A Review of Selection, Mutation and Recombination in Genetic Algorithms

Milena Karova[1]

Abstract – **Genetic Algorithms have been applied to a number of optimization problems. This paper presents a new analysis of principal genetic operators: selection, mutation and recombination. New methods are described. Recombination (crossover) is also a multitude of methods to create a coherent set of genes from two parent sets. GA selection perform the equivalent role to natural selection. Mutation enables the GA to maintain diversity whilst also introducing some random search behavior.**

Keywords – **genetic algorithms, selection, mutation, crossover, recombination, inversion, genotype, phenotype.**

I. Introduction

A Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome – like data structure and apply a recombination operators to these structures so as to preserve critical information. Genetic algorithms (GA) are often viewed as function optimizers, although the range of problems to which GA have been applied is quite broad.

An implementation of GA begins with a population of (typically random) chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes which represent a better solution to the target problem are “given” more chances to “reproduce” than those chromosomes which are poorer solutions. The “goodness” of a solution is typically defined with respect to the current population.

The term Genetic Algorithm has two meanings. In a strict interpretation the genetic algorithm refers to a model introduced and investigated by John Holland (1975) [5]. It is still the case that most of the existing theory for GA applies either solely or primarily to the model introduced by Holland. It comes from the fact that individuals are represented as strings of bits analogous to chromosomes and genes. In addition to recombination by crossover, we also throw in random mutation of these bit-strings every so often.

A GA is any population-based model that uses selection and recombination operators to generate new simple points in a search space. Many GA models have been introduced by researchers largely working from an experimental perspective. Many of these researchers are application oriented and are typically interested in GA as optimization tools.

[1]Milena Karova is with the the department of Computer Science, Studentska 1, Technical University Varna Email: mkarova@ieee.bg

II. Functioning of GA

The fitness or objective function is used to map the individual’s bit strings into a positive number which is called the individual’s fitness. There are two steps involved in this mapping (however in some problems these two steps are essentially accomplished as one). The first step we will call “decoding” and the second, “calculating fitness”. To understand decoding it helps to partition individuals into two parts commonly called the genotype or genome and the phenotype. These terms are borrowed from biology. The genotype, as its name implies, specifically refers to an individual’s genetic structure or for our purpose, the individual’s bit string(s).

The phenotype refers to the observable appearance of an individual (pheno comes from Greek for “to show” - phainein).

The principle of GA is simple:

1. Encoding of the problem in a binary string.
2. Random generation of a population. This one includes a genetic pool representing a group of possible solutions.
3. Reckoning of a fitness value for each subject. It will directly depend on the distance to the optimum.
4. Selection of the subjects that will mate according to their share in the population global fitness.
5. Genomes crossover and mutations.
6. And then start again from point 3.

The functioning of a GA can also be described in reference to genotype (GTYPE) and phenotype (PTYPE) notions.

1. Select pairs of GTYPE according to their PTYPE fitness.
2. Apply the genetic operators (crossover, mutation…) to create new GTYPE.
3. Develop GTYPE to get the PTYPE of a new generation and start again from 1.

III. Selection

Selection is one of the most important elements of all GA’s. Selection determines which individuals in the population will have all or some of its “genetic material” passed on to the next generation of individuals. The object of the selection method employed in a GA is to give exponentially increasing trials to the fittest individuals. The most common way in which this is accomplished is by a technique called “roulette-wheel” selection. As you will see, it is the implementation of roulette-wheel selection which necessitates positive fitness values where higher values indicate greater fitness. Roulette wheel selection gets its name from the fact that the algorithm works like a roulette wheel in which each slot on the wheel is paired with an individual in the population. This is done such that the size of each slot is proportional to the corresponding individuals fitness. It should be obvious then that maximization problems fit directly into this paradigm - larger slot implies larger fitness. Negative values are not allowed because how can you have a slot of negative size?

A common way to implement roulette wheel selection is to:

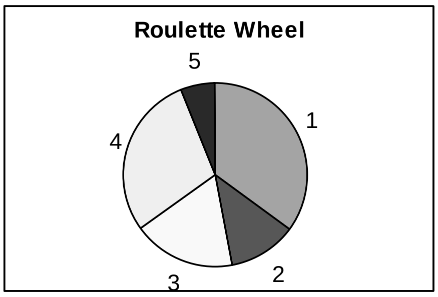
1. Sum up all the fitness values in the current population, call this value SumFitness. SumFitness is in effect the total area of the roulette wheel.

2. Generate a random number between 0 and 1, called Rand.

3. Multiply SumFitness by Rand to get a number between 0 and SumFitness which we will call RouletteValue (RouletteValue = SumFitnesss x Rand). Think of this value as the distance the imaginary roulette ball travels before falling into a slot.

4. Finally we sum up the fitness values (slot sizes) of the individuals in the population until we reach an individual which makes this partial sum greater or equal to RouletteValue [Fig. 1]. This will then be the individual that is selected.

It is not always intuitively obvious that this algorithm actually implements a weighted roulette wheel. To see that it



Individual	Fitness	Slot Size %
1	30	35
2	10	12
3	15	18
4	25	29
5	5	6
SumFittnes:	85	

Fig. 1. Roulette Wheel Selection with Five Individuals of Varying Fitness

does lets look at some extreme situations. Imagine an individual, I, whose fitness is equal to SumFitness (implying all other individuals have a fitness of zero).

Clearly no matter what number is generated for RouletteValue, I will always throw the partial sum over the top, thus having a selection probability of 1. This corresponds to a roulette wheel with just one slot.

On the other extreme, an individual, i, with fitness zero can never cause the partial sum to become greater than RouletteValue, so it has a zero probability of getting selected. This corresponds to a slot that does not exist on the wheel. All other individuals between these extremes will have a probability of throwing the partial sum over the top that is proportional to their size, which is exactly how we would expect a weighted roulette wheel to behave.

The important quality of all legitimate GA selection techniques is to reward fitter individuals by letting them reproduce more often. This is one of the important ways in which a GA differs from random search.

IV. Recombination (Crossover)

A. 1-point crossover

The traditional GA uses 1-point crossover, where the two mating chromosomes are each cut once at corresponding points, and the sections after the cuts exchanged. It is here that two individuals selected in the previous step are allowed to mate to produce offspring. Crossover is the process by which the bit-strings of two parent individuals combine to produce two child individuals.

There are many ways in which crossover can be implemented. Some of the ways are broadly applicable to all types of problems and others are highly problem specific. Here we will talk about the most primitive (but also highly effective) form of crossover, single-point crossover [Fig. 2]. Single point crossover starts by selecting a random position on the bit string, called a cut point or cross point. The bits from the left of the cut point on parent1 are combined with the bits from the right of the cut point in parent2 to form child1. The opposite segments are combined to form child2.

Thus child1 and child2 will tend to be different from either of their parents yet retain some features of both. If the parents each had high fitness (which is likely by the fact that they were selected) then there is a good chance that at least one of the children is as fit or better than either parent. If this is the case, then selection will favor this child’s s procreation, if not than selection will favor the child’s extinction.

There is of course a possibility (albeit small) that most or

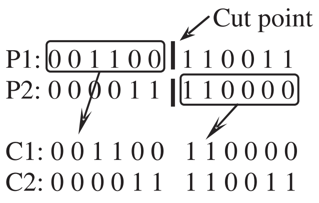

Cut point
P1: 0 0 1 1 0 0 1 1 0 0 1 1
P2: 0 0 0 0 1 1 1 1 0 0 0 0
C1: 0 0 1 1 0 0 1 1 0 0 0 0
C2: 0 0 0 0 1 1 1 1 0 0 1 1

Fig. 2. Example of single-point crossover

all of the crosses produce children of less fitness. To counter this possibility, a parameter, px – the probability of crossover, is introduced. Before crossover is performed a simulated coin is flipped that is biased to come up heads (TRUE) with probability px. If it does, then crossover is performed, if not than the parents are passed into the next generation unchanged. Since without crossover there is no hope for advancement, px is usually high ($0.5 < px < 1.0$).

However, many different crossover algorithms have been devised, often involving more than one point. [2]

B. 2-point crossover

In 2-point crossover, (and multi-point crossover in general), rather than linear strings, chromosomes are regarded as loops formed by joining the ends together. To exchange a segment from one loop with that from another loop requires the selection of two cut points, as shown in Fig. 3.

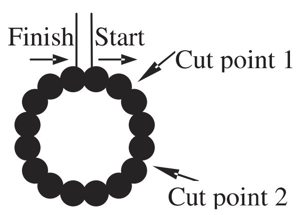


Fig. 3. Chromosome Viewed as Loop

In this view, 1-point crossover with one of the cut points fixed at the start of the string. Hence 2-point crossover performs the same task as 1-point crossover (i.e. exchanged a single segment), but is more general. A chromosome considered as a loop can contain more building blocks – since they are able to "wrap around" at the end of the string. Researchers now agree that 2-point crossover is generally better than 1-point crossover.

C. Uniform crossover

Uniform crossover is radically different to 1-point crossover. Each gene in the offspring is created by copying the corresponding gene from one of the other parent, chosen according to a randomly generated crossover mask. Where there is a 1 in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask, the gene is copied from the second parent, as shown in Fig. 4. The process is repeated with the parents exchanged to produce the second offspring. A new crossover mask is randomly generated for each pair of parents.

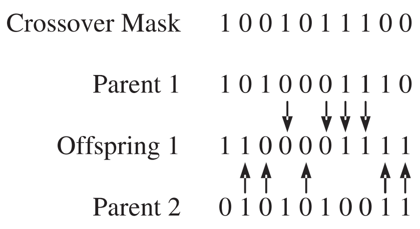

Crossover Mask 1 0 0 1 0 1 1 1 0 0
Parent 1 1 0 1 0 0 0 1 1 1 0
Offspring 1 1 1 0 0 0 0 1 1 1 1
Parent 2 0 1 0 1 0 1 0 0 1 1

Fig. 4. Uniform Crossover

Offspring therefore contain a mixture of genes from each parent. The number of effective crossing points is not fixed, but will average L/2 (where L is the chromosome length).

D. Which technique is best?

Arguments over which is the best crossover method to use still rage on. Syswerda [7] argues in favor of uniform crossover. Under uniform crossover the number of specify bit values, is equally likely to be disrupted. uniform crossover has the advantage that the ordering of genes is entirely irrelevant. This means that reordering operators such as inversion are unnecessary. GA performance using 2-point crossover drops dramatically if the recommendations of the building block hypothesis are not adhered to. Uniform crossover , on the other hand, still performs well – almost as well as 2-point crossover used on a correctly ordered chromosome. Uniform crossover therefore appears to be robust.

Spears & DeJong [6] are very critical of multi-point and uniform crossover. They stick by the theoretical analyses which show 1- and 2-point crossover are optimal. 2-point crossover will perform poorly when the population has largely converged, due to reduced crossover productivity.

In a slightly later paper DeJong &Spears [6] conclude that modified 2-point crossover is best for large populations, but the increased disruption of uniform crossover is beneficial if the population size is small (in comparison to the problem complexity), and so gives a more robust performance.

Goldberg [2] describes a rather different crossover operator, partially matched crossover (PMX), for use in order-based problems. (In an order-based problem, such as the traveling salesperson problem, gene values are fixed, and the fitness depends on the order in which they appear). In PMX it is not the values of the genes which are crossed, but the order in which they appear. Offspring have genes which inherit ordering information from each parent. This avoids the generation of offspring which violate problem constraints.

V. Mutation

Another important GA operator is mutation. Although mutation is important, it is secondary to crossover. Many people have the erroneous belief that mutation plays the central role in natural evolution. This is simply not the case. The reason is that mutation is more likely to produce harmful or even destructive changes than beneficial ones. An environment with high mutation levels would quickly kill off most if not all of the organisms. In genetic algorithms, high mutation rates cause the algorithm to degenerate to random search [Fig. 5].

Unlike crossover, mutation is a unary operator - it only acts on one individual at a time. As bits are being copied from a

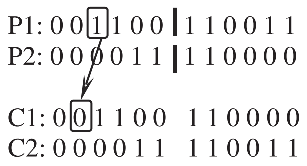

P1: 0 0 1 1 0 0 1 1 0 0 1 1
P2: 0 0 0 0 1 1 1 1 0 0 0 0
C1: 0 0 1 1 0 0 1 1 0 0 0 0
C2: 0 0 0 0 1 1 1 1 0 0 1 1

Fig. 5. Example of mutation

parent individual to a child. a weighted coin is flipped, if it comes up TRUE than the bit is inverted before copying. The probability of the simulated coin coming up TRUE is called pm – the probability of mutation. As previously stated pm is small ($0 \leq \text{pm} \leq 0.1$).

VI. Inversion

Another genetic operator is called inversion. Inversion is not used as often as crossover and mutation in most GA's. Inversion is a process that shifts the locus of one or more gene in a chromosome from one point to another. This does not change the meaning of the genotype in the sense that a genotype before and after inversion will still decode to they same phenotype. If this is true, then why bother with inversion at all? The theory behind inversion is that there are groups of two or more genes in a chromosome that work together to yield a high fitness. If these genes are physically close together than single point crossover is much less likely to disturb these groups. Although this argument seems reasonable, inversion used in practice as achieved very mixed results. This is why many GA's ignore inversion all together.

VII. Conclusion

GA are original systems based on the supposed functioning of the Living. The method is very different from classical optimization algorithms.

- Use of the encoding of the parameters, not the parameters themselves.
- Work on a population of points, not a unique one.
- Use the only values of the function to optimize, not their derived function or other auxiliary knowledge
- Use probabilistic transition function not determinist ones.
- Using selection alone will tend to fill tend the population with copies of the best individual from the population.
- Using selection and crossover operators will tend to cause the algorithms to converge on a good but sub-optimal solution.
- Using mutation alone induces a random walk through the search space.
- Using selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm.

References

[1] Goldberg D. L., "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989

[2] Goldberg D, "Web Courses", http://www.engr.uiuc.edu/OCEE, 2000.

[3] Mitchell M., "An Introduction to Genetic Algorithms", Massachusetts Institute of Technology, 1996.

[4] Paechter B., Rankin R., Cumming A., "Timetabling the Classes of an Entire University with an Evolutionary Algorithm", Napier University, Edinburgh, Scotland.

[5] Holland, John H., "Adaption in Natural and artificial systems", the Mit Press, 1992.

[6] Spears, W. M. and DeJong K., An analysis of multi-point crossover", Foundations of Genetic Algorithms, pp. 301-315, Morgan Kaufmann, 1999.

[7] Syswerda, G., Uniform crossover in genetic algorithms, Procedings of the Third International Conference on Genetic Algorithms, pp. 2-9, 1995

[8] Ladd, S. R., Genetic Algorithm in C++, 1999-2000