Coevolving Functions in Genetic Programming: Classification using K-nearest-neighbour

Manu Ahluwalia
Intelligent Computer Systems Centre
Faculty of Computer Studies and Mathematics
University of the West of England
Bristol BS16 1QY, U.K.
manu@ics.uwe.ac.uk

Larry Bull
Intelligent Computer Systems Centre
Faculty of Computer Studies and Mathematics
University of the West of England
Bristol BS16 1QY, U.K.
larry@ics.uwe.ac.uk

Abstract

In this paper we introduce a new approach to the use of automatically defined functions within Genetic Programming for classification tasks.The technique consists of coevolving a number of separate sub-populations of functions, each of which acts as a feature extractor for the K-nearest-neighbour algorithm. Using two well-known classification tasks it is shown that our coevolutionary approach performs better than the equivalent traditional function mechanism used in Genetic Programming. The approach is then extended to include explicit feature selection at the level above the coevolving extractor functions. In this way features which are not needed for the task can be ignored more effectively than relying on the evolution of extractors which achieve the same effect. It is shown that this approach performs better than the first coevolutionary technique, and hence better than the traditional approach.

1 INTRODUCTION

The use of function subroutines is ubiquitous in high-level computer programming languages. Functions provide the ability to reuse code efficiently, generating a modular and hierarchical structure to programs. Since its formalization, Genetic Programming (GP) [Koza 1992] has included the ability to exploit functional subroutines, termed "automatically defined functions" (ADFs) [Koza 1992, p.534]; modular functions can form part of the genetic make-up of an evolving program.

In this paper we present an extension, specifically for classification tasks, to our previous work on a coevolutionary approach to the use of automatically defined functions in genetic programming [e.g. Ahluwalia et al. 1997]. In the general approach each identified ADF is assigned its own independent sub-population which coevolves with other ADF sub-populations and a population of main program trees or "result-producing branches" (RPBs). For each evaluation, ADFs from each sub-population are selected randomly to be used by a program, where fitness can be assigned globally or locally. In the global case all ADFs and the main tree receive the same fitness. In the local case the main tree receives the global fitness, but each ADF receives the fitness available for that aspect of the task with which it is concerned. In either case selection and reproduction are done independently within each sub-population.

Recently, we have also introduced a version of the approach termed "evolution defined functions" (EDFs) [Ahluwalia & Bull 1998] which uses the coevolutionary strategy in conjunction with the two mutation operators, compression and expansion, of the "genetic library builder" (GLiB) [Angeline & Pollack 1994]. We showed that the automatic specification of EDF sub-populations via compression is beneficial when the existence of a particular function is determined by a measure of its worth/ recent usage. We then extended the approach further to allow any number of functions to be created during evolution, rather than having an a priori fixed number of EDFs, again using the measure of existing function worth. It was shown that improvements can be achieved over the previous coevolutionary approach.

In this paper we introduce a version of our coevolutionary technique specifically for classification tasks, based on the work of Raymer et al. [1996], in which ADFs are feature preprocessors/extractors for the well-known K-nearest-neighbour (Knn) classification algorithm. The Knn algorithm holds a number of data exemplars in memory (the training set) and uses the Euclidean distance of a presented data item to the memory members to determine its class/ identification.

The paper is arranged as follows: the next section details the basic approach. Section 3 describes the problems used to do the comparisons and section 4 presents results from its use. Finally, all findings are discussed.

2 FUNCTIONS AS FEATURE EXTRACTORS IN GENETIC PROGRAMMING

2.1 AUTOMATICALLY DEFINED FUNCTIONS

Koza [1992] presented automatically defined functions as a refinement to genetic programming with the aim of enabling the composite evolution of larger programs. Here each identified ADF is genetically joined to the main program tree such that a child's ADFs are a mix of its parents'; each joined ADF recombines with the corresponding ADF of the other parent. Whenever a call to a particular type of ADF is made, the joined example individual is used. The number of ADFs available during evolution is fixed a priori and the ADFs exist in a hierarchy, i.e. ADF_0 can call all others, ADF_1 all others except ADF_0, etc. (see [Koza 1994] for a full review).

This idea has been extended by Spector [1996] to allow for the modular use of macros, rather than full function subroutines, in GP - termed automatically defined macros (ADMs).

Raymer et al. [1996] have presented an alternative approach to the use of ADFs specifically for classification tasks. Rather than evolve classification programs, they evolve sets of feature preprocessors, or feature "extractors", for use in conjunction with a K-nearest-neighbour algorithm. Here each feature is altered by a GP ADF tree, evolved for that feature only, with the aim of increasing the separation of pattern classes in the feature space. The modified set of features are then used by a Knn algorithm to evaluate the effectiveness of the evolved extractors; each individual consists of a set of ADFs, one for each feature.

2.2 COEVOLVING AUTOMATICALLY DEFINED FUNCTIONS

As noted above, we have previously suggested a coevolutionary approach to the use of GP with ADFs for larger programs. The use of coevolutionary or multi-agent/population techniques has been shown to be beneficial in a number of domains, e.g. [Husbands & Mill 1991]. In the approach presented each type of ADF evolves within its own separate sub-population via a standard Genetic Algorithm [Holland 1975] (steady state algorithms are used [Syswerda 1989]). As each evolved main tree is evaluated, making one or more calls to a particular function type, an individual from the corresponding sub-population is randomly selected, i.e. over time an individual ADF sub-population member can find itself being used by a number of evolving programs and any other ADFs used within them. We have suggested that in this way a greater amount of genetic mixing, and hence searching, is possible than in Koza's original approach.

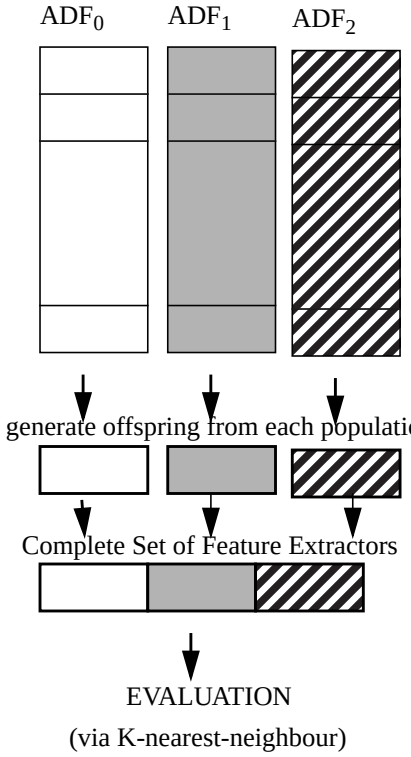


Figure 1: Showing the coevolutionary approach to the use of ADFs for classification. Each population generates a feature extractor for its particular feature, which are then used as preprocessors for the K-nearest-neighbour classification algorithm. All individuals receive the resulting rating/score.

In [Raymer et al. 1996] each ADF is a feature extractor for a Knn algorithm and hence each individual consists of α ADFs for a task containing α features, with no main program. To apply our coevolutionary approach to their approach we use a system containing α sub-populations of coevolving feature extractors, where one individual from each population is required for an evaluation (Figure 1). Since we use a steady-state system, after evaluating each corresponding individual in the initial populations, the offspring of each population are partnered for an evaluation.

Raymer et al.'s approach uses feature extraction, rather than feature selection. As noted above, feature extraction

alters the scaling of the feature space, whereas feature selection indicates whether a feature should be used. Hence feature selection is possible in the former when a scaling of zero is applied. For other (early) examples of evolutionary computation approaches to these two techniques, see [Siedlecki & Sklansky 1988] for a genetic algorithm-based feature selection approach using Knn and [Kelly & Davis 1991] for a similar genetic algorithm-based feature extraction approach also using Knn. In this paper we also present a version of our approach which includes a population of traditional genetic algorithm binary feature selectors, such that these individuals are analogous to the main programs in GP with ADFs and our (more general) previous coevolutionary techniques. Here, when a main program contains a '1' in a given locus, an individual from the corresponding ADF sub-population is generated for use in conjunction with the Knn algorithm (after all initial sub-population members have been used). This allows both the benefits of feature extraction and feature selection to be exploited; feature selection is possible without waiting for feature functions to evolve scales of exactly zero. For completeness we also present an EDF version of this system, such that usage/worth counters are employed to determine whether the population of main (feature selector) programs have found the current logic of a particular function (feature extractor) beneficial. Such sub-populations are randomly re-initialised if they have not been significantly used. We use uniform crossover [Syswerda 1989] and mutation (rate per bit 0.01) to evolve the feature selectors here.

3 THE CLASSIFICATION TASKS

In this paper we use two well-known classification problems to compare the different strategies (see [Eiben et al.1997] for an example of the comparative performance and potential benefits of GP for classification tasks).

3.1 AUSTRALIAN CREDIT CARD

The Australian Credit Card data set [Statlog] contains 690 examples, each of which contains the customer's details in 14 input variables (floating values between 0 and 1) and a classification bit (output) which is '0' or '1'. A '1' indicates that a particular customer should have a credit card and the '0' indicates not. The Knn memory is generated by dividing the full data set into (roughly) half and then using a random three-quarters of that. The Classifier System is trained using the same set of 400 of the examples and evaluations are run until a system correctly classifies all of the training data or no further improvement is seen. Also, K=1 is used here.

The function set used for all populations is {+,-,/,*} and the terminal set is {A, $\mathcal{R}$}, where $\mathcal{R}$ is an ephemeral random constant, range 0.0-1.0, and A is the feature value. The initial population is created with tress of depth size 10.

3.2 LETTER IMAGE RECOGNITION

In this task [Frey & Slate 1991] the objective is to identify each of a large number of black-and-white rectangular

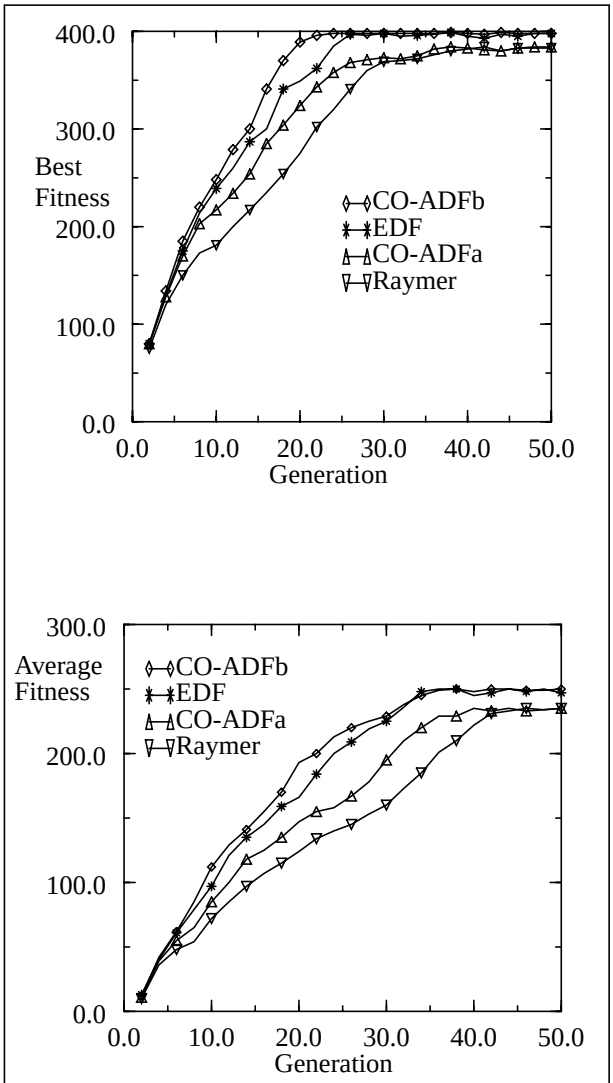
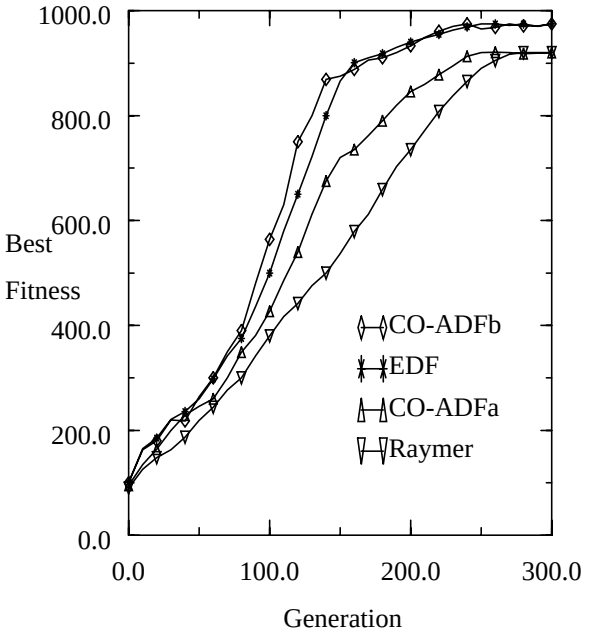


Figure 2: Comparative performance of the Knn techniques on the Australian data set.

pixel displays as one of the 26 capital letters in the English alphabet. The character images are based on 20 different fonts and each letter within these 20 fonts is randomly distorted to produce a file of 20,000 unique stimuli. Each

stimulus is converted into 16 primitive numerical attributes (statistical moments and edge counts) which are



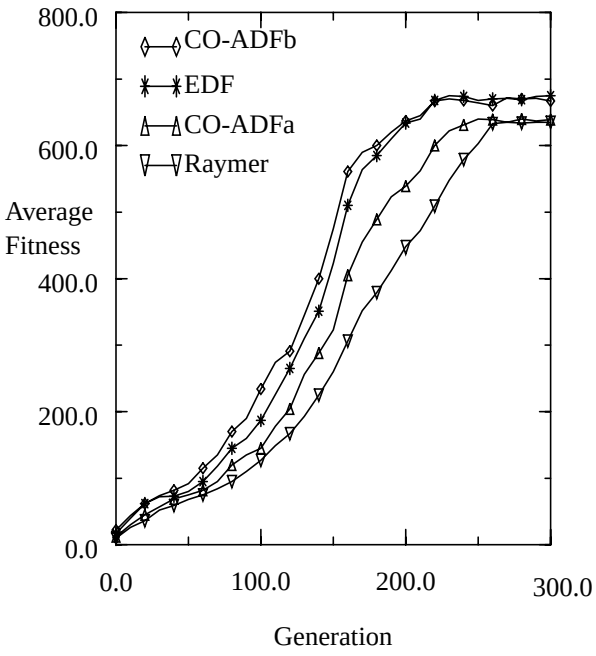


Figure 3: Comparative performance of the Knn techniques on the Letter Recognition task.

then scaled to fit into a range of integer values from 0 through 15. The Knn memory is generated by halving the full data set and picking three-quarters of that half. A training set of 1000 examples from the same half is used here, randomly recreated once every ten evaluations for efficiency, until a system which can correctly classify all of the training data is found. Again, K=1 here.

The function and terminal sets are the same as those for the Australian Credit Card task and the tress in the initial population are restricted to depth size 10.

It has been found that, for both tasks, the best results are obtained when crossover (for each population) is performed on a small percentage (20%) of the population. A mutation rate of 0.02 per node and roulette-wheel selection are used throughout.

4 RESULTS

4.1 AUSTRALIAN CREDIT CARD

Figure 2 shows the average results from fifty runs using these three versions of our coevolutionary approach and Raymer et al.'s on the Australian Credit Card task. Here each feature sub-population contains 101 individuals for the first coevolutionary version of Raymer et al.'s approach (CO-ADFa) and 94 for the other two (CO-ADFb and EDF) since they require an extra sub-population for the binary feature selector main programs (~1410 in total).

In the EDF version of the system the usage counter was checked every 5 generations to see if any sub-populations had not been used by the main programs, i.e. whether each feature was being used for the majority of classifications (>50%).

From figure 2 it can be seen that the two techniques which make explicit use of feature selection (CO-ADFb and EDF) do better than those which rely purely on feature extraction, both in terms of mean and best performance, with all coevolutionary approaches doing better than Raymer et al.'s approach. The CO-ADFb and EDF approaches produce sets of selector/extractors which, coupled with the Knn algorithm, classify 98% of the data set correctly using (the same) 10 features. The use of the EDF dynamic function creation mechanism here gave no benefits in terms of functionality, only speed-up, and is in fact slightly slower than the CO-ADFb approach. The CO-ADFa and Raymer et al.'s approaches produce classifiers capable of around 95% accuracy on the full data set.

4.2 LETTER IMAGE RECOGNITION

Figure 3 shows the average results from fifty runs of the same techniques on the letter recognition task. Each sub-

population contains 100 individuals for Raymer et al.'s

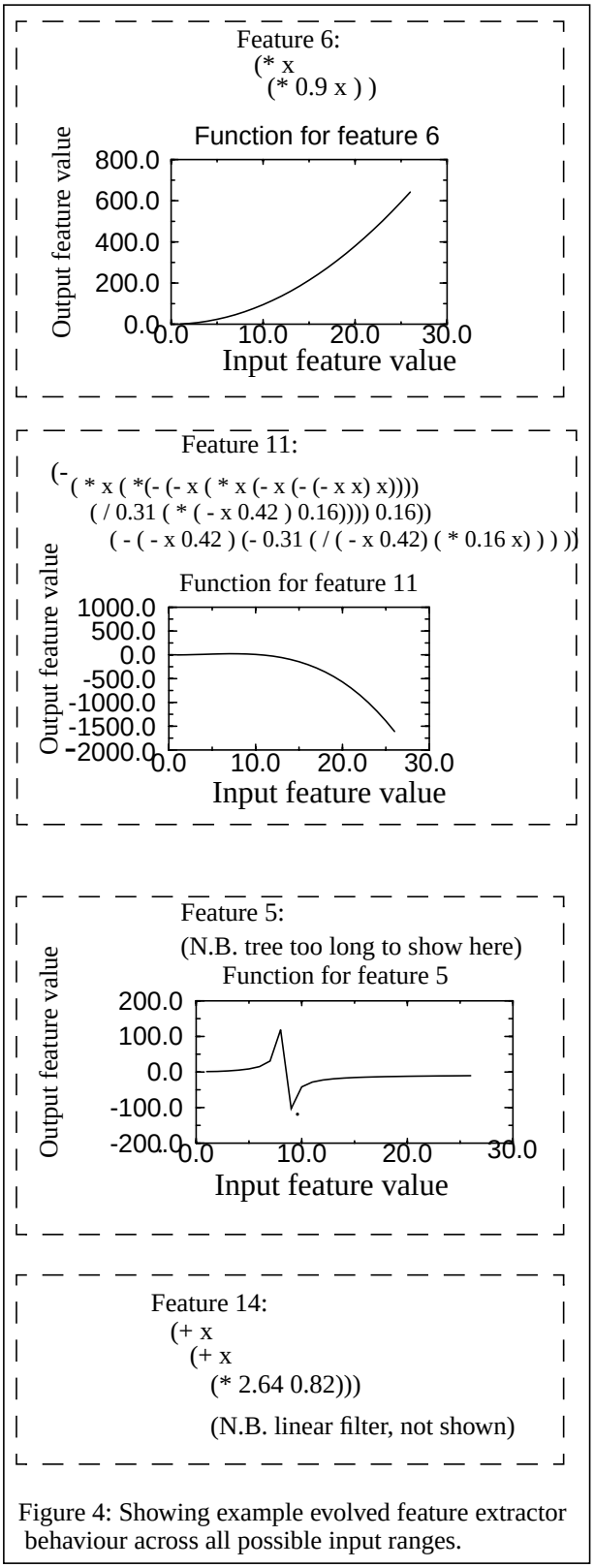


Figure 4: Showing example evolved feature extractor behaviour across all possible input ranges.

and the CO-ADFa approaches and 94 for the other two (~1600 in total). It can be seen that, again, the use of explicit feature selection proves beneficial, both in terms of mean and best performance, with CO-ADFb and EDF both producing classifiers with 94% accuracy on the complete task. The CO-ADFb approach is found to use 9 features here, with the EDF approach using the same 9 plus one other feature on average. The CO-ADFa approach produces classifiers with around 92% accuracy and Raymer et al.'s approach also produces classifiers with 92% accuracy.

Since the EDF approach with the binary feature selector used an extra feature to the ADF approach using feature selection (CO-ADFb), the evolved feature extractors have been examined for this task.

It was found that both resulting classifiers used features 5-8, 11 and 13-16. The EDF approach also used feature 9. We examined what the evolved feature extractors do to the input variable (x) by presenting all possible feature values to the evolved S-expressions and plotting the results. Figure 4 shows examples of each type of feature extractor seen, where their behaviour can be described as "linear", "exponential", "polynomial" and "pulse". Feature 6 is an exponential, feature 14 is linear, features 5, 7, 8, 15 and 16 are pulses, and features 11 and 13 are polynomial. The evolved extractor for feature 9, the extra feature used by EDF, has the same characteristics as feature 6, i.e. it is an exponential. It was assumed that the extractor for feature 9 would be simple, i.e. linear, since its use does not appear to provide any significant benefit. However, it is assumed that if a cost was added to encourage the use of the least possible number of features, this use of feature 9 would disappear; feature 9 is not critical/misleading.

5 Conclusions

In this paper we have presented a coevolutionary approach to the use of automatically defined functions (ADFs) in genetic programming for classification tasks. This was based on the work of Raymer et al. [1996] in which each ADF is a feature extractor for the Knn algorithm. We examined our coevolutionary approach's performance, along with a coevolutionary and an EDF-like version in which the main program is a binary feature selector rather than a full S-expression tree. It was found that all coevolutionary approaches performed better than Raymer et al.'s approach and that the use of feature selection in conjunction with feature extraction performed best of all.

It is noted that the different approaches have different computational overheads since some involve the creation

of sub-populations via mutation. Table 1 shows the relative number of evaluations required by each approach to classify the training data correctly and the performance of the resulting system on the full data set, for both tasks. It can be seen that the our coevolutionary approaches always require less computational effort to achieve greater performance.

Table 1: Tableau for total number of individuals processed during training and resulting performance. The total number of individuals processed depends on number of individuals processed per generation and the total cost which is compression and expansion events in each case.

Tecnnique	Number of individual processed	Classification Rate
Raymer	Aust: 76230 Letter: 537600	Aust: 95% Letter: 92%
CO-ADF*	Aust: 59290 Letter: 489600	Aust: 95% Letter: 92%
CO-ADF**	Aust: 42375 Letter: 461040	Aust: 98% Letter: 94%
EDF**	Aust: 48683 Letter: 482922	Aust: 98% Letter: 94%

References

Ahluwalia M, Bull L, & Fogarty T C (1997), “Coevolving Functions in Genetic Programming: A Comparison in ADF Selection Schemes”, in J R Koza, K Deb, M Dorigo, D B Fogel, M Garzon, H Iba & R Riolo (eds.) *Proceedings of the Second Annual Conference on Genetic Programming*, Morgan Kaufmann, pp3-8.

Ahluwalia M & Bull L (1998), “Coevolving Functions in Genetic Programming: Dynamic ADF Creation using GLiB”, in V W Porto, N Saravanan, D Waagen & A E Eiben (eds.) *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, Springer-Verlag, pp809-818.

Angeline P J & Pollack J B (1994), "Coevolving High-Level Representations”, in C G Langton (ed.) *Artificial Life III*, Addison-Wesley, pp55-72.

Eiben A E, Euverman T J, Kowalczyk W and Slisser F (1997), “Modelling Customer Retention with Statistical Techniques, Rough Data Models and Genetic Programming”, in A Skowron and S K Pal (eds.), *Fuzzy Sets, Rough Sets and Decision Making Processes*, Springer, in press.

Frey P W & Slate D J (1991), “Letter Recognition Using Holland-style Adaptive Classifier Systems”, *Machine Learning* 6(2):161-182.

Husbands P & Mill F (1991), “Simulated Coevolution as the Mechanism for Emergent Planning and Scheduling”, in R L Belew & L B Booker (eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp264-270.

Koza J R (1992)(ed.), *Genetic Programming*, MIT Press.

Koza J R (1994)(ed.), *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press.

Raymer M L, Punch W, Goodman E D & Kuhn L (1996), “Genetic Programming for Improved Data Mining - Application to the Biochemistry of Protein Interactions”, in J R Koza, K Deb, M Dorigo, D B Fogel, M Garzon, H Iba & R Riolo (eds.) *Proceedings of the First Annual Conference on Genetic Programming*, Morgan Kaufmann, pp375-380.

Siedlecki W & Sklansky J (1988), “On Automatic Feature Selection”, *International Journal of Pattern Recognitiona and Artificial Intelligence* 2:197-220.

Statlog (), data and documentation at fttp.ncc.up.pt/pub/statlog

Syswerda G (1989), "Uniform Crosssover in Genetic Algorithms", in J D Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp2-9.