Simple Genetic Algorithms for Pattern Learning: The Role of Crossovers

Predrag Tosic[*], **Gul Agha**
Open Systems Laboratory, Department of Computer Science
University of Illinois at Urbana-Champaign
{p-tosic, agha}@cs.uiuc.edu

Abstract

Genetic algorithms (GA) are a biology-inspired class of evolutionary stochastic optimization techniques. The three basic operations in GA are selection, mutation and crossover. Of these three, we argue that only the crossover is a uniquely GA operation. We offer alternative interpretations of the basic GA operations, and then focus on how the speed of convergence changes with the change in crossover frequency in case of some simple pattern learning problems. Our position is that only those combinatorial optimization and search problems where crossovers significantly affect performance warrant an application of GA instead of simpler random search approaches.

Keywords: *genetic algorithms, stochastic optimization, pattern learning*

1 Brief Introduction to GA

Genetic algorithms (GA) [1, 2] are a class of stochastic optimization and search algorithms inspired by biology and, in particular, evolution and natural selection. *GA* have found numerous applications in a number of problem domains where a *randomized local search* of the parameter space is applicable. Some examples of such applications are various pattern detection and recognition problems [3] and training the weights of artificial neural networks [4].

GA belong to a class of *evolutionary algorithms*: the solution of a particular problem is sought via evolving a population of agents (represented as a set of strings of genes) from one generation to the next, until, after some number of generations, the desired quality of the population is reached. The goodness of individual organisms, as well as entire populations, is quantified by an appropriate *fitness function* over the gene strings.

The three basic operations in *GA*, in accordance with the original biological inspiration, are called *selection, mutation* and *crossover*.

Selection assures that, among the current pool of genes, those gene patterns whose *fitness* is comparably high are more likely to be kept in the next generation gene pool than the gene patterns of lower fitness. In the context of stochastic optimization, the "moves" or choices that have produced more desirable outcomes in the past are more likely to be chosen in the future than those moves or choices that have led to less favorable outcomes.

Mutation is a random, typically small perturbation that introduces some randomization into the search. It can be viewed as a version of "flipping a bit" (or, in this case, a gene) in gene sequences of some, randomly selected, "organisms" present in the current generation. While often detrimental in case of individual living organisms, mutations can ensure maintaining or even creating some *genetic diversity* in populations, as we discuss in §2. In particular, in the context of stochastic optimization, mutations help in exploring more of the search space than what would be typically explored with selection and crossovers only[1].

Crossover is an operation characteristic of the *sexual reproduction*: two different gene sequences are selected, and then appropriately chosen subsequences exchanged. Thus the two "children" sequences inherit some of their genes from each of the two parents; however, these children, assuming nontrivial subsequences have been crossed over, are also different from each of their parents as well as from one another. It has been argued that, in case of the natural evolution and many stochastic optimization problems alike, the interaction of these three basic operations and the synergy among them is what makes *GA* powerful in finding high fitness solutions, and that crossovers play a central role in this synergy.

In this paper, we mainly focus on the role of the crossover operation: how the convergence of a *GA* is affected as the *crossover rate* is varied, and, in particular, as it approaches zero. We argue that crossover is the only among the basic *GA* operations that is truly a specific feature of genetic algorithms, rather than a familiar operation from other random search and machine learning techniques, only "in disguise". That non-negligible frequency of crossovers is quite essential for successful optimization via *GA* in some cases, while for all practical purposes crossovers are altogether unnecessary in other, is illustrated with some simple pattern learning examples. Some lessons drawn from our experiments (discussed in §3) are that (i) the choice of the actual crossover frequency, in case of those problems where crossovers are useful, is likely to be of utmost importance for the speed of convergence of the initial population to the one of maximal fitness, and that (ii) the good crossover and mutation frequencies (or frequency ranges) can be dependent on one another. The main lesson that we emphasize herein is that (iii) if the *GA* convergence rates are independent or nearly independent of the crossover rates, than the problem at hand is not a good *GA* application; in particular, we deduce from the examples in §3 that (iv) if a particular *GA* does quite well in a given context when the probability of crossovers is kept

[*] Contact author

[1] Strictly, this need not always be the case - but often enough it is; the potential benefits of random mutations depend on the concrete problem at hand and its representation, the population size, and the "genetic diversity" of the initial (usually randomly chosen) population.

very small or even set to zero, then a GA-based approach is likely an "overkill" for that particular problem and/or application domain. Due to space constraints, in the discussion of our experiments in §3 we mostly focus on (iii)-(iv) and only briefly reflect on the other points above.

2 Interpreting GA Operations

We next briefly discuss the three basic GA operations. We interpret and characterize selection, mutation and crossover in terms of the basic concepts from *learning* and *optimization*. Our main point is that, while selection and mutation, in our view, can be readily understood without any reference to biology, natural evolution or genetic algorithms *per se*, the crossover operation is peculiarly a GA operation. Therefore, since any problem that can be solved by GA can also be solved by other randomized local search techniques, only those problems where a non-negligible probability of crossovers makes a considerable performance difference, truly deserve to be considered GA applications. A rigorous mathematical formulation of the three basic GA operations can be found, e.g., in *Chapter §2* of [3].

We fix some notation and terminology used throughout the paper. For concreteness, a "gene" is a symbol from either the ternary alphabet $\{A, B, C\}$, or its binary subset $\{A, B\}$. The "organisms" or agents are going to be represented as finite strings of genes over the alphabet. The evaluation function in GA is called *fitness*; in our case, it maps some finite set of binary or ternary strings into a bounded set of integer values. Average fitness of a generation of organisms is simply the arithmetic mean of all individual organisms' fitness values. We shall use biology, mathematics and computer science terminologies interchangeably throughout the paper; for instance, the terms *organism*, *agent*, *individual* and *string (of genes or symbols)* are used synonymously herewith.

2.1 Selection

Selection is the GA way of achieving *reinforcement* or *exploitation* of the desirable properties that have already been discovered. In our simple model of selection, given a collection of strings and their individual fitness values, one or more low fitness strings are chosen to be discarded, and replaced by copies of the same number of high fitness strings. Thus, the size of the population remains the same, but its "genetic qualities" are improved via selection. Then genetic operations of mutation and crossover are performed on thereby improved population.

More generally, selection is a mechanism that ensures that the more fit individuals are to propagate their *genetic material* to the future generations with higher probability than the less fit individuals. Hence, selection can be viewed as a *reinforcement mechanism*: good gene patterns, in general, tend to be reinforced from one generation to the next, whereas low fitness gene patterns have a below-average probability of survival in each generation and, consequently, tend to disappear altogether in the long run [2, 5].

2.2 Mutation

There are at least two different roles of mutations, and hence two different ways of interpreting them. One is the small perturbation view: in order to prevent the algorithm from getting "stuck" (by, say, prematurely converging to a local optimum), a random mutation breaks the current equilibrium until, eventually, a truly globally stable configuration is reached. Another role of mutations is that they enable the exploration of those parts of the configuration space that, in case of bad luck with the initial configuration, would have never been reached solely via the selection and crossover operations. A simple example is an optimization problem over ternary alphabet $\{A, B, C\}$ where the string encoding of any globally optimal solution includes the symbol C, yet the initial population of strings includes symbols A and B only. Clearly, without a nonzero probability of mutation of either A or B into C, no sequence of selection and crossover operations *alone* would ever be able to reach a globally optimal configuration.

Thus, while selection provides *exploitation* of the good traits in the present generation, mutations provide *randomization*, thereby expanding the search and enabling *exploration* of those configurations that otherwise may be inaccessible. In biological terms, while the selection operation, just as in case of *natural selection*, favors the survival, reproduction and ultimate numerical dominance of the fittest, the mutation operation enables creation and long-term persistence of *population diversity*.

2.3 Crossover: Uniquely GA

While selection is a model of reinforcement and mutation a form of random perturbation, explaining *crossover* in the terms familiar to an engineer or a computational scientist with no knowledge of biology is much more challenging. Biologically, crossovers are a peculiar feature of *sexual reproduction*. While a single crossover operation between two strings of symbols can in principle always be simulated via an appropriate sequence of single-symbol mutations, this does not lead to a natural interpretation of crossovers in terms of mutations. Namely, mutations are supposed to be infrequent, random, and independent of one another, rather than very common[2] and, in the case of 'mutations' simulating the same single crossover, highly mutually correlated and non-random[3].

It has been argued that the power of GA stems primarily from the power of crossovers [3]. We rephrase this by arguing in §3 that those GA applications that use little or no crossovers, and/or perform independently of the crossover probabilities, are not genuine GA, in that the underlying problems can be readily and just as efficiently solved without genetic algorithms altogether.

3 Pattern Learning Examples

We now describe our experiments, outline the main features of the two pattern learning problems we have studied, present some of the experimental results, and then focus on their interpretation and implications.

In both problems, the (initial) string (or *"chromosome"*) length is eight. In the first set of our simple pattern learning experiments, the goal is to evolve a population of organisms or agents to the one where each agent, represented as a binary string over the alphabet $\{A, B\}$, contains in its representation each of these three patterns: AAA, BBB, and BAB. Given a string, its fitness is increased by 1 for the presence of each of these three substrings. The award is given irrespective of the number of repetitions;

[2] In practice, the crossover rates are usually by at least one order of magnitude higher than the mutation rates.

[3] Indeed, in case of binary alphabets, such mutations are uniquely determined once the two strings to be crossed over, and *'the point of crossover'*, have been chosen.

thus, fitness of $AAAAAAA$ is only 1, and so is that of $BABBAABB$. String $BABAAAAA$ is of fitness 2, whereas string $BBBABAAA$ has fitness 3. In stochastic optimization terms, the fitness function reaches its global maximum for strings such as $BBBABAAA$ above. In the reinforcement learning terms, the populations are iteratively trained to learn that the most desirable configurations are those such as $BBBABAAA$, good but suboptimal those such as $BBBAAAAAA$, while $BABABABABA$ is outright bad (zero fitness). We call this pattern learning task Problem 1.

In the second set of experiments, the optimal gene strings are those whose *longest all-A substrings* are of length four. That is, strings such as $AABBAAAA$ or $ABAAAABA$ are globally optimal (max. fitness 3), while, e.g., $BBAAABAA$ and $BAAAAABA$, with exactly three and five consecutive As in their *longest-all-A substrings*, respectively, are assigned fitness 2, the strings with exactly two or six consecutive As are assigned fitness 1, and all other strings are of fitness zero. We refer to this pattern learning problem as Problem 2.

Let us mention some of the main features of these two problems. It has been often stressed (e.g., §3 in [3]) that *GA* show their strength to the fullest extent in *multi-modal* optimization problems that, in addition to one or more global optima, also have several local optima. For such problems, it has been argued that *GA* distinguish themselves among other randomized search techniques for their ability to get out of locally optimal solutions and reliably eventually reach globally optimal configurations. For the string lengths of eight or greater, both of our problems allow several globally optimal solutions. An example of a locally optimal solution in case of our problems would be a string of fitness 2 whose fitness cannot be increased to 3 by flipping a single gene. In the case of Problem 1, string $BABBBBBB$ is locally optimal in this sense. Problem 2, on the other hand, may or may not have such local maxima, depending on the string length. While $AAA(BAAA)^k$ is a strict local maximum for string lengths $4k+3$ (where $k = 1, 2, ...$), there are no interesting local optima for other string lengths, including length 8 that we have used in most of our experiments[4]. Indeed, for length-eight strings, any relatively good but globally suboptimal string such as, e.g., $BBBAAABB$, can be made optimal by flipping a single symbol. That is, no such globally suboptimal string is nontrivially locally optimal for Problem 2, as there are always strings with higher fitness within Hamming distance 1 from it. Therefore, one may intuitively expect that *GA* would perform better in Problem 1 than in Problem 2. If "doing better" means that a typical genetic algorithm would, on average, converge faster in case of an instance of Problem 1 than in case of a comparable instance of Problem 2, this conjecture turns out to be wrong, as our experimental results indicate. A better measure of the applicability and usefulness of *GA*-based learning, in our view, is the variability of performance with respect to the *uniquely-GA* parameter, the crossover rate.

We used a simple, single-point crossover operator. We did allow this operator to be non-symmetric, in that the lengths of the offspring strings could differ from the lengths of their parents. In the case of Problem 2, this was utterly irrelevant - not surprisingly, given that crossovers themselves had very little, if any, impact there. However, this feature was absolutely crucial for allowing a reasonably rapid convergence in Problem 1 (see below).

Tables 1-2 pertain to the experiments for Problems 1 and 2, respectively, for small population sizes. We have tested a broad range of crossover and mutation rate values; as our focus is on dependence of the speed of convergence on the crossover rate, we show some results for fixed mutation rate of 2%.

Table 1: Problem 1 (population: 10)

mutation rate: $p_m = 2\%$	# of agents: 10	# of experimental runs: 500
crossover rate, p_c	mean	standard deviation
0.90	290.90	277.32
0.80	128.76	129.61
0.70	66.70	56.51
0.60	43.50	31.63
0.50	33.48	21.03
0.45	32.66	22.96
0.40	33.34	21.06
0.35	30.45	22.48
0.30	31.36	23.28
0.25	29.53	21.61
0.20	31.12	24.10
0.15	34.53	30.24
0.10	39.39	31.23
0.05	57.55	52.46
0.02	80.11	83.88
0.01	102.46	110.90
0.001	192.11	257.69
0.00	207.34	302.80

Table 2: Problem 2 (population: 10)

mutation rate: $p_m = 2\%$	# of agents: 10	# of experimental runs: 500
crossover rate, p_c	mean	standard deviation
1.00	17.48	19.70
0.90	16.08	16.92
0.80	18.28	18.97
0.70	19.60	21.65
0.60	17.91	21.14
0.50	18.98	18.12
0.40	18.74	19.82
0.30	18.16	19.42
0.20	17.91	19.76
0.10	19.11	18.47
0.01	18.64	19.69
0.00	18.13	19.76

Tables 3-4 show the mean rates of convergence and the standard deviations when the population sizes are increased from 10 to 50. As the selection mechanism was kept the same (a single worst string in a given generation is overwritten by a copy of a single best string), the convergence is slower in case of bigger population sizes, as one would expect.

Table 3: Problem 1 (population: 50)

mutation rate: $p_m = 1\%$	# of agents: 50	# of experimental runs: 100
crossover rate, p_c	mean	standard deviation
0.60	225.59	358.55
0.50	94.48	52.27
0.45	77.58	29.08
0.40	73.13	19.49
0.35	67.88	18.50
0.30	67.17	16.38
0.25	63.83	13.71
0.20	65.42	17.19
0.15	64.45	17.97
0.10	66.38	24.39
0.05	72.72	33.57
0.02	93.53	62.62
0.01	106.73	90.92
0.001	218.17	314.74
0.00	306.36	626.58

[4]One can view a string such as $BBBBBBBB$ as a trivial local maximum, in a sense that the fitness of such a string cannot be improved by flipping just one of the genes. However, we don't consider any such zero-fitness example an interesting locally optimal solution.

Table 4: Problem 2 (population: 50)

mutation rate: $p_m = 2\%$	# of agents: 50	# of experimental runs: 100
crossover rate, p_c	mean	standard deviation
1.00	48.67	3.63
0.90	49.09	4.96
0.80	49.25	3.61
0.70	48.84	3.67
0.60	48.43	3.32
0.50	48.63	3.05
0.40	48.53	3.47
0.30	49.00	3.19
0.20	49.32	4.92
0.10	48.79	3.51
0.00	49.44	3.24

In the case of Problem 1, as the crossover rate approaches zero, the convergence slows down considerably, as both *Table 1* and *Table 3* indicate. This dependence on the crossover rate p_c is highly nonlinear in two ranges: when p_c approaches zero, and once the crossover probability exceeds the upper bound of an experimentally obtained "good range". This good range of p_c values depends on the population size: in *Table 1*, the speed of convergence starts rapidly decreasing once p_c exceeds about $0.50-0.60$, while, in *Table 3*, this phenomenon is observed once p_c exceeds $0.35-0.40$. We point out that the good or optimal range for p_c, in addition to the population size, is also dependent on the mutation rate, p_m. While much of the *GA* literature recommends p_c roughly within the range $[0.60, 0.90]$ (see, e.g., §3 in [3]), in *all* of our experiments (including many not presented herein), and for all mutation rates we have tested (roughly from 10% down to 0.1%), the fastest convergence has been typically obtained for values of p_c well below 0.60. Different population sizes and different values of p_m have been demonstrated to lead to different "flat regions" in p_c for which the convergence rates are at or around the optimum.

In the case of Problem 2, given the selection strategy and the mutation probability, the mean convergence rate is nearly constant with respect to changes in the crossover probability. This holds for all population sizes between 10 and 50 that we have experimented with. In case of larger populations (*Table 4*), the average speed of convergence is slightly improved by reducing the mutation probability p_m from 2% down to 0.1% (not shown above); however, the overall behavior remains just as "flat" with respect to, i.e., basically independent of, any changes in the crossover rate. Let us also notice that, in Problem 2, the variance across different experimental runs decreases considerably with an increase in the population size.

Another interesting observation is that, when the crossovers are required to produce offspring of the same string length as their parents, the performance in Problem 1 generally tends to deteriorate dramatically. On the other hand, letting some strings grow to about 10-12 genes in length apparently enables mutations and crossovers to efficiently evolve such longer "chromosomes" into the optimal solutions that contain all three desired patterns. Another possible "fix" for the slow convergence in Problem 1 is to consider the same goal patterns and symmetric crossovers only, but for (constant) string lengths greater than eight.

In another toy example, the goal was to make a population "learn" that optimal strings ought to contain both patterns AAA and BBB - but pattern BAB was considered irrelevant this time. Statistically, the performance for this, modified version of Problem 1 turned out to be rather similar to that for Problem 2 with the same corresponding population sizes and values of p_m. In particular, "learning" $AAA+BBB$ in the space of *fixed-length-eight* binary strings via GA rendered crossovers about just as irrelevant as learning $AAAA$ did.

We conclude this brief and necessarily incomplete analysis with some remarks on the impact of the mutation rate, p_m. As expected, larger populations, in general, prefer lower p_m. However, this dependence is not linear. For both problems, changing p_m from 2% to 1% in case of larger population sizes had a bigger impact on the rate of convergence than reducing p_m from 1% down to 0.1%. Finding an optimal or close to optimal p_m is particularly important in case of the overall better behaved Problem 2, where the relative significance of mutations is higher, given the irrelevance of crossovers. Finally, in case of Problem 1, mutual dependence between p_c and p_m has been experimentally confirmed; due to space constraints, we leave the analysis of this dependence for another occasion.

4 Conclusions

Genetic algorithms are a useful computational paradigm for a variety of optimization and search problems. In particular, *GA* can be used effectively in many pattern recognition and pattern learning applications. However, even if a *GA* approach can be used for the given problem and a relatively fast convergence can be reached, we argue that this still does not imply that *GA* necessarily *should* be used in place of simpler search techniques. Whether a given problem truly warrants *GA* can be measured, in our view, by the relevance of the single uniquely *GA* operation, viz., the crossover. Thus, a problem that requires a non-negligible probability of crossovers and where the convergence rates are dependent on crossover rates deserves to be considered a worthy *GA* application. In contrast, a problem insensitive to crossover rates, and just as efficiently solvable by selection and mutations alone, as we see it, does not warrant the use of *GA*. We have illustrated these points with some simple pattern learning problems, seemingly quite similar to one another, yet where in some cases convergence critically depends on the crossover probability, while in others the choice of the crossover rate turns out to be practically irrelevant. Generalizing these simple examples and finding analytical characterizations of the two tentative classes of pattern learning problems appear to be worthy future research endeavors.

Acknowledgments: We are greatly indebted to Alfred Hubler (Center for Complex Systems Research), and we also sincerely thank Reza Ziaei and Kirill Mechitov (Open Systems Laboratory), all from University of Illinois, for their assistance and feedback. The work presented herein was partly supported by the **DARPA IPTO TASK Program**, contract number **F30602-00-2-0586**.

Bibliography

[1] Holland, J. H., *Genetic Algorithms and the Optimal Allocation of Trials*, SIAM J. Comp., 2, 1973

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989

[3] Pal, S. H., P. P Wang (ed.), *Genetic Algorithms for Pattern Recognition*, CRC Press, 1996

[4] A. J. F. van Rooij, L. C. Jain, R. P. Johnson, *Neural Network Training Using Genetic Algorithms*, Machine Perception & Artificial Intelligence, vol. 26, World Scientific Publishing Co., 1996

[5] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992