99.72	0.5163	0.0009	99.67	0.4310	0.0012	119	49

Table D.6 Shuttle 3 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric (TNR+TPR Metric the same)								
RSS-DSS								
4.48	94.93	0.9781	0.0037	94.98	0.9759	0.0039	63	34.25
5.56	98.08	0.9967	0.0197	98.02	0.9974	0.0203	119	58
7.61	99.55	0.9996	0.0407	99.55	1.0000	0.0421	171	93
DSS-DSS								
4.22	95.62	0.9810	0.0035	95.48	0.9813	0.0039	63	35.5
5.65	97.93	0.9981	0.0211	97.79	0.9981	0.0234	127	61
8.00	99.69	0.9997	0.0390	99.67	1.0000	0.0389	171	89.75
Balanced Block (50/50)								
4.18	87.49	0.5514	0.0513	87.44	0.5531	0.0529	63	20.5
5.53	88.72	0.5864	0.0668	88.60	0.5916	0.0680	87	33
6.85	90.26	0.7940	0.0756	90.25	0.8019	0.0772	123	56.5

Table D.7 Shuttle 4 Algorithm Comparison

Run Time	Train Acc	Train DR	Train FPR	Test Acc	Test DR	Test FPR	Size	Size Final
Hits Metric (TNR+TPR Metric the same)								
RSS-DSS								
2.94	99.99	1.0000	0.0000	99.99	1.0000	0.0001	76.75	17.5
4.89	100.00	1.0000	0.0000	99.99	1.0000	0.0001	139	29.5
6.24	100.00	1.0000	0.0001	99.99	1.0000	0.0001	181	74.5
DSS-DSS								
3.10	99.99	0.9996	0.0000	99.99	1.0000	0.0001	77	17.25
5.20	100.00	1.0000	0.0000	99.99	1.0000	0.0001	147	41
6.22	100.00	1.0000	0.0001	99.99	1.0000	0.0001	183	68
Balanced Block (62.5/37.5)								
2.90	96.45	0.5315	0.0006	96.34	0.5439	0.0003	63	20
4.62	97.52	0.9714	0.0020	97.55	0.9691	0.0019	119	41
6.13	99.81	0.9983	0.0082	99.77	0.9988	0.0077	207	68