Your Brains and My Beauty: Parent Matching for Constrained Optimisation

Robert Hinterding
Dept. of Comp. and Math. Sciences
Victoria University of Technology
PO Box 14428 MCMC,
Melbourne 8001, Australia
rhh@matilda.vut.edu.au

Zbigniew Michalewicz*
Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA
zbyszek@uncc.edu

Abstract— **During the last years, several methods have been proposed for handling constraints by evolutionary algorithms for parameter optimisation problems. These methods include those based on penalty functions, preservation of feasibility, decoders, repair algorithms, as well as some hybrid techniques.**

Most of these techniques have serious drawbacks (some of them may return infeasible solution, others require many additional parameters, etc). Moreover, none of these techniques has utilized knowledge on which constraints are satisfied, and which are not. In this paper we introduce a new element to evolutionary algorithms for constrained parameter optimization problems: the parent matching mechanism. The preliminary results show that the proposed technique works very well on selected test cases.

I. Introduction

The general nonlinear programming (NLP) problem is to find $\vec{x}$ so as to

optimize $f(\vec{x})$, $\vec{x} = (x_1, \ldots, x_n) \in \boldsymbol{R}^n$,

where $\vec{x} \in \mathcal{F}$. $\mathcal{F}$ is the set of feasible solutions and is defined by a set of m constraints ($m \geq 0$):

$g_j(\vec{x}) \leq 0$, for $j = 1, \ldots, q$, and $h_j(\vec{x}) = 0$, for $j = q+1, \ldots, m$.

The NLP problem, in general, is intractable: it is impossible to develop a deterministic method for the NLP in the global optimization category, which would be better than the exhaustive search. This makes a room for evolutionary algorithms, which aim at complex objective functions (e.g., non-differentiable or discontinuous). Consequently, during the last few years, evolutionary techniques were extended by some constraint-handling methods. In a recent survey paper [14] various methods incorporated by these algorithms were grouped into four categories: (1) methods based on preserving feasibility of solutions, (2) methods based on penalty functions, (3) methods which make a clear distinction between feasible and infeasible solutions, and (4) other hybrid methods.

The first category requires the design of specialized operators (e.g., arithmetical crossover, non-uniform mutation [10], [11], geometrical crossover [13], sphere crossover [22], etc). Clearly, to maintain feasibility of an individual, a specialized operator (which incorporates the knowledge about problem-specific constraints) should be used [21]. The methods based on penalty functions require many additional parameters, the major ones being the penalty coefficients themselves. Apart from static penalties [3], there were some experiments reported on the use of dynamic penalties [4], [9] and adaptive penalties [1], [24], where the search length and the feedback about constraint violation, respectively, was incorporated. The third category of methods emphasizes the distinction between feasible and infeasible solutions. For example, one particular method [23] (called a "behavioral memory" approach) considers the problem constraints in a sequence; a switch from one constraint to another is made upon arrival of a sufficient number of feasible individuals in the population. The final category includes methods which combine evolutionary computation techniques with deterministic procedures for numerical optimization problems [26], [15]. In particular, the method by Myung et al. [15] divides the whole optimization process into two separate phases. During the first phase, evolutionary algorithm optimizes a modified objective function, whereas during the second phase the optimization algorithm of Maa and Shanblatt [6] is applied to the best solution found during the first phase.

In this paper we report on an evolutionary algorithm with additional features, which are very appropriate for constrained parameter optimization problems. Some of these features are similar to those proposed in the past, but some other features are new. It seems that combination of all these features is responsible for very good performance of the new algorithm.

The paper is organized as follows. The next section presents the general ideas which were incorporated in the construction of the new system. Section III discusses the features of this system, and section IV reports experimental results. Section V concludes the paper.

II. General ideas

In evolutionary computation the main link between the algorithm and the problem is the evaluation function. This

*also at the Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland, *zbyszek@ipipan.waw.pl*.

function influences the rating of individuals in the population: better individuals have better chances to survive and reproduce. However, in NLP the ranking of individuals is not straightforward: apart from optimizing the objective function f, the solution should satisfy a set of constraints. The main issue is: how should we compare two (infeasible) individuals, $\vec{x_1}$ and $\vec{x_2}$, if $f(\vec{x_1}) < f(\vec{x_2})$,[1] and $\vec{x_1}$ satisfies two (out of $m = 4$) constraints, whereas $\vec{x_2}$ satisfies three constraints? Which of these two is better? Should we also take into account the amount of constraint violation?

In most methods, which have been proposed and experimented with during the last few years, the evaluation function returns just a single numeric value which is used to compare individuals. The most popular approach is based on the concept of penalty functions, which penalize infeasible solutions, i.e., try to solve an unconstrained problem using the modified fitness function:

$$eval(\vec{x}) = \begin{cases} f(\vec{x}), & if\ \vec{x} \in \mathcal{F} \\ f(\vec{x}) + penalty(\vec{x}), & otherwise, \end{cases}$$

where $penalty(\vec{x})$ is zero, if no violation occurs, and is positive, otherwise. Usually, the *penalty* function is based on the distance of a solution from the feasible region $\mathcal{F}$; in many methods a set of functions f_j $(1 \le j \le m)$ is used to construct the penalty, where the function f_j measures the violation of the j-th constraint in the following way:

$$f_j(\vec{x}) = \begin{cases} \max\{0, g_j(\vec{x})\}, & if\ 1 \le j \le q \\ |h_j(\vec{x})|, & if\ q+1 \le j \le m. \end{cases}$$

There are many possible ways of constructing penalty functions. One particular approach was developed by Powell and Skolnick [17]: the method distinguishes between feasible and infeasible individuals by adopting an additional heuristic rule (suggested earlier by Richardson et al. [18]): any feasible solution is better than any infeasible one. Thus, infeasible individuals have increased penalties: their values cannot be better than the value of the worst feasible individual.

Of course, we are not limited to a singleton numerical value. For example, some methods use of the values of objective function f and penalties f_j $(j = 1, \ldots, m)$ as elements of an evaluation vector and apply multi-objective techniques to minimise all components of the vector. For example, Schaffer's [20] Vector Evaluated Genetic Algorithm (VEGA) selects $1/(m+1)$ of the population based on each of the objectives. Such an approach was incorporated by Parmee and Purchase [16] in the development of techniques for constrained design spaces. On the other hand, in the approach by Surry et al. [25], all members of the population are ranked on the basis of constraint violation. Such rank r, together with the value of the objective function f, leads to the two-objective optimization problem. Chu and Beasley [2] also considered two-objective optimization problem; a vector of two values consisted of the value of the objective function and a measure of the "unfitness" of an individual.

[1]Let us assume here a minimisation problem.

The proposed approach used is based on following observations:

- As in the behavioural memory approach, the search should run in two phases: (1) the phase of searching for feasible solutions and (2) the optimization phase. In this respect, the approach is similar to that of [15], however, our phases have somewhat fuzzy boundaries. During the first phase of the search the values of objective function are ignored due to the fact that all individuals are infeasible (assuming, of course, that this is the case). Gradually, with larger and larger numbers of feasible individuals in the population, the value of the objective function is of growing importance, and the second (optimization) phase of the search plays the major role (i.e., fine-local tuning of the best individuals).
- The evaluation function should assist us in comparing two individuals. If both individuals are feasible, the value of the objective function, of course, should be taken into account. If both individuals are infeasible, the value of the objective function is meaningless (contrary to all approaches based on penalty functions). In such cases, information on constraint violation should be considered. Additionally, it seems reasonable to assume that feasible individuals are always better than infeasible ones (as in [17]).
- The evaluation function can also assist us in other ways. In particular, it might be useful in selecting a second parent for a given one (for the crossover operator). For example, if the evaluation function keeps a record of which constraints are satisfied, then the second parent can be selected to maximize the number of constraints which are satisfied by both parents (in this way the second parent complements the first parent and the generated offspring hopefully satisfies a larger number of constraints).

The idea of matching parents is not new to EAs, the idea is discussed by Ronald [19], but for the applications he presented, the selection of the second parent did not depend on the first parent at all. Hence, the function to select the second parent, *seduction()*, acted as a secondary fitness function using a different global criterion from the main fitness function.

III. The CONGA system

The evolutionary algorithm developed, CONGA (COnstraint based Numeric Genetic Algorithm) is a genetic algorithm for finding optimal or near optimal solutions for numeric problems with constraints. The fitness function *eval* (as described in the previous section) returns information on which constraints are not satisfied and the extent of the violations as well as the value of the objective function:[2]

$$eval(\vec{x}) = \langle f(\vec{x}), v, s, \vec{c} \rangle,$$

[2]The aim here was to mimic more closely parent selection practices in nature: a number of characteristics of prospective mates are considered, and the characteristics which are considered to be desirable in a mate influence its selection.

where

- $f(\vec{x})$ — is the value of the objective function;
- v — is the number of violated constraints ($0 \leq v \leq m$); clearly, $v = 0$ implies that the solution $\vec{x}$ is feasible. This value can be used in comparison between two individuals (smaller v, the better);
- $s = \sum_{j=1}^{m} f_j^2(\vec{x})$; this sum of squares of the measures of constraint violations provides with information on unfeasibility of each individual. This value can be used to break some comparison ties (e.g., when two individuals satisfy the same number of constraints);
- $\vec{c} = \langle c_1, \ldots, c_n \rangle$ — is the constraints satisfied mask. This is a binary vector of length m; bit i is set iff the i^{th} constraint is satisfied. This vector can be used to select individuals for crossover (individuals would complement each other in terms of constraints satisfied).

New individuals are generated by either crossover or mutation, but not both. Hence we can have two selection functions, "SelectFirst" which selects an individual for mutation or the first parent for crossover, and "SelectFor" which, given a selected parent, finds it a mate for crossover. For both functions the information used in selecting an individual depends on the characteristics of the individuals involved. CONGA uses tournament selection with a tournament size of 2 and the rules for winning the tournament in "SelectFirst" are given below:

- if both individuals are feasible, select individual with the better value of the objective function;
- if only one individual is feasible, select it;
- if both individuals are infeasible, select individual with
  - smaller number of violated constraints v. If equal, select individual with
  - smaller constraint violation measure s if they satisfy the same constraints $\vec{c}$, else use random choice.

In "SelectFor" procedure, which chooses a mate for a parent, we can be more creative. To choose between two potential mates which are both infeasible and satisfy an equal number of constraints, we choose the individual which has *the least* number of satisfied constraints in common with the already chosen parent. Here we select the potential mate that best "complements" the already selected parent by satisfying the constraints that the first parent does not satisfy, in a hope that crossover will create a new individual that satisfies more of the constraints than either parent does.

In CONGA the function variables (genes) are represented as fixed point reals using a 20 bit binary representation with Gray encoding. The range of values represented is the smallest range common to all the domain constraints. The percentage of the population to be replaced is set by a parameter; the worst of the current generation is replaced by the new generation. Any duplicate individuals generated are discarded. Six point binary crossover and Cauchy mutation were used. Cauchy rather than Gaussian mutation was used as the longer tails with the Cauchy distribution was thought to be beneficial [5]. Mutation is implemented by first decoding the gene an an unsigned integer and adding a Cauchy distributed random number of mean zero and spread t which is scaled by multiplying it by half the maximum possible value of the unsigned integer. This new value is then encoded into the gene.

Self-adaptation for the mutation strength was implemented by adding an extra gene to the chromosome which encoded the "spread parameter" for the Cauchy distribution. This was initialised to a small range of values, and self-adaptation is enabled when this gene was allowed to participate in mutation. This gene is only allowed to mutate in feasible individuals. The reason is that the search for feasible individuals was considered to be an exploratory phase and it was not considered appropriate to tune the mutation strength during this phase. The number of genes to be mutated in a chromosome is controlled by λ, the mean for a Poisson distributed random number. Hence if we have n genes in a chromosomes, each with a $1/n$ probability of mutation, we would use $\lambda = 1$.

We compare the results with those of Genocop [8], [12] and show it produces comparable or better results. For these experiments a population size of 100 was used, 50% of the new individuals were produced by crossover and 50% by mutation. It should be noted that most of the parameters of the EA were set to values which were thought to be "reasonable", no proper testing has been done to determine if these values are the most appropriate ones to use. All the results reported in the next section were produced from batches of 20 runs.

IV. Experiments and results

The test functions G1 – G5 are taken from [7] and cover a range of constraint based problems with both linear and non-linear constraints. These test functions are summarised in Table I; for each test function (TF) we list the number n of variables, type of function, the number of constraints of category (linear inequalities LI, nonlinear equations NE and inequalities NI), the number a of active constraints at the optimum, together with the value "Opt." of the objective function at the optimum.

TABLE I
Summary of test functions

TF	n	Type of f	LI	NE	NI	a	Opt.
G1	13	quadratic	9	0	0	6	−15.00
G2	8	linear	3	0	3	6	7,049.33
G3	7	polynomial	0	0	4	2	680.63
G4	5	nonlinear	0	3	0	3	0.054
G5	10	quadratic	3	0	5	6	24.31

For test function G4, the problem was reduced by hand to one of two variables by using the three nonlinear equations. There are other methods to deal with the equality constraints but they have not yet been tested. An equality constraint $F(x) = 0$ could be replaced by two constraints $F(x) \leq \sigma$ and $F(x) \geq -\sigma$ for some suitably small σ.

It was experimentally determined that a replacement rate of 97% gave best results for function G2, and this setting was used for all the test cases. Figure 1 shows plots of the average best function value *versus* number of evaluations for a number of replacement rates for the test function G2. The other feature to notice in this graph is that the function value tends to rise first as feasible solutions are sought, and then decreases as the optimum feasible solution is sought. What we see here is that as the population is driven towards the feasible region, and more of the constraints are satisfied the function value of the best individuals increases[3], and hence demonstrates that the function value can be misleading while the individuals are infeasible.

Figure 2 shows the plots of a number of other statistics gathered on the function G2. "Feas" gives the ratio $(0-1)$ of the population that is feasible, "Crx" gives the ratio of individuals produced by crossover that are equal or better than both the parents, "Mut" gives the ratio of new individuals produced by mutation that are equal or better than its parent, and "mut str * 10" gives the average self-adaptive mutation strength multiplied by 10. The graph shows the increase in the number of feasible individuals up to 20,000 evaluations when feasible individuals have been found in all 20 runs, and that an average of about 20% feasible individuals is maintained for the rest of the runs. The graph also shows that the effectiveness of mutation is maintained by self-adapting the mutation strength. That is, the effectiveness of mutation remains roughly constant during the runs, rather than decreasing when self-adaption is not used.

Figures 3 & 4 shows what happens in more detail during the first phase of the runs on function G2 when feasible individuals are sought. Figure 3 plots the sum of constraint violations (Violations) of the best individual, and the ratio of feasible individuals in the population (note: the Y axis is log scale), while figure 4 plots the objective function value (Fn. value) of the best individuals. Feasible individuals first occur at about 6,000 evaluations, and all runs have found feasible individuals by 16,000 evaluations (Violations drops to 0). Also, the sum of violations decreases as the EA progresses, and the function value first increases and then starts to decrease again as feasible individuals are found.

A series of runs was performed on all the test functions using a value of λ (the average number of function variables mutated in a chromosome) set to the square root of the number of test function variables. These results are in Table II, and are given where the mutation strength was held fixed, and where it was allowed to self-adapt. Each run used 140,000 evaluations so the results could be compared to those of GENOCOP. In this and for Table III, the value of λ, the maximum number of evaluations to find a feasible solution in all the 20 runs (feas.), and the mean[4] and standard deviation (σ) of the average best values found is given.

For Table III, the best setting for λ for each of the functions was determined experimentally, and the runs were done for 140,000 evaluations. The percentage of the population that is feasible at the end of the run (feas %) is also

[3]Remember these are minimisation problems

[4]Note the value followed by # is the maximum no. of evals to find the optimum in all 20 runs.

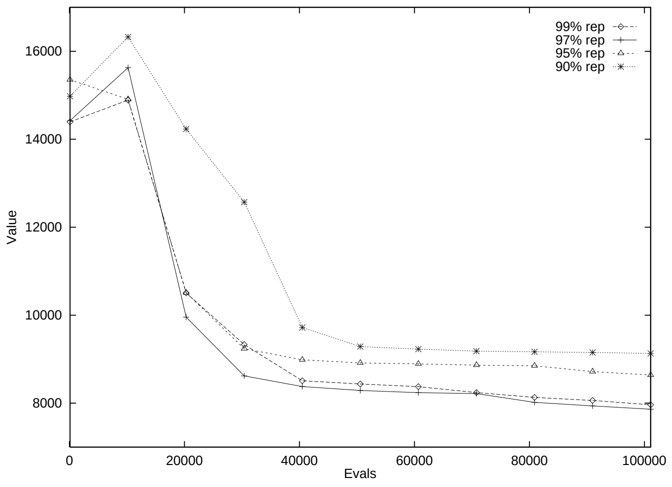

Fig. 1. Replacement rate for G2:ave. 20 runs

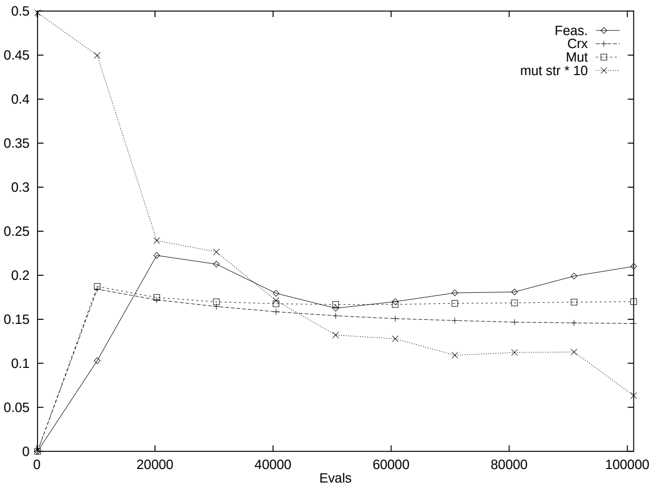

Fig. 2. Stats on G2:ave. 20 runs

given, and this varies considerably.

In Table IV we give a comparison of the results with GENOCOP, for GENOCOP the results are from ten runs on each of the functions. For GENOCOP 2, the median result and the number of constraint violations for the median result are given. GENOCOP 3 and CONGA always give feasible results and for these the best result is given. Although GENOCOP and CONGA are very different, and use very different operators and representation, these results give some indication that comparable or superior results can be obtained.

V. Conclusions

The approach we used here to deal with constraints in an EA was to get it to utilise more knowledge about the constraints rather than using penalty functions or specialised operators. The evaluation function, the link between the problem and the EA, was modified to return information about which constraints were satisfied and the extent of the violations as well as the function objective value. This al-

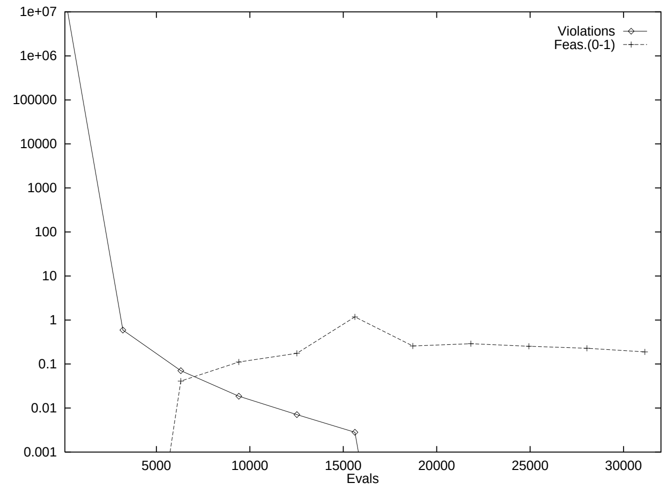

Fig. 3. Performance on G2: violations & feasibity

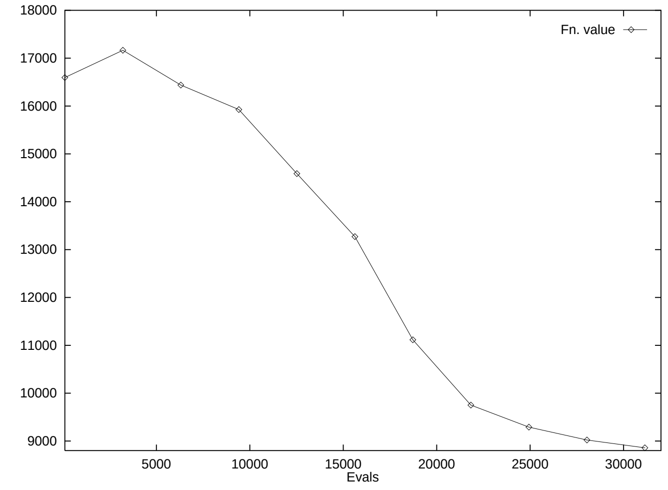

Fig. 4. Performance on G2: function value

lowed selection and parent matching to be implemented as a set of preferences. These preferences cover cases where the individuals are feasible and infeasible and define the global behaviour of the EA.

Infeasible individuals are penalised, without using penalties, as they are not preferred as parents, and because of the same preferences any feasible individual is better than an infeasible one. Also the EA appears to do a two-phase search but this is just a result of the mix of feasible/infeasible individuals in the population. The EA *always* trys to optimize feasible individuals and to produce feasible individuals from infeasible ones, but of course feasible individuals must first be found if none are in the initial population.

As mentioned earlier, the idea of matching parents is not new to EAs [19], but the idea of matching an infeasible parent with one that satisfies the constraints it does not, is novel for EAs. The utility of this and the other preferences have not been tested separately.

The results show that for constrained numerical optimisation problems tested, comparable results to that of

TABLE II
CONGA RESULTS

			Self-adaptive		Fixed mutation	
TF	λ	feas.	mean	σ	mean	σ
G1	3.6	28,230	-14.89	4.5e-1	-14.71	6.8e-1
G2	2.8	28,230	8,475.40	1.3e3	8,673.74	1.5e3
G3	2.7	14,165	680.70	4.8e-2	681.51	6.7e-1
G4	1.4	100	8,248#	0	12,322#	0
G5	3.2	14,165	25.74	1.0e0	27.21	2.3e0

TABLE III
OPTIMISED CONGA RESULTS

	Self-adaptive mutation				
TF	λ	feas.	mean	σ	feas %
G1	1	14,165	-15.00	4.3e-4	46%
G2	3.5	42,295	7,804.33	5.7e2	19%
G3	4	14,165	680.72	5.6e-2	63%
G4	1.5	100	12,322#	0	99%
G5	3.5	14,165	25.61	5.7e-1	46%

GENOCOP (the best of the evolutionary algorithms so far) can be obtained by using more sophisticated selection techniques rather than using penalty functions and repair methods.

One of the interesting experimentally determined parameter settings for CONGA is the replacement rate of the population. The value of 97% seems quite high from previous experience with unconstrained optimisation. A possible reason for this high setting could be that the high setting allows a greater percentage of infeasible solutions to exist during the second phase of the search, and hence increase the probability of infeasible individuals being chosen for reproduction. It would appear that this is advantageous as infeasible individuals can carry useful information. Another reason could be that such a high setting minimises the risk of getting stuck at a local optimum, by keeping a smaller history of the good individuals already found, and hence encouraging greater exploration.

While the results are preliminary, the EA developed shows promise and further research is warranted to see what other improvements can be obtained by refining the preferences and by using more sophisticated reproduction operators.

TABLE IV
COMPARISON WITH GENOCOP

	GENOCOP 2		GENOCOP 3	CONGA
TF	median	viol.	best	best
G1	-15.00	0	-15.00	-15.00
G2	8,206.15	0	7,286.65	7,083.21
G3	680.72	0	680.64	680.65
G4	0.064	0	—	0.054
G5	24.42	1	25.88	24.44

Acknowledgments:

This material is based upon work partially supported by the grants IRI-9322400, IRI-9725424 from the National Science Foundation.

References

[1] Bean, J. C. and A. B. Hadj-Alouane (1992). A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan.

[2] Chu, P.C. and J.E. Beasley (1997). Constraint handling in genetic algorithms: The set partitioning problem. The Management School, Imperial College, Technical Report.

[3] Homaifar, A., S. H.-Y. Lai, and X. Qi (1994). Constrained optimization via genetic algorithms. *Simulation 62*(4), 242–254.

[4] Joines, J. and C. Houck (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pp. 579–584. IEEE Press.

[5] Kappler, C. (1996). Are Evolutionary Algorithms Improved by Large Mutations? W. Ebeling, and H.-M. Voigt (Eds.), *Proceedings of the* 4th *Conference on Parallel Problems Solving from Nature*, pp. 346–355. Springer Verlag.

[6] Maa, C. and M. Shanblatt (1992). A two-phase optimization neural network. *IEEE Transactions on Neural Networks 3*(6), 1003–1009.

[7] Michalewicz, Z. (1995). Genetic Algorithms, Numerical optimization, and Constraints. In L. J. Eshelman (Ed.), *Proceedings of the* 6th *International Conference on Genetic Algorithms*, pp. 151–158.

[8] Michalewicz, Z. (1996). *Genetic Algorithms+Data Structures=Evolution Programs.* New-York: Springer Verlag. 3rd edition.

[9] Michalewicz, Z. and N. Attia (1994). Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp. 98–108. World Scientific.

[10] Michalewicz, Z. and C. Z. Janikow (1991). Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the* 4th *International Conference on Genetic Algorithms*, pp. 151–157. Morgan Kaufmann.

[11] Michalewicz, Z., T. Logan, and S. Swaminathan (1994). Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp. 84–97. World Scientific.

[12] Michalewicz, Z. and G. Nazhiyath (1995). Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In D. B. Fogel (Ed.), *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pp. 647–651. IEEE Press.

[13] Michalewicz, Z., G. Nazhiyath, and M. Michalewicz (1996). A note on usefulness of geometrical crossover for numerical optimization problems. In P. J. Angeline and T. Bäck (Eds.), *Proceedings of the 5th Annual Conference on Evolutionary Programming.*

[14] Michalewicz Z. and Schoenauer, M. (1996). Evolutionary computation for constrained parameter optimization problems. Evolutionary Computation, Vol.4, No.1, pp.1–32.

[15] Myung, H., J.-H. Kim, and D. Fogel (1995). Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pp. 449–463. MIT Press.

[16] Parmee, I. and G. Purchase (1994). The development of directed genetic search technique for heavily constrained design spaces. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control*, pp. 97–102. University of Plymouth.

[17] Powell, D. and M. M. Skolnick (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In S. Forrest (Ed.), *Proceedings of the* 5th *International Conference on Genetic Algorithms*, pp. 424–430. Morgan Kaufmann.

[18] Richardson, J. T., M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer (Ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 191–197. Morgan Kaufmann.

[19] Ronald, E. (1995). When Selection Meets Seduction. In L. J. Eshelman (Ed.), *Proceedings of the* 6th *International Conference on Genetic Algorithms*, pp. 167–173.

[20] Schaffer, D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the* 1st *International Conference on Genetic Algorithms.* Laurence Erlbaum Associates.

[21] Schoenauer, M. and Z. Michalewicz (1996). Evolutionary computation at the edge of feasibility. W. Ebeling, and H.-M. Voigt (Eds.), *Proceedings of the* 4th *Conference on Parallel Problems Solving from Nature*, Springer Verlag.

[22] Schoenauer, M. and Z. Michalewicz (1997). Boundary Operators for Constrained Parameter Optimization Problems. Proceedings of the 7th International Conference on Genetic Algorithms, July 1997.

[23] Schoenauer, M. and S. Xanthakis (1993). Constrained GA optimization. In S. Forrest (Ed.), *Proceedings of the* 5th *International Conference on Genetic Algorithms*, pp. 573–580. Morgan Kaufmann.

[24] Smith, A. and D. Tate (1993). Genetic optimization using a penalty function. In S. Forrest (Ed.), *Proceedings of the* 5th *International Conference on Genetic Algorithms*, pp. 499–503. Morgan Kaufmann.

[25] Surry, P., N. Radcliffe, and I. Boyd (1995). A multi-objective approach to constrained optimization of gas supply networks. In T. Fogarty (Ed.), *Proceedings of the AISB-95 Workshop on Evolutionary Computing*, Volume 993, pp. 166–180. Springer Verlag.

[26] Waagen, D., P. Diercks, and J. McDonnell (1992). The stochastic direction set algorithm: A hybrid technique for finding function extrema. In D. B. Fogel and W. Atmar (Eds.), *Proceedings of the 1st Annual Conference on Evolutionary Programming*, pp. 35–42. Evolutionary Programming Society.