International Journal of Systems Science, 1998, volume 29, number 2, pages 207–212

Self-crossover—a new genetic operator and its application to feature selection

NIKHIL R. PAL†, SUCHISMITA NANDI‡ and MALAY K. KUNDU‡

Crossover is an important genetic operation that helps in random recombination of structured information to locate new points in the search space, in order to achieve a good solution to an optimization problem. The conventional crossover operation when applied on a pair of binary strings will usually not retain the total number of 1s in the offsprings to be the same as that of their parents, but there are many optimization problems which require such a constraint. In this article, we propose a new crossover technique called 'self-crossover', which satisfies this constraint as well as retaining the stochastic and evolutionary characteristics of genetic algorithms. This new operator serves the combined role of crossover and mutation. We have proved that self-crossover can generate any permutation of a given string. As an illustration, the effectiveness of this new technique has been demonstrated for the feature selection problem of pattern recognition. Performance of self-crossover for feature selection is also compared with that of ordinary crossover.

1. Introduction

Genetic algorithms (GAs) are probabilistic heuristic search processes based on natural genetic system. They are capable of solving a wide range of complex optimization problems using three simple genetic operations (selection–reproduction, crossover and mutation) on coded solutions (strings–chromosomes) for the parameter set (not the parameters themselves) in an iterative fashion. There are several interesting features which make GAs so popular. GAs consider several potential solution points in the search space simultaneously, which reduces the chance of convergence to a local optima. GAs use only the pay-off or penalty function (objective function) called the fitness function and do not need any other auxiliary information. GAs are theoretically and empirically proven to provide a robust search in complex spaces (Goldberg 1989) even if the functions are not smooth or not continuous. Such functions are very difficult (sometimes impossible) to optimize using calculus-based methods.

Normally, GAs use simple crossover which in case of binary coded solution space, may change the total number of 1s in the offsprings. Crossover and mutation are usually adequate for solving a wide class of optimization problem. However, there are families of problems which require to maintain certain constraints on the number of 1s (in the case of binary coding) in the strings. Some such problems are selection of an optimal subset of k, $k > 0$, features for pattern recognition application and selection of a subset of k, $k > 0$, data points for designing a nearest-neighbour (NN) classifier. In addition there are several other combinatorial optimization problems such as the travelling-salesman problem, which require some such constraints.

In this paper we show that, if the genetic recombination operator is chosen properly, then GAs can be competitive with the best known techniques for a large set of problems with this type of constraint. To achieve this we propose a new crossover operation. We call it 'self-crossover' because of its similarity with conventional crossover. The self-crossover is performed with a single parent string instead of a pair of strings. The new operator preserves the probabilistic and evolutionary characteristics of GAs. We use self-crossover for solving the feature selection problem. We compared the performance of self-crossover with the conventional crossover for feature selection using an 18-dimensional data set.

2. Genetic algorithms

GAs (Goldberg 1989) are general-purpose optimization algorithms which can solve problems resistant to other known optimization methods. This search technique is population based which evolves from generation to generation. The criterion of 'survival of the fittest' provides evolutionary pressure for populations to grow with increasingly fit individuals. Although there are many variants, the basic mechanism of GA (conventional GA) consists of the following steps.

Step 1. Start with an initial population (a set of strings - chromosomes).

Step 2. Evaluate fitness of every string and selection of appropriate candidate strings to form the mating pool.

Step 3. Perform crossover and mutation.

Step 4. Repeat Steps 2 and 3 until the system ceases to improve or some stopping criterion is reached.

Each string in the population is represented by a fixed-length coded string. Although different coding schemes are possible, we confine ourselves to binary coding only. Selection or reproduction creates the population for the next generation using a probabilistic selection process which gives a string with higher fitness a greater chance of selection. Mutation corresponds to flipping (probabilistically) one or more bits of an individual string. Mutation increases the variability in the population and ensures that the probability of attaining any point in the search space is greater than zero. The simplest implementation of crossover selects two parents (randomly) from the mating pool and then, after choosing a random position, each parent string exchanges its tail at that position. The resulting offsprings form the subsequent population for the next generation.

There are several variations of conventional GA. Kuo and Hwang (1993) proposed a new selection operator, named *disruptive selection*, which rests on the fact that a worse solution also contains information that is useful for biasing the search. So, unlike directional selection, disruptive selection tends to eliminate offsprings with moderate fitness values for the next generation. Syswerda (1989) presented a new crossover, *uniform crossover*, and empirically showed its superiority to one-point crossover and two-point crossover. For uniform crossover, a crossover mask is considered. The mask is a string of bits of the same size as that of the chromosomes to be crossed. If any mask bit is '1', then parents will exchange their corresponding bits; otherwise they will retain their original bits, resulting in two new offsprings.

Bhandari *et al.* (1994) proposed a new mutation operator called *directed mutation*, which is motivated from the concept of induced mutation in biological systems. The generation of the new string is directed by the potential strings of some previous populations. At each generation a new string is generated on the basis of the information gathered from the gradient (or some crude estimate of the gradient) of the fitness function in the search space. They empirically showed that GAs with this new mutation operator needs fewer iterations than the ordinary GAs to obtain a good solution. Homaifar *et al.* (1993) devised a new crossover technique named *matrix crossover* (MC) to solve the travelling-salesman problem using GA. This new operator is a natural extension of the conventional single-point or double-point crossover to matrix strings. MC operates on matrices and deals with column positions rather than bit positions. MC just exchanges all the entries of the two parents determined by the crossover site (the chosen column position).

3. Feature extraction

Many workers (Fukunaga and Koontz 1970, Foley and Sammon 1978, Johnson and Wichern 1988, Pal 1992, De *et al.* 1997, Pal and Chintalapudi 1997) dealt with the problem of feature vectors extraction in q space from data in p space where $q \ll p$. Feature extraction is used for two different but somewhat related problems: dimensionality reduction and visual display. Reducing p to $q \ll p$ reduces the space and time complexity of computations that use the extracted data. Another issue is that some of the original p features may result in confusion in the feature space. Can we do just as well (with respect to a problem under consideration) with $q < p$ new features derived from the original features? For example, transformed features can work better than the original data for purposes such as classifier design. We show that exactly this happens with the *mango-leaf* data for nearest prototype (NP) classifier.

Another important use of feature extraction is to obtain two-dimensional ($q = 2$) displays of data for visual exploratory data analysis which helps us to guess the clustering tendencies, that is the distributional densities through visual inspection of scatter plots of two-dimensional extracted data.

Several techniques such as principal components (PCs) (Johnson and Wickern 1988) and the (Foley–Sammon 1978) algorithm (FSA) for extracting important feature subsets are available. Fukunaga and Koontz (1970) pointed out that one of the disadvantages of PCs analysis (Johnson and Wichern 1988) is that this technique does not necessarily produce the best features for pattern classification. Conversely, Foley and Sammon (1978) observed that the best extracted features

for discrimination may not serve adequately for visual display. PCs and the FSA extract features using different criterion. PCs can be used to select features that can account for a large share of variance while the FSA extracts features preserving interpoint distances.

Feature selection, a special type of feature extraction, on the other hand, selects a subset of the original features so that several pattern recognition tasks can be performed with the reduced set of features to a satisfactory level. For reducing the dimension of the feature space, we should eliminate those features which are less important or redundant for *discriminating* the classes. We can achieve this through ranking of features according to their importance in discrimination. In our investigation we shall call a feature subset A better than a subset B if the performance of an NP classifier with the subset A is better than that with B. There exist a number of methods for feature extraction–ranking, each having its own merits and demerits (Foley and Sammon 1978, Johnson and Wichern 1988, Kelly and Davis 1991, Pal 1992).

We now mention some earlier attempts to solve the feature selection problem using GAs. Siedlecki and Sklansky (1989) used a k-NN rule to find a small subset of features for which the classifier's performance does not deteriorate below a specified level. They did this by constructing a GA chromosome consisting of a binary string whose length equalled the number of features. If a bit is '1', that feature is selected for evaluating the performance of the classifier.

Kelly and Davis (1991) and Punch *et al.* (1993) solved the same feature selection problem using GAs. Unlike Siedlecki and Sklansky they multiplied each feature by a real-valued weight and then used that weighted feature for computing distances required for implementation of k-NN classifier. GAs have been used to learn these weights. Features with high values for the learned weights are considered important features and vice versa.

We make the following remarks about the preceding GA-based approaches. These methods cannot be used to select a fixed number of good features, that is say q, good features. The algorithm may terminate at a point where the total number of 1s in the solution string may not be equal to q (Siedlecki and Sklansky 1989), while for other two methods (Kelly and Davis 1991, Punch *et al.* 1993) those q features having highest weights can be selected. However, this can create another problem. Suppose that there is a feature which is more or less constant for all classes. For this feature, irrespective of the weight, the classifier performance is not going to be changed. Since GAs are probabilistic search techniques, the algorithm might terminate at a point with high weight for this indifferent feature, thereby indicating a false importance!

4. Self-crossover

Unlike the conventional crossover mechanism, the self-crossover mechanism alters the genetic information within a *single* potential string selected *randomly* from the mating pool to produce an offspring. This is done in such a manner that the stochastic and evolutionary characteristics of GAs are preserved.

Let

$$S = 00010010011001011011$$

be a string of length 20 slected from the mating pool. For self-crossover, first we select a random position $p(0 < p < L)$ and generate two substrings s_1 and s_2 : s_1 = bits 1 to p of S and s_2 = bits $p+1$ to L of S. Now we select two random positions p_1, $0 \leqslant p_1 \leqslant p$, and p_2, $0 \leqslant p_2 \leqslant L - p$. Then four substrings are generated as follows:

$$s_{11} = \text{bits 1 to } p - p_1 \text{ of } s_1,$$

$$s_{12} = \text{bits } p - p_1 + 1 \text{ to } p \text{ of } s_1,$$

$$s_{21} = \text{bits 1 to } L - p - p_2 \text{ of } s_2,$$

$$s_{22} = \text{bits } L - p_2 + 1 \text{ to } L \text{ of } s_2.$$

Using operations similar to crossover we generate $S^1 = s_{11}|s_{22}$ and $S^2 = s_{21}|s_{12}$. Finally, the self-crossovered offspring of S is generated as $S_1 = S^1|S^2$. It is easy to see that the numbers of 1s in S and S_1 are the same. We now explain it with the example string S of length 20:

$$S = 00010010011001011011.$$

A random position, $p = 9$, is selected for splitting the string into two substrings (s_1, s_2) as follows:

$$s_1 = 000100100,$$

$$s_2 = 11001011011.$$

Now two random positions, $p_1 = 4$ and $p_2 = 7$, are selected for s_1 and s_2 respectively. After splitting s_1 and s_2 at the fourth and seventh positions respectively, we get

$$s_{11} = 00010,$$

$$s_{12} = 0100,$$

$$s_{21} = 1100,$$

$$s_{22} = 1011011.$$

The two new substrings S^1 and S^2 are then obtained as

$$S^1 = 000101011011,$$

$$S^2 = 11000100.$$

Finally, the offspring S_1 is generated by concatenating S^1 and S^2 as

$$S_1 = 00010101101111000100.$$

Thus, self-crossover exchanges substrings s_{12} and s_{22}. If the parent string consists of all 0s or all 1s, the offspring generated through self-crossover will resemble its parent because of the underlying constraint on the total number of 1s in the string. It is also clear that, if we do not start the GA with an all-'1' or all-'0' string, the GA with the self-crossover technique will never generate such strings as offsprings. So, self-crossover will evolve new offsprings as iterations go on.

We can see clearly that mutation is ineffective in producing such constrained offsprings, but self-crossover can regenerate any lost genetic information. So we may avoid mutation when we use this new technique in constrained GA application.

Next we show via a Lemma that self-crossover (without mutation) can generate any target string.

Lemma: *Given a string of symbols, self-crossover operations can generate any arbitrary permutation of the symbols.*

Proof: We can represent any arbitrary string P by $P = S_1|S_2|S_3|S_4$ where S_i represents a subsequence of symbols. S_i can be empty sequence as well. Now, if we are able to prove that a parent string $P = S_1|S_2|S_3|S_4$ can produce an offspring $O = S_1|S_2|S_3|S_4$ using a finite number of self-crossover operations, then we can iterate the process to generate a sequence of self-crossover operations to reach any target offspring.

The position for splitting P is chosen such that two subsequence s_1 and s_2 are formed as

$$s_1 = S_1|S_2$$

and

$$s_2 = S_3|S_4.$$

Now two random positions for s_1 and s_2 are selected such that after splitting of s_1 and s_2 at these two positions we get

$$s_{11} = S_1,$$

$$s_{12} = S_2,$$

$$s_{21} = S_3,$$

and

$$s_{22} = S_4.$$

So the resultant child is obtained as

$$P_1 = S_1|S_4|S_3|S_2.$$

We apply once more the self-crossover operation on intermediate child P_1. Now the random position for splitting P_1 is chosen such that

$$s_1 = S_1|S_4$$

and

$$s_2 = S_3|S_2.$$

Again we select two random positions for s_1 and s_2 such that after splitting of s_1 and s_2 at these two positions we get

$$s_{11} = S_1,$$

$$s_{12} = S_4,$$

$$s_{21} = S_3|S_2$$

and

$$s_{22} = \text{empty sequence.}$$

The offspring now becomes

$$O = S_1|S_2|S_3|S_4$$

which is simply what we wanted to produce. □

Since S_1 could be a null string, so any symbol from the parent string can be brought at the beginning of the offspring through substring S_3 by two successive self-crossover operations. The lemma also ensures that any substring consisting of symbols starting from the beginning of a parent string can be preserved in the child through substring S_1. Hence, any target permutation can be grown from the left side. Proceeding this way in the terminal phase of the process S_2 and S_4 will be empty; S_1 will contain the entire target substring except the last symbol which will be in S_3.

Note that the lemma does *not* say that there is no need for mutation in the GA with the self-crossover technique. It simply says that for combinatorial problems such as the travelling salesman problem, use of self-crossover without mutation can generate all possible valid solution strings. For problems such as feature selection, data editing for the NN classifier where we want to select the best subset of features or data points of a pre-fixed cardinality, self-crossover without mutation is sufficient. In fact, conventional mutation for such problems may produce invalid solutions, that is it may generate a substring of arbitrary cardinality, not equal to the pre-fixed cardinality.

At first sight, it might appear that self-crossover is simply a parallel random search, but this is not the case for two reasons. Self-crossover is done only on a randomly selected subset of strings and self-crossover does not alter the substring s_{11}. It exchanges on s_{22} and s_{12}. Consequently, the evolutionary characteristics of the GA are preserved. The similarity between the parents and offsprings will be more if we take $p_1 = p_2 = p'$ (say) equation to a random number selected between 1 and $\min(p, L-p)$, that is $0 < p_1 = p_2 = p' < \min(p, L-p)$. In this case, the bits in positions 0 to p' and bits from $p+1$ to $L-p'$ will remain unaltered. Consequently, the evolution of the process will be faster.

5. Algorithm for feature selection using the genetic algorithm

Let $X = \{x_1, x_2, \ldots, x_n\}$, $x_i \in R^p$; the p components of x_i correspond to values of p features $F_1, F_2, \ldots, F_p$. We have to select a set of q features, say $\{F_{i1}, F_{i2}, \ldots, F_{iq}\} \subset \{F_1, F_2, \ldots, F_p\}$ such that the selected feature subset can do different pattern recognition jobs well. To use the GA for feature selection we need an objective (fitness) function to guide the feature selection process. The fitness function should reflect the performance of the reduced data set for different pattern recognition tasks. For an unlabelled (where class information is not available) data set the fitness function may reflect the performance of a clustering algorithm, while for labelled data (where class information is available) the fitness function may be defined to measure the performance of a classifier. Here we consider the latter case and the fitness function is defined to be the performance of the NP classifier. Thus the fitness function f is given by $f(F_{i1}, F_{i2}, \ldots, F_{iq}, Y_q, V)$ equal to the number of correct classification, where $Y_q = \{y_1, y_2, \ldots, y_n\}$, $y_i \in R^q$, and the kth component of y_i, that is y_{ik}, is equal to some lth component x_{il} of $x_i \in R^p$, $V = \{v_1, v_2, \ldots, v_c\}$, and $v_i \in R^q$ is the set of q-dimensional prototypes defined by

$$v_i = \frac{1}{|C_i|} \sum_{k \in C_i} y_k, \tag{1}$$

where c is the number of classes and C_i denotes the ith class.

Note that the prototypes may be generated in many other ways.

We now represent a feature subset by a binary string of length p. A set of M binary strings of length p and cardinality k is taken as the initial population where the cardinality of a binary string is defined as the total number of 1s in the string. If the jth feature is selected for the chosen subset, then the jth position of the binary string is filled up with a '1', or otherwise with a '0'. Thus, a string of cardinality k denotes a feature subset of size k. Corresponding to each string the fitness function value f is calculated. The best member of the population is stored. The mating pool can now be generated by replicating potential strings (potential as reflected by their fitness values) more times. Some randomly selected members or all members of the current generation then undergo the self-crossover mechanism to generate the population for next generation. We find the current best member of the newly generated population, compare it with the previously stored best member and finally store the best of these two. The entire process is repeated for a desired number of times or until we find no improvement in the fitness value for several generations.

6. Results and discussion

We used a data set named mango-leaf. Mango-leaf is a $p = 18$ dimensional data with 166 points. It has three classes representing three kinds of mango. The feature set consists of the following measurements: area A, perimeter Pe, maximum length L, maximum breadth B, petiole P, length + petiole $L + P$, length/petiole L/P, length/maximum breadth L/B, $(L + P)/B$, A/L, A/B, A/Pe, upper midrib/lower midrib, upper Pe/lower Pe and so on. The terms upper and lower are used with respect to maximum breadth position. Although ordinary crossover may not select a feature subset with the desired number of features, we implemented if for comparison. In this investigation, the population size is taken as 20 and the process is run for 200 generations. For the implementation with ordinary crossover, we used a mutation probability of 0.1.

The Table shows the best results obtained by self-crossover and ordinary crossover for the mango-leaf data. Note that the final population with ordinary crossover had all strings with either five or six non-zero elements and the best feature sets with cardinality five and six are reported. In the Table the column headed Match

Results with the mango-leaf data

Number of features extracted	Our result with self-crossover: Feature subset	Number of generations	Match	Results with ordinary crossover: Feature subset	Number of generations	Match
1	9	11	103	—	—	—
2	9, 14	18	127	—	—	—
3	9, 13, 14	51	131	—	—	—
4	9, 13, 14, 17	80	132	—	—	—
5	9, 13, 14, 16, 17	74	133	9, 13, 14, 17, 18	15	132
6	5, 9, 12, 13, 16, 17	32	128	7, 9, 13, 14, 17, 18	35	133
18	(Full set)	—	106	—	—	—

indicates the total number of data points whose derived classification and actual classification agreed, that is the value of the objective function.

The Table reveals several interesting things. First, more features are not always good; some features may create confusion in the decision surface. For example, with all 18 features, the NP classifier can recognize only 106 poiints while, with just two features, the recognition score increases to 127. It is interesting to see that the best recognition score is obtained with only five features.

7. Conclusions

There is a class of optimization problems which requires the number of 1s in each string to be constant. Conventional crossover–mutation does not guarantee this. We have introduced a new crossover operation named *self-crossover* which preserves this constraint. Moreover, this new operator plays the combined role of mutation and as well as of the conventional crossover. The effectiveness of GAs equipped with self-crossover is demonstrated for solving the feature selection problem. Experimental investigation shows that the new operator is quite effective for the feature selection problem.

References

BHANDARI, D., PAL, N. R., and PAL, S. K., 1994, Directed mutation in genetic algorithms. *Information Sciences*, **79**, 251–270.

DE, R., PAL, N. R., and PAL, S. K., 1997, Feature analysis: neural network and fuzzy set theoretic approaches, *Pattern Recognition*, **30** (10), 1579–1590.

FOLEY, D. H., and SAMMON, J. W., 1978, An optimal set of discriminant vectors. *IEEE Transactions on Computers*, **24**, 271–278.

FUKUNAGA, K., and KOONTZ, W. L. G., 1970, Application of the Karhunen–Loeve expansion to feature selection and ordering. *IEEE Transactions on Computers*, **19**, 311–318.

GOLDBERG, D. E., 1989, *Genetic Algorithms: Search, Optimization and Machine Learning* (Reading, Mass.: Addison-Wesley).

HOMAIFAR, A., GUAN, S., and LIEPINS, G. E., 1993, A new aproach on the traveling salesman problem by genetic Algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms*, University of Illnois at Urbana-Champaign, IL, pp. 460–466.

JOHNSON, R. A., and WICHERN, D. W., 1988, *Applied Multivariate Statistical Analysis* (Englewood Cliffs, N.J: Prentice-Hall).

KELLY, J. D., JR., and DAVIS, L., 1991, A hybrid genetic algorithm for classification. *Proceedings of the International Joint Conference on Artificial Intelligence*, Darling Harbour, Sydney, 1991, Vol. 1, pp. 645–650.

KUO, T., and HWANG, S. Y., 1993, Genetic algorithm with disruptive selection. *Proceedings of the 5th International Conference on Genetic Algorithms*, University of Illinois at Urbana-Champaign, IL, pp. 65–69.

PAL, S. K., 1992, Fuzzy set theoretic measures for automatic feature evaluation: ii. *Information Sciences*, **64**, 165–179.

PAL, N. R., and CHINTALAPUDI, K., 1997, A connectionist system for feature selection. *Neural, Parallel and Scientific Computations*, 5 359-382.

PUNCH, W. F., GOODMAN, E. D., PEI, M., SHUN, L. C., HOVLAND, P., and EMBODY, R., 1993, Further research on feature selection and classification using genetic algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms*. University of Illinois at Urbana-Champaign, IL, pp. 557–564.

SIEDLECKI, W., and SKLANSKY, J., 1989, A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, **10**, 335–347.

SYSWERDA, G., 1989, Uniform crossover in genetic algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, George Mason University, Fairfax, VA, pp. 2–9.