LOCAL VERSUS GLOBAL WRAPPER FUNCTIONS IN GENETIC PROGRAMMING

by

Ashley L. George

Submitted in partial fulfillment of the requirements

for the degree of Master of Computer Science

at

Dalhousie University

Halifax, Nova Scotia

March 2006

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**LOCAL VERSUS GLOBAL WRAPPER FUNCTIONS IN GENETIC PROGRAMMING**" by **Ashley L. George** in partial fulfillment of the requirements for the degree of **Master of Computer Science**.

Dated: March 28th, 2006

Supervisor: _____
Dr. Malcolm Heywood

Readers: _____
Dr. Syed Sibte Raza Abidi

_____
Dr. Qigang Gao

DALHOUSIE UNIVERSITY

DATE: **March 28th, 2006**

AUTHOR: **Ashley L. George**

TITLE: **LOCAL VERSUS GLOBAL WRAPPER FUNCTIONS IN GENETIC PROGRAMMING**

DEPARTMENT OR SCHOOL: **Computer Science**

DEGREE: **M.C.Sc.** CONVOCATION: **May** YEAR: **2006**

Table of Contents

List of Tables . . . . . **vi**

List of Figures . . . . . **vii**

Abstract . . . . . **xi**

Chapter 1 Introduction **1**

Chapter 2 Previous Work **4**

2.1 Evolutionary Computation 4

2.1.1 Techniques in Evolutionary Computation 5

2.1.1.1 Genetic Algorithms 6

2.1.1.2 Genetic Programming 7

2.2 Page-based Linear Genetic Programming 9

2.2.1 Representation 9

2.2.2 Initialisation 10

2.2.3 Evaluation 11

2.2.4 Selection 11

2.2.5 Search (Crossover and Mutation) 12

2.3 Problem Representations 12

2.4 Robustness and Generalisation 15

Chapter 3 Methodology **16**

3.1 Error Functions 16

3.2 Clustering and Local Wrappers for GP Evaluation Metrics 18

3.2.1 Class Separation Distance Maximisation 18
3.2.2 Localised Wrapper-based GP Classifers 20

Chapter 4 Results 23
4.1 Initial Parameters for Genetic Programming 23
4.2 Discussion 25
4.2.1 Fitness (Hits) 26
4.2.2 Program Length 28
4.2.3 Tournament Fitness 28
4.2.4 Summary 31
4.3 Error Functions 31
4.3.1 Median Hits 33
4.3.2 Median Program Length 36
4.3.3 Summary 36
4.4 Clustering and Local Wrapper for GP Evaluation Metrics 39
4.4.1 Raw Hits for the Local Wrapper Method on Raw GP Output . 40
4.4.2 Pruned Program Lengths for Raw GP Methods 42
4.4.3 Consequences 42

Chapter 5 Conclusion 45

Appendix A Tournament Fitness 47

Bibliography 54

List of Tables

Table 2.1 Instruction types for our linear genetic programming model . . 9

Table 3.1 Functions used in Experiment 2. (The $equal?(a, t)$ function for hits-based error returns 1 if a and t are equal, and 0 otherwise.) 17

Table 4.1 Data sets used and their attributes 24

Table 4.2 Parameters for the classification and regression problems described in Table 4.1 25

Table 4.3 Parameters for the error function experiments 32

List of Figures

Figure 1.1 a) a switching type wrapper which obscures error information; b) an error function with a smooth transition between class labels 3

Figure 3.1 Distribution of GP output with and without successful clustering on the GP output number line. 19

Figure 3.2 Identification of in-class (+) and out-class (×) raw GP values using a Gaussian model. 20

Figure 4.1 Median hits (for training and test) by initial population size and maximum page count for the BREAST classification problem 26

Figure 4.2 Median hits (for training and test) by initial population size and maximum page count for the C-HEART classification problem 26

Figure 4.3 Median hits (for training and test) by initial population size and maximum page count for the LIVER classification problem 27

Figure 4.4 Median hits (for training and test) by initial population size and maximum page count for the TS classification problem . . 27

Figure 4.5 Median hits (for training and test) by initial population size and maximum page count for the SEXTIC regression problem 27

Figure 4.6 Median hits (for training and test) by initial population size and maximum page count for the TWOBOX regression problem 28

Figure 4.7 Median program length by initial population size and maximum page count for the BREAST classification problem 29

Figure 4.8 Median program length by initial population size and maximum page count for the C-HEART classification problem 29

Figure 4.9 Median program length by initial population size and maximum page count for the LIVER classification problem 29

Figure 4.10 Median program length by initial population size and maximum page count for the TS classification problem 30

Figure 4.11 Median program length by initial population size and maximum page count for the SEXTIC regression problem 30

Figure 4.12 Median program length by the initial population size and maximum page count for the TWOBOX regression problem 30

Figure 4.13 Median training and test percentage hits for the BREAST classification problem with a maximum page count of 24 and initial population size of 125. 33

Figure 4.14 Median training and test hits for the C-HEART classification problem with a maximum page count of 24 and initial population size of 125. 34

Figure 4.15 Median training and test hits for the LIVER classification problem with a maximum page count of 24 and initial population size of 125. 34

Figure 4.16 Median training and test hits for the TS classification problem with a maximum page count of 24 and initial population size of 125. 34

Figure 4.17 Median training and test hits for the SEXTIC regression problem with a maximum page count of 24 and initial population size of 125. 35

Figure 4.18 Median training and test hits for the TWOBOX regression problem with a maximum page count of 24 and initial population size of 125. 35

Figure 4.19 Quartile pruned program length for the best individuals on the BREAST classification problem with a maximum page count of 24 and initial population size of 125. 36

Figure 4.20 Quartile pruned program length for the best individuals for the C-HEART classification problem with a maximum page count of 24 and initial population size of 125. 37

Figure 4.21 Quartile pruned program length for the LIVER classification problem with a maximum page count of 24 and initial population size of 125. 37

Figure 4.22 Quartile pruned program length for the TS classification problem with a maximum page count of 24 and initial population size of 125. 37

Figure 4.23 Quartile pruned program length for the SEXTIC regression problem with a maximum page count of 24 and initial population size of 125. 38

Figure 4.24 Quartile pruned program length for the TWOBOX regression problem with a maximum page count of 24 and initial population size of 125. 38

Figure 4.25 Quartile training and test hits for the BREAST dataset with global and local wrapper functions 40

Figure 4.26 Quartile training and test hits for the C-HEART dataset with global and local error wrapper functions 41

Figure 4.27 Quartile training and test hits for the LIVER dataset with global and local error wrapper functions 41

Figure 4.28 Quartile training and test hits for the TS dataset with global and local error wrapper functions 41

Figure 4.29 Quartile pruned program lengths for the BREAST dataset with four classification metrics based on raw GP output 42

Figure 4.30 Quartile pruned program lengths for the C-HEART dataset with four classification metrics based on raw GP output 42

Figure 4.31 Quartile pruned program lengths for the LIVER dataset with four classification metrics based on raw GP output 43

Figure 4.32 Quartile pruned program lengths for the TS dataset with four classification metrics based on raw GP output 43

Figure A.1 Median bezier smoothed minimum, median average and median maximum fitness at each tournament for the BREAST classification problem 48

Figure A.2 Median minimum, median average and median maximum fitness at each tournament for the C-HEART classification problem 49

Figure A.3 Median minimum, median average and median maximum fitness at each tournament for the LIVER classification problem 50

Figure A.4 Median minimum, median average and median maximum fitness at each tournament for the TS classification problem . . . 51

Figure A.5 Median minimum, median average and median maximum fitness at each tournament for the SEXTIC regression problem . 52

Figure A.6 Median minimum, median average and median maximum fitness at each tournament for the TWOBOX regression problem 53

Abstract

Genetic programming offers freedom in the definition of the cost function that is unparalleled in the realm of supervised learning algorithms. However, this freedom goes largely unexploited in previous work. Here, we revisit the design of fitness functions for genetic programming by explicitly considering the contribution of the wrapper and cost function.

Within the context of supervised learning, as applied to classification problems, a clustering methodology is introduced using cost functions which encourage maximisation of separation between in and out of class exemplars. Through a series of empirical investigations of the nature of these functions, we demonstrate that classifier performance is much more dependable than previously the case under the genetic programming paradigm. In addition, we also observe solutions with lower complexity than typically returned by the classically employed hits (or even sum square error) based cost functions.

Chapter 1

Introduction

One of the purported advantages of Genetic Programming (GP) relative to other supervised learning algorithms is that there is much more freedom in how the fitness (cost) function is expressed.

A cost function, as in neural networks, typically measures the distance between the actual output of a solution and the desired output of a solution, indicating the suitability of that solution for solving a particular problem [7]. In GP, the cost function is paired with a wrapper function which constrains the value of the cost function within a desired interval. Because the wrapper acts as an interface to the cost function, mediating between the ranking subsystem in GP and the feedback of the cost function, GP permits additional freedom in the definition of the cost function.

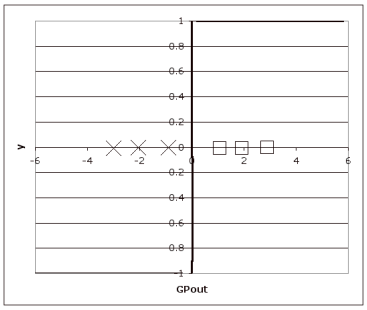
For example, neural networks are typically required to have a cost function that is smooth and therefore differentiable [7] whereas no such requirement exists for GP [10]. To date, however, GP fitness functions do not necessarily build on this freedom in a manner designed to encourage the identification of robust solutions [12]. In this work the design of fitness functions for classification problems is revisited by explicitly considering the contributions made by wrapper and cost function. Specifically, the GP wrapper is used to transform the 'raw' GP output, a value limited only by the numerical range of the computing platform, to an interval appropriate for distinguishing class. Here binary classification problems are considered, thus typical ranges would be $[0, 1]$ or $[-1, 1]$. Common practice has been to utilize a wrapper based on a binary switching function, as in Figure 1.1 (a).

$$f(x) = \begin{cases} -1 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{1.1}$$

The ensuing fitness (cost) function then merely counts the number of misclassified training exemplars. The hypothesis of this work is that such an approach to designing a wrapper-cost function combination results in sub-optimal classifiers with poor robustness properties. Specifically, the switching type wrapper hides useful information: it does not explicitly encourage the raw GP output values to be distributed away from the switching point of the wrapper as in Figure 1.1 (a). In particular, as long as the GP output points for each class fall on the correct side of the switching function transition, there will be zero error, irrespective of their distance from the wrapper class transition. Conversely, a wrapper with a finite transition region between the two class labels would provide the basis for a more informative cost function, therefore quantifying the degree of any error or the degree of separation currently achieved between two classes, as in Figure 1.1 (b). In this case, points on the GP output axis at the transition region of the wrapper function will result in a nonzero error. Error minimisation now corresponds to both mapping points to the correct side of the wrapper transition point and maximising the distance from the transition point. We consider such a solution as more robust as the greater separation between two classes on the GP output axis is less likely to result in previously unseen exemplars being mapped to the wrong side of the wrapper transition[1] (a misclassification). Moreover, the new wrapper also provides the basis for establishing certainty in the classification, as opposed to merely presenting the result as a binary in class or out of class answer.

In this work, we propose taking this concept further by concentrating on explicitly maximising the separation between in and out of class exemplars by expressing the

[1] Training data is implicitly assumed to be representative of the wider (unseen) test data, as per any machine learning classifier.



(a) Wrapper with switching function

(b) Wrapper function with finite transition region

Figure 1.1: a) a switching type wrapper which obscures error information; b) an error function with a smooth transition between class labels

problem in terms of the original GP output values, that is, the horizontal axis in Figure 1.1. We identify the cluster means for each class and measure the error by finding the separation between cluster means. We then use a nearest neighbour strategy to determine the class membership of previously unseen data. We intend this to encourage the development of a robust separation between classes, based on the training data. We also employ a local wrapper for predicting in or out of class membership, placing increased emphasis not only on the class separation but also on predictable class membership behaviour, as defined in terms of variance from the mean. In both cases, an increased emphasis on a clear separation between clusters may offer an improvement in post-training GP performance.

This thesis is organised into several chapters. Chapter 2 summarises previous work in genetic programming related to our hypothesis. Chapter 3 describes our methodology and details of our linear genetic programming framework and evaluation methods. Chapter 4 summarises our experimental design and explains our observations. Chapter 5 offers our conclusions and suggests future avenues of exploration in this area of research.

Chapter 2

Previous Work

For classification and regression problems, previous studies in evolutionary computation have introduced a variety of techniques for expressing error in the genetic programming evaluation phase. Here we describe the relevant methodologies which led to and informed our exploration of clustering methodologies for genetic programming classifiers. In the following, raw GP output denotes the value returned by the GP model before application of the wrapper function, that is, a value found on the horizontal axis as opposed to the vertical axis of Figure 1.1.

2.1 Evolutionary Computation

Evolutionary computation is a family of computational search methods inspired by the incremental and adaptive action of the process of biological evolution [5]. Biological evolution, as it is commonly understood, is an emergent process occurring in populations composed of individuals engaging in mutual competition for resources, whose physical characteristics are determined by an inherited scheme passed on during reproduction. Observing the "multitude of forms"[1] produced by the systematic effect of biological evolutionary operators, there can be derived a nominal motivation for the evolution of computational methods, using biological evolution for metaphorical inspiration [3, 10].

[1] "On separate continents, and on different parts of the same continent when divided by barriers of any kind, and on outlying islands, what a multitude of forms exist [...]", Charles Darwin in [3].

2.1.1 Techniques in Evolutionary Computation

Within the field of evolutionary computation, genetic algorithms and genetic programming embody the notions of using evolutionary methods to produce both numerical solutions to problems (optimisation) and produce programmatic solutions to problems (modelling), respectively.

Genetic algorithms and genetic programming address fundamental machine learning design problems by taking advantage of the incremental improvement offered by an evolutionary approach.

In the construction of a machine learning algorithm, we are faced with a variety of design choices affecting the algorithm's applicability and performance in assorted operational environments. A learner must be able to determine how suitable a chosen strategy is for reaching its goal. This implies that a target function must be judiciously chosen based on *prior* knowledge of the problem environment to provide the learner with adequate feedback about its progress. A learning machine must also use an appropriate representation for the problem and the environment in which solutions will be located. Finally, a learner which uses a number of steps in succession to solve a problem must be able to discern which steps proved valuable in solving the problem so that positive or negative credit may be assigned to those steps [17].

Evolutionary computation addresses the credit assignment problem through the action of evolutionary modification operators. When mutation or crossover occur for a given individual or pair of individuals, any change in fitness can be ascribed to the change caused by the last mutation or crossover in the individual's composition. Thus, every change in an individual that is the direct result of evolutionary operation is evaluated and consequently scored in the next generation or the next evaluation. Credit is assigned automatically.

Representation and goal measurement are typically handled through *a priori* or

expert knowledge. Even in these cases, some adjustments may be left to the evolutionary process where possible, resulting in a self-tuning mechanism.

2.1.1.1 Genetic Algorithms

In the case of genetic algorithms, evolutionary operators operate on linear sequences of genes, or strings. The representation for such a string takes one of three forms: bit, integer or real-valued, selected on their relative appropriateness to the problem domain. There is typically a representative mapping between the genes of this string and potential solutions to the problem in question. In [5], two examples of optimisation problems which may be solved through the application of genetic algorithms include

- the Eight Queens Problem (where the solution space is represented by eight numbers indicating the row on which each queen ought to be placed, noting that any two queens may not share a column, diagonal or row. The fitness for a solution is quantified by counting the number of illegal queen placements.)

- the 0-1 Knapsack Problem (where the solution space is represented by a binary string of length n, where each gene represents the inclusion or exclusion of an item i with cost c_i and value v_i. The fitness for a solution is quantified by the magnitude of the sum of the products of the costs and values of the included items.)

In each of the above example applications of genetic algorithms, a string is modified by repeated application of evolutionary selection and search operators. The string of numbers itself directly represents a solution to the problem. This process may be halted at any time or when a chosen stop criterion is reached. The best solution is collected from the population of evolved solutions.

2.1.1.2 Genetic Programming

Where genetic algorithms techniques focus on the evolution of solutions to problems from discrete and continuous optimisation, genetic programming evolves programs which solve modelling problems. Here, the fitness of a candidate solution is determined by executing and comparing its output with the correct output for a problem, that is, a supervised or reinforcement learning context. The representation for genetic programming has historically taken the form of tree-based functional expressions but it can take other forms, including linear programs, to which we shall return in section 2.2.

Among other applications, genetic programming can be used in the case of classification problems and regression problems, as in [10], where genetic programming is used to

- produce decision trees for the intertwined spirals problem;
- model an unknown function through symbolic regression.

Genetic programming typically follows an algorithm similar to that of genetic algorithms, with the main difference being that the evaluation step involves execution of program instructions [5, 8]. Algorithm 1 on the following page outlines the form of the generic evolutionary computation paradigm.

Returning to the machine learning design choices discussed in Section 2.1.1, we find that the credit assignment problem is partially addressed in the effects of steps 2 to 4. In step 2, each candidate solution (individual) is evaluated according to the fitness function. Steps 3 and 4 select individuals for mutation and reproduction based directly on these fitness values. Because the population is initially systematically diversified, fitness is also initially very diverse. Each mutation and crossover results in changes to individuals - those changes are evaluated in the next iteration of the

Algorithm 1 The typical genetic programming algorithm. [5]

1. Initialise a population of N individuals. Choose the probability of mutation, $p(m)$, and probability of crossover, $p(c)$.
2. Evaluate each individual, i, according to the fitness function, recording the maximum fitness, f_m as well as each individual's fitness, f_i.
3. Assign to every individual a probability of reproduction $p(r) = \frac{f_i}{f_m}$.
4. Test $p(r)$ for each individual to determine inclusion as a parent in a mating pool, P.
5. For each parent in the mating pool, apply the mutation operator with probability $p(m)$ and apply crossover (with a randomly selected partner) with probability $p(c)$, producing one or two children.
6. Replace each parent with its child.
7. If the population has not yet converged and not yet reached a maximum number of generations, go to step 2.

algorithm. Thus, every incremental change toward or away from a particular strategy is evaluated, ensuring that credit is assigned for all changes.

The genetic programming environment has certain characteristics selected in common with other methods for evolutionary computation. These include many options of which a few are listed here in parentheses.

- representation (tree-structured, linear instructions, graph based)
- initialisation (random or partially constrained)
- evaluation (fitness functions)
- selection operators (fitness-proportional, tournament based, degree of elitism)
- search operators (crossover, mutation)

In our work, we have used the modified genetic programming algorithm described

Instruction Type	Generic Form
reg-reg	$R_x \Leftarrow R_x$ *opcode* R_y
reg-input	$R_x \Leftarrow R_x$ *opcode* I_y
constant	$R_x \Leftarrow$ 'n' bit integer constant

Table 2.1: Instruction types for our linear genetic programming model

here as Algorithm 2. The main differences in our approach include (steady state) tournament selection rather than a generational approach and a linear representation for individuals rather than tree-based as in Koza's canonical genetic programming model [10, 11]. These are described more fully in the following section. The contribution of this thesis, however, is independent of the particular form of GP employed.

2.2 Page-based Linear Genetic Programming

We employ a fixed length, linear GP representation in which individuals take the form of instruction sequences, grouped into pages of a common instruction count, as in [8]. To define maximum program length, we *a priori* state maximum page count. The natural implication of this is that the initial population is initialised over the total range of permitted program lengths; whereas a variable length representation begins with individuals initialised over a limited range (of short programs) and lets them grow up to some *a priori* specified size limit. The fixed length representation is enforced by limiting crossover to the exchange of a single equal length page of instructions between two parents. Each 2-address instruction is represented by an integer and therefore a page of instructions is a sequence of integers which are decoded at run-time.

2.2.1 Representation

The linear programs operate upon a (virtual) register-based machine. The instruction format supports three instruction types: register-register, register-input and constants, see Table 2.1. Opcodes considered within this work are limited to the four

Algorithm 2 Algorithm for the L-GP architecture described in Section 2.2.

1. Generate the population randomly. Let T be the number of completed tournaments (initially zero).
2. Randomly select 4 tournament participants from the population.
3. Evaluate each individual in the tournament and rank them.
4. Copy the best two individuals over the worst two individuals.
5. Perform crossover and mutation according to their respective probabilities of occurrence.
6. Increment T; if T is less than the tournament limit (50000) and the population has not yet converged, go to 2.
7. Record the best individual, other desired statistics.

arithmetic instructions alone (addition, subtraction, multiplication, protected division), and operate on either two registers or a register and input from the data set. The range of register references R_x and R_y is defined *a priori* by the number of (general purpose) registers allocated to the (virtual) register machine. Naturally the range of references to inputs, I_y, is defined by the number of features in the data set. Previous work has established empirically that the arithmetic operators are sufficient for solving a wide range of problems, including classification [13] and intrusion detection [18].

2.2.2 Initialisation

Individuals are initialised by first selecting the maximum page count, with uniform probability over the interval $[1, ...MaxPages]$. Each instruction is then initialised by first selecting an instruction type where instructions defining constants are half as likely as either register-register instructions or register-input instructions. That is to say, without such a bias, half of the instructions comprising an individual would take the form of register-register and register-input instructions, and the other half would

describe constants [8].

2.2.3 Evaluation

In the genetic programming evaluation phase, individuals are assigned fitness values based on a problem-dependent fitness metric. Fitness for an individual is typically defined in terms of the individual's ability to correctly solve the problem. For example, in classification, the fitness of an individual may correspond to the number of correctly classified exemplars. For a regression problem, the fitness of an individual might be interpreted as a function of the distance between the correct points and the candidate points produced by the GP individual.

The fitness function scores or ranks individuals, enabling though biasing the ranking and selection phase. The selection of appropriate fitness functions for classification problems forms the contribution of this thesis and is presented in detail in Section 3.

2.2.4 Selection

We used steady-state tournament selection with a tournament size of four. The four individuals are selected with uniform probability from the population and ranked according to the evaluation function. The best two individuals in the tournament are selected for reproduction. They are copied over the worst two tournament individuals and crossover and mutation applied.

The 'copy' operator replaces the losers of the tournament in the original population and is inherently elitist (best of the population is guaranteed to survive). Moreover, such a scheme is known to result in a higher takeover rate[2] than generational selection [6].

[2]Poor performing individuals die out more quickly than with the case of proportional selection.

2.2.5 Search (Crossover and Mutation)

The crossover operator consists of swapping randomly selected single pages of equal instructions between two parent individuals. This guarantees that the length of each individual remains fixed. An annealing schedule is used to incrementally modify the number of instructions constituting a page during training [8].

In addition to the crossover operator, L-GP typically employs two types of mutation. The first consists of randomly choosing an instruction within an individual and performing the XOR operation between that instruction and a random integer. Mutation therefore provides an avenue for introducing instructions not currently in the population. The second mutation strategy is to swap two randomly chosen instructions, again within the same individual. This establishes a path for investigating alternative instruction orders within the same individual. Both have associated probabilities defining the frequency of application.

2.3 Problem Representations

Hits A classic measure of fitness for classification problems is the hits-based metric, as in Koza's genetic programming experiments [10]. A hits-based metric measures the fitness of a particular problem solving method in terms of the number of subinstances of the problem which that method correctly solves. In terms of genetic programming, the hits-based fitness of an individual would correspond to the number of exemplars in a training data set which are correctly classified by the individual into their respective classes.

Typically, the hits-based metric is enabled by the use of a wrapper function which maps the raw output of a GP individual to a discrete classification interval, as in Equation (1.1) on page 2, where the discrete value indicates the GP classification. The drawback of the switching-type wrapper is that we are limited to binary classification

problems, although this may be overcome to some degree through a combination of experts, as in ensemble programming, boosting or bagging [9] or through problem decomposition [15].

MSE / SSE The mean squared error and sum squared error metrics are described by Banzhaf *et al* in [1].

The sum squared error metric characterises error as the sum, over all exemplars, of the squares of the magnitudes of the differences between the actual outputs[3], a and the desired outputs, d. In [1], Banzhaf *et al* use the following,

$$\sum_{i=1}^{n} (a_i - d_i)^2$$

The mean squared error metric is the sum of the squared error, divided by the number of outputs, n, as in [1],

$$\frac{1}{n}\sum_{i=1}^{n} (a_i - d_i)^2$$

A useful feature of the squared error metric is that error is not distributed uniformly. The error rate increases quadratically as the magnitude of the absolute error increases. Additionally, dividing by n to obtain the mean squared error will temper large error values. Additionally, the error rate now is a continuous value, permitting greater sensitivity to changes in the GP performance.

Weighted Hits Eggermont *et al*, in [4], describe a method involving the combination of an error metric with weights which are adjusted by a predetermined quantum at each generation. Eggermont *et al* describe the overall error $f(x)$ for an individual

[3][1] makes no reference to the wrapper employed. Thus it is not clear whether the 'actual output' implies a linear wrapper (that is, the raw GP output is used directly) or the raw GP output is mapped to an interval appropriate for interpretation in terms of a (binary) classification.

x,

$$f(x) = \sum_{r \in D} w_r \cdot error(x, r)$$

where $error(x, r)$ may be a switching-type function, for example,

$$error(x, r) = \begin{cases} 1 & \text{if } x \text{ classifies data record } r \text{ incorrectly} \\ 0 & \text{otherwise} \end{cases}$$

or $error(x, r)$ could be a continuous function, as in mean squared error, described previously. The weights magnify errors which occur more frequently over time, thereby penalising individuals which persist in misclassifying a particular record. This technique showed improvement over standard GP on some datasets and produced comparable or inferior performance on others.

Static and Dynamic Range Selection Tree structured GP is limited to a single output. Thus, when used with the switching-type wrapper, Equation (1.1), it is formulated as a function with only two distinct output values. This becomes a problem in the case of a classification problem involving multiple classes, unless composite techniques are used to combine multiple, separately evolved, binary classifiers to solve a single problem.

One method for performing multiple classifications without evolving separate classifiers is described as range selection, by Loveard *et al* in [15]. It is an extension of the concept of a binary switching function. The binary switching function Equation (1.1) associates two ranges of output values, $(-\infty, 0]$ and $(0, \infty)$, with the class labels 0 and 1, respectively. The range selection strategy divides the one dimensional space associated with the raw GP output into further intervals, with as many labels as there are class intervals.

Loveard *et al* propose two strategies. The first is static range selection, where the

intervals are fixed. However, they had more success with dynamic range selection, where, for each individual, its range intervals are determined at each generation by setting aside a portion of the training data. Thus, each interval's limits are determined using a nearest neighbour algorithm on the outputs for these exemplars. Fitness takes the form of counting the number of times that exemplars from the remainder of the training set produce a raw GP output value lying in a region assigned the same class label.

2.4 Robustness and Generalisation

An important criterion of quality in any machine learning algorithm is the ability to produce robust and, more importantly, general solutions. In [12], Kushchu defines robustness as "the desired successful performance of the solution when it is applied to an environment similar to the one it was evolved for," whereas he defines generalisation to refer to the "performance of the [solution] during the testing process".

We contend that switching-type wrappers contribute to poor generalisation in that they obscure fine differences in the raw GP output to such an extent that the GP algorithm is rewarded for behaviour not conducive to identifying robust solutions.

For example, a drawback of range selection in terms of robustness is that, as with the switching-type wrapper function, it obscures distinctions between output values within the selected intervals. No reward is provided for mapping exemplars to raw GP output values that both correspond to the relevant region and are distant from the region boundaries.

This consideration is important in our work as our clustering methodology, by virtue of its close association with the raw output of the genetic programming system, may explicitly reward a mapping that maximises the separation between classes, in comparison with methods based on switching functions.

Chapter 3

Methodology

3.1 Error Functions

Since Koza popularized the Genetic Programming approach, the wrapper for classification problems has taken the form of a switching function, Figure 1.1 (a) [10, 11]. As indicated in the introduction, we note that such a wrapper effectively throws away a lot of useful information that could adversely affect the generalisation performance of the resulting GP classifier. There is, however, a long history of wrapper function development within the context of neural networks[1]. Moreover, the selection of a wrapper also has direct implications for the nature of the corresponding fitness (cost) function. A wrapper based on the switching function limits the fitness function to a mere count of the number of correctly classified exemplars. That is to say, the corresponding distance metric is binary. Conversely, neural networks are typically required to have an activation function that is smooth (i.e. differentiable). If we consider the particular case of global activation functions[2], this results in a requirement for a monotonically increasing function, where the most popular operator for achieving this is the sigmoid function [7], Figure 1.1 (b).

As established in the introduction, this now provides the basis for exemplar errors that increase as the transition point of the activation function is approached, Figure 1.1 (b), as well as when exemplars are explicitly misclassified (wrong side of the

[1]The neural network literature refers to wrapper functions as activation functions [7].

[2]Local activation (wrapper) functions have also been widely utilised in the neural network literature, particularly within the context of hidden layer neurons (e.g. Radial basis function networks and Support Vector Machines).

Error Name	Wrapper Used	Error (a=actual, t=target)
absolute	$2 * (1.0 + \exp(-gp_{out}))^{-1} - 1$	$\lvert actual - target \rvert$
Bernoulli	$(1.0 + \exp(-gp_{out}))^{-1}$	$-\log(a + t - 1.0)$
Hits-Based	$(gp_{out} > 0)?\ a = 1\ else\ a = 0$	$1 - equal?(a, t)$
Minkowski	$2 * (1.0 + \exp(-gp_{out}))^{-1} - 1$	$\lvert actual - target \rvert^{r}$
squared	$2 * (1.0 + \exp(-gp_{out}))^{-1} - 1$	$\lvert actual - target \rvert^{2}$

Table 3.1: Functions used in Experiment 2. (The $equal?(a, t)$ function for hits-based error returns 1 if a and t are equal, and 0 otherwise.)

wrapper transition point). Moreover, as each error distance is now real valued, we are also free to build a fitness (cost) function that penalizes or weights errors in different ways.

In this work we will consider fitness functions based on one of five forms of error distance - absolute, Bernoulli, Minkowski, and squared - including the switching type wrapper. The selection of these functions was informed by previous neural network literature [7, 19], where cost functions have been extensively evaluated, in comparison with the GP literature. Table 3.1 summarizes the association between wrapper and error metric. Specifically, the Bernoulli fitness function assumes a probabilistic model for the classification problem (labels are binary) [20], thus the activation function maps the 'GPout' axis to the unit interval, using a sigmoid type global mapping, e.g. Figure 1.1 (b), limited to the interval $[0, 1]$. The associated Bernoulli error metric applies an exceptionally high penalty to any exemplar misclassification. We also note that such a formulation is equivalent to the entropy penalty function. The absolute error assumes an equal penalty for any error, and makes use of the wider interval of the hyperbolic tangent wrapper function, Figure 1.1 (b). Minkowski and square error apply an increasing (decreasing) penalty to larger (smaller) exemplar errors relative to an error distance of unity, thus the wider range of the sigmoid wrapper function is again appropriate. In all cases the fitness function is merely the sum of error taken across all training exemplars for a given wrapper / error distance metric combination.

3.2 Clustering and Local Wrappers for GP Evaluation Metrics

In the previous section, we introduced a rationale for utilising non-switching wrapper functions and their corresponding error metric to encourage a wider class separation over training exemplars. In this section, we take a different approach to encouraging such a separation. To do so we recognise that the overall objective is to maximise the distance between points on the 'raw' GP output axis representing the same class. Viewed from this perspective, the objective is to map the class data into separate clusters (on the 'raw' GP output axis) whose intracluster variance is minimised, but intercluster distance is maximised, as in Figure 3.1. Such a rationale does not explicitly use a wrapper function for fitness evaluation.[3]

Section 3.2.2 proposes a local wrapper function approach. In this case, error is described in terms of two Gaussian distributions, one for in class and one for out class. Error is minimized with respect to boh class and (local) wrapper membership. Class separation is now less directly represented, but takes the form of an indirect property of minimising membership of the wrong distribution.

3.2.1 Class Separation Distance Maximisation

Let us consider the scenario of evaluating a single individual on all the exemplars of a training or test data set for a binary classification problem. The data set, R, contains training exemplars. Each exemplar p_i is itself a pair, consisting of an ordered tuple of input features and an *a priori* classification label l_i. Executing a GP individual (program) on each exemplar p_i generates a set S of pairs of the form $(gp_{out}(i), l_i)$, where $gp_{out}(i)$ is the raw GP output on exemplar i.

Now that we have GP outputs and a label for each GP output assigning it to a particular class, we can now characterise the respective class distributions in the

[3]The wrapper function may be considered linear or an identity function.

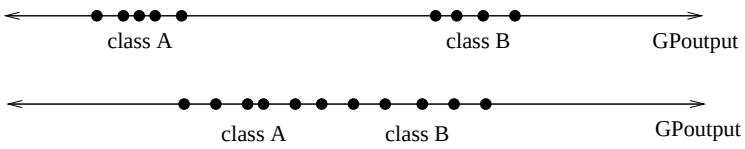


Figure 3.1: Distribution of GP output with and without successful clustering on the GP output number line.

GP output space. For this we draw on metrics from pattern recognition and feature selection, particularly the mean, variance and class separation distance, described in [2, pp. 516-517]. We reproduce these here for reference.

Let us define S_k to be a set containing every $gp_{out}(i)$ in S for which $l_i = k$. That is, S_k contains all the GP output values in S that were generated for a record i with label $k \in \{0, 1\}$. Then the approximate class mean[4] is calculated as in Equation (3.1). (Here $gp_{out}(j)$ refers to elements of S_k.) Similarly, the variance is expressed in Equation (3.2) using the approximate class mean.

$$\hat{\mu}_k = \frac{1}{|S_k|} \sum_{j=1}^{|S_k|} gp_{out}(j) \tag{3.1}$$

$$\hat{\sigma}_k^2 = \frac{1}{|S_k|} \sum_{j=1}^{|S_k|} (gp_{out}(j) - \hat{\mu}_k)^2 \tag{3.2}$$

Finally, Equation (3.3) characterises the distance between the approximate class means, normalised by the approximate class variance.

$$\hat{D}_{k_1 k_2} = \frac{|\hat{\mu}_{k_1} - \hat{\mu}_{k_2}|}{\sqrt{\hat{\sigma}_{k_1}^2 + \hat{\sigma}_{k_2}^2}} \tag{3.3}$$

In our experiments, we use the value $\hat{D}_{k_1 k_2}$ to rank individuals during the selection phase. Thus, individuals are rewarded for generating a more distinct separation between the two classes, that is, maximising Equation (3.3). In summary, GP is

[4]From [2]: "The caret ... remind[s] us that these are estimates of the class means based upon the training set[.]"

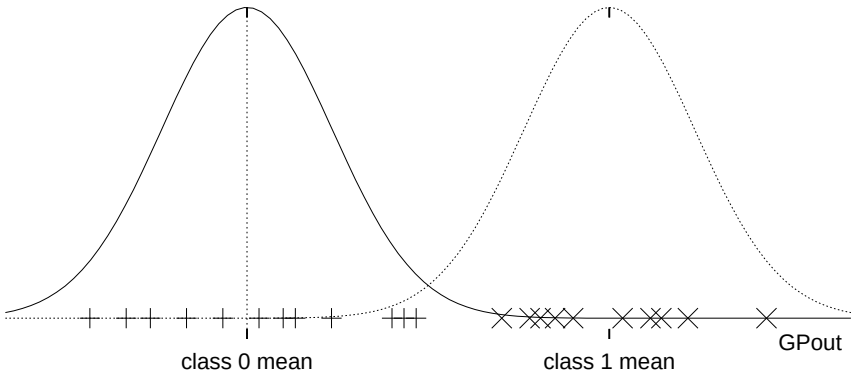


Figure 3.2: Identification of in-class (+) and out-class (×) raw GP values using a Gaussian model.

explicitly rewarded for providing a mapping from a high dimensional input space to a one-dimensional output space such that the class separation distance of the raw GP output space is maximised. Post-training, the model is applied to unseen data using a nearest neighbour algorithm to determine class membership.

3.2.2 Localised Wrapper-based GP Classifers

The classification metric described in the previous section provides the opportunity to base the classifier output on a wrapper described by a local membership function. (e.g. a Gaussian). In this case, when unseen data is mapped to regions of the GP output axis that do not correspond to the (cluster) mapping identified during training, they should be regarded as distinct from the distribution used to develop the GP model, Figure 3.2.

The wrapper for such a local membership function is built by estimating the mean and variance of each class over the corresponding raw GP output values (gp_{out}). Error is then expressed relative to each of the individual classes insofar as their members conform to a normal distribution. The objective is to encourage as many of the in-class exemplars to appear as close to the mean of the class' Gaussian membership function as possible, and to distance themselves as far as possible from the mean of

the opposite class, or (where x is the in-class and y is the out-class),

$$inclasserror = \sum_{i=1}^{N_x} \left(1 - \exp\left(-\frac{1}{2\sigma_x^2} \left\| gp_{out}(i) - \mu_x \right\|^2\right)\right) \tag{3.4}$$

$$outclasserror = \sum_{i=1}^{N_x} \left(\exp\left(-\frac{1}{2\sigma_y^2} \left\| gp_{out}(i) - \mu_y \right\|^2\right)\right) \tag{3.5}$$

where N_x is the size of the in-class x and where the fitness of the individual is the sum of of the class errors for both classes and the minimum value denotes the fittest individual. Note that we evaluate these equations twice, once for each set of points representing classes 0 and 1, to obtain the total error (that is, once with $x = 0, y = 1$ to obtain the total error for class 0 and again with $x = 1, y = 0$ to obtain the total error for class 1). Note also that the normalisation ($\frac{1}{\sqrt{2\pi\sigma}}$) typically associated with a Gaussian distribution (enforcing a unit integral area) is not utilised. This ensures that the minimal distance condition, $gp_{out}(i) = \mu_1$, returns a maximum membership (of 1). Without this constraint, the error metric of (3.4) and (3.5) will not be appropriately scaled.

Relative to the cluster separation approach of section 3.5, the localised wrapper approach has the implicit advantage that, as both in-class and out-class error are accumulated, selection pressure encouraging separation between clusters is maintained. The drawback of the local wrapper approach relative to the cluster separation metric is computational; three passes through the training data are necessary before fitness of an individual may be expressed. A pass through the training data is required to estimate the mean and a second pass estimates the variance of the in-class data. A third pass through the entire training dataset is required to estimate class membership, Equation (3.4) and (3.5). However, the program expressed by the individual is only run once through the entire process (to produce the raw GP output values).

Post-training, we classify test data by using the Gaussian function explicitly as a

membership function, Equation 3.6, where x corresponds to an exemplar from the test data set; $gp_{out}(x)$ is the raw GP output produced by the individual when evaluated on exemplar x; and i is the class for which we are testing membership.

$$membership(x, i) = \exp\left(-\frac{1}{2\sigma_i^2} \left\| gp_{out}(x) - \mu_i \right\|^2\right) \tag{3.6}$$

Then, the class i to which a GP individual assigns an exemplar x from the test data is decided by choosing $i \in \{0, 1\}$ such that the value of $membership(x, i)$ is maximised.

Chapter 4

Results

This section reports on two sets of experiments. In the first case, the significance of population size and program complexity (instruction count limit) are investigated using the hits fitness function. These are typically considered the principal design parameters associated with GP models [10]. In doing so, we demonstrate that neither parameter has a significant impact on solution quality. Population size does however appear to be correlated with additional solution complexity. The second study considers the contribution of local and global wrapper operators as discussed in Chapter 3. The local wrapper operators are shown to be significantly more robust than their global counterparts. Thus, from a pragmatic perspective, more consideration should be given to establishing an appropriate representation and fitness function than testing the impact of size and complexity constraints.

4.1 Initial Parameters for Genetic Programming

We hypothesised that the maximum page count and initial population size, beyond necessary minimum values, would have no significant effect on the median fitness and median program length of best-of-run solutions (programs). We used six data sets, of which two are regression problems and four are classification problems. These are summarised in Table 4.1.

The datasets we have chosen bear certain characteristics. The 'breast' and 'c-heart' datasets are medical diagnosis classification sets which are known to support accuracies in the 80% to 90% range [14]. The 'liver' classification problem dataset

Name	# of exemplars	Problem type	Function
breast	699	classification	-
c-heart	303	classification	-
liver	345	classification	-
ts	192	classification	-
sextic	50	regression	$f(x) = x^6 - 2x^4 + x^2$
twobox	10	regression	$f(x) = x_0 y_0 z_0 - x_1 y_1 z_1$

Table 4.1: Data sets used and their attributes

is also based on medical diagnoses. The 'liver' set typically supports a classification accuracy of about 60% to 70% [14]. The 'ts' dataset is an artificial benchmark classification problem frequently used in neural network and genetic programming research [8]. The dataset contains points drawn from two intertwined spirals on a two dimensional Cartesian space. The two regression problems, 'sextic' and 'twobox', are both artificial datasets which represent well known GP benchmarks [11]. We emphasise that our objective is to provide for the comparative evaluation of the error-cost function, as opposed to establishing new levels of performance on these datasets.

Each dataset was split into training and test partitions. The training partition contained 75% of the exemplars in the original dataset while the test partition contained 25% of the exemplars in the original dataset. These partitions were generated by randomly selecting exemplars with the constraint that the ratio of class membership among exemplars in the original dataset must be preserved in the training and test sets.

The experimental design is enumerated in Table 4.2. For each dataset, we performed 50 independent runs where we varied two parameters, maximum page count (for values of 24, 48 and 72) and initial population size (for values of 125, 500, 1000 and 5000). For these runs, we pregenerated 50 random seeds and used this set of seeds for each combination of maximum page count and initial population size. We therefore blocked for the effect of random initialisation in our experiments.

Objective	(Correctly classify exemplars) OR (fit a curve)
Terminal Set	x_1, x_2, x_3, x_4
Functional Set	+,-,*,%, load constant
Fitness Cases	# of exemplars in dataset
Fitness	Sum Squared Error
Selection	Tournament ($\lambda = 4$) Selection
Hits (Classification)	# of correct classifications
Hits (Regression)	# of cases with abs. error < 0.01
Maximum Page Size	8
Maximum Page Count	(24, 48, 72)
Init. Population Size	(125, 500, 1000, 5000)
Termination	50000 tournaments or 100% hits
Experiments	50 independent runs

Table 4.2: Parameters for the classification and regression problems described in Table 4.1

4.2 Discussion

Using the experimental framework described in Section 4.1, we examined the performance of linear genetic programming while varying the static parameters of initialisation.

We measured two aspects of performance for individuals in our experiments. We recorded the fitness, in terms of the quartile hits scored by that individual on the training and test data, and the quartile program length, in terms of the individual's instruction count after training. Intron removal was typically employed when evaluating solution instruction counts, where this might result in a 70% to 80% reduction in instruction count [1].

For classification problems, hits are directly measured by comparing the actual output of the switching-type wrapper with the desired output from the data set's class labels. We formulated the regression problems as data sets generated from the desired function, complete with inputs and outputs. A hit for a GP individual on a regression problem was defined as a match between the output of the individual and the desired output from the data set, within a tolerance of ± 0.01.

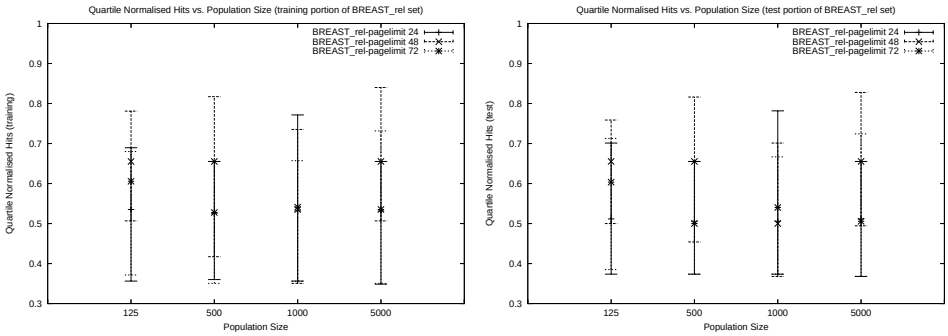


Figure 4.1: Median hits (for training and test) by initial population size and maximum page count for the BREAST classification problem

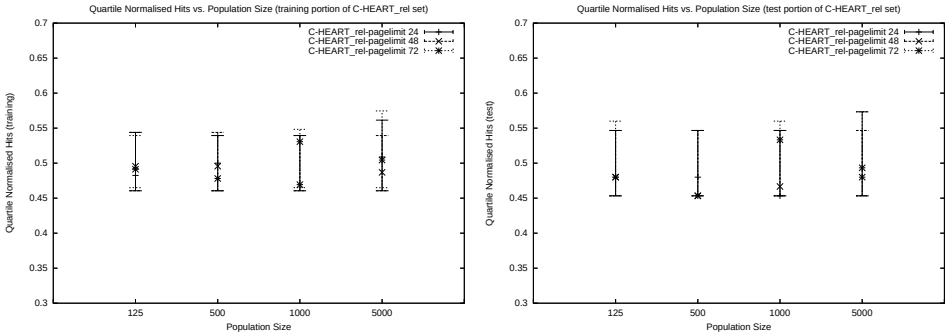


Figure 4.2: Median hits (for training and test) by initial population size and maximum page count for the C-HEART classification problem

Additionally, we recorded the minimum, average and maximum fitness scored in each tournament. Using this statistic, we monitored the convergence in fitness of the GP search process.

The results we observe support the hypothesis that the population size, as well as the maximum page count, largely fail to affect the outcome of the linear genetic programming framework, encouraging exploration of more dynamic means for modifying performance. Observations supporting this conclusion are detailed in Sections 4.2.1 to 4.2.3.

4.2.1 Fitness (Hits)

Figures 4.1 through 4.6 show the quartile hits for each combination of initial population size (125, 500, 1000, 5000) and maximum page count. (24, 48, 72). As the

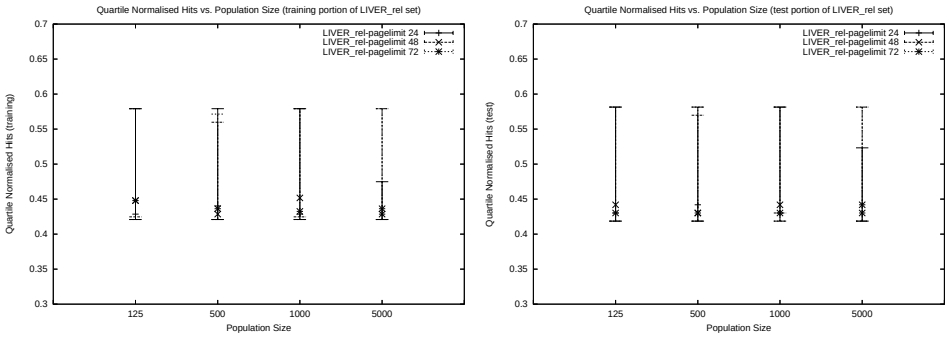


Figure 4.3: Median hits (for training and test) by initial population size and maximum page count for the LIVER classification problem

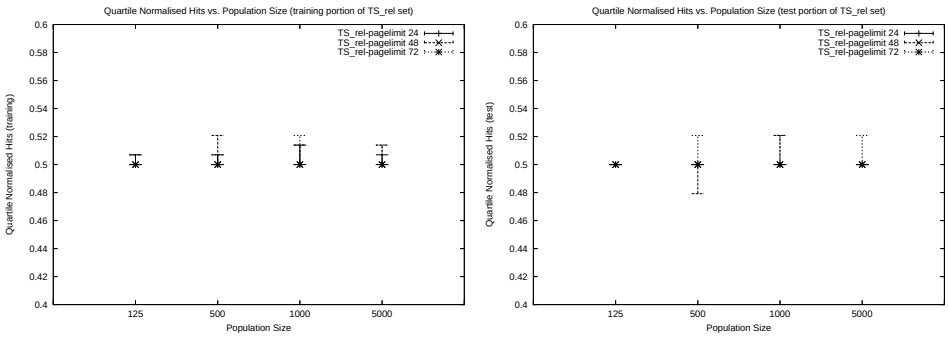


Figure 4.4: Median hits (for training and test) by initial population size and maximum page count for the TS classification problem

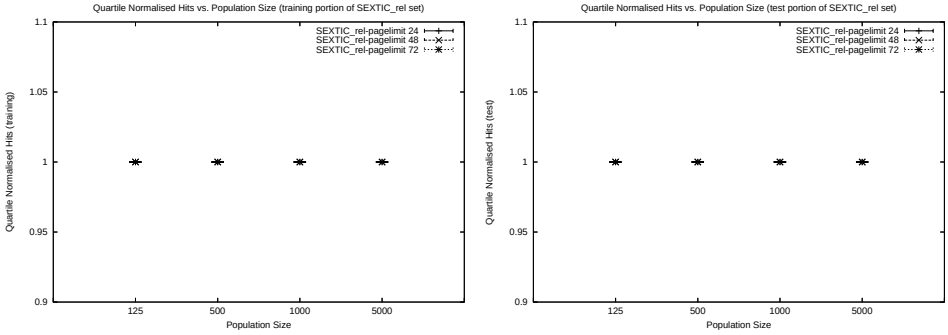


Figure 4.5: Median hits (for training and test) by initial population size and maximum page count for the SEXTIC regression problem

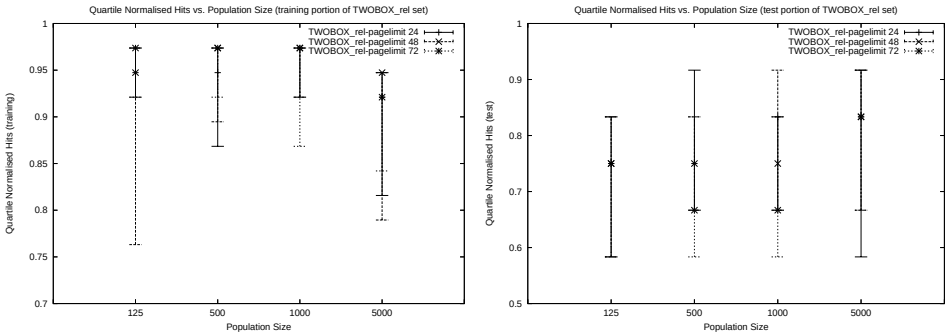


Figure 4.6: Median hits (for training and test) by initial population size and maximum page count for the TWOBOX regression problem

initial population size and maximum page count are varied, we can observe no significant change in the median number of hits. Note that for the sextic problem, Figure 4.5, the quartiles and median are all the same value. This implies that over 75% of individuals solve the problem correctly.

4.2.2 Program Length

Figures 4.7 through 4.12 show the quartile program lengths for each combination of initial population size and maximum page count. The page count limits of 24, 48 and 72 have corresponding instruction limits of 192, 384 and 576. (That is, a maximum of 8 instructions per page.) There appears to be a correlation between an increasing maximum page count and the quartile program lengths. A maximum page count of 72 produces a much wider spread than a maximum page count of 24. Moreover, as larger populations are employed, the solution length tends to increase, relative to the smallest population (125 individuals).

4.2.3 Tournament Fitness

In the interest of verifying the convergence of fitness, we recorded the median minimum, median average and median best fitness for each tournament. We manipulated the initial population size and maximum page count as before. Figures A.1 through

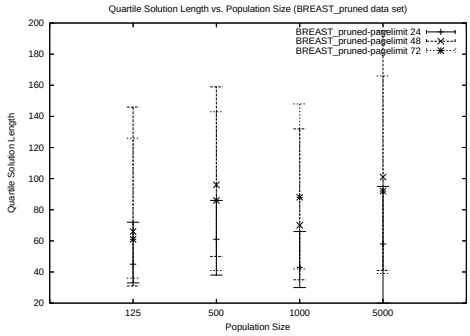


Figure 4.7: Median program length by initial population size and maximum page count for the BREAST classification problem

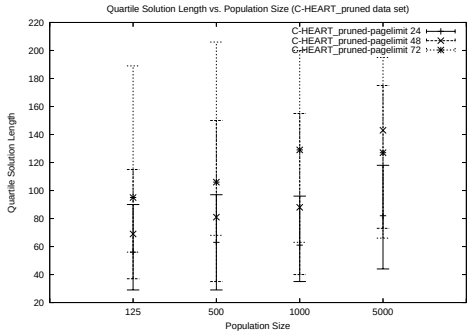


Figure 4.8: Median program length by initial population size and maximum page count for the C-HEART classification problem

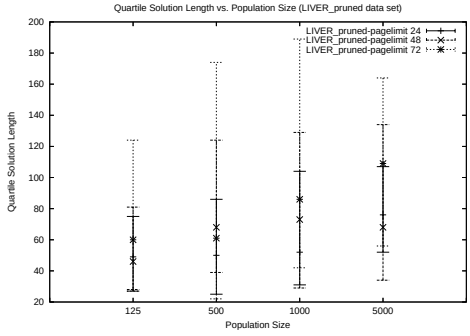


Figure 4.9: Median program length by initial population size and maximum page count for the LIVER classification problem

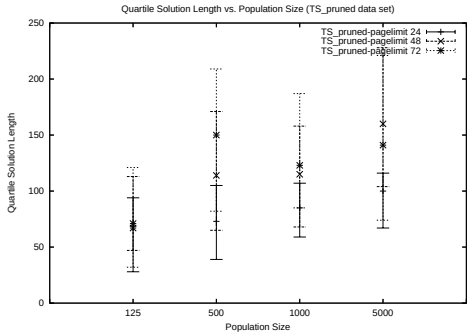


Figure 4.10: Median program length by initial population size and maximum page count for the TS classification problem

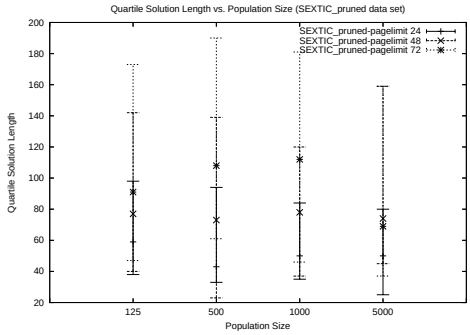


Figure 4.11: Median program length by initial population size and maximum page count for the SEXTIC regression problem

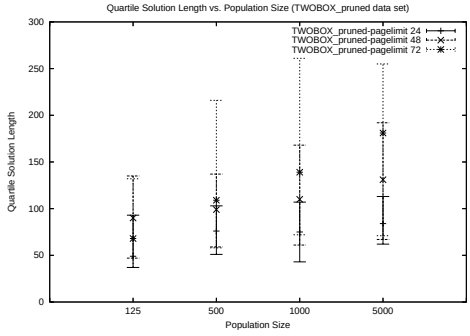


Figure 4.12: Median program length by the initial population size and maximum page count for the TWOBOX regression problem

A.6, on pages 48 through 53 (in Appendix A), display the results we recorded, with bezier curve approximation. Observing these diagrams, we can see that, for all problem datasets, the minimum fitness tends toward zero over 50000 tournaments. In the case of the SEXTIC problem, increasing the maximum page count leads to an increase in the number of tournaments required for convergence. In addition, the smaller population models appear to express more variation in the minimum-average-maximum fitness. Factors influencing this would include the higher likelihood of (poorly performing) children replacing fitter individuals in the population (or replacement error) under a small population model than under a large population model.

4.2.4 Summary

The results in the preceding sections support the following observations:

- experimentally varying the population size and maximum page count produces a negligible difference in quartile performance (in terms of hits); and
- an increase in population size and maximum page count appears to correlate with an increase in program length.

Having observed stable results under the modification of these parameters, we turned our attention to experimentally varying other aspects of the genetic programming model, in particular, our object of interest, the error function.

4.3 Error Functions

Given that our initial parameters for linear genetic programming failed to influence overall fitness and program length, we proceeded to examine the effect of modifying the error function, running 50 trials for each error function on each data set. In our initial analysis we concentrate on the case of global wrapper functions, as introduced

Objective	(Correctly classify exemplars) OR (fit a curve)
Terminal Set	x_1, x_2, x_3, x_4
Functional Set	+,-,*,%, load constant
Fitness Cases	# of exemplars in dataset
Fitness	(Each function from Table 3.1)
Selection	Tournament ($\lambda = 4$) Selection
Hits (Classification)	# of correct classifications
Hits (Regression)	# of cases with abs. error < 0.01
Maximum Page Size	8
Maximum Page Count	24
Init. Population Size	125
Termination	50000 tournaments or 100% hits
Experiments	50 independent runs

Table 4.3: Parameters for the error function experiments

in Section 3.1, Table 3.1. The comparison between local and global wrappers follows in Section 4.4. Each of these functions was tested with the parameters described in Table 4.3.

For this experiment, we used the results of the previous experiment as a guideline. To minimise execution time, we fixed the initial population size and the maximum page count at values of 125 and 24, respectively. The varied quantity was the fitness function used during evaluation of programs. From the recorded observations, we extracted the median hits, median program length, median distances between class means and raw GP output.

The error function is the only direct feedback available to genetic programming regarding the solution space which it explores. Varying the descriptiveness of information supporting this function should affect the quality and quantity of feedback afforded to the GP search process.

Because each function quantifies error differently, it is impossible to directly compare their error values. Thus, we included the hits-based metric as a baseline for error and, post-training, applied a switching-type wrapper to the remaining functions to obtain a comparable count of the hits.

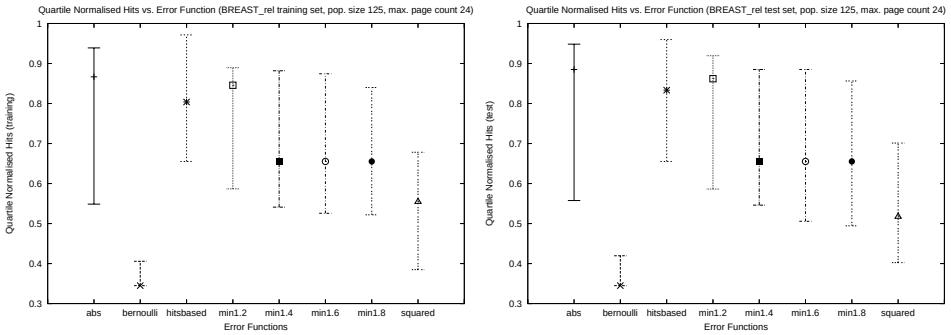


Figure 4.13: Median training and test percentage hits for the BREAST classification problem with a maximum page count of 24 and initial population size of 125.

4.3.1 Median Hits

The median training and test hits were recorded for each of the six datasets and are shown in Figures 4.13 to 4.18. As such, better solutions are associated with a larger hits count.

For classification problems, the distinction most apparent is that between all error functions and the Bernoulli error formulation. This error metric is outperformed by the majority of other functions on the BREAST dataset - the easiest of the problems. We omitted this function in further experiments. We next notice that the hits type metric actually performed respectably across all classification problems considered, with the absolute error metric also returning consistently good results. All other wrapper-metric combinations did not perform as well. On specific data sets, the other metrics were able to return results with less variance, however, there is little consistency to this property across different data sets.

For the regression problems, we can also ignore the Bernoulli function as its definition only holds in the case of classification problems. For both the regression datasets (SEXTIC and TWOBOX), the most reliable results are seen in the spectrum of Minkowski error functions between the absolute error function and the squared function. For the SEXTIC problem, we see that the absolute error function (and the similar low end of the Minkowski spectrum of functions) produces poor performance.

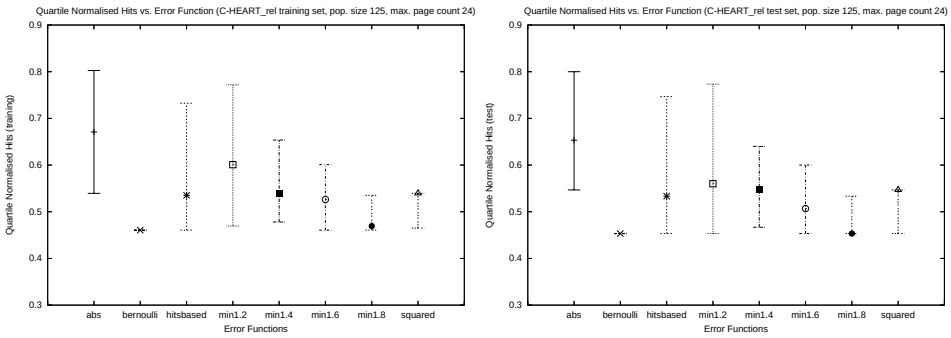


Figure 4.14: Median training and test hits for the C-HEART classification problem with a maximum page count of 24 and initial population size of 125.

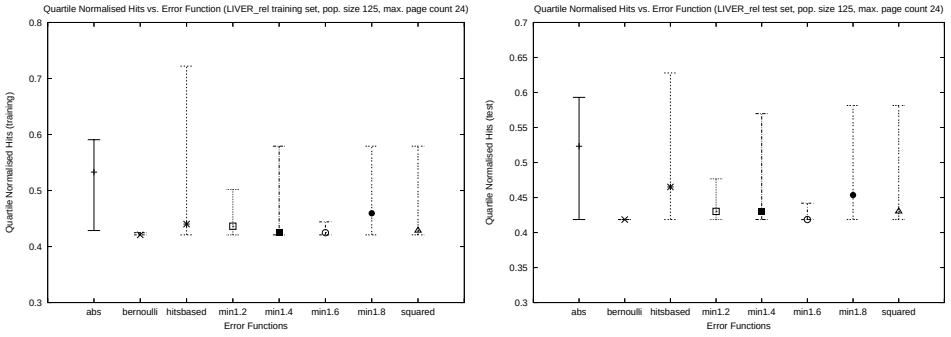


Figure 4.15: Median training and test hits for the LIVER classification problem with a maximum page count of 24 and initial population size of 125.

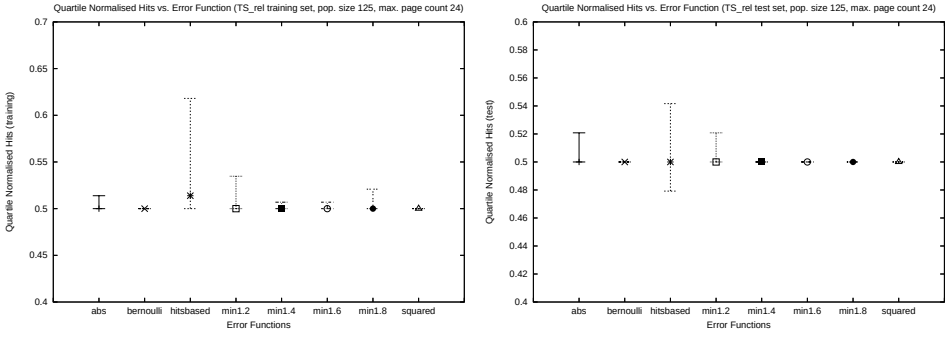


Figure 4.16: Median training and test hits for the TS classification problem with a maximum page count of 24 and initial population size of 125.

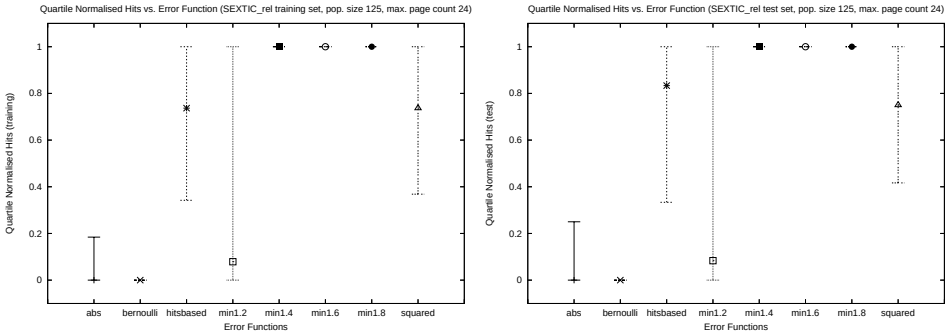


Figure 4.17: Median training and test hits for the SEXTIC regression problem with a maximum page count of 24 and initial population size of 125.

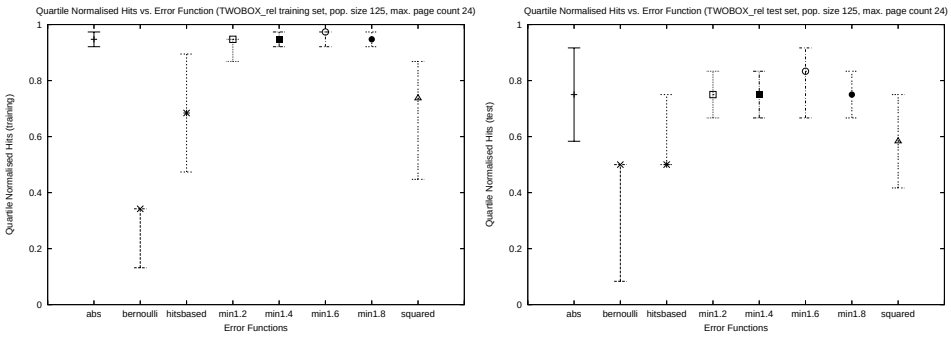


Figure 4.18: Median training and test hits for the TWOBOX regression problem with a maximum page count of 24 and initial population size of 125.

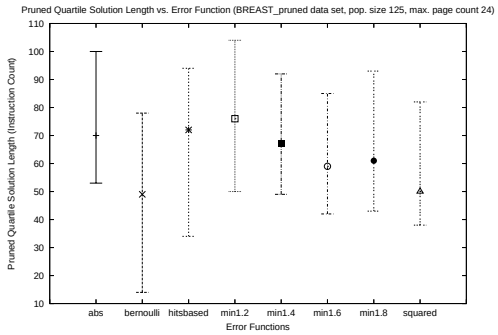


Figure 4.19: Quartile pruned program length for the best individuals on the BREAST classification problem with a maximum page count of 24 and initial population size of 125.

4.3.2 Median Program Length

We collected the quartile pruned program lengths for the best-of-run programs, Figures 4.19 to 4.24. For the regression problems we considered (SEXTIC and TWOBOX), the hits-based and squared error metrics produced programs with significantly greater lengths (as measured at the 95% confidence interval using a standardised student T-test). For classification problems, the hits and square error metrics again resulted in more complex individuals, but was no longer necessarily significant at the 95% confidence interval. (e.g. BREAST returned similar complexity across all metrics whereas TS again established hits and square error as the most complex solutions.) However, we do note that as problem difficulty increases, the hits and square error metrics tend to result in more complex solutions relative to the other metrics considered.

4.3.3 Summary

The results in the preceding sections support the following observations:

- global error functions are mostly indistinguishable in terms of hits-based performance;
- most global error functions support generalisation to the test set, but show poor robustness in that they do so unreliably; and

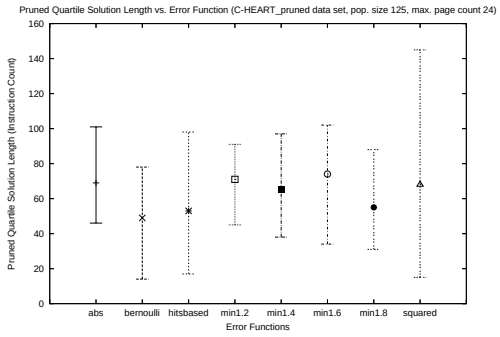


Figure 4.20: Quartile pruned program length for the best individuals for the C-HEART classification problem with a maximum page count of 24 and initial population size of 125.

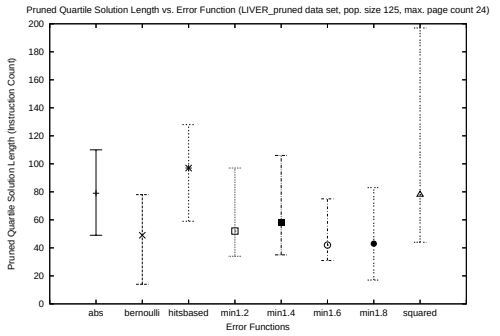


Figure 4.21: Quartile pruned program length for the LIVER classification problem with a maximum page count of 24 and initial population size of 125.

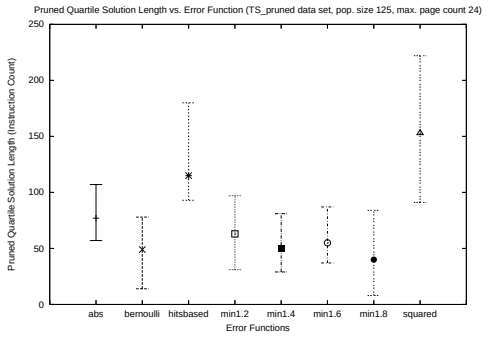


Figure 4.22: Quartile pruned program length for the TS classification problem with a maximum page count of 24 and initial population size of 125.

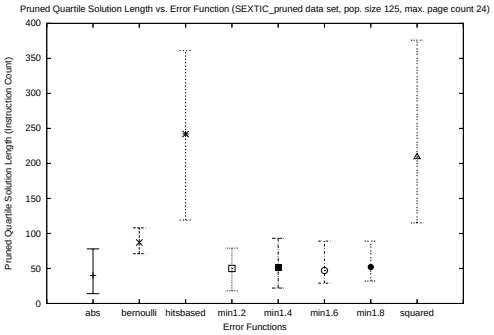


Figure 4.23: Quartile pruned program length for the SEXTIC regression problem with a maximum page count of 24 and initial population size of 125.

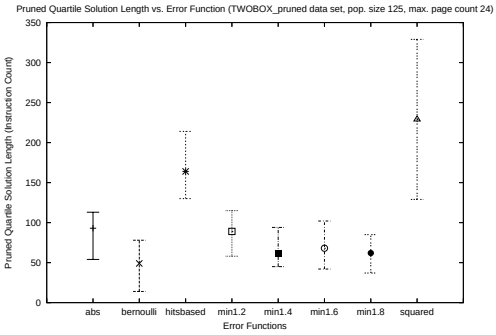


Figure 4.24: Quartile pruned program length for the TWOBOX regression problem with a maximum page count of 24 and initial population size of 125.

- in terms of program length, the hits-based and sum squared error metrics produce the most complex individuals.

In this result, we observe that, while global error functions demonstrate potential for high performance across many initialisations, high performance repeats only intermittently in the sample. In the next experiment, we examined localised methods in an effort to promote more reliable discovery of solutions.

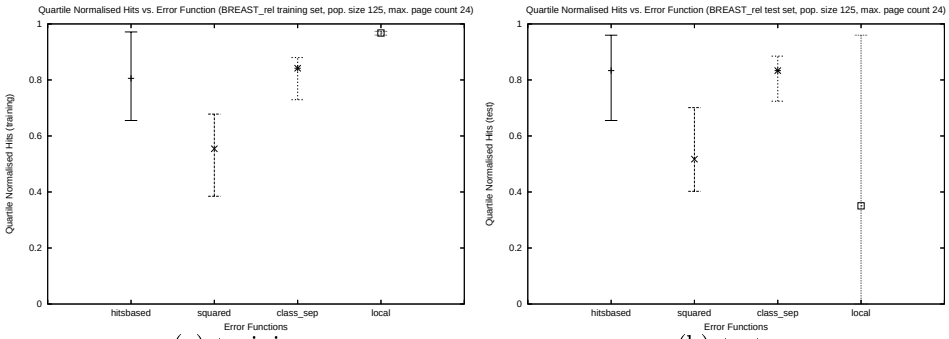
4.4 Clustering and Local Wrapper for GP Evaluation Metrics

In this section, we introduce the proposed class separation distance metric and a local wrapper error metric. We used the hits-based and square error metrics as a baseline.

To produce comparable hits totals for the latter three methods, we had to express each metric's output in terms of hits (post training) through unique methods. For the squared error metric, we used the switching-type wrapper post-training, as in previous examples.

For the class separation distance metric, during the test phase, we determined class membership using a nearest neighbour strategy. For each exemplar, we determined raw GP output, then found the nearest neighbouring class mean (on the GP output axis) and assigned this exemplar to the class corresponding to that mean. The class means are obtained during the training phase.

For the local wrapper case, we establish the mean and variance for both classes during training and use these during testing to select class membership according to a maximum membership strategy. That is, unseen exemplars are classified according to the Gaussian function which maximises their membership, where the Gaussian in question is generated from the mean and variance associated with each respective class during the training phase. This is the assignment rule described in Section 3.2.2.

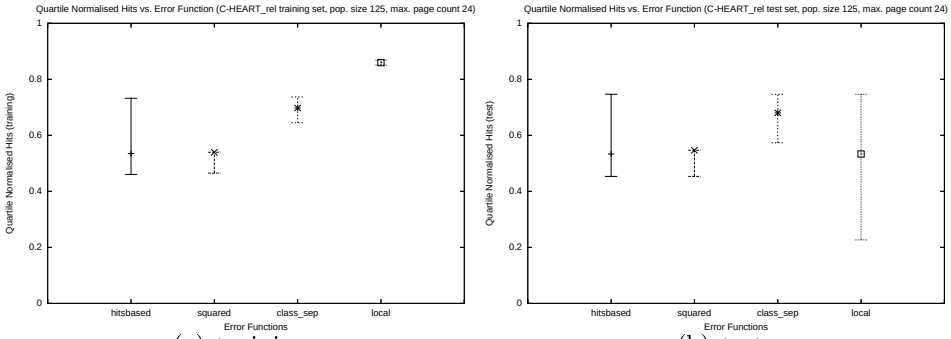


(a) training (b) test

Figure 4.25: Quartile training and test hits for the BREAST dataset with global and local wrapper functions

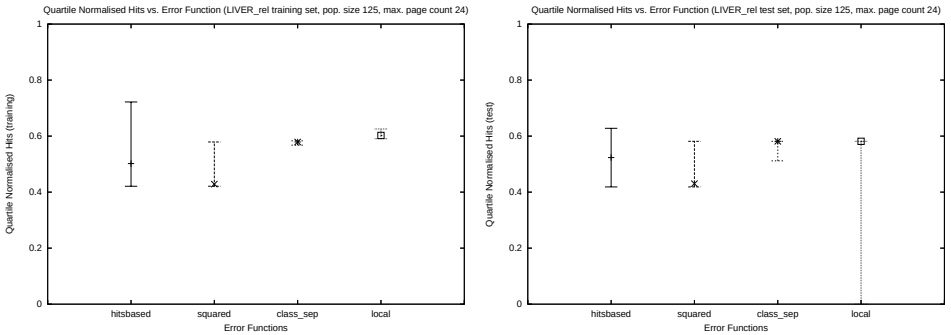
4.4.1 Raw Hits for the Local Wrapper Method on Raw GP Output

In examining the quartile hits on the classification datasets, shown in Figures 4.25 through 4.28, we observe that the variance in the case of classifiers trained using a hits metric and squared error metric is significantly higher than that for the cluster separation error function. We also see that the consistency of performance across all four classification datasets for the class separation distance metric is particularly good, whereas the local wrapper metric appears to overspecialise on training data on specific datasets. That is to say, the variance on training data is very low whereas that on test data is significantly higher. With respect to the squared error metric, a lower variance in classifier performance was established relative to the hits based metric, indicating that the additional feedback provided by this metric was useful, although not necessarily sufficient to provide better median performance. The hits based wrapper tends to return the most variation in classifier behaviour, a characteristic that frequently provides the best single classifier over a set of initialisations (encourages a lot of exploration during training) but at the expense of repeatability in the search process. Conversely, the cluster separation wrapper appears to provide the most consistent classifier behaviour, as indicated by the high median and tight error bars, irrespective of dataset.



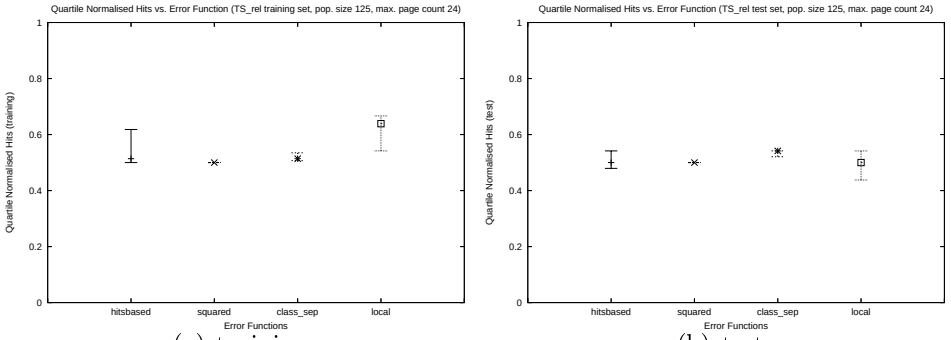
(a) training (b) test

Figure 4.26: Quartile training and test hits for the C-HEART dataset with global and local error wrapper functions



(a) training (b) test

Figure 4.27: Quartile training and test hits for the LIVER dataset with global and local error wrapper functions



(a) training (b) test

Figure 4.28: Quartile training and test hits for the TS dataset with global and local error wrapper functions

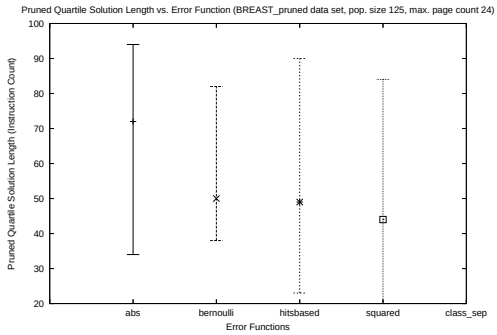


Figure 4.29: Quartile pruned program lengths for the BREAST dataset with four classification metrics based on raw GP output

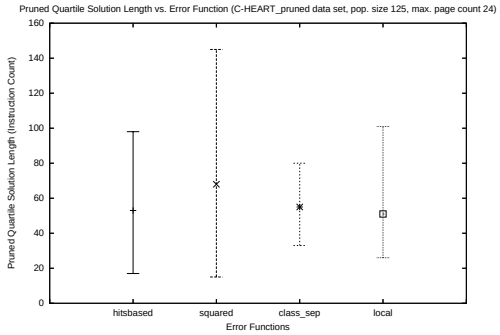


Figure 4.30: Quartile pruned program lengths for the C-HEART dataset with four classification metrics based on raw GP output

4.4.2 Pruned Program Lengths for Raw GP Methods

Figures 4.29 through 4.32 summarise quartile pruned program length, following the removal of useless (ie intron) instructions. On the simpler problems of BREAST and C-HEART, similar solution complexities are returned. In the case of the more challenging problems (Liver and Two Spirals), the cluster separation and local wrapper metrics returned significantly shorter programs, where Figure 4.27 indicates that no penalty has come to the classification performance.

4.4.3 Consequences

The results in the preceding sections support the following observations:

- the hits-based wrapper generalises to the test set but cannot be called a robust

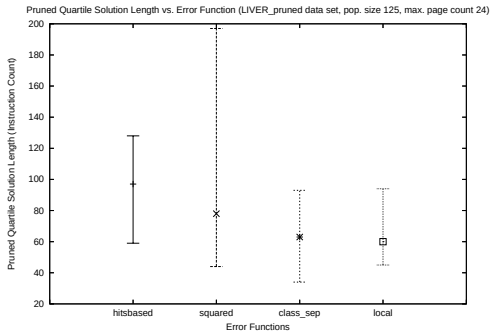
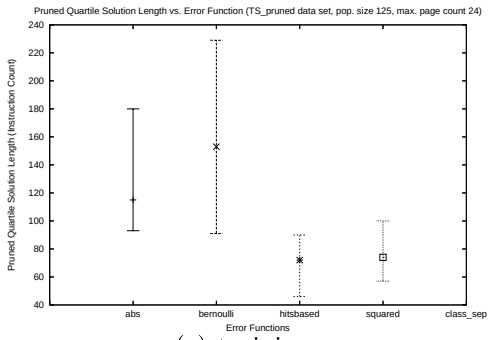


Figure 4.31: Quartile pruned program lengths for the LIVER dataset with four classification metrics based on raw GP output



(a) training

Figure 4.32: Quartile pruned program lengths for the TS dataset with four classification metrics based on raw GP output

solution, in that it appears to suffer from poor repeatability (that is, it is dependent on the initialisation and suffers wide variance in quartile performance);

- the local wrapper method repeatably produces good individuals on the training set but only partially generalises to the test set (nevertheless, the upper quartile and median nearly overlap);

- the class separation distance metric combines both good generalisation and robust (repeatable) performance.

Although the local wrapper method failed to produce clearly robust solutions for the problems we considered, the class separation distance metric provided both general and robust behaviour consistently across all data sets.

Chapter 5

Conclusion

Within a (linear) genetic programming framework, we examined the influence of initial population parameters on the outcome of genetic programming models on selected datasets representing classification and regression problems.

Our results indicate that initial parameters do not significantly influence the fitness of resulting solutions within the context of a fixed length representation. Population size does appear to be correlated with solution complexity. This result encourages investigation of the effects of functional parameters, motivating our study of error functions in genetic programming. That is to say, a better method for influencing solution quality is through the definition of the fitness function than selection of population parameters.

We examined the role of wrapper-based error functions in GP fitness. In this case, we observed that the hits-based metric performed most consistently across classification problems, followed by the absolute error metric. For regression problems, we found that the Minkowski series of error functions provided consistent performance in comparison with other functions. We question whether the switching-type wrapper function obscured information content for more complex functions including absolute error and squared error.

Our next trials investigated the properties of error metrics explicitly designed to encourage a robust mapping on the raw GP output axis, specifically a cluster separation metric and a local wrapper function was introduced. Our class separation metric was intended to emphasise the difference between two clusters in the raw GP

output. We found that this metric returned the most consistent results across the set of classification problems evaluated.

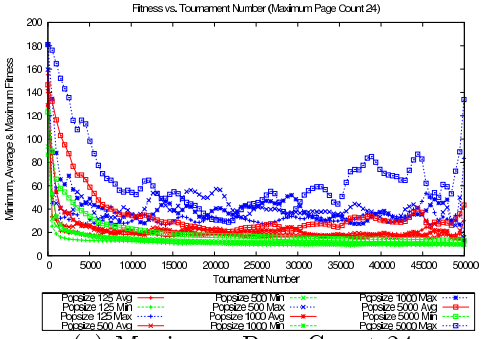
In terms of future work, we believe that the reformulation of the classification problem as a clustering problem will provide the basis for one class classifiers and the use of multi-objective optimisation techniques for problem decomposition. In essence, by focusing on the behaviour of the raw GP output values, the classification problem takes the form of a clustering problem. When this is the case, methods from multiobjective optimisation are applicable, thus providing for problem decomposition and multi-model solutions [16].

Moreover, one class classification may now be appropriate as the task is now defined in terms of establishing a mapping from input to output spaces (over exemplars from one class) in which the objective is expressed in terms of establishing raw GP output values with a specific local distribution. (e.g. a Gaussian distribution).

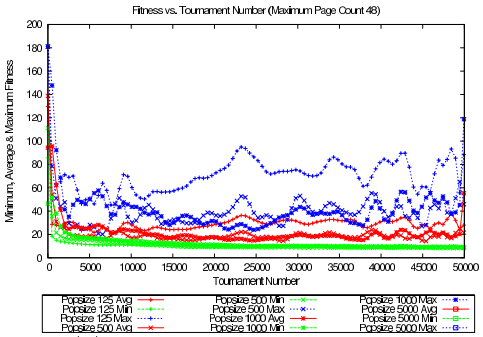
Finally, we note that the local wrapper approach may lead to a Bayes model formulation of the classification problem. Such a scheme is not applicable to the current local wrapper model (e.g. as implemented using a post-processing technique) as the underlying assumptions for the two models are inherently different.

Appendix A

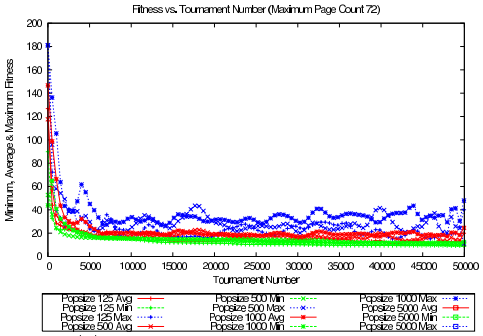
Tournament Fitness



(a) Maximum Page Count 24

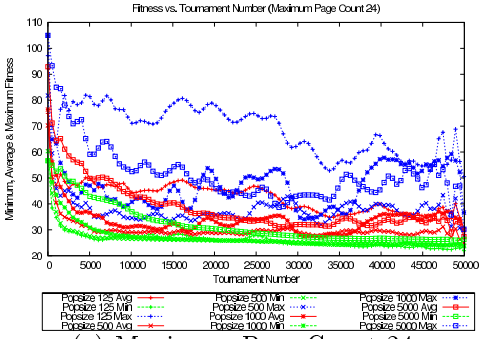


(b) Maximum Page Count 48

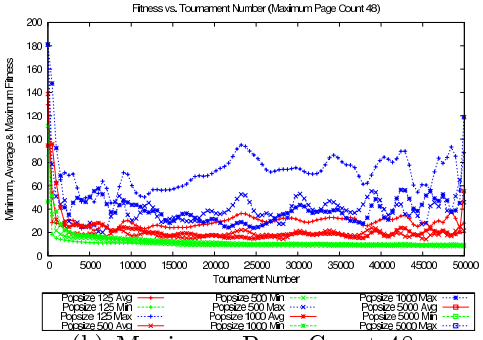


(c) Maximum Page Count 72

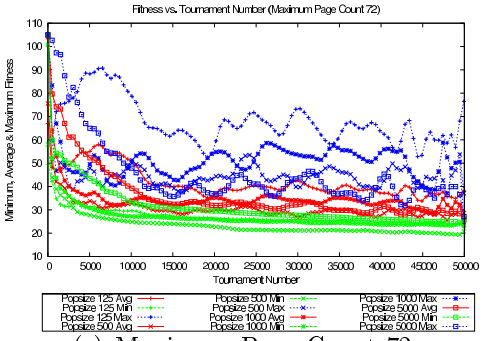
Figure A.1: Median bezier smoothed minimum, median average and median maximum fitness at each tournament for the BREAST classification problem



(a) Maximum Page Count 24

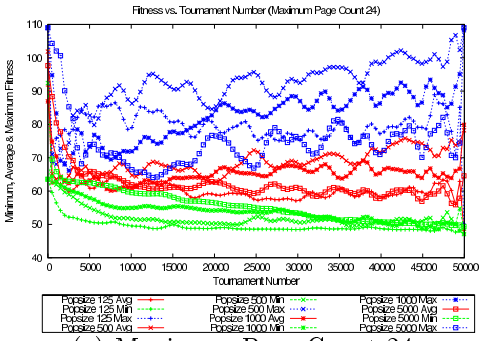


(b) Maximum Page Count 48

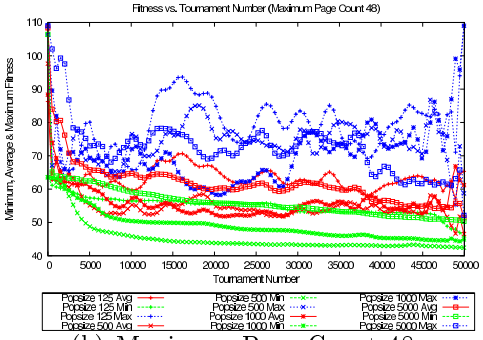


(c) Maximum Page Count 72

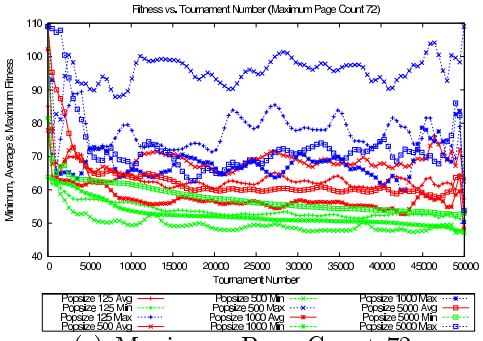
Figure A.2: Median minimum, median average and median maximum fitness at each tournament for the C-HEART classification problem



(a) Maximum Page Count 24

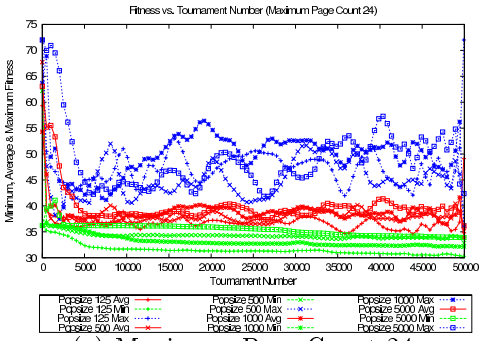


(b) Maximum Page Count 48

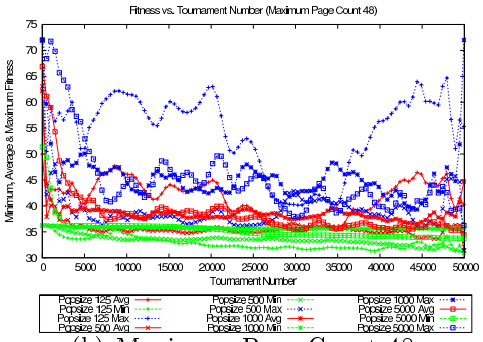


(c) Maximum Page Count 72

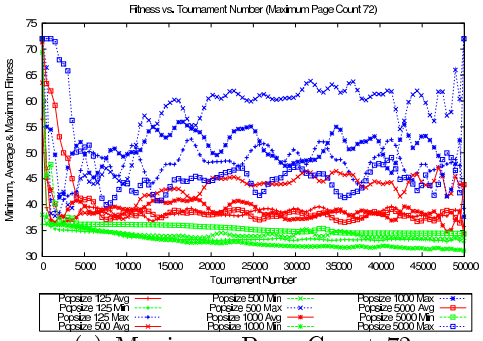
Figure A.3: Median minimum, median average and median maximum fitness at each tournament for the LIVER classification problem



(a) Maximum Page Count 24

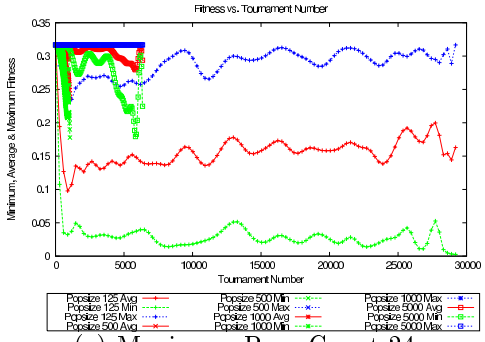


(b) Maximum Page Count 48

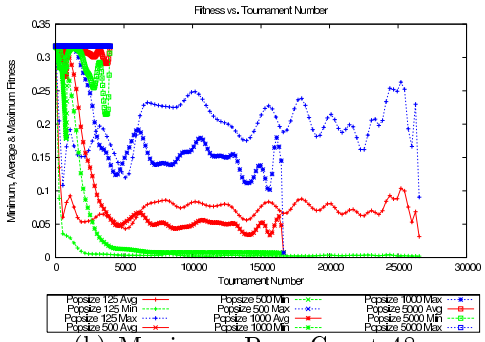


(c) Maximum Page Count 72

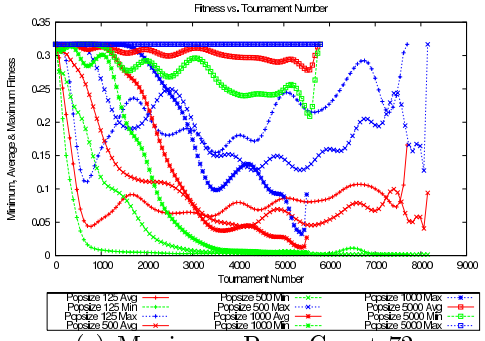
Figure A.4: Median minimum, median average and median maximum fitness at each tournament for the TS classification problem



(a) Maximum Page Count 24

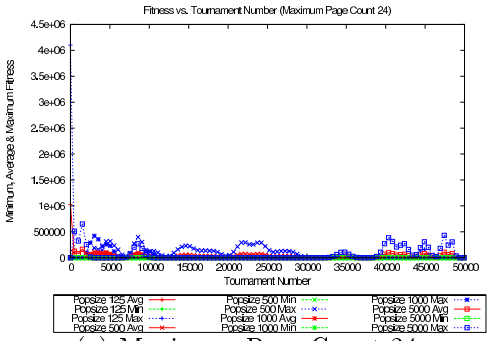


(b) Maximum Page Count 48

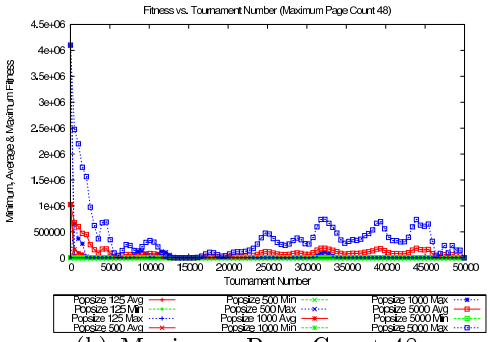


(c) Maximum Page Count 72

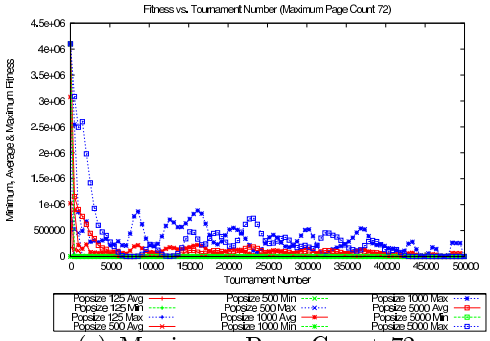
Figure A.5: Median minimum, median average and median maximum fitness at each tournament for the SEXTIC regression problem



(a) Maximum Page Count 24



(b) Maximum Page Count 48



(c) Maximum Page Count 72

Figure A.6: Median minimum, median average and median maximum fitness at each tournament for the TWOBOX regression problem

Bibliography

[1] Brameier, M., and Banzhaf, W. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation 5*, 1 (Feb. 2001), 17–26.

[2] Castleman, K. R. *Digital Image Processing.* Prentice Hall, Englewood Cliffs, NJ, USA, 1996.

[3] Darwin, C. *The Origin of Species.* Modern Library, New York, 1998.

[4] Eggermont, J., Eiben, A. E., and van Hemert, J. I. Adapting the fitness function in GP for data mining. In *Genetic Programming, Proceedings of EuroGP'99* (Goteborg, Sweden, 26-27 May 1999), R. Poli, P. Nordin, W. B. Langdon, and T. C. Fogarty, Eds., vol. 1598 of *LNCS*, Springer-Verlag, pp. 193–202.

[5] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing.* Springer, 2003.

[6] Goldberg, D. E. Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Orlando, Florida, USA, 13-17 July 1999), W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 1, Morgan Kaufmann, pp. 212–219.

[7] Haykin, S. *Neural Networks. A Comprehensive Foundation.* Prentice Hall, New Jersey, USA, 1999.

[8] Heywood, M. I., and Zincir-Heywood, A. N. Dynamic page based crossover in linear genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics 32*, 3 (June 2002), 380–388.

[9] Keijzer, M., and Babovic, V. Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In *Genetic Programming, Proceedings of EuroGP'2000* (Edinburgh, 15-16 Apr. 2000), R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, Eds., vol. 1802 of *LNCS*, Springer-Verlag, pp. 76–90.

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

[11] Koza, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, May 1994.

[12] Kushchu, I. Genetic programming and evolutionary generalization. *IEEE Transactions on Evolutionary Computation 6* (Oct. 2002), 431–442.

[13] Lichodzijewski, P., Zincir-Heywood, N., and Heywood, M. Cascaded GP models for data mining. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation* (Portland, Oregon, 20-23 June 2004), IEEE Press, pp. 2258–2264.

[14] Lim, T., Loh, W., and Shih, Y. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning 40* (2000), 203–228.

[15] Loveard, T., and Ciesielski, V. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation* (COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001), vol. 2, IEEE Press, pp. 1070–1077.

[16] McIntyre, A. R., and Heywood, M. I. Multiobjective genetic programming: Genetic programming problem decomposition using multiobjective optimization. *Submitted to Genetic and Evolutionary Computation Conference (GECCO)* (2006).

[17] Mitchell, T. M. *Machine Learning*. McGraw-Hill, 1997.

[18] Song, D., Heywood, M. I., and Zincir-Heywood, A. N. A linear genetic programming approach to intrusion detection. In *Genetic and Evolutionary Computation – GECCO-2003* (Chicago, 12-16 July 2003), E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, Eds., vol. 2724 of *LNCS*, Springer-Verlag, pp. 2325–2336.

[19] Trappenberg, T. P. *Fundamentals of Computational Neuroscience*. Oxford University Press, Great Clarendon Street, Oxford, 2002.

[20] Werbos, P. J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.