COMPUTATIONAL ANALYSIS OF OPTIMISATION ALGORITHMS WITH ARTIFICIAL INTELLIGENCE

A. SADEGHIEH, Ph. D.
Department of Industrial Engineering
University of Yazd, P. O. Box: 89195-741, YAZD, I. R. of Iran
email: Sadeghieh@yazduni.ac.ir

Abstract - Artificial intelligence is a way of making a computer behave 'intelligently'. This can be accomplished by: studying how people think when they are trying to make decisions and solve problems; breaking those thought processes down into basic steps, and finally designing a computer program that solves problems using those same steps. AI thereby provides a simple, structured approach to designing complex decision making programs. The goal of an AI system is to analyse human behaviour in the fields of perception, comprehension and decision making with the ultimate hope of reproducing the behaviour on a machine, namely a computer. One major category of AI techniques is 'genetic algorithm'. Although it is recognised that the performance of an evolutionary system such as GA is affected by the parameters that are employed to implement them, there is hardly any work known to us that has shed much light on the interdependencies and interactions between these parameters. Most studies on the effects of these parameters on performance of GA-based systems have focused on a parameter, at a time, without considering the effect of other parameters on that parameter and vice versa. Consequently, there is hardly any theory about the interactions and interdependencies of these parameters. This paper contributes towards correcting the situation mentioned above by examining empirically the relationship between three parameters of GAs.

Keywords - Linear Programming, Iterative Methods, Genetic Algorithm, Transportation Problem, Integer Programming, Non-linear Programming, Discrete Structures.

INTRODUCTION

Before applying GA to find the solution to a particular problem, it has to be determined how the solution is to be encoded as a chromosome, and how that chromosome will subsequently be translated into an appropriate measure of its relative fitness. Basically, encoding schemes fall into one of two broad categories: those with binary valued and those with real valued chromosomes.

A binary chromosome consists of binary digits. A 1 or 0 in the string may, in a Boolean scheme, correspond to whether some condition is true or false, or bits may be strung together to form binary words that will be translated either directly or indirectly into continuous valued variables, as shown in Figures 1 and 2 respectively.

Alternatively, a chromosome may be formed by a string of integers or real valued variables, with each integer or real number representing a single parameter (e.g. applied to the parametric design of an aircraft). An ordered list representation may also be used as applied to schedule optimisation [1-5].

0	0	1	1	0	1	1	0	1	1
Off	Off	On	On	Off	On	On	Off	On	On

Figure 1: Each bit in a binary chromosome may represent some Boolean condition

Bit string:	0000000000	0000001000	0000000001	0000000011
Decimal value:	0	8	1	3
Parameter:	P_1	P_2	P_3	P_4

Figure 2: Binary string representation of parameters

Several types of representation can be found in the literature, for example, 'binary encoding', 'uniform encoding' and 'Gray encoding' [6-10]. It is essential that an appropriate representation is chosen because different representations may result in very different performances in terms of computation time and rate of convergence to the solution. In this research, binary encoding is used. This representation has several advantages: it is easy to create and manipulate; many types of information can be easily encoded, and the genetic operators are easy to apply [11].

THE CHROMOSOME REPRESENTATION USED HERE

The chromosome representation used here starts with an initial population of possible solutions. Each member of the population comprises a bit string, called a chromosome, to represent a particular set of possible transmission line power capacities, P_1, P_2, ..., P_n. Each individual line capacity is encoded by sufficient bits to cover its allowable range of values. For example, if P_x has a maximum capacity of 400 MW, in either direction, then the range is ± 400 MW. This range (with a resolution of 1 MW) requires 10 bits with -400 MW being encoded as all zeros and +400 MW as 1100100000. The bit strings for each P_i are concatenated to form a chromosome. The initial population is generated randomly, that is, each bit in each chromosome is set randomly to either 1 or 0. Whenever a new chromosome is generated, it is checked to see if in decoded form it produces valid values for the P_i's. In the above example, it is possible for P_x to be given a value greater than +400 MW due to the length of the bit string. When an invalid value is produced, the chromosome is discarded and a new one is generated [7].

FITNESS AND CONSTRAINTS

The fitness value of a chromosome is a measure of how well it meets the desired objective. In this case, the objective is the minimisation of the network's cost function,

so the fitness value is the rciprocal of this cost. The fitness of any individual belonging to a given population is another important concept in GAs. The fitness function plays the role of the environment in natural evolution. Choosing and formulating an appropriate objective function is crucial to the efficient solution of any given genetic algorithm problem. When designing an objective function for an optimisation problem with constraints, penalty functions can be introduced and applied to those individuals that violate the imposed constraints, so that a constrained problem is transformed into an unconstrained problem by associating a cost or penalty with all constraint violations. This cost is included in the objective function evaluation. Consider, for example, the original costrained problem in minimisation form:

Minimise $Z(x)$ (1)

subject: to

$g_i(x) \geq 0 \qquad i = 1, 2, \ldots, n$ (2)

where x is an m vector.

Transforming this to the unconstrained form gives:

$$\text{Minimise } Z(x) + r \sum_{i=1}^{n} \Phi\ [g_i(x)]$$

Where:

Φ is a penalty function;

r is a penalty cocfficient.

A number of alternatives exist for the penalty function Φ. The unconstrained solution converges to the constrained solution as the penalty coefficient r approaches infinity [12]. In this paper, the penalty method is used in network planning

REPRODUCTION

When it comes to reproduction, a GA may operate in a generation mode or in a 'steady-state' mode. In generation mode, each iteration of the Genetic Algorithm produces a whole new generation of chromosomes. In contrast, the steady-state GA produces, at each iteration, just one new 'child' chromosome from two selected parents. This child is added to the existing population and the least fit member of the population is, then, deleted to maintain the population size. The steady-statc GA is used in this rescarch. The advantagc of using steady-state reproduction is that unlike generational replacement, where after rcplacement many of the best individuals may not be produced at all and their genes may be lost, all the genes are not lost.

Steady-state reproduction is a better model of what happens in longer-lived species in nature. This allows parents not only to nurture and teach their offspring, but also to give rise to competition between them [13].

The reproduction (selection) operator can be implemented in a varicty of ways. Three of the selection operators found more commonly in the literature are described in this section.

ROULETTE WHEEL SELECTION

Roulette whcel selection is a well known method of parent selection [5]. Figure 3 shows the reproduction operator implementcd using a biased roulette wheel. Each string in the population has a slot on the roulette wheel. The width of the slot is proportional to the fitness of the string. Strings having higher fitness are, therefore, given higher probabilitics of being selected during reproduction as is the case in nature. Each time a parcnt is required to be selectcd, the roulette wheel is spun.

TOURNAMENT SELECTION

Tournament selection chooses a number of individuals randomly from a population (with or without replaccment), then selects thc best individual from this group for further genetic processing. Tournaments are oftcn held between pairs of individuals, although larger tournaments can be used [14].

String Number	Initial Population	Decoded Integer Value (X)/100	Fitness $Z(X)=-(X^2-X)$	$Z_i / \sum Z_i$ (%)
1	1010000	0.80	0.16	22.30
2	0001010	0.10	0.09	12.54
3	0111100	0.60	0.24	33.45
4	0100011	0.35	0.2275	31.71
Total			0.7175	100.00

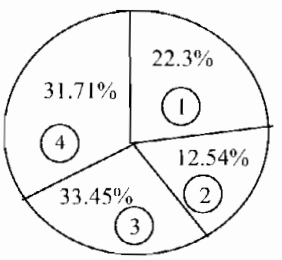

Figure 3: Reproduction implemented using a biased roulette wheel

RANKIN-BASED SELECTION

Baker was the one who suggested rank selection [15]. Sort the population from best to worst, assign the number of copies that each individual should receive according to a no-increasing assignmcnt function, and then perform proportionate selection according to thc assignment [9]. Ranking is a two-step process. First, the list of individuals must be sorted, and thcn the assignment values must be used in some form of proportionate selection. Some qualitative theory regarding such schemes was presented by

Grefenstette [16]. A 'rank-based method' is used here. The members are ranked in order of their fitness and the probability of selection is inversely rated to this ranking. The advantage of a 'rank-based approach' is that it helps to avoid too rapid a rate of convergence that may lead to the population being swamped by a local optimum due to the loss of diversity.

CROSSOVER OPERATOR

The purpose of a crossover operator is to produce new chromosomes (children) that are distinctly different from their parents, yet retain some of their characteristics. In the steady-state GA implementation used here, when two selected parents perform crossover, the proportion of the genes coming from the first parent into the child is defined by the crossover rate, the remainder coming from the second parent. This is in marked contrast to the generational GA, for which the crossover rate defines the probability that the two selected parents perform crossover as opposed to being carried straight across into the new generation. Three of the crossover operations found more commonly in the literature are considered here.

ONE-POINT CROSSOVER

One-point crossover was, first, proposed by Holland [17]. Crossover may proceed in two steps. First, two parent strings are chosen for crossover. Then, an integer position k along the strings is selected according to uniformly distributed random number between 1 and (ℓ-1) where, ℓ is the length of the string. Two new strings are created by crossing over the bits on either side of this position as illustrated in Figure 4.

Parent string 1:	0	0	0	1	0	1	0	0	0	0
Crossover position:							*			
Parent string 2:	0	0	0	0	1	1	1	1	0	0
Child string 1:	0	0	0	1	0	1	0	1	0	0
Child string 2:	0	0	0	0	1	1	1	0	0	0

Figure 4: Simple one-point crossover

TWO-POINT CROSSOVER

In two-point crossover, two crossover points are selected instead of just one. The part of the parent strings that comes between these two points is, then, swapped to generate two children. An example is shown in Figure 5. Empirical studies have shown that two-point crossover usually provides better randomisation than one-point crossover [12].

Parent string 1:	0	0	0	1	0	1	0	0	0	0
Crossover position:						*		*		
Parent string 2:	0	0	0	0	1	1	1	1	0	0
Child string 1:	0	0	0	1	0	1	1	1	0	0
Child string 2:	0	0	0	0	1	1	0	0	0	0

Figure 5: Two-point crossover

UNIFORM CROSSOVER

Syswerda [18] proposed uniform crossover which works as follows. Two parents are selected and two children are produced. For each bit position on the two children, it is decided randomly as to which parent contributes its bit value to which child. Figure 6 shows uniform crossover in action. For each bit position in the parents, a random binary template indicates which parent will contribute its value in that position to the first child. The second child receives the bit value in that position from the other parent.

Parent string 1:	0	0	0	1	0	1	0	0	0	0
Parent string 2:	0	0	0	0	1	1	1	1	0	0
Binary template:	0	0	0	1	1	0	1	0	0	1
Child string 1:	0	0	0	1	0	1	0	1	0	0
Child string 2:	0	0	0	0	1	1	1	0	0	0

Figure 6: Uniform crossover

Uniform crossover generally works better than one and two-point crossovers. It was shown by Syswerda that in almost all cases, uniform crossover is more effective for combining schemata than one or two-point crossovers. Empirically, uniform crossover is shown to be more effective on a variety of function optimisation problems [11]. In this paper, one child is formed by taking a mixture of bits from its two parents according to a random bit string. The proportion of bits coming from the best parent is defined by the user-defined crossover rate in the range zero to one, where a '1' in the random bit string indicates that in that position the child inherits the corresponding bit from parent-1, whilst a '0' causes inheritance from parent-2.

MUTATION OPERATOR

Until now, reproduction and crossover effectively search and recombine the existing

chromosomes. They do not create any new genetic material in the population. Mutation is capable of overcoming this shortcoming. It gives random movement about the search space and, thus, preventing the GA from becoming trapped in 'local optima' or 'dead corners' during the search. In this paper, a recipe GA mutation is performed by looking at each gene individually. A random number, from a uniform distribution across the range [0,1], is generated for each gene in the chromosome, and if the gene gets a number that is less than or equal to the mutation rate, then the gene is mutated. Mutating a binary gene means switching it from 0 to 1, or from 1to 0. A high mutation rate will introduce excessive randomness into the search, making it too diverse and impeding the convergence. Conversely, too low a mutation rate will reduce diversity which is likely to cause premature convergence to a local optimum.

BASIC PARAMETERS OF A GENETIC ALGORITHM

The basic parameters of a GA include population size, crossover and the mutation rates. By changing these parameters, the convergence of the algorithm changes too. Thus, to maintain the robustness of the algorithm, it is important to assign appropriate values to these parameters. There has been a great deal of discussion on parameter setting in the literature. De Jong's [19] experiments indicate that the best population size is 50-100 individuals, the best crossover rate is 0.6 and the best mutation rate is 0.001 per bit. Later, Schaffer et al [20] found that a population size of 20-30, crossover rate of 0.75-0.95 and mutation rate of 0.005-0.01 are the best settings regardless of the problem in their test suite.

In this paper, combinations of population sizes in the rang [20-80], crossover rates in the range [0.2-0.8] and mutation rates in the range [0.001-0.016] are tested. The GA stops when a convergence criterion is reached. In the experiments performed here, this is when the change, in the best fitness in the last 100 trials, is less than 0.0001 unit cost.

IMPLEMENTATION USING SPREADSHEETS

Spreadsheets are used widely in industry due to factors such as their versatility, ease of use, rapid development and ease of modification. For over a decade, spreadsheets have been providing intuitive applications. Bodily [21] stated that the adoption of spreadsheets as decision making aids by end-users was due to the natural interface that existed for model building, the ease of use in terms of inputs, solutions and report generation, and the ability to perform 'what-if ' analysis. He continued that, because of these key properties, the spreadsheet medium could be used as a stepping stone for bringing operations, research models and techniques to the end-user.

Recent literature supports Bodily's conclusions, with successful applications in queuing systems, inventory management, aggregate planning, analysis of manufacturing systems, financial planning, warehousing and transportation. Techniques and models include linear programming, integer programming, dynamic programming, simulation and heuristics. Spreadsheets are particularly suitable for network planning due to their

fundamental representation of data in the form of easily understood tables.

The work presented here was carried out using the Microsoft^{R} Excel^{TM} spreadsheet and an add-in to provide the GA. This add-in is called 'Evolver^{TM}' and is developed and supplied by Axcelis [22]. The use of this proprietary software demonstrates how simple it is to implement the GA approach to the minimum cost flow problem (MCFP) and transportation problem (TP) optimisation. This software enables the immediate implementation, by any reader, of the methods presented here. This is a key aspect of this paper.

The model of network planning developed in this research is built in Excell^{TM} using the spreadsheet's built-in functions. After building the model, GA is run to optimise the network given an objective function.

The fitness value and decision variables are passed back to the GA component which is independent of the spreadsheet model. At the end of the GA run, when the stopping criterion is met, the best network is presented in a tabular form in the spreadsheet.

GA APPLIED TO NETWORKS

Networks arise in numerous settings and in a variety of forms. Transportation, electrical and communication networks pervade our daily lives. Network representations are also widely used for problems in such diverse areas as manufacturing systems analysis, logistics, project planning, facilities location and resource management. In fact, a network representation provides such a powerful visual and conceptual aid for portraying the relationships between the components of systems that are used in virtually every field of scientific, social, and economic endeavour.

One of the most significant developments in operational research in recent years has been the rapid advance in both the methodology and application of network optimisation models.

A network is a system of lines or channels connecting different points. Some examples of networks are communication lines, railway networks, pipeline systems, road networks, shipping lines, aviation networks, etc. The primary concern in all these networks is how to send some specified commodity from certain supply points to some demand points. For example, in a pipeline system, the task is sending water, oil or gas from supply stations to demanding customers. Many of the network flow problems (for example, transportation, minimum-cost flow, etc.) can be formulated as different forms of mathematical programming, e.g. linear, non-linear and integer.

There are many good and well established methods of analysis that can be applied at the scheme evaluation stage. However, satisfactory methods for scheme formation are still evolving and they have been the subject of much current research. Scheme formation can be a complex task subject to many constraints and a non-linear object function. 'Optimal' network designs are important because they can result in large cost savings. There is clearly a need and strong justification for the development of methods for the design of networks that are as near to 'optimal' as possible. This paper aims to provide such methods. This paper introduces the application of Genetic Algorithms to

the Minimum Cost Flow Problem and the Transportation Problem.

The minimum cost flow problem provides a unified approach to many applications because of its general structure. The minimum cost flow problem holds a central position among network optimisation models, both because it encompasses such a broad class of applications and because it can be solved extremely efficiently.

As an example, let Tables 1 and 2 represent a car company with plants in two locations 1 and 7, the total production of car units respectively. The negative numbers next to the nodes representing 1 and 7 indicate the available supply of cars at these locations. The demand for cars at each of the other locations is represented by positive numbers next to the remaining nodes, e.g. 3 has a demand for 60 cars. The values on the arcs connecting the various nodes represent the unit cost of transporting cars between the indicated locations, e.g. it costs 50 to transport a car from 7 to 6. The problem here is to determine the plan with a minimum cost for transporting cars from 1 and 7 to meet the demand for cars at the other locations.

Table1: Arcs and cost per unit in a car company

Arcs	1-2	1-4	2-3	3-5	5-3	5-4	5-6	6-5	7-4	7-5	7-6
Costs	30	60	80	35	45	35	45	25	95	45	50

Table 2: Nodes and supply/demand

Nodes	1	2	3	4	5	6	7
Supply or Demand	-200	90	60	90	180	70	-300

The first step in the application of a GA is the coding of the variables that describe the problem. The most common coding method is to transform the variables to a binary string of specific length. This string represents the chromosome of the problem and the length of the chromosome represents the number of zeros and ones in the binary string. By decoding the individuals of the initial population, the solution for each specific instance is determined and the value of the objective function that corresponds to this individual is evaluated. This applies to all members of the population.

Genetic algorithms require that the natural parameter set of the optimisation problem be coded as a finite length string. For the car distribution example, there are eleven decision variables X_{ij} as follows: $X_{12}, X_{14}, X_{23}, \ldots, X_{76}$ (4)

The final step in the implementation of GA is the application of the fitness function. A fitness function is used to calculate the fitness value of each individual. Before this function is applied, the string must be decoded. Then, once the real value for each parameter is available, the fitness for that specific individual can be estimated.

The basic parameters of a genetic algorithm are the population size, the crossover rate and the mutation rate. By changing these parameters, the rate of convergence of GA changes too. Thus, to maintain the robustness of the algorithm, it is important to assign appropriate values to these parameters. A population size of 50, a crossover rate of 0.5

and a mutation rate of 0.001, 0.006 and 0.011 are used in the experiments presented here. With a mutation rate of 0.006 or 0.011, GA reaches, within 4000 iterations, the optimal solution (25300) identified by linear programming. In contrast, with a mutation rate of 0.001, the rate of convergence is much poorer.

One of the advantages of GA seen here is that the algorithm is only using fitness values for searching and, thus, solving the problem, in contrast to linear programming which needs a spanning tree and other specific knowledge of the application. The other attractive property of GA that is demonstrated is its ease of application.

As another example, a car manufacturing company has plants in three locations: A, B and C. During the past week the total production of a special car unit out of each plant has been 750, 640, and 450 units respectively. The company wants to ship 240 units to distribution centre number 1, 270 to 2, 340 to 3, 290 to 4, and 230 to 5. The supply and demand and the per unit costs for each lorry (a lorry can carry up to 6 units) are given in Table 3. What is the best shipping strategy to follow?

Table 3: The supply and demand and the per unit shipping costs.

	Distribution Centres					
Plants	**1**	**2**	**3**	**4**	**5**	**Supply (units)**
A	0.56	1.37	0.22	0.13	0.33	750
B	0.50	0.45	0.21	0.41	0.54	640
C	1.20	0.15	0.45	0.50	0.75	450
Demand (units)	240	270	340	290	230	

This shows the optimal solution and that the total cost of this transportation plan is 57.96 units.

Now consider a situation in which the shipping costs are not liner. For example, transportation costs may depend on how many lorries are needed for the shipment. To illustrate this, consider an example in which a lorry can carry up to 6 units, so that shipping 48 units will require 8 lorries. Integer programming can be used to solve this type of problem. The apparent drawback is the large number of variables and constraints, which may limit its application for large systems. The total cost of this transportation plan is 1482 units and is not a spanning tree.

A GA is applied, here, to the integer transportation problem. This application shows that it is much easier to implement GAs for a complex transportation problem, compared with constructing mathematical programming formulations, since it is a very simple matter to implement a complex cost function and solution constraints when using a GA.

CONCLUSIONS

GAs are based on concepts on natural genetic and evolutionary mechanisms working on populations of solutions in contrast to other search techniques that work on a single solution. An important aspect of GAs is that although they do not require any prior

knowledge or any space limitations such as smoothness, convexity or unimodality of the function to be optimised, they exhibit very good performance in most applications. They only require an evaluation function to assign a quality value (fitness value) to every solution produced. Another interesting feature is that they are inherently parallel (solutions are individuals and unrelated to each other). Therefore, their implementation on parallel machines would reduce the CPU time required significantly: GAs are suitable for traversing large search space since they can do this relatively rapidly and because the mutation operator diverts the method away from local optima, which will tend to become more common as the search space increases in size. Other key advantages of GAs are their generality and their ease of applicability. Empirical analysis of the performance of GAs and the effects of GAs parameters on this performance, in addition to the aforementioned advantages, show that GAs are feasible, robust and practical engineering tools.

REFERENCES

[15] Baker, J. E., *Adaptive selection methods for GAs*, in Grefenstette, J. J. ed., Proceedings of First International Conference on GAs, Erlbaum, 1985, pp. 101-111.

[21] Bodily, S. E., *Spreadsheet modelling as a stepping stone*, Interfaces, vol. 16, No. 5, 1986, pp. 34-52.

[4] Bramlette, M. F. and Bouchard, E. E., *GA in parametric design of aircraft*, in Handbook of GAs, Editor: Davis, L., New York: Van Nostrand Reinhold, 1991.

[11] Davis, L., *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York, 1991.

[19] DeJong, K. A., *An analysis of the behaviour of a calss of genetic adaptive system*, Ph. D., Discretion, University of Michigan, Ann Arbor, 1975.

[22] *Evolver user's guide*, Axcelis, Inc., Seattle, WA, USA, 1995.

[12] Goldberg, D. E., *Genetic algorithms in search optimization, and machine learning*, Addison Wesley, 1989.

[14] Goldberg, D. E. and Deb, K., *A Comparative analysis of selection schemes used in GAs*, in Rawlins, G. ed., Foundations of GAs, organ Kaufmann, 1991.

[16] Grefenstette, J. J. and Baker, J. E., *How GAs work: a critical look at implicit parallelism*, Proceedings of the Third International Conference on GAs, 1989, pp. 20-27.

[17] Holland, J. H., *Adaptation in natural and artificial systems*, Ann Arbor: Univ. of Michigan, 1975.

[10] Karaboga, D., *Design of fuzzy logic controllers using GAs*, University of Wales, Cardiff, Ph.D, 1994.

[7] Sadeghieh, A., *Iterative improvement method and mixed-integer programming in system planning,* 16th International Power System Conference, Tehran, Iran, October 2001, pp. 22-24.

[1] Sadeghieh, A., *Mathematic and simulation with biological, economical and*

musicoacoustical application, Editors: D'Attellis, C. E., Kluev, V. V., Mastorakis, N., World Scientific and Engineering Society Press, 2001, pp. 11-16.

[9] Sadeghich, A., *Network planning using iterative improvement and heuristic method*, International Journal of Engineering, vol. 15, No. 1, 2002, pp. 63-74.

[8] Sadeghich, A., *Optimization in spreadsheet model for network problem*, Far East Journal of Mathematical Sciences, vol. 4, No. 1, 2002, pp. 100-115.

[6] Sadeghich, A. and Drake, P. R. *Anvances in scientific computing, computational intelligence and applications,* Editors: Mastorakis, N., Mladenov, V., Suter, B. and Wang, L. J., WSES Press, 2001, pp. 215-221.

[2] Sadeghieh, A. and Drake, P. R., *Network optimization using linear programming and genetic algorithm*, Neural Network World, International Journal of Non-Standard Computing and Artificial Intelligence, vol. 11, No. 3, 2001, pp. 223-233.

[3] Sadeghieh, A. and Drake, P. R. *Recent advances in applied and theoretical mathematics*, Editor: Mastorakis, N., World Scientific and Engineering Society press, 2000, pp. 150-155.

[20] Schaffer, J. D., Caruana, R. A., Eshelman, L. J. and Das, R. *A study of the control parameters affecting the performance of GAs for function optimisation*, in Schaffer, J. D. ed., Proceedings of the Third International Conference on GAs, Morgan Kaufmann, 1989.

[5] Syswerda, G., *Schedule optimization using algorithms*, in Handbook of GAs, Editor: Davis, L., New York: Van Nostrand Reinhold, 1991, pp. 129-144.

[18] Syswerda, G., *Uniform crossover in genetic algorithms*, in Schaffer, J. D. ed., Proceedings of the third International Conference on Genetic Algorithms, San Mateo, Calif Morgan Kaufmann Publishers, 1989, pp. 2-9.

[13] Whitley, D., *GENITOR: a different genetic algorithm*, in Proceedings of the Rocky Mountain Conference on Artificial Intelligence, Denver, Colorado, 1988.