Further Research on Feature Selection and Classification Using Genetic Algorithms

Appeared in ICGA93, pg 557-564, Champaign Ill.

W.F. Punch[1], E.D. Goodman[2], Min Pei[2,3], Lai Chia-Shun[2], P. Hovland[2] and R. Enbody[4]

[1] Knowledge Based Systems Laboratory
Computer Science Department, Michigan State University
E. Lansing, MI. punch@cps.msu.edu
[2] Case Center for Computer-Aided Engineering and Manufacturing
College of Engineering, Michigan State University
[3] Beijing Union University, Beijing China
[4] Parallel Processing Laboratory
Computer Science Department
Michigan State University

Abstract. This paper summarizes work on an approach that combines feature selection and data classification using Genetic Algorithms. First, it describes our use of Genetic Algorithms combined with a K-nearest neighbor algorithm to optimize classification by searching for an optimal feature weighting, essentially warping the feature space to coalesce individuals within groups and to separate groups from one another. This approach has proven especially useful with large data sets where standard feature selection techniques are computationally expensive. Second, it describes our implementation of the approach in a parallel processing environment, giving nearly linear speed-up in processing time. Third, it will summarize our present results in using the technique to discover the relative importance of features in large biological test sets. Finally, it will indicate areas for future research.

1 The Problem

We live in the age of information where data is plentiful, to the extent that we are typically unable to process all of it usefully. Computer science has been challenged to discover approaches that can sort through the mountains of data available and discover the essential features needed to answer a specific question. These approaches must be able to process large quantities of data, in reasonable time and in the presence of "noisy" data i.e., irrelevant or erroneous data.

Consider a typical example in biology. Researchers in the Center for Microbial Ecology (CME) have selected soil samples from three environments found in agriculture. The environments were: near the roots of a crop (rhizosphere), away from the influence of the crop roots (non-rhizosphere), and from a fallow field (crop residue). The CME researchers wished to investigate whether samples from those three environments could be distinguished. In particular, they wanted to see if diversity *decreased* in the rhizosphere as a result of the symbiotic relationship between the roots and its near-neighbor microbes, and if so in what ways.

Their first experiments used the Biolog© test as the discriminator. Biolog consists of a plate of 96 wells, with a different substrate in each well. These substrates (various sugars, amino acids and other nutrients) are assimilated by some microbes and not by others. If the microbial sample processes the substrate in the well, that well changes color which can be recorded photometrically. Thus large numbers of samples can be processed and characterized based on the substrates they can assimilate. The CME researchers applied the Biolog test to 3 sets of 100 samples from each of rhizosphere, non-rhizosphere and crop residue soil. Each sample was tested on the 96 features provided by Biolog. The questions to be asked of these results were twofold: can Biolog be used to distinguish the three sample sets and if so, which of the Biolog features are most important for making the discrimination and which are simply "noise" – that is, non-contributing features.

Note a number of characteristics of this application. First, the feature space is quite large, 3 x 100 x 96, and therefore computationally expensive for traditional approaches. Second, the data is *very* noisy. The microstructure of soils is extremely heterogeneous, so the

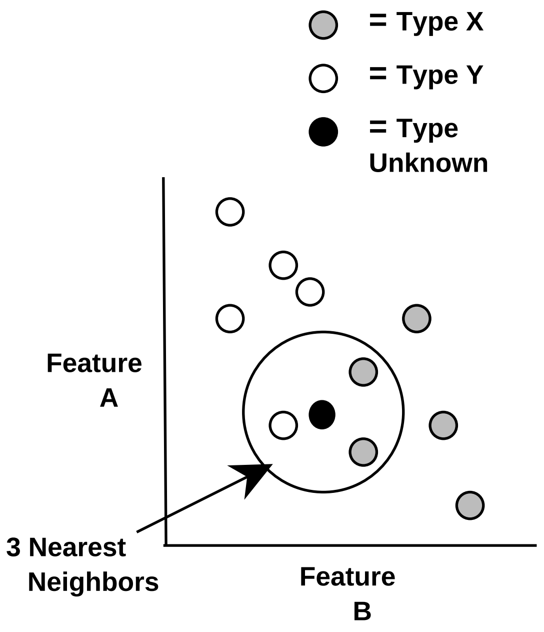

Figure 1: A Simple Knn example

potential for non-uniformity among samples is high. Third, it is desirable for the results to indicate the *relative importance* of the Biolog features, not only for discrimination of unknowns, but also for subsequent analysis of *why* those particular features are important in this discrimination – i.e., what is the biological significance of those features.

2 The Approach

The basis of our approach is to use a K nearest neighbor algorithm within the evaluation function of a Genetic Algorithm (GA) to learn how to classify samples. This approach was inspired by work first reported by Siedlecki and Sklansky [Siedlecki and Sklansky 1990] but modifies and extends their approach in a number of important ways (see section 2.2 for details).

2.1 The Knn algorithm

A simple Knn approach to classification is shown in Figure 1. Each feature of the test set is a dimension in the classification search space. For simplicity's sake, let us assume that there are only two features in the test we conduct, represented by the x and y axis in the figure. Known examples are then placed into this space as points based on their *known* feature values and labeled according to their known classification. An unknown sample can then be placed in the same space based on its feature values and it can be classified based on its K *nearest neighbors*, where K is set to some integer value. Assume the $K = 3$; then in Figure 1, the unknown sample is labeled as type X based on the fact that 2 of the 3 of its nearest neighbors are of type X. In the case of the rhizosphere data, there are 96 dimensions to the Knn space and 324 examples that are placed in the space as known classifications.

Techniques exist for reducing the problems encountered using only a Knn – for example, finding representative subsets using editing and condensing [Devivjer and Kittler 1982], but these essentially require throwing away data and do not take into account similar noise encountered in the unknown evaluations. The technique presented here will use the GA to overcome some of the shortcomings of the classical Knn, by allowing the system to learn to warp the n-dimensional feature space so as to maximize the clustering of individuals within a class, and at the same time maximize the separation between classes.

2.2 The GA approach

The Knn algorithm does not tell us which of the 96 dimensions/features are important for discriminating the known samples. Moreover, it does not tell the *relative* importance of these features. It only allows us to measure similarity between known samples. Its performance could be improved if we could determine which of the available features are most important for the discrimination and which are acting primarily as "noise". This is an important question in itself: how to *select features* (represented as the dimensions in the Knn) such that we can optimize the classification of the known samples and therefore, presumably, the unknown test samples. Thus we hope to accomplish two things at the same time: feature selection in the data and optimal classification of those data based on our feature selection, in a generalized feature space as defined below.

To accomplish this optimization we use a genetic algorithm with the Knn as the main part of the evaluation function. Siedlecki and Sklansky's original work used the Knn in the following way. Their goal was to find the "the smallest or least costly subset of features for which the classifier's performance does not deteriorate below a certain specified level." [Siedlecki and Sklansky 1990]. Essentially, they did this by constructing a GA chromosome which consisted of a binary encoding whose length equaled the number of features. If a bit was one, then that feature was used in evaluating the usefulness of the Knn on the known data sets. Each such string was penalized based on its performance (more penalty for worse performance in classifying the known Knn samples) and its length (more penalty for more 1's in the chromosome). They found their approach to be as good as many standard approaches like exhaustive search and branch and bound and much better on large feature sets (20 features or more) where the standard ap-

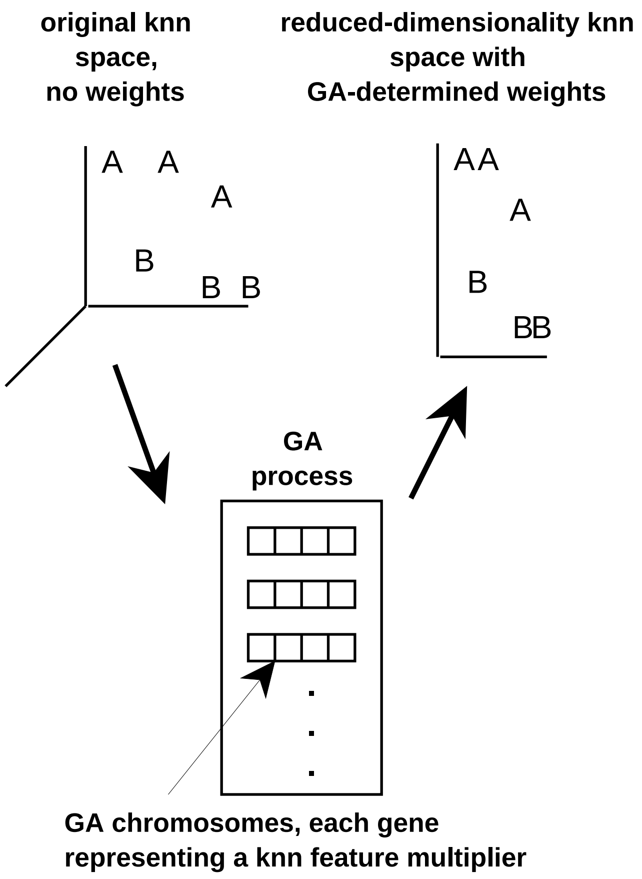
original knn space, no weights
reduced-dimensionality knn space with GA-determined weights
A A A B B B
AA A B BB
GA process
GA chromosomes, each gene representing a knn feature multiplier

Figure 2: An example warping of the Knn space by weighting features. Note that, after weighting, A and B are closer to others in their class, and relatively further from individuals in other classes. Had A and B been interchanged, a similar shortening of the vertical axis would have had a similar effect.

proaches began to show combinatorial explosion and therefore poorer computational performance.

An addition to this approach is shown in Figure 2. One can view the Siedlecki-Sklansky chromosome as an example of a 0/1 weighting of the importance of the features. That is, before the Knn is evaluated each feature dimension is multiplied by a weight (in this case 0 or 1) and this modified Knn used to classify the known samples. Rather than allowing the weights to have values of only 0 or 1, we allow those weights to range over a broader scale, such as 0 to 10. As such we are now searching for a *relative weighting* of features that gives *optimal* performance on classification of the known samples. In effect, this weighting is a warping/modification of the Knn space. Those weights that move towards 0 indicate that their corresponding features are not important for the discrimination task. Essentially, those features "drop out" of the Knn space and are not considered. Any weight that moves towards the maximum weight indicates that the classification process is sensitive to small changes in that feature. That feature's Knn dimension is elongated, which probably reflects that an increased separation between the classes gave better resolution. This is precisely the approach we have taken, using the GA to discover real-valued weights that modify the Knn space. We pursued this work independently of work done in 1991 by Kelly and Davis [Kelly and Davis 1991]. Their core idea was exactly the same, namely using a GA to generate real-valued weights to improve Knn performance, though the emphasis of the two groups is somewhat different (our work incorporates parallel processing and "hidden feature" processing, which is discussed in subsequent sections).

Features are prenormalized to the range 0,1], giving weights an interpretation of relative importance of features to the classification task. However, it is important that they are also function *in combination* to change the boundaries between groups, thus allowing *superior* classification performance to the best "unscaled" Knn algorithm.

The end result of running the GA-Knn combination is a scaling of each of the Knn features such that an optimal class separation can be achieved between the known classes. This weighting vector, once discovered, can then be used to classify unknown samples very quickly (Knn speed) and can also be used to indicate which differences are important for telling the known classes apart, providing focal points for further research in the domain.

2.3 Fitness Functions

The functional parts of the GA-Knn weighting algorithm are shown in Figure 3. The Knn training data are first normalized so that each feature has equal weight in the beginning. For a typical problem, weights were chosen to range among 256 values between 0 and 10, using 8 bits per feature (Gray coded) to represent each weight. After translation from Gray to their standard binary representation weights were mapped exponentially to the real range [0.01,10) plus the value 0.This mapping was done to allow "fine tuning" below 1.0, and to allow mutation to have a similar proportional effect throughout the range of weights.

If weights fell below a set threshold value, in this case 1, then the weight was effectively set to 0 for the evaluation. This is an empirical modification based on the fact that weights tended to very rarely reach 0, despite the fact that their weights were indeed quite low and apparently not contributing to the classification. We also performed experiments with weights constrained to discrete values of 0 and 1 to compare to the original

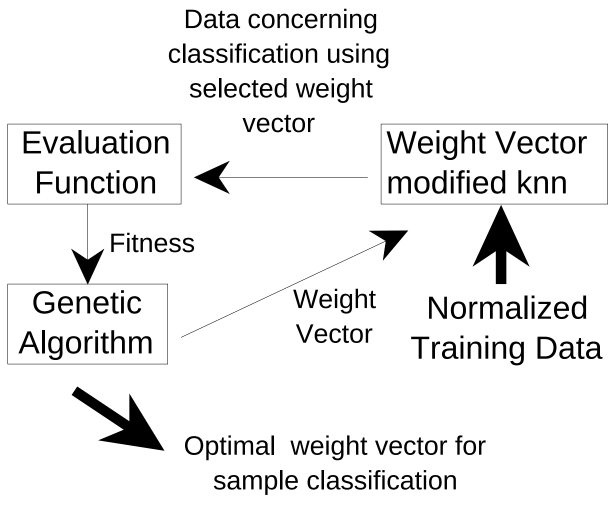
Data concerning classification using selected weight vector
Evaluation Function
Weight Vector modified knn
Fitness
Genetic Algorithm
Weight Vector
Normalized Training Data
Optimal weight vector for sample classification

Figure 3: The GA-knn feature weighting algorithm

Class	f1	f2	f3	f4
A	{0-1}	{0-0.1}	{0-0.1}	0.2+[f3]
B	{0-1}	{0-0.1}	{0-0.1}	0.2+[f3]
C	{0-1}	{0-0.1}	{0.1-0.2}	0.3–[f3]
D	{0-1}	{0-0.1}	{0.1-0.2}	0.3–[f3]
E	{0-1}	{0.1-0.2}	{0-0.1}	0.2+[f3]
F	{0-1}	{0.1-0.2}	{0-0.1}	0.2+[f3]
G	{0-1}	{0.1-0.2}	{0.1-0.2}	0.3–[f3]
H	{0-1}	{0.1-0.2}	{0.1-0.2}	0.3–[f3]

Class	f5	f6	f7-15
A	$0.5*[f3]^2$	$0.5+5*[f3]^2$	{0-1}
B	$0.5*[f3]^2$	$1-25*[f3]^2$	{0-1}
C	$0.5*[f3]^2$	$0.5+5*[f3]^2$	{0-1}
D	$0.5*[f3]^2$	$1-25*[f3]^2$	{0-1}
E	$0.5*[f3]^2$	$0.5+5*[f3]^2$	{0-1}
F	$0.5*[f3]^2$	$1-25*[f3]^2$	{0-1}
G	$0.5*[f3]^2$	$0.5+5*[f3]^2$	{0-1}
H	$0.5*[f3]^2$	$1-25*[f3]^2$	{0-1}

Table 1: Dataset 1, a simple generated test set

Siedlecki-Sklansky-type performance[5]. All Knn runs were done with K=5 i.e., 5 nearest neighbors.

We have experimented with a number of evaluation functions based on the classification of the known samples using the weight-modified Knn. The simplest is shown in equation 1:

$$Fitness = (TotPats - CorrectPats)/TotPats \quad (1)$$

where $TotPats$ is the number of total patterns to be examined and $CorrectPats$ is the number of patterns correctly classified by the weight-modified Knn. While this function gives reasonable performance, it does not generate a weight setting that gives the *maximum* class separation. Instead, it only gives a weight setting that optimizes separation based on the threshold for rating each example as a member of one class or another. Equation 2 takes both into account, giving a better optimization:

$$Fitness = \gamma \frac{TotPats - CorrectPats}{TotPats} + \delta \frac{nmin/K}{TotPats} \quad (2)$$

where $nmin$ is the cardinality of the near-neighbor minority set (i.e., the number of neighbors that were *not* used in the subsequent classification) and k is the number of nearest neighbors. This gives credit for not only getting the examples to be classified by a majority of the nearest neighbors, but also for getting the majority to the highest value possible (K). The constants γ and δ are used to tune the processing of the algorithm, presently 2 and 0.2 respectively.

3 Results

3.1 Generated Data Tests

As described in the previous sections, we used Equation 2 as the fitness function with a population of strings where each string used 8 bits to represent a single weight to modify the Knn. The actual length of the weight string (chromosome) depended on the number of features to be encoded. All work reported in this section was done using either the Genesis [Schraudolph and Grefenstette 1992] program or modifications to that program, and a Knn written locally, with K=5.

Our first work used generated data sets of which Table 1 is typical. The table represents a template from which we could generate as many examples as we needed. The expression $\{0.0, 1.0\}$ represents *uniform* random values generated in the range of 0.0 to 1.0. The expression $0.5 + 5 * [f3]^2$ represents a generation expression where [f3] means that the present value of the f3 field is used in the calculation. Thus fields f1 and f7-f15 represent uniform random noise generated in the range $\{0.0 - 1.0\}$, f2 – f6 are fields that can be used to distinguish the eight classes A–H, where fields f4 and f5 make use of values generated in f3. Note that using *uniform* random values up to contiguous field boundaries (in contrast to *normally* distributed values with a specified mean and variance) yields a dataset much

[5]We did not reproduce their published results as we did not not have access to their data.

# of unknown patterns	Errors normalized Knn	Errors 0/1 weighted GA-Knn	Errors 0-10 weighted GA-Knn	Errors 0-10 weighted GA-Knn after selection
1200	35.93%	6.40%	4.13%	3.0%

0-10 weights no selection filter	0.000	5.154	4.656	7.622	0.000
	4.248	0.000	0.000	0.000	0.000
	0.000	0.000	0.000	0.000	0.000

Table 2: Results on Dataset 1

harder for a Knn algorithm to classify than (for example) for an algorithm based on discrete thresholds. String length for this data set was $(15 * 8) = 120$ bits. The population size was 50.

The GA was "trained" on a set of 30 examples of each of the 8 classes for a total of 240 examples. It was then presented with a set of 1200 new examples generated from the same template. The results of using the weighted-Knn with weights generated in the range of $\{0.0 - 10.0\}$ are shown in Table 2. These results are an average of 3 runs. Note that in classifying the training data, the Knn evaluation function did not use the point being classified as its own "neighbor". This "leave-one-out" strategy was used make the training data resemble the test data more closely, since a test datum will not have itself as a near neighbor during Knn evaluation.

Note the distinct improvement of the weighted-Knn over the standard and normalized Knn. Further note that the results of 0/1 and 0-10 weightings are quite similar. The results of the 0-10 weighting would have been better, but appeared to settle at a local minimum. This was likely due to the small signal/noise ratio of the artificially generated, uniformly random dataset. This is supported by the experiment in which we ran the 0/1 weighting as a feature selection filter followed by the 1-10 weightings, where we obtained a much better result (3% error).

3.2 Exploiting Additional Non-Independence of Fields

Having established the usefulness of the approach on test data, we examined the search capabilities of a weighted-Knn on a different kind of data, one not searchable by a regular Knn. We investigated ways to use the correlations between fields in improving classifications. Of course, nearest neighbor methods are sensitive to some inter-field correlations, but sometimes not in the sharpest sense. We wondered, for example, if we created a data set in which a correlation between two apparently non-discriminatory fields was important, could the GA be made to discover the usefulness of that pairing in our weighted Knn scheme. Clearly, what we could *not* do was to create a field for every combination of pairs of fields. This is because, for high dimensionality data sets, the size of the chromosome would be quite large (1 gene for each feature as well as for each feature *pair*) and potentially with non-useful information. Instead, we coded some special fields, termed "index fields", in the genome representing pairs of field indices (instead of weights) plus additional weight fields to apply to these index-pair-generated terms. Thus, for example, one 8-bit position could contain 4 bits indexing field 9, the other four bits indexing field 5. The evaluation function was modified such that values in the fields pointed to by the two indices would be passed through some pre-determined function (in this case just simple multiplication). This essentially creates the product of these two fields as a new attribute of the individuals to be classified. The GA can then search for a weight for that product field. Only a tiny subset of all possible pairs of fields is being examined by the GA at any point, but whenever crossover or mutation alters an index pair, the GA essentially begins to search for a weight for that new pair. For the runs to be discussed, only two weights were possible for each index pair, represented by a single bit in the index part of the chromosome. The weight was either set to 10.0 or 0.10 depending on whether the weight bit was on or off. We have conducted experiments where the GA actually searches through a larger weight space as well, but this simpler approach seems to work very well for these test data. If the GA chooses either index to be 0, then that index field is not used in the evaluation function, thus allowing the GA to select for no index terms. The search strongly discourages the use of the same index twice in a pair (e.g. 3,3) or the repetition of a pair in a string (e.g. 2,5,5,2), by assigning a very bad fitness to any string with such a pair.

We created some example data sets in which the fields would individually not allow good classification, but taken as pairs under multiplication would be significant. Table 3 is an example of such data.

The data all consist of 15 uniform random values in the range $\{0.0 - 1.0\}$, and two "hidden features" consisting of multiplication of the f1 and f2 or the f3 and f4 fields; that is, the data were sieved to make the products significant, but no products actually appear among the data values. The goal was to see if the GA, using the approach of searching simultaneously for index pairs and weights on that pair, would work effectively. We modified the evaluation function as described, and allowed for 3 index mpairs at the end of each string which indicated the index pair and the weight associated with that pair. We trained the algorithm on 10 examples from each of the 8 classes (80 total) then tested the trained algorithm on 1200 newly generated test sets. The results of running this modification are shown in Table 4. The Knn alone performed at around

Class	Feature 1-15	Hidden Feature 1
1	{0.0-1.0}	$0.42 <$ [f1]$*$[f2] < 0.44
2	{0.0-1.0}	$0.42 <$ [f1]$*$[f2] < 0.44
3	{0.0-1.0}	$0.42 <$ [f1]$*$[f2] < 0.44
4	{0.0-1.0}	$0.42 <$ [f1]$*$[f2] < 0.44
5	{0.0-1.0}	$0.46 <$ [f1]$*$[f2] < 0.48
6	{0.0-1.0}	$0.46 <$ [f1]$*$[f2] < 0.48
7	{0.0-1.0}	$0.46 <$ [f1]$*$[f2] < 0.48
8	{0.0-1.0}	$0.46 <$ [f1]$*$[f2] < 0.48

Class	Hidden Feature 2
1	$0.54 <$ [f3]$*$[f4]< 0.56
2	$0.58 <$ [f3]$*$[f4]< 0.60
3	$0.62 <$ [f3]$*$[f4]< 0.64
4	$0.66 <$ [f3]$*$[f4]< 0.68
5	$0.54 <$ [f3]$*$[f4]< 0.56
6	$0.58 <$ [f3]$*$[f4]< 0.60
7	$0.62 <$ [f3]$*$[f4]< 0.64
8	$0.66 <$ [f3]$*$[f4]< 0.68

Table 3: A test using the indexing concept

Algorithm	Pop. size	Trials	Errors, training set	Errors, 1200 patterns
weighted Knn with with Index	50	25,600	0.0%	0.0%
Original Knn				88.17%

Table 4: Results of the Indexed feature experiment.

at 88% errors for the unknowns. The index algorithm did indeed find the appropriate indices in all of several runs and got the error down to 0.0% by finding the two index pairs (1,2) and (3,4), setting their weights to 10, and dropping all other weights to low levels.

3.3 Rhizosphere Data, a Real-World Example

Having established the usefulness of the weighted-Knn on various test data, we explored classification of the rhizosphere data already discussed in Section 1. Each data element was a 96-bit vector, where a positive result was a 1 and a negative result, a 0. Here, the genome string length was a considerably longer $(8 * 96) = 768$ bits. We had 324 examples, about 100 from each of the three sets. We trained the GA in the following manner. We used 100 samples from each of the three groups of 300 test individuals, but trained the Knn using the "leave-one-out" strategy. That is, we trained on all the sample data but at each point in the test, the test sample was not included in the Knn.

# of patterns	Errors, original Knn	Errors, 0/1 weighted Knn	Errors, 0–10 weighted Knn
test 60	27.72%	35.00%	20.10%
training 300	29.00%	19.58%	18.00%

Table 5: Results on Rhizosphere data

This simulates a leave-one-out strategy in the training process, training on all but one sample in the population. Testing of these weights was then performed on a random selection of 60 samples from the original 300. In all, 10 sets of these samples of 60 were tested and averaged for the results shown in Table 5. However, note that the 0/1 weightings are the result of a single run.

There are a number of things to note. First, the weighted-Knn performed significantly better than the unmodified Knn. Second, the 0/1 weighting scheme no longer performed adequately on such a large data set with considerable noise. Third, while the error rates were relatively high, the results were deemed quite good by the rhizosphere researchers. Therefore, the algorithm performed adequately on the rhizosphere data given the evaluation criteria described above. Of course, these are new data and other analyses applied to this data have not yet yielded criteria for classification of this data, therefore it is not clear how much better we can get. See Section 5 for more discussion on this point.

4 The Need for Speed, Parallel Processing

While we had chosen Genesis for its portability, speed and proven usefulness, its running time on the rhizosphere data was long, on the order of 14 *days* on a SPARC workstation for the results shown in Table 5. This was unacceptable for all but the simplest experiments. It was clear that some means would be necessary to increase performance if the work was to continue. The approach we took was to apply parallel processing to the problem.

There are many ways to exploit the parallelism inherent in GA searches. Large parts of the proceedings of the last four International Conferences on Genetic Algorithms and their Applications have been devoted to "Parallel Genetic Algorithms." For example, under the specific appelation "Parallel Genetic Algorithms", Grefenstette et. al. [Pettey, Leuze and Grefenstette 1987] assigns subpopulations to separate processors, with occasional migration of selected strings among processors. Bianchi and Brown [Bianchi and Brown 1992] have recently

Machine	Pop. Size	# of Trials	# of Processor Elements	Time dd:hh:mm
SPARC 4/390	50	29,000	1	14:19:02
GP1000	50	46,000	51	1:14:47
TC2000	50	39,000	6	1:16:14
TC2000	50	39,000	11	1:00:58

Table 6: Speed increases using micro-grained parallelism. All on Training Set Size of 240.

done a nice job of comparing a variety of such approaches. These approaches typically require only limited inter processor communication, and assist in avoiding premature convergence on local optima, while assuring that fit individuals from subpopulations ultimately can have an influence on the global search. Tanese [Tanese 1989], calls this same sort of approach a "distributed" genetic algorithm.

The "distributed" or, sometimes called "fine-grained" approach, as used, for example, in some of our previous work [Sannier and Goodman 1987] and that of others such as Muehlenbein [Muehlenbein 1989] and Manderick and Spiessens [Manderick and Spiessens 1989] distributes individuals in a landscape (sometimes with spatially-varying environments) and biases reproduction according to spatial location of individuals.

The approach described below is simple, builds on existing tools, and produces nearly linear speedup for the application at hand. For this application, the evaluation of the fitness of a string requires much more time than the remainder of the GA operations, so it is possible to use what we shall call *micro-grained parallelism*. In this approach, a single population is kept, so that the output is exactly what would result from a classical serial GA. Sets of strings are simply passed out to the individual processors for evaluation *only*. So long as the evaluation of fitness dominates the rest of the GA calculations, this technique can produce a nearly linear speedup in GA performance with the number of processors available to calculate fitnesses. We therefore implemented this micro-grained parallelism first, since we could implement it easily within a nearly unmodified Genesis program and had access to a parallel processor. Our first set of changes modified the appropriate part of the Genesis [Schraudolph and Grefenstette 1992] code to run on the Butterfly series of parallel machines. We had available two classes of Butterfly, the GP1000, consisting of 68020 nodes, and the TC2000, consisting of 88000 nodes. The change in running time for the rhizosphere data is shown in Table 6.

A single SPARC 4/390 server took more than 14 days to get 29,000 generations and a 27% error rate. The GP1000, using 51 68020 processors (1 for each string in the population and one master node) did 46,000 generations in about $1\frac{1}{2}$ days. The TC2000 with 6 nodes did about the same, 39,000 generations in $1\frac{1}{2}$ days; with 11 nodes, the time was reduced by 40%, a near linear speed-up. Accounting for the differences in processor speed, all of these runs show essentially linear speed-up (proportional to N-1 actually) for this GA problem in which the fitness evaluation dominates the computation.

However, because one of our goals was to make the changes as portable as possible, we have since re-coded our parallel changes in the P4 [Butler and Ewing 1992] language. P4 allows for compatibility across a number of shared memory parallel architectures, making the implementation more portable. Furthermore, P4 allows operation in a distributed environment, such as a network of SPARC workstations on an ethernet. We have gotten the same results on the Butterfly architectures using the P4-code algorithm and are now experimenting with running the algorithm on a distributed SPARC network.

4.1 "Nice" Parallelism

As mentioned above, many authors have investigated the usefulness of coarse-grained parallelism, as a means of promoting efficient search while avoiding premature convergence. We are working on adding to our P4 implementation a coarse-grained approach to work with or to supplant our existing micro-grained approach.

Furthermore, we are investigating the practical issues of running the P4 implementation on a distributed network of heterogeneous workstations, where those workstations exist in a university computing environment. We are developing robust parallel GA's which allow computation in an unstable distributed workstation environment, with minimal effect when a computing node is lost. Issues that we need to address include:

- Monitoring station usage so as to not "hog" resources,
- Deciding to drop a node when it becomes too loaded, with as little overhead as possible,
- Caching intermediate results (if possible) when a node goes down, for possible restart or transfer, but allowing all other nodes to continue with the GA regardless.

5 Discussion

The approach of doing feature extraction and warping of the feature space to optimize classification has been demonstrated to be very powerful in dealing with some artificially-generated data sets. Furthermore, our first experience with real-world data, in this case the rhizosphere data, has shown reasonable promise. Typical results on this data via weighted Knn were a 20%

error rate, a 25% reduction in error over the standard Knn's 27% error rate. The 0-10 weightings also appear to work better than an approach similar to the 0/1 weighting approach of Siedlecki and Sklansky, based on the statistical evaluation using the "leave-one-out" approach. There are a number of cautions, however. First, the rhizosphere data is indeed noisy, and no lower error classification based on these data may exist. In the future we plan to assess some standard archival data sets with known classification properties, in order to better demonstrate the utility of our approach. Second, it is clear that despite experimentation with about 20 evaluation functions, we have not yet found one that allows us to reach the global optimum in reasonable time. We know that we can achieve somewhat lower classification error rates using the weighted Knn approach (e.g., the 0-10 scaling), since we have been able to demonstrate the existence of lower error-producing weight sets by first using 0/1 selection, and then using 0-10 scaling. The evaluation functions used in the 0-10 weighted-Knn alone have not yet been able to find those better solutions. Our evaluation functions are still converging prematurely on local optima given a reasonable amount of computation/time. We are continuing to explore various scenarios to improve the performance of the system, and these explorations are the subject of intense research in our laboratories.

Furthermore, the practical problems of running such large strings and using computationally expensive evaluation functions has forced us to investigate parallel processing issues. Our first experiences were using parallel machines (BBN Butterflies), but this is not the most attractive approach for production runs as these are shared resources with many clients. We are therefore exploring the distributed workstation approach, but hoping to provide machinery that will allow our algorithms to run such that we will not interfere with normal use. To date, we have experience running the algorithm on distributed workstations (Sparc and HP) with the same speed-up properties using the P4 implementation, but need to address using workstation resources that are not already occupied, and how to "unoccupy" a workstation that becomes busy. Both sides of this work still present interesting possibilities for further research.

References

[Bianchi and Brown 1992] Bianchi and Brown, "Parallel Genetic Algorithms on Distributed-Memory Architectures", Technical Report 436, Computer Science Department, University of Rochester, Rochester, New York, August, 1992.

[Butler and Ewing 1992] Butler, Ralph, and Lusk Ewing, "User's Guide to the p4 Programming System", Technical Report ANL-92/17, Argonne National Laboratory, Argonne, IL, 1992.

[Devivjer and Kittler 1982] Devijver, P. A. and J. Kittler, Pattern Recognition: A Statistical Approach, Prentice Hall International, London, 1982.

[Kelly and Davis 1991] Kelly, James and Lawrence Davis, "Hybridizing the Genetic Algorithm and the K Nearest Neighbors Classification Algorithm", Proceedings of the 4th International Conference on Genetic Algorithms and their Applications, Morgan Kaufman Publishers, 1991.

[Manderick and Spiessens 1989] Manderick, B. and Piet Spiessens, "Fine-Grained Parallel Genetic Algorithms", Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications, Morgan Kaufman Publishers, 1989.

[Muehlenbein 1989] Muehlenbein, H." Parallel Genetic Algorithms, Population Genetics, and Combinatorial Optimization", Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications, Morgan Kaufman Publishers, 1989.

[Pettey, Leuze and Grefenstette 1987] Pettey, Chrisila B., Michael R. Leuze, and John J. Grefenstette, "A Parallel Genetic Algorithm", Genetic Algorithms and their Applications, Proceedings of the 2nd International Conference on Genetic Algorithms, 1987, pp.155-161.

[Tanese 1989] Tanese, R. "Distributed Genetic Algorithms", Proceedings of the Third International Conference on Genetic Algorithms and their Applications, Morgan Kaufman Publishers, 1989.

[Sannier and Goodman 1987] Sannier, A. and E. Goodman, "Genetic Learning Procedures in Distributed Environments," Proceedings of the Second International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates, 1987.

[Schraudolph and Grefenstette 1992] Schraudolph, N and J. Grefenstette, "A User's Guide to GAucsd 1.4", Technical Report CS92-249 from Technical Reports, CSE Department, UC San Diego, La Jolla, CA 92093-0114, 1992.

[Siedlecki and Sklansky 1990] Siedlecki, W. and J. Sklansky, "A note on genetic algorithms for large-scale feature selection", *Pattern Recognition Letters*, 10(1989), 335-347.