Improving the Human Readability of Features Constructed by Genetic Programming

Matthew Smith
Faculty of Computing, Engineering & Mathematical Sciences,
University of the West of England,
Bristol BS16 1QY, U.K.
matthew.smith@thinkinglogic.com

Larry Bull
Faculty of Computing, Engineering & Mathematical Sciences,
University of the West of England,
Bristol BS16 1QY, U.K.
larry.bull@uwe.ac.uk

ABSTRACT
The use of machine learning techniques to automatically analyse data for information is becoming increasingly widespread. In this paper we examine the use of Genetic Programming and a Genetic Algorithm to pre-process data before it is classified by an external classifier. Genetic Programming is combined with a Genetic Algorithm to construct and select new features from those available in the data, a potentially significant process for data mining since it gives consideration to hidden relationships between features. We then examine techniques to improve the human readability of these new features and extract more information about the domain.

Categories and Subject Descriptors
I.2.2 [**Artificial Intelligence**]: Automatic Programming – *Automatic analysis of algorithms, Program modification, Program synthesis, Program transformation, Program verification.*

General Terms
Algorithms, Performance, Experimentation, Human Factors.

Keywords
Genetic programming, genetic algorithm, feature construction, feature selection, classification, human readability, parsimony, post-processing, knowledge discovery.

1. INTRODUCTION
Classification is one of the major tasks in data mining, involving the prediction of class value based on information about the other attributes. In this paper we examine the combination of Genetic Programming (GP) [12] and a Genetic Algorithm (GA) [8] to improve the performance of a classification algorithm through feature *construction* and feature *selection*. Feature construction is a process that aims to discover hidden relationships between features, inferring new composite features. In contrast, feature selection is a process that aims to refine the list of features used thereby removing potential sources of noise and ambiguity. We use GP individuals consisting of a number of separate trees/automatically defined functions (ADFs) [12] to construct features. A GA is simultaneously used to select over the new set of constructed features. Previous results [21] have shown that the system is able to construct a new set of features that improve the individual performance of 3 classification algorithms: C4.5 [17]; IBk, a k-nearest neighbour classifier [1] with k=1; and Naïve Bayes, a probability based classifier [9] on a number of datasets held at the UCI repository (http://www.ics.uci.edu /~mlearn/MLRepository.html). The system is also able to select the most appropriate classifier for the dataset at hand. We then show how using a parsimony measure based on that of Bojarczuk et. al. [4] can reduce the size and number of GP trees to improve human readability without appearing to significantly harm the performance of the algorithm. The parsimony measure is then compared with a method of post-processing, similar to reduced error pruning [15], to improve human readability.

Raymer et al. [18] have used ADFs for feature *extraction* in conjunction with the k-nearest-neighbour algorithm. Feature extraction replaces an original feature with the result from passing it through a functional mapping. In Raymer et al.'s approach each feature is altered by an ADF, evolved for that feature only, with the aim of increasing the separation of pattern classes in the feature space; for problems with *n* features, individuals consist of *n* ADFs. Ahluwalia and Bull [2] extended Raymer et al.'s approach by coevolving the ADFs for each feature and adding an extra coevolving GA population of feature selectors; extraction and selection occurred simultaneously in *n*+1 populations for use by a k-nearest-neighbour algorithm. For other (early) examples of evolutionary computation approaches to data mining see [19] for a GA-based feature selection approach using k-nearest-neighbour and [10] for a similar GA-based approach also using k-nearest-neighbour.

Vafaie and DeJong [23] demonstrated a combination of GP and a GA for use with C4.5. They used the GA to perform feature selection for a face recognition dataset where feature subsets were evaluated through their use by C4.5. GP individuals were then evolved which contained a variable number of ADFs to construct new features from the selected subset, again using C4.5. Our approach is similar to Vafaie and DeJong's but feature construction and selection are combined within a single stage instead of starting with feature selection and then alternating between construction and selection. We find that our approach performs as well or better than Vafaie and DeJong's [21]. Krawiec [13] has also presented a similar approach.

More recent work using GP to construct features for use by C4.5 includes that of Otero et al. [16]. They use a population of GP trees to evolve a single new feature using information gain as the fitness measure (this is the criteria used by C4.5 to select attributes to test at each node of the decision tree). This produces a single feature that attempts to cover as many instances as possible – a feature that aims to be generally useful and which is appended to the set of original features for use by C4.5. Ekárt and Márkus [6] use GP to evolve new features that are useful at specific points in the decision tree by working interactively with C4.5. They do this by invoking a GP algorithm when constructing a new node in the decision tree – e.g., when a leaf node incorrectly classifies some instances. Information gain is again used as the fitness criterion but the GP is trained only on those instances relevant at that node of the tree.

While Naïve Bayes is often used as a bench mark comparison for GP algorithms relatively little work seems to have been done to combine GP with Naïve Bayes. Langdon and Buxton [14] have combined GP and Naïve Bayes by using the output of Naïve Bayes (along with a number of other classification algorithms) as an input to GP, but not used GP to improve the performance of a Naïve Bayes algorithm.

One of the advantages of Genetic Programming is that it has the potential to produce solutions that are amenable to human readability. The results can be made easier to read by reducing their complexity, either through post-processing (removing redundant sub-trees) or during their evolution by introducing a fitness function that encourages simplicity. Simpler solutions can also reduce the well known problem of over-fitting. Much of the work on enforcing simplicity in GP has focused on controlling bloat in decision trees: De Jong et al. [5] and Bernstein et al. [3] have used multi-objective techniques to control bloat, while Thomas and Sycara [22] have enforced simplicity by controlling the maximum depth of GP trees for data mining in financial data. Bojarczuk et al. [4] have applied a fitness function that combines predictive accuracy and simplicity to medical datasets, with the primary intention of improving human readability. Most theories that explain bloat in decision trees rest on the fact "that it is easier to add code to a program than it is to remove it. That is, it is quite difficult to remove code from an effectively functioning program without heavily impacting on that functionality and thus reducing its fitness" [3]. As this does not apply in the same way to constructed features (a sub-tree can be removed from a single feature without significantly reducing overall fitness) bloat does not pose such a problem, and instead we concentrate here on improving readability.

This paper is arranged as follows: the next section describes the algorithm; section 3 presents results from its use on a number of well-known datasets and discusses the results. Two measures are then introduced to improve human readability in section 4; in section 5 the algorithm is applied to a dataset used for computer aided vision, followed by some conclusions and future directions in section 6.

2. THE GAP ALGORITHM

In this work we have used the WEKA [24] data mining toolset to examine the performance of our Genetic Algorithm and Programming (GAP) approach [21]. This is a wrapper approach [11], in which the fitness of individuals is evaluated by performing 10-fold cross validation using the same inducer as used to create the final classifier. The algorithm is capable of operating with any WEKA-compatible classifier, in this paper we will concentrate on its use with the WEKA implementation of C4.5 (known as J48), k-nearest neighbour (IBk) and Naïve Bayes. Our algorithm is now briefly described (see [21] for a more detailed description and some analysis of the feature sets created).

A population of 101 genotypes is created at random. Each genotype consists of *n* trees, where *n* is the number of numeric valued features in the dataset, subject to a minimum of 7. This minimum is chosen to ensure that, for datasets with a small number of numeric features, the initial population contains a large number of compound features. A tree can be either an original feature or an ADF. That is, a genotype consists of *n* GP trees, each of which may contain 1 or more nodes. The chance of a node being a leaf node (a primitive attribute) is determined by:

$$P_{leaf} = 1 - \frac{1}{(depth + 1)} \tag{1}$$

Where *depth* is the depth of the tree at the current node. Hence a root node will have a depth of 1, and therefore a probability of 0.5 of being a leaf node, and so on. If a node is a leaf, it takes the value of one of the original features chosen at random. Otherwise, a function is randomly chosen from the set {*, /, +, -, %, log} and two child nodes are generated (one child in the case of log). In this manner there is no absolute limit placed on the depth any one tree may reach but the average depth is limited (see [21] for a comparison with the more traditional ramped half and half method).

During the initial creation no two trees in a single genotype are allowed to be alike and no two genotypes are allowed to be the same, though these restrictions are not enforced in later stages. Additionally, nodes with '–', '%' or '/' for functions cannot have child nodes that are equal to each other. In order to enforce this, child nodes within a function '*' or '+' are ordered lexicographically to enable comparison (e.g. [width + length] will become [length + width]). Each tree has an associated activity – a boolean switch set randomly in the initial population that determines if the feature created by the tree will be used. Each genotype also has a flag specifying the classifier it is to use (one of C4.5, IBk, Naïve Bayes).

An individual is evaluated by constructing a new dataset with one feature for each active tree in the genotype. This dataset is then passed to the specified classifier (using default parameters), whose performance on the dataset is evaluated using 10-fold cross validation. The percentage correct is then assigned to the individual and used as the fitness score.

Once the initial population has been evaluated, several generations of selection, crossover, mutation and evaluation are performed. We use tournament selection to select the parents of the next generation, with a tournament size of 6. The classifier flag is copied from parent to child, subject to mutation (changed to a randomly selected, but different, classifier) with a probability of 0.2. There is a 0.6 probability of uniform crossover occurring between the ADFs of the two selected parents (whole trees are exchanged between genotypes). The activity switch remains associated with the tree.

There is an additional 0.6 probability that crossover will occur within two ADFs at a randomly chosen locus (sub-trees are exchanged between trees in the same position in the child genotypes). There is a 0.24 probability per individual of mutating tree structure (in which a single node in one active tree is replaced with a random sub-tree[1]), and an additional 0.24 probability of mutating tree activity (in which the activity bit of a single tree is flipped). We also use a form of inversion with low probability (0.02), whereby the order of the trees between two randomly chosen loci is reversed.

The fittest individual in each generation is copied unchanged to the next generation. The evolutionary process continues until the following conditions are met: at least 20 generations have passed, and the fittest individual so far is at least 12 generations old. At this point, in order to reduce over-fitting the training dataset is randomly reordered, the fitness of the population re-evaluated, and evolution continues until the termination criteria are reached a second time. This provides a different split for the cross-validation and has been found to reduce over-fitting.

This is a lengthy process, as performing 10-fold cross validation for each member of the population is very processor intensive. The extra time required for cross-validation can be justified here by the improvement in the results over using, e.g., a single train and test set (results not shown). Information Gain, the fitness criterion employed by both Otero and Ekárt, is much faster but is only applicable to a single feature – it cannot provide the fitness criterion for a set of features (it is also specific to C4.5, and the GAP algorithm is designed to be classifier-neutral).

3. INITIAL EXPERIMENTATION

3.1 Datasets

We have used ten well-known data sets from the UCI repository to examine the performance of the GAP algorithm. The UCI datasets were chosen because they consisted entirely of numeric attributes (though the algorithm can handle nominal attributes, as long as there are two or more numeric attributes present). Table 1 shows the details of the ten datasets used here.

For performance comparisons the tests were performed using ten-fold cross-validation, and in which 90% of the data was used for training and 10% for testing. An additional set of ten runs using ten-fold cross validation were made (a total of twenty runs - two sets of ten-fold cross-validation) to allow a paired *t*-test to establish the significance of any improvement over the comparison classifier. During each of the 20 runs all 3 unaided classifiers were evaluated against the training data and the most accurate classifier was selected as the comparison for evaluation on the test data.

[1] The new sub-tree is created using equation 1 but using an initial depth of 1, regardless of where in the tree it will be inserted.

Table 1. UCI dataset information

Dataset	**Features**	**Classes**	**Instances**
BUPA Liver Disorder (Liver)	6	2	345
Glass Identification (Glass)	9	6	214
Ionosphere (Iono.)	34	2	351
New Thyroid (NT)	5	3	215
Pima Indians Diabetes (Diab.)	8	2	768
Sonar	60	2	208
Vehicle	18	4	846
Wine Recognition (Wine)	13	3	178
Wisconsin Breast Cancer – New (WBC New)	30	2	569
Wisconsin Breast Cancer – Original (WBC Orig.)	9	2	699

3.2 Results

The highest classification score for each dataset is shown in Table 2 in bold. The first two columns show the performance of the GAP algorithm and comparison on the test data (with standard deviation in brackets), and the last column shows the results of the paired *t*-test comparing the GAP algorithm to the selected comparison classifier. Results that are significant at the 95% confidence level are shown in bold. Table 3 shows the classifier that was selected most often for the dataset, and the number of times it was selected. Table 4 shows the performance of each of the unaided classifiers for comparison purposes (see [20] for details of how the GAP algorithm successfully improves the performance of each of these classifiers individually).

Table 2. Performance of GAP algorithm and comparison

Dataset	**GAP (S.D.)**	**Comparison (S.D.)**	**Paired *t*-test**
Liver	**68.71** (7.98)	64.76 (8.99)	**2.44**
Glass	**73.86** (9.09)	68.78 (11.47)	**2.16**
Iono.	88.08 (6.69)	89.82 (4.79)	-1.10
NT	96.52 (3.63)	96.26 (4.19)	0.32
Diab.	74.55 (5.28)	76.05 (4.87)	-1.12
Sonar	88.30 (7.66)	86.65 (6.56)	0.95
Vehicle	71.93 (4.79)	72.22 (3.33)	-0.20
Wine	96.32 (4.27)	**97.99** (3.85)	**-2.29**
(WBC New)	94.32 (3.71)	95.09 (3.06)	-0.93
(WBC Orig.)	95.70 (2.60)	95.99 (1.84)	-0.74
Overall	84.83	84.36	1.03

The initial results indicate that GAP algorithm performs little better than selecting the most appropriate classifier for the dataset at hand, at a significant increase in cost (time). Part of the lack of improvement may be due to the fact that C4.5 appears more prone to over-fitting than the other classifiers and the fitness (not shown) of C4.5 genotypes often overshadows other, ultimately more accurate, genotypes. For instance on the WBC original dataset C4.5 is selected 10 times, while selecting Naïve Bayes instead would have resulted in a higher test score.

Table 3. Selection of classifiers by dataset

Dataset	Classifier Most Used	count
Liver	C4.5	16
Glass	IBk	20
Iono.	C4.5	15
NT	NaiveBayes	11
Diab.	NaiveBayes	14
Sonar	IBk	20
Vehicle	C4.5	12
Wine	NaiveBayes	11
(WBC New)	IBk	9
(WBC Orig.)	C4.5	10

Table 4. Performance of unaided classifiers

Dataset	C4.5(J48)	IBk	Naïve Bayes
Liver	66.37	62.62	54.19
Glass	68.28	68.79	48.50
Iono.	89.82	86.95	82.37
NT	92.31	96.95	97.20
Diab.	73.32	69.90	75.13
Sonar	73.86	86.65	67.16
Vehicle	72.22	70.03	43.98
Wine	93.27	95.44	97.99
(WBC New)	93.88	95.44	93.26
(WBC Orig.)	94.42	95.42	96.06
Overall	81.77	82.82	75.58

However there is still scope for successful use of the GAP algorithm. Being probabilistic in nature the GAP algorithm will produce a different set of features every time it is run – some of which will be better than others. The results have been obtained from 20 runs per dataset, every run providing a distinct set of features and the score is an average over the 20 feature sets (or feature/classifier combinations) - each set tested on a single data fold. In a practical situation we are not interested in the average utility of a number of feature sets, but in using the best available feature set.

The best of the feature sets tends to show a marked improvement over the average, and over the performance of an unaided classifier. In section 5 we will be able to demonstrate this improvement on a larger dataset in which a third of the data has been held back for this purpose.

4. IMPROVING HUMAN READABILITY

4.1 Applying a Parsimony measure

The parsimony measure presented here is modelled on the fitness function used by Bojarczuk et al. [04]. They applied a fitness function that combines predictive accuracy and simplicity to a GP system which operates on medical datasets with a binary classification (in which a person either has a specific condition or they do not). Their fitness function is defined as follows:

$$Fitness = Sensitivity * Specificity * Simplicity \tag{2}$$

Where *Sensitivity* measures the rate of true positives against false negatives, and *Specificity* measures the rate of true negatives against false positives. As our algorithm is designed to work with datasets having multiple classifications, sensitivity and specificity are not appropriate, so these are replaced with the existing fitness measure: predictive accuracy – the proportion of instances correctly identified during 10-fold cross validation.

Bojarczuk et al.'s measure of simplicity is defined as:

$$Simplicity = \frac{(maxnodes - 0.5 * numnodes - 0.5)}{(maxnodes - 1)} \tag{3}$$

"Where *numnodes* is the current number of nodes (functions and terminals) of an individual (tree), and *maxnodes* is the maximum allowed size of a tree (set to 45)." [04] For our purposes we take *numnodes* to be defined as the total number of nodes in all active trees in an individual. Because in our algorithm there is no maximum size for a GP tree *maxnodes* is defined, somewhat arbitrarily, as twice the number of nodes in the largest individual in the initial population.

If we modify the fitness function of the GAP algorithm to be:

$$Fitness = PredictiveAccuracy * Simplicity \tag{4}$$

With simplicity is defined as in equation 3, simplicity quickly overwhelms predictive accuracy. Before the simplicity measure is applied the average number of nodes in the fittest individual is approx. 23 nodes in 9 trees; with the simplicity measure applied it drops to roughly 7 nodes in 6 trees, i.e., feature construction is almost completely eliminated. The overall performance also drops, from 84.3% to 83.3%. This is because Bojarczuk et al.'s measure is designed to reduce the complexity of a GP classifier, not reduce the complexity of features passed to a classifier. A GP classifier with just a single node is doomed to failure, while, say, C4.5 may still perform well given just one feature (at least to the extent that the loss in accuracy is less than the gain from simplicity).

It is possible to amend Bojarczuk et al.'s measure of simplicity to include a *strength* parameter:

$$Simplicity = \frac{(maxnodes - strength * numnodes - (1 - strength))}{(maxnodes - 1)} \tag{5}$$

If *strength* is 0.5 then the simplicity measure is the same as Bojarczuk et al.'s, if *strength* is reduced to 0 simplicity is eliminated from the fitness function. Figures 1 and 2 illustrate the effect of varying values of the strength parameter. Initially strength values in the series 0, 0.05, 0.1, 0.15,…0.45 were tried (not shown), then the range was narrowed down to 0, 0.01, 0.02,…0.1. All values are averages over all ten datasets listed in table 1, with 20 runs per dataset.

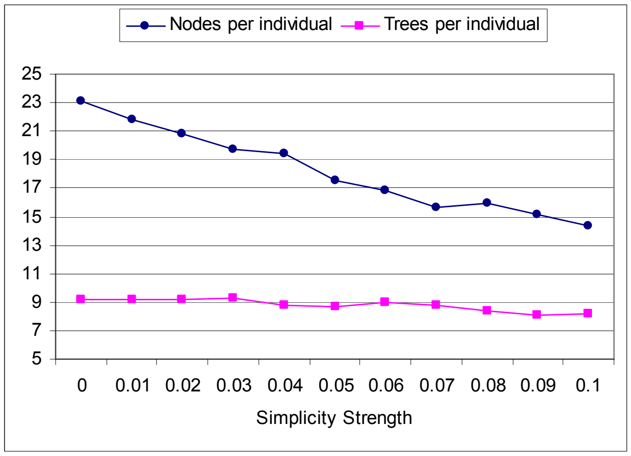

Figure 1. No. Nodes and Trees vs. Simplicity Strength

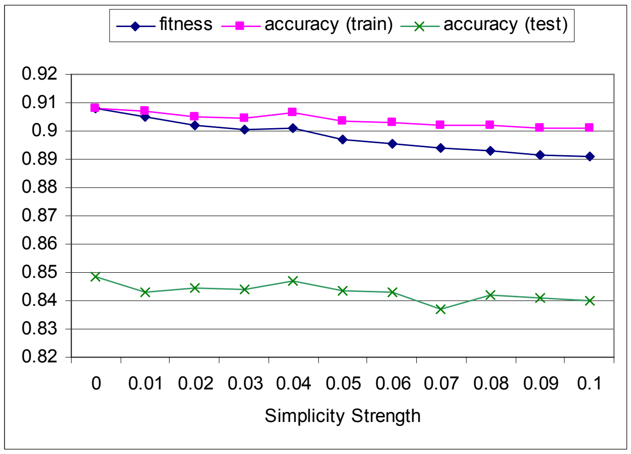

Figure 2. Fitness and Accuracy vs. Simplicity Strength

A value of 0.04 was finally chosen for the simplicity strength parameter (i.e., the simplicity of a genotype accounts for 4% of its fitness). 0.04 was chosen because it successfully simplifies the individuals (reducing the average number of nodes per tree from roughly 2.5 to just over 2) while allowing feature construction to continue playing a meaningful part in the algorithm. It also occupies a slight hump in the graph of predictive accuracy for both train and test scores (Figure 2).

Table 5 shows the results for the algorithm including parsimony on the UCI datasets. The figures are very similar to those without the parsimony measure. Results for the Liver dataset are no longer significant but otherwise there is no significant change in the performance, while the average number of nodes in the output has been reduced by 16%. However this average hides some variation between datasets. More nodes are removed on datasets with many attributes (and correspondingly many possible trees) than on those with few attributes. The accuracy on the test set remains almost exactly the same when setting simplicity strength to 0.04 (as compared to no parsimony pressure), while the average standard deviation drops slightly from 5.6% to 5.0%. The average number of generations drops by just under 11%, from 72.5 to 64.7 generations.

Table 5. Performance with parsimony

Dataset	GAP (S.D.)	Comparison (S.D.)	Paired *t*-test
Liver	66.64 (9.77)	64.76 (8.99)	0.80
Glass	**73.46** (8.50)	68.78 (11.47)	**2.17**
Iono.	89.34 (4.68)	89.82 (4.79)	-0.37
NT	96.28 (3.22)	96.26 (4.19)	0.03
Diab.	75.13 (4.39)	76.05 (4.87)	-1.26
Sonar	87.74 (5.85)	86.65 (6.56)	0.88
Vehicle	71.82 (4.98)	72.22 (3.33)	-0.31
Wine	94.90 (4.83)	**97.99** (3.85)	**-2.90**
(WBC New)	95.88 (2.14)	95.09 (3.06)	1.42
(WBC Orig.)	96.36 (1.68)	95.99 (1.84)	1.05
Overall	84.75	84.36	0.92

4.2 Post-Processing the Final Genotype

Even with parsimony pressure applied by the simplicity function there is no guarantee that the final genotype does not contain redundant or unhelpful nodes that obscure useful information. There are two techniques that we apply to clean the final genotype. Both techniques use the full dataset and have *not* been applied to the genotypes used to obtain the results shown above. Note that as the post-processing techniques involve the full dataset it is impossible to give a fair estimate of the change in predictive accuracy, as the cleaned genotypes have effectively 'seen' the test data (the UCI datasets are too small to keep back a separate validation set).

The first technique is to identify any sub-trees that evaluate to a constant and replace them with that constant. A sub-tree is said to evaluate to a constant if it returns the same value for every instance in the dataset. Constants do not appear in the final genotype very often – in less than 1 in 20 runs, but this step certainly aids human readability when they do occur.

The second is to identify and remove any trees or sub-trees that do not contribute to the predictive accuracy of the genotype. This has the same purpose as the simplicity measure but operates in a deterministic fashion, and is similar to the reduced error pruning technique applied to decision trees [15] except using cross-validation instead of separate validation data (see also [7] for related work). First, the genotype is evaluated by ten-fold cross validation to provide a baseline score. Then each active tree in turn is deactivated and the genotype is re-evaluated. If the new score is the same as or better than the baseline score then the tree remains deactivated and the new score becomes the baseline. Otherwise the tree is re-activated.

After the trees have been evaluated in such a fashion, sub-trees are similarly evaluated. This is done by replacing the top node of each sub-tree by each of its operands in turn, re-evaluating the genotype at each step. Once again if the new score is the same as or better than the baseline score then the function is permanently replaced by its operand. Otherwise the process continues in a depth-first, left-right fashion. For example, if a sub-tree consists of the function (A + B) then the function will be replaced by A (so that the function is replaced by a leaf node) and the genotype re-evaluated. If using A instead of (A + B) offers the same or improved performance then the function will be replaced by A. Otherwise the function is instead replaced by B and the genotype re-evaluated again.

Post-processing reduces the number of nodes in the final genotype by an average of 6.9 nodes (or 30% of the total) with no parsimony pressure applied, and 5.3 nodes (27.4%) with parsimony pressure. With a simplicity strength of 0.04 there is evidently still scope for the post-processing to remove a fairly large proportion of redundant nodes from the final genotype. Conversely, after cleaning the parsimonious genotypes remain slightly smaller than their counterparts (an average of 14.1 nodes compared with 16.1) indicating that the parsimony pressure may be eliminating some potentially useful nodes. The lower standard deviation of the parsimonious solutions suggests that at least some of those nodes are over-fitting the training data, however.

5. APPLICATION TO A VISION DATASET

The machine vision dataset we used consists of measurements of video data feed from a camera in a controlled environment. The aim is to identify and provide a colouring for various objects in a room on a mobile display. The dataset consists of 9963 instances, 31 real-number attributes and 6 classes. The attributes are Luminance, Red, Green, Blue, and 27 Harr Wavelet coefficients[2] (8 size scales, 3 orientations). The class values are Obstacle, Boundary, Floor, Chair, Ball and Box. The target performance is 93%, as achieved by an MLP in a previous study (unpublished).

The dataset was randomly divided into a training set of 6676 instances (67%) and a validation set of 3287 instances. The training set was then treated as the source for 20 runs of the algorithm, each using 90% of the data for training and 10% for test.

Table 6. Performance of unaided classifiers on Vision Dataset

Classifier	Accuracy (train)	Accuracy (test)
C4.5	87.43 (1.06)	88.56
IBk	90.14 (0.84)	90.57
Naïve Bayes	65.36 (2.05)	65.14

The performance of the unaided classifiers on the training set is shown in Table 6 (accuracy on the train set is from cross-validation, followed by standard deviation in brackets). The accuracy figure in the final column was obtained from a single run on the test data. It can quickly be seen that of the three classifiers, IBk is the most accurate by a convincing margin, but falls short of the performance achieved by the MLP neural network.

[2] Harr Wavelet coefficients can be used to identify edges in images.

5.1 Applying Parsimony Pressure

The GAP algorithm was tested with and without parsimony on the training data, results are shown in Table 7 (at this point no post-processing has been done, and genotypes not evaluated on the validation data). Due to the performance implications of using a large dataset the actual number of instances used during fitness evaluation for the GAP algorithm was limited to 2000 (randomly selected)[3].

Table 7. Performance of GAP on Vision training set

	Accuracy (S.D.)	Paired *t*-test
Without Parsimony	91.56 (1.66)	4.26
With Parsimony	91.42 (1.26)	4.53

The accuracy is virtually the same with or without parsimony, but applying parsimony pressure reduces the number of nodes by approx. 48% and in doing so reduces the standard deviation. Applying parsimony pressure provides some robustness to the algorithm by reducing variation between solutions with minimal cost to accuracy (thus improving the *t*-test value). The *t*-test values are obtained by comparing the accuracy against that of IBk on the training data.

Despite the fact that on its own IBk is clearly the best of the three classifiers for this dataset, over 25% of the final genotypes used C4.5 as the base classifier, rising to just over 50% when parsimony pressure is applied.

Although the algorithm provides a significant improvement over an unaided classifier it still falls short of the accuracy of the MLP neural network used in the previous study. However we have not yet performed any post-processing to clean the genotypes, and the presence of a separate validation set allows us to test the best single genotype (as identified by cross-validation on the training set).

5.2 Effect of Post-Processing

Post-processing to clean the final genotypes reduces the number of nodes as shown in Table 8 - figures show nodes before and after post-processing, followed by accuracy using cross-validation on the training set. Recall that as post-processing uses the entire (training) set it cannot be fairly compared to the accuracy of the initial genotypes shown in Table 7, although it is indicative.

[3] At the point where the termination criteria have been met the first time and the dataset would be randomly reordered, it was instead re-sampled to obtain a different random selection of 2000 instances from the training set.

Table 8. Effect of post-processing

	Avg. Nodes (before)	Avg. Nodes (after)	Accuracy (S.D.)
Without Parsimony	52.2	28.6	92.17 (0.93)
With Parsimony	27.2	21.2	91.52 (1.06)

The cleaned genotypes with no parsimony are over 25% bigger than the cleaned genotypes with parsimony applied, but they are more accurate (by over half a percent) and the standard deviation is now smaller than that with parsimony.

Comparing the number of nodes after post-processing, it appears that applying parsimony pressure may be causing useful information to be lost. However it is possible that the extra nodes in the genotypes without parsimony are over-fitting the training data and will prove detrimental on the validation data.

5.3 Applying the Best-of-Run Genotypes to Validation Data

The individuals with the best accuracy using cross-validation on the training set (after post-processing) were applied to the validation set to obtain the results shown in Table 9. Both individuals offer a marked improvement over the average shown in Table 8.

Table 9. Best-of-Run Genotypes

	Accuracy (train)	Accuracy (validation)
Without Parsimony	94.01	94.83
With Parsimony	93.68	94.07

The best of run genotype without parsimony is three quarters of a percent better than that with parsimony (both of which use IBk as the base classifier), and indeed now offers almost two percent improvement over the neural network. Thus results on the validation data do not support the theory that the extra nodes removed by parsimony pressure are over-fitting the training data for the dataset. To determine the impact of post-processing on the overall accuracy we also evaluated the same genotypes as they were before post processing on the validation data. In both cases the cleaned genotype performed better on the validation data than the original genotype (the original without parsimony obtained 94.34% and the original with parsimony obtained 93.7%). That is, post-processing appears to reduce over-fitting as well as improving human readability.

5.4 Human Readability of the Results

Visual inspection of the best single genotype, while easier with the cleaned genotype than the original, did not provide any immediately useful information to a subject matter expert. This is perhaps because the genotype still contains 18 features (of which 7 are trees/ADF's) using 17 of the original attributes, and it is not clear which of the features are the most useful in classification. More useful information may be obtained using a process that estimates the utility of each feature (by removing each one in turn and determining the accuracy of the tree without it, also noting any features that occur more than once[4]). Using this yardstick 3 ADF's were identified as being the most important features: Blue/Luminance, Green/Luminance and Log(Red), along with one of the original coefficients. Dividing a colour by its luminance was identified by the expert as a method by which the true colour of an object may be determined (by factoring out the level of light falling on it).

Additional use was made of the human readability of the decision trees created by C4.5. These were created using the features of the both the best of run genotype (based on IBk) and of the best genotype based on C4.5 – see Figures 3 and 4. The presence of ratios of colour/luminance at the top of both decision tree hierarchies added weight to their importance in identifying objects in images.

Analysis of the frequency with which attributes appear in all of the final (cleaned) genotypes also provides some indication of how useful they are in classification. The 3 colours and luminosity are the most useful attributes (they always appear in all 20 of the final genotypes regardless of parsimony pressure, and are the only attributes to do so). These are followed by 2 of the coefficients which appear in over 75% of the final genotypes. Similar analysis of the frequency of ADF's confirms the utility of the 3 ADF's identified above (which appear in at least a quarter of the final genotypes), and also the ratios of Green and Blue to Red (ADF's that did not appear in the best of run).

6. CONCLUSIONS

When pre-processing data, and in the absence of any form of post processing, applying parsimony pressure successfully reduces the number of nodes in the final solution, making it easier for a human to read and adding some robustness by reducing variance in the accuracy of the final solutions. However, it does introduce a risk of removing some useful information from the solutions, particularly the best-of-run.

Performing post-processing to clean the final genotypes has a similar impact on human readability as parsimony pressure, but does not run the same risk of losing useful information. It can also help to reduce over-fitting.

Additional measures to determine the utility of individual features (both within the single best solution and across all solutions) provide a further aid to human readability.

Future work will investigate the application of the GAP algorithm to larger datasets, and investigate combining all 20 final genotypes into an ensemble classifier.

7. REFERENCES

[1] Aha, D., & Kibler, D. Instance-based learning algorithms. *Machine Learning* vol.6, 1991, 37-66.

[2] Ahluwalia, M. & Bull, L. Co-Evolving Functions in Genetic Programming: Classification using k-nearest neighbour. In *GECCO-99: Proceedings of the Genetic and Evolutionary*

[4] Because the genotype has been processed to remove any part that does not contribute toward its accuracy, we may be certain that IBk is benefiting from having extra copies of each of two features: Blue/Luminance and log (Red); and one of the coefficients.

Computation Conference. Morgan Kaufmann, 1999 pp. 947–952.

[3] Bernstein, Y., Li, X., Ciesielski, V., Song, A.: Multiobjective parsimony enforcement for superior generalisation performance. In: *Proceedings of the Congress for Evolutionary Computation 2004 (CEC'04)*, 2004 pp. 83-89.

[4] Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., Michalkiewicz, E.L., A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets, *Artificial Intelligence in Medicine* 30 (1), 2004, 21–48.

[5] De Jong, E. D., Watson, R. A., Pollack, J. B. Reducing Bloat and Promoting Diversity using Multi-Objective Methods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, 2001, pp. 11-18.

[6] Ekárt, A. & Márkus, A. Using Genetic Programming and Decision Trees for Generating Structural Descriptions of Four Bar Mechanisms. In *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, volume 17, issue 3, 2003.

[7] Garcia-Almanza, A.L., Tsang, E.P.K. Simplifying Decision Trees Learned by Genetic Programming. *IEEE Congress on Evolutionary Computation, CEC 2006*, pp 2142- 2148.

[8] Holland, J.H. *Adaptation in Natural and Artificial Systems*. Univ. Michigan. 1975.

[9] John, G.H & Langley, P. Estimating Continuous Distributions in Bayesian Classifiers. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, San Mateo. 1995, 338-345.

[10] Kelly, J.D. & Davis, L. Hybridizing the Genetic Algorithm and the K Nearest Neighbors Classification Algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp377-383.

[11] Kohavi, R. & John, G. H. Wrappers for feature subset selection. Artificial Intelligence Journal vol. 1-2: 273-324. 1997.

[12] Koza, J.R. *Genetic Programming*. MIT Press. 1992.

[13] Krawiec, K. Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks. *Genetic Programming and Evolvable Machines* vol. 3 no. 4: 329-343. 2002.

Langdon, W. B. & Buxton, B. F. Genetic programming for improved receiver operating characteristics. In *Second International Conference on Multiple Classifier System*, volume 2096: 68-77. 2001.

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

[15] Otero, F. E. B., Silva, M. M. S., Freitas, A. A. & Nievola J. C. Genetic Programming for Attribute Construction in Data Mining. In *Genetic Programming: 6th European Conference, EuroGP 2003, Essex, UK, April 2003, Proceedings.* Springer, pp. 384-393.

[16] Quinlan, J.R. *C4.5: Programs for Machine Learning.* Morgan Kaufmann. 1993.

[17] Raymer, M.L., Punch, W., Goodman, E.D. & Kuhn, L. Genetic Programming for Improved Data Mining - Application to the Biochemistry of Protein Interactions. In *Proceedings of the Second Annual Conference on Genetic Programming*, Morgan Kaufmann, 1996, 375-380.

[18] Siedlecki, W. & Sklansky, J. On Automatic Feature Selection. *International Journal of Pattern Recognition and Artificial Intelligence* 2:197-220. 1988.

[19] Smith, M. & Bull, L. Using Genetic Programming for Feature Creation with a Genetic Algorithm Feature Selector. In *Parallel Problem Solving from Nature - PPSN VIII, X.* Springer-Verlag, 2004.

[20] Smith, M. & Bull, L. Genetic Programming with a Genetic Algorithm for Feature Construction and Selection. *Genetic Programming and Evolvable Machines* vol. 6 no. 3: 265-281. 2005

[21] Thomas, J. & Sycara, K. The Importance of Simplicity and Validation in Genetic Programming for Data Mining in Financial Data. Proceedings of the joint AAAI-1999 and GECCO-1999 Workshop on Data Mining with Evolutionary Algorithms, July, 1999.

[22] Vafaie, H. & De Jong, K. Genetic Algorithms as a Tool for Restructuring Feature Space Representations. In *Proceedings of the International Conference on Tools with A.I.* IEEE Computer Society Press. 1995.

[23] Witten, I.H. & Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann. 2000.

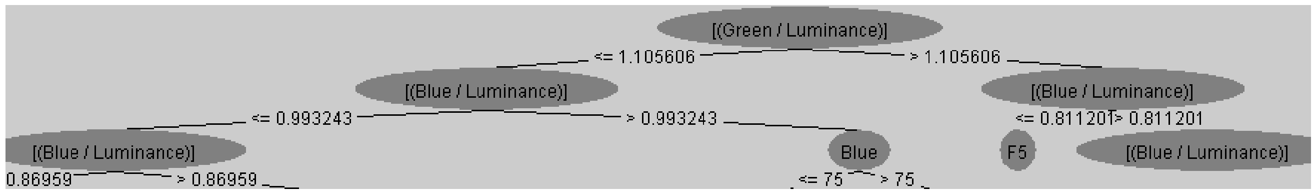

Figure 3. Top section of the C4.5 decision tree for the best IBk genotype

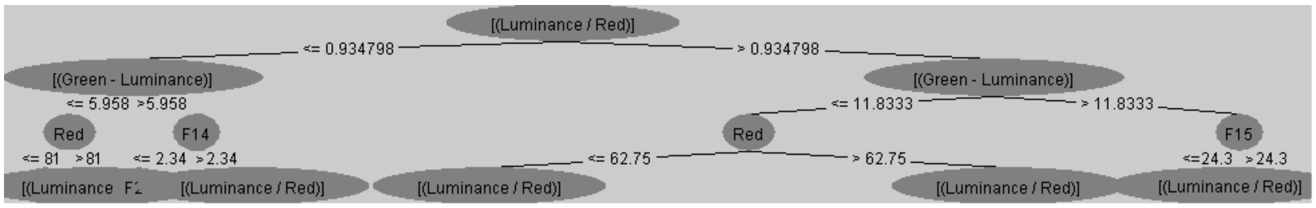

Figure 4. Top section of the C4.5 decision tree for the best C4.5 genotype